

código.py

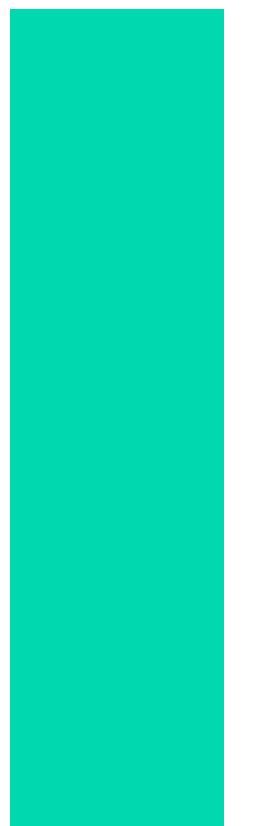
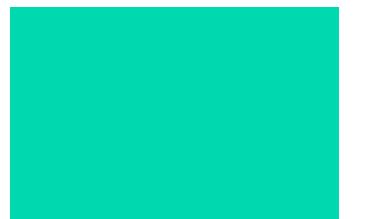
GALÁXIA 13

Machine Learning
e Deep Learning



Introdução

Seja muito bem vindo a nossa última galáxia do curso! Ao fim deste módulo você vai saber tudo que precisa sobre inteligência artificial, Machine Learning e Deep Learning, além disso estará mais do que pronto para aplicar seus conhecimentos de Python em todas as áreas e projetos relacionados a finanças. Bora lá!



Mundo 1

O que é IA? Ela oferece riscos à raça humana?

A Inteligência artificial originalmente foi desenvolvida no início do século XX, mas só agora conseguimos colocar em prática com as ferramentas que possuímos. Existem também dois tipos de IA:

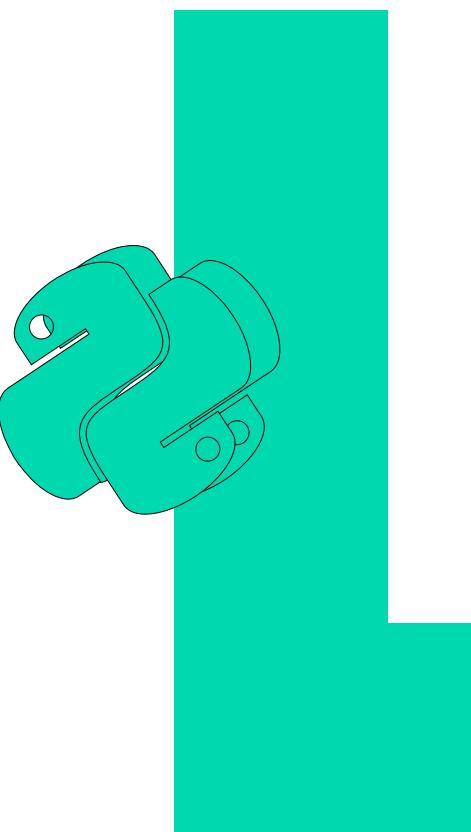
IA forte

- Autoconsciência.
- É capaz de observar diferentes problemas e resolvê-los.
- É capaz de ver problemas diferentes que foi treinado e resolver a partir de conhecimento prévio. Ela juntou conhecimento de outras áreas.
- É capaz de se auto treinar e criar outras IA mais inteligentes que ela.

IA Fraca

- É limitada a um escopo.
- Não consegue resolver o que não treinou.
- É uma função matemática que recebe um input, otimiza via pesos e forma um output.
- Todas são fracas e a IA forte foi abandonada.

Hoje em dia não tem risco: entenda que qualquer IA é fraca e é só um amontoado de vetores e funções que tentam buscar os melhores pesos para, dado um input, usar esses pesos para gerar um output que faça sentido. É uma grande otimização, muito mais simples do que parece ser.



Mundo 2

IA X Machine learning x Deep learning

Não são a mesma coisa, é muito importante entender a diferença entre elas.

- IA: Sei que todos estão acostumados a imaginar uma IA sendo algo muito complexo, mas na verdade são computadores que podem realizar qualquer tarefa antes feita por humanos de maneira autônoma.

Ex.:

- Ligar uma luz quando a luminosidade baixar ou enviar um e-mail automaticamente.
- Machine learning: Os computadores começam a aprender a tomar decisões que não estão programadas com IFs. Ele aprende com os dados.

Ex.:

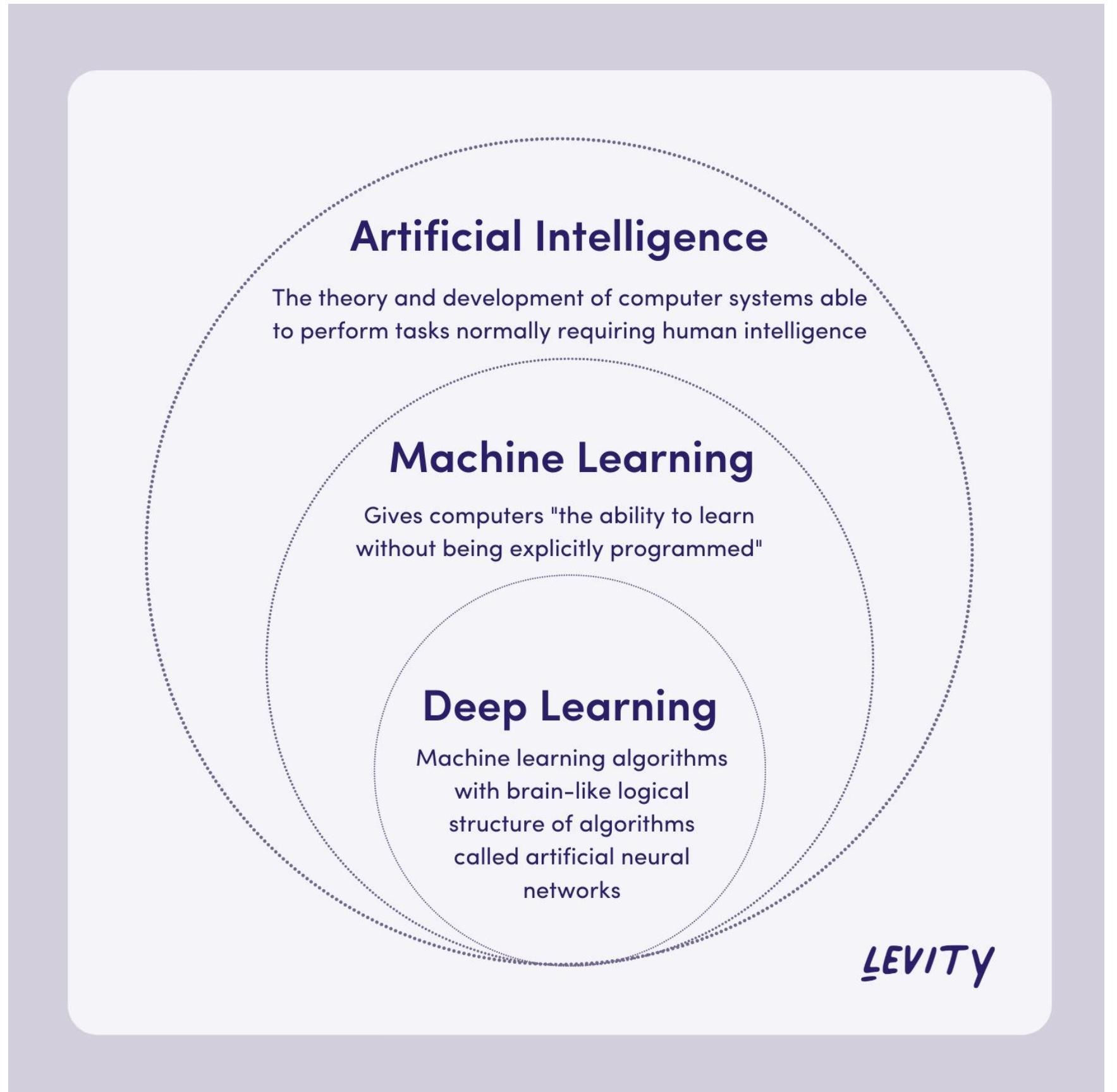
- Classificar se um e-mail é SPAM. Por tentativa e erro e muito treinamento com o algoritmo, reconhecendo padrões.
- Projetar a receita de uma empresa baseado em alguns parâmetros.

ML é uma IA que usa dados para aprender sobre a tarefa que foi designada.

- Deep learning: Algoritmos de machine learning que usam redes neurais para aprender tarefas

Ex.:

- ChatGPT.



Mundo 3

Existem 3 tipos de aprendizados: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço. Nenhum sendo melhor que o outro, são apenas tipos diferentes de problemas a serem resolvidos que vão demandar diferentes tipos de aprendizados.

Aprendizado supervisionado.

É o mais comum.

Exemplo de problema que resolve: círculo, quadrado ou triângulo? Separa a amostra em 70/30, nos 70 de treinamento você diz pra ele o que é o que e deixa ele tentar adivinhar nos outros 30.

O ChatGPT possui esse aprendizado, pois nada mais é que um completador de texto, por isso quanto melhor o input melhor será o output.

Aprendizado não supervisionado.

Geralmente associado com algoritmos de classificação, como classificar clientes.

Você dá um conjunto de dados pro algoritmo e ele tenta classificar esses dados seguindo alguma forma não óbvia, ou seja, ele se vira e tenta achar uma relação. Mas esses dados devem ter coisas objetivas.

Exemplo: Como posso separar meus clientes em 4 grupos?

Aprendizado por reforço.

Tem um sistema de recompensas que é usado para situações menos simples, como IA aprendendo a dirigir, quando acerta é recompensado e quando erra é penalizado. Na situação X, qual a melhor coisa a se fazer? O sistema de recompensa parece muito com o ser humano. É o mais utilizado atualmente.

Exemplo: Carro com piloto automático da Tesla.

Mundo 4

Tipos de modelos: Regressão x Classificação X Multi decisão.

Modelos de regressão.

Já foi mostrado anteriormente aqui no curso modelos de regressão, mas para relembrar, basicamente seria:

- Dados meus “Xs”, qual meu “Y”?
- Ou qual o preço da ação amanhã dado x indicadores?

Modelos de classificação.

Quando tem variáveis categóricas, usa-se esse modelo

- Dados meus Xs, é A ou B? (Ou é ou não é)
- Um cliente vai dar default no crédito ou não?
- Quantos grupos de clientes existem na minha base de consumidores?

Multi decisões.

Não vamos usar aqui no curso por ser muito complexo.

- ChatGPT: Um modelo multi decisão, com aprendizado supervisionado em deep learning.

Mundo 5

A coisa mais importante de IA

Quando for resolver seu problema, vai ter que fazer escolhas e entender a natureza do seu problema. Escolher o tipo de aprendizado e de modelo.

Porém, primeiro é preciso identificar qual é meu problema. Uma vez que você entender vai conseguir responder perguntas como:

- Como meu problema é classificado?
- Qual modelo utilizar?

No entanto, ninguém é vidente e sabe exatamente o que precisa, com isso, você pode testar diferentes modelos para uma mesma solução!

Mundo 6

Machine learning em finanças.

Vantagens:

- Aprender a tomar decisões melhores não programadas.
- Você não consegue identificar todos os padrões no mercado e como projetar certas coisas. Nem todas as relações do mercado são lineares.
- Explorar micro estruturas e reconstruir dados de alta frequência.

Desvantagens

- Falta de dados, principalmente fundamentalistas.
- Overfitting, mercado é incrivelmente sem padrão no curto prazo, que é onde existem mais dados.

Mundo 7

Como os projetos da galáxia foram organizados.

Projeto 1: Usando regressões simples que prevê o preço das ações.

- Aqui eu quero mostrar algo que já foi explicado em factor e como isso também é IA.

Projeto 2: Fazendo o mesmo projeto, mas agora usando redes neurais.

- Observar diferentes resultados de diferentes modelos para uma mesma situação.
- Entender como funcionam redes neurais e como deep learning é uma evolução da IA. Essa evolução vai ficar muito clara com um projeto depois do outro.

Projeto 3: Problemas de classificação. Vamos pro outro oposto - não supervisionado e com um problema categórico.

Projeto 4: Problema de classificação binária. Ou é ou não é. Muito usado em bancos para prever risco de crédito, afinal, ou uma pessoa paga ou ela não paga.

Projeto 1, Mundo 1: Prevendo o preço de ações.

Vamos criar o passo a passo de um modelo de investimento que usa IA para descobrir se uma ação vai cair ou subir no dia seguinte. A parte matemática com o código e com isso operar comprado ou vendido.

Também, teremos a parte teórica, aprendendo tudo que é necessário para criar um modelo desse tipo.

Projeto 1, Mundo 2: O que são regressões lineares?

A regressão linear

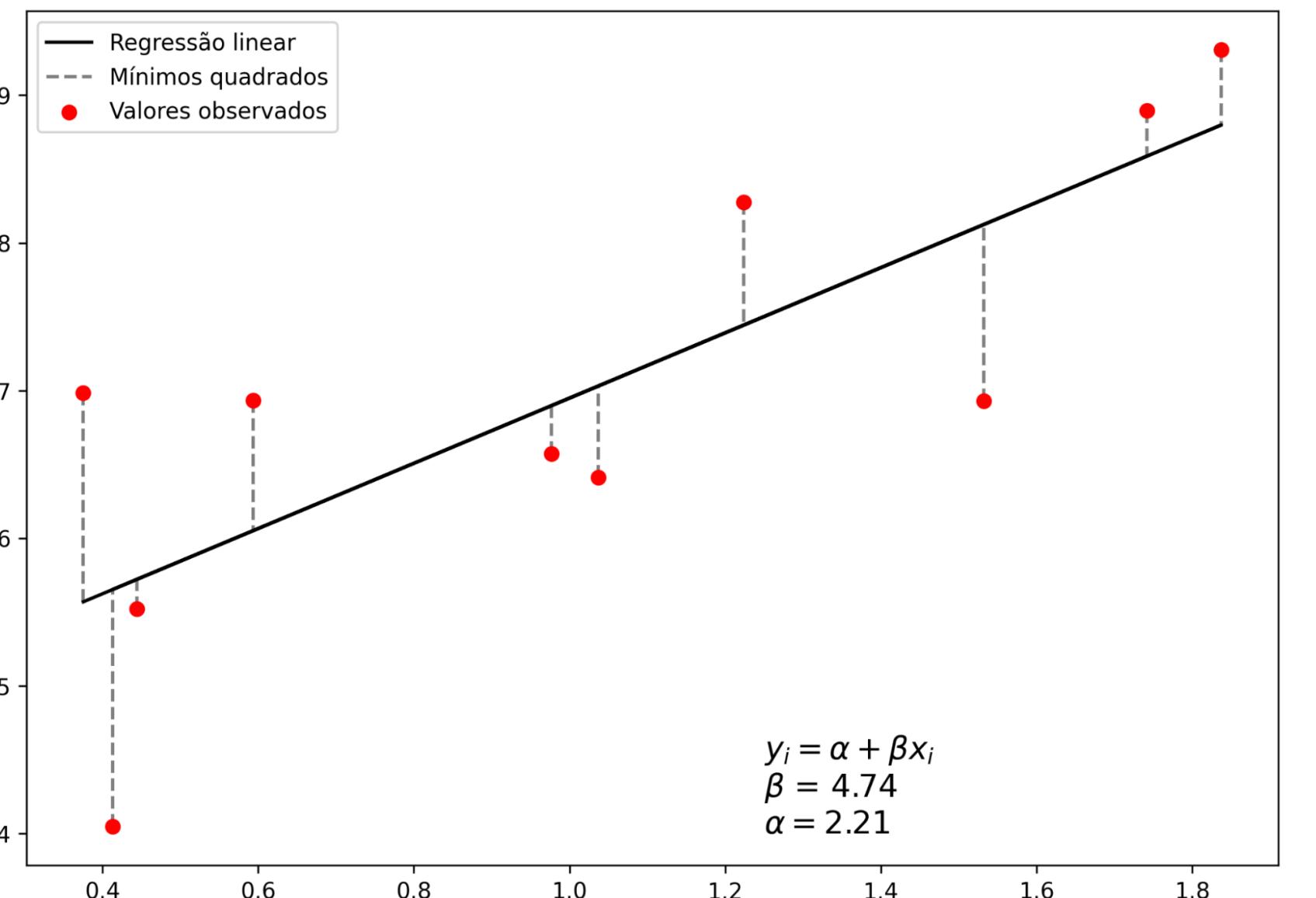
Regressão linear é um modelo estatístico poderosíssimo para prever ou explicar uma variável Y a partir de outras (X).

$$Y = \beta_0 + \beta_1 X + e$$

Variável resposta	Intecepto	Coeficiente angular	Variável explicativa	Erro
-------------------	-----------	---------------------	----------------------	------

Modelos mais complexos só aumentam a chance de overfitting.

Uma das formas de estimar os betas usando a regressão linear é utilizar o método de mínimos quadrados:



Nosso objetivo é minimizar a somatória do e^2 para traçar a reta mais próxima dos dados reais históricos.

A grande diferença entre um modelo de regressão em IA e um modelo de regressão em data science é que na IA, nós só estamos interessados em prever o Y, não na significância dos coeficientes.

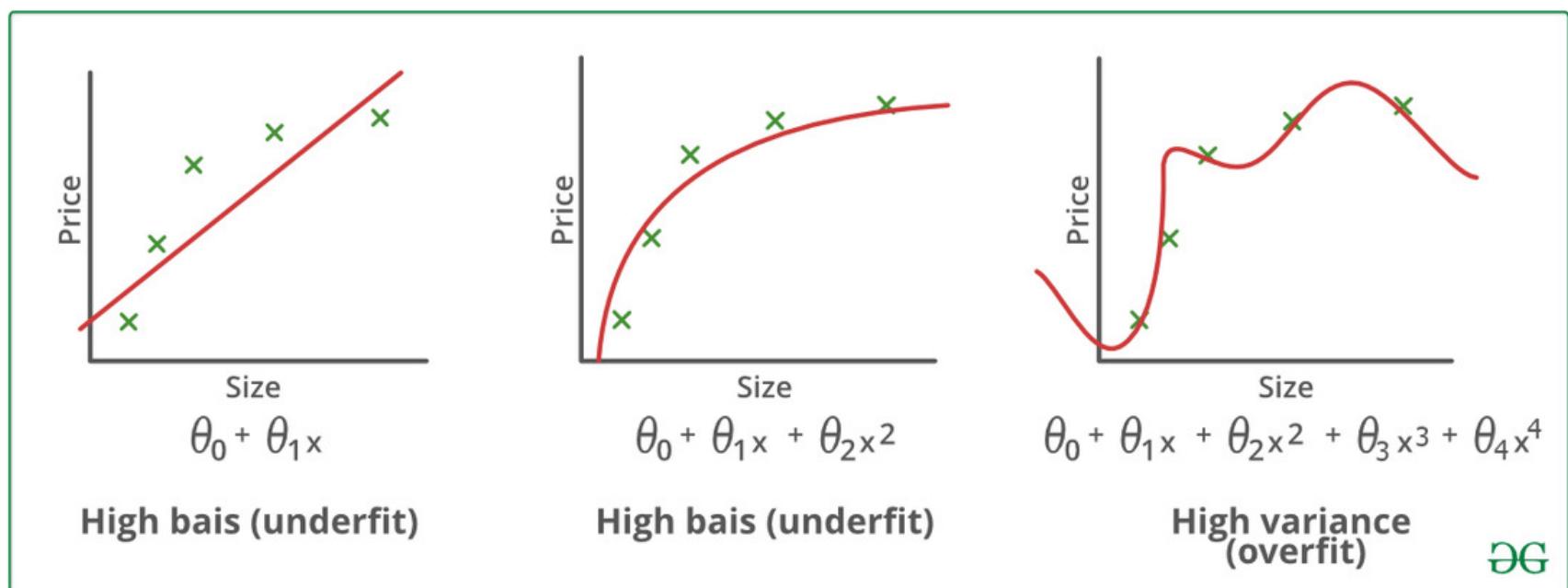
Nós não estamos buscando uma relação de causalidade, como foi no factor investing e é utilizado em estudos econômicos. **Nós precisamos, pura e simplesmente, prever o Y. Os motivos econômicos dos Xs não importam tanto.**

A gente precisa de um modelo que funcione, não um modelo que explique.

Isso às vezes anda junto, mas nem sempre. Até porque, um modelo que funciona só precisa acertar mais de 50% e ganhar mais do que perder, não precisa ser 95% do tempo assertivo.

Projeto 1, Mundo 3: Regressão múltipla e overfitting.

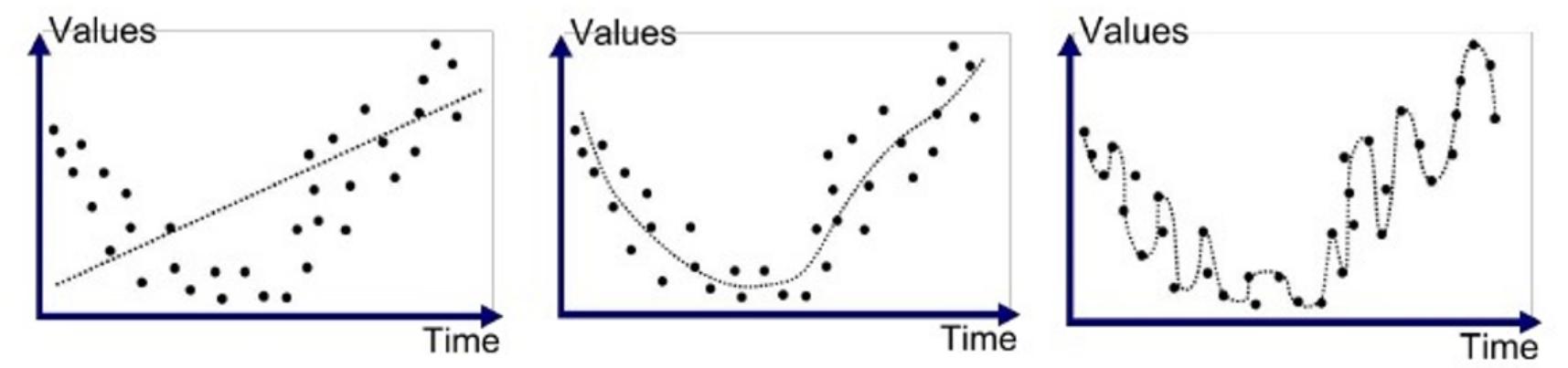
A regressão linear como o nome já diz, é uma linha, porém se você adicionar outros termos ela vira uma regressão múltipla. A desvantagem é que isso pode causar um overfitting.



Um Underfitted é quando a linha não consegue representar os dados corretamente, sendo um modelo muito pobre para explicar os dados.

No caso abaixo, seria ideal adicionar um termo quadrático para que a regressão consiga se encaixar melhor nos dados, mesmo não passando exatamente em cima de cada ponto, conseguindo identificar o padrão.

O modelo Overfitted passa em cima de cada ponto, mas não se engane, uma vez feito isso, não é possível prever os dados futuros, se tornando um modelo inútil.



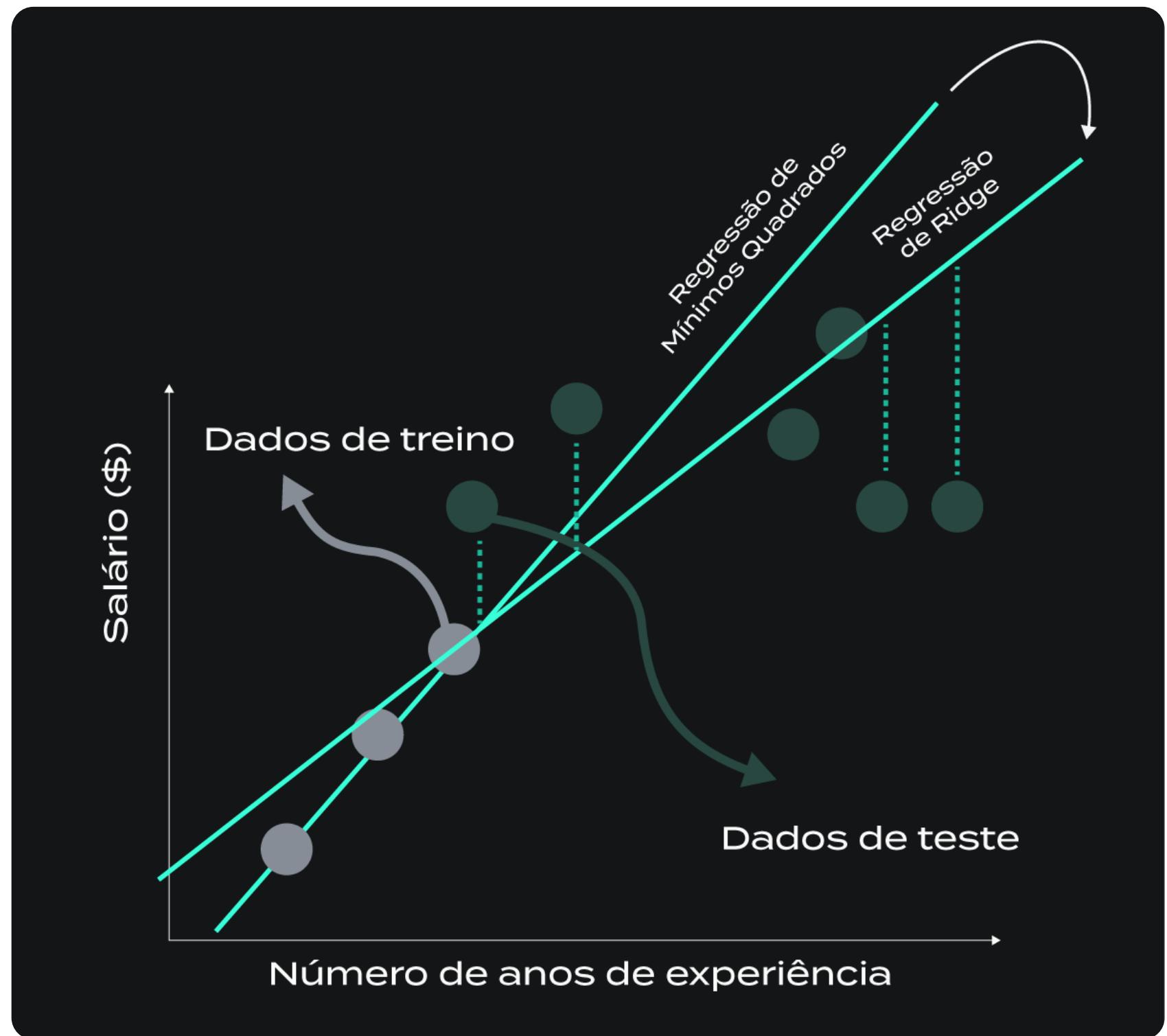
Aumentar o polinômio pode ajudar o fit, mas também aumenta a chance de overfitting.

Uma única equação é muito pouco para explicar fenômenos complexos.

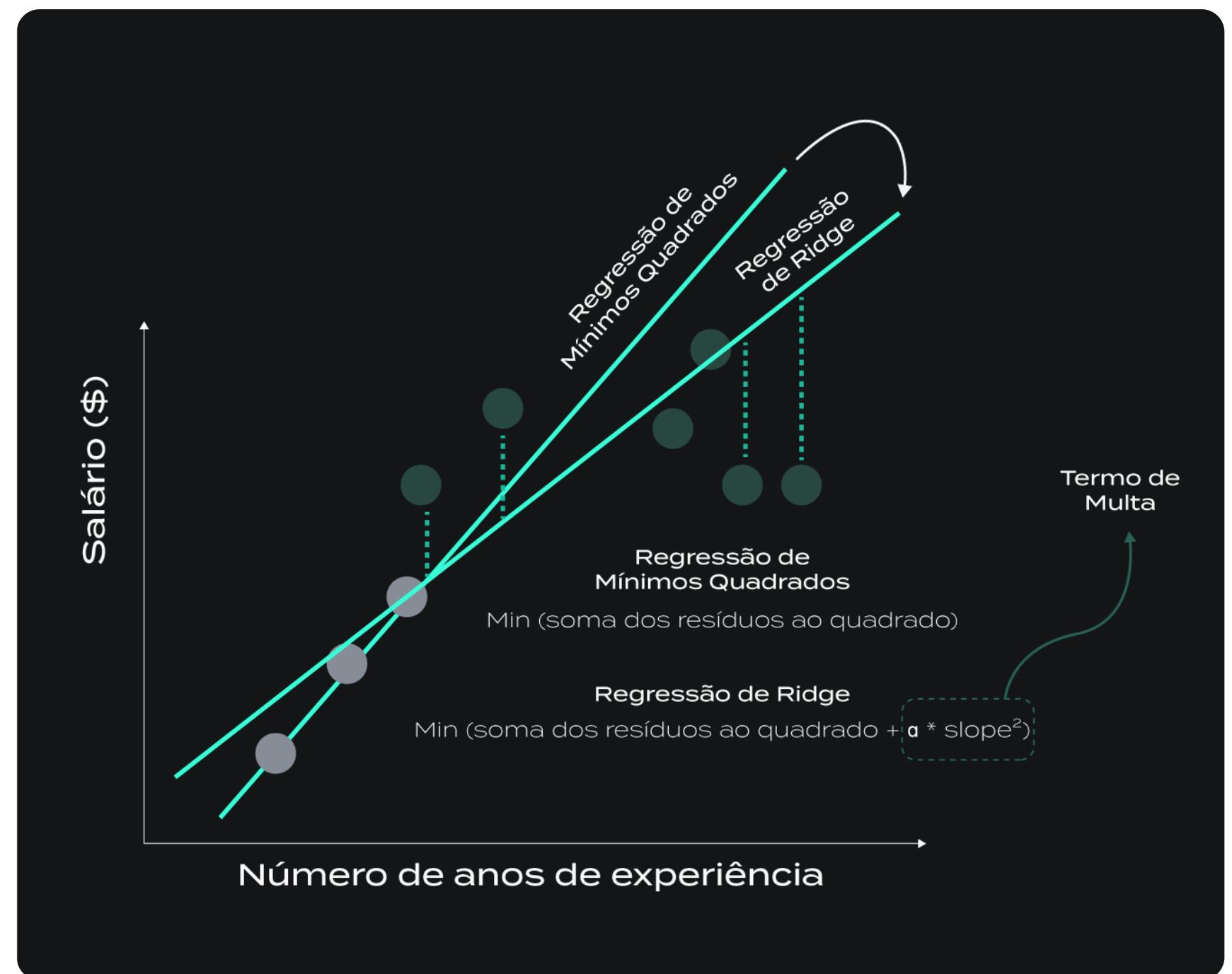
Ajuda na otimização: regressão de ridge

Projeto 1, Mundo 4: Regressão Ridge.

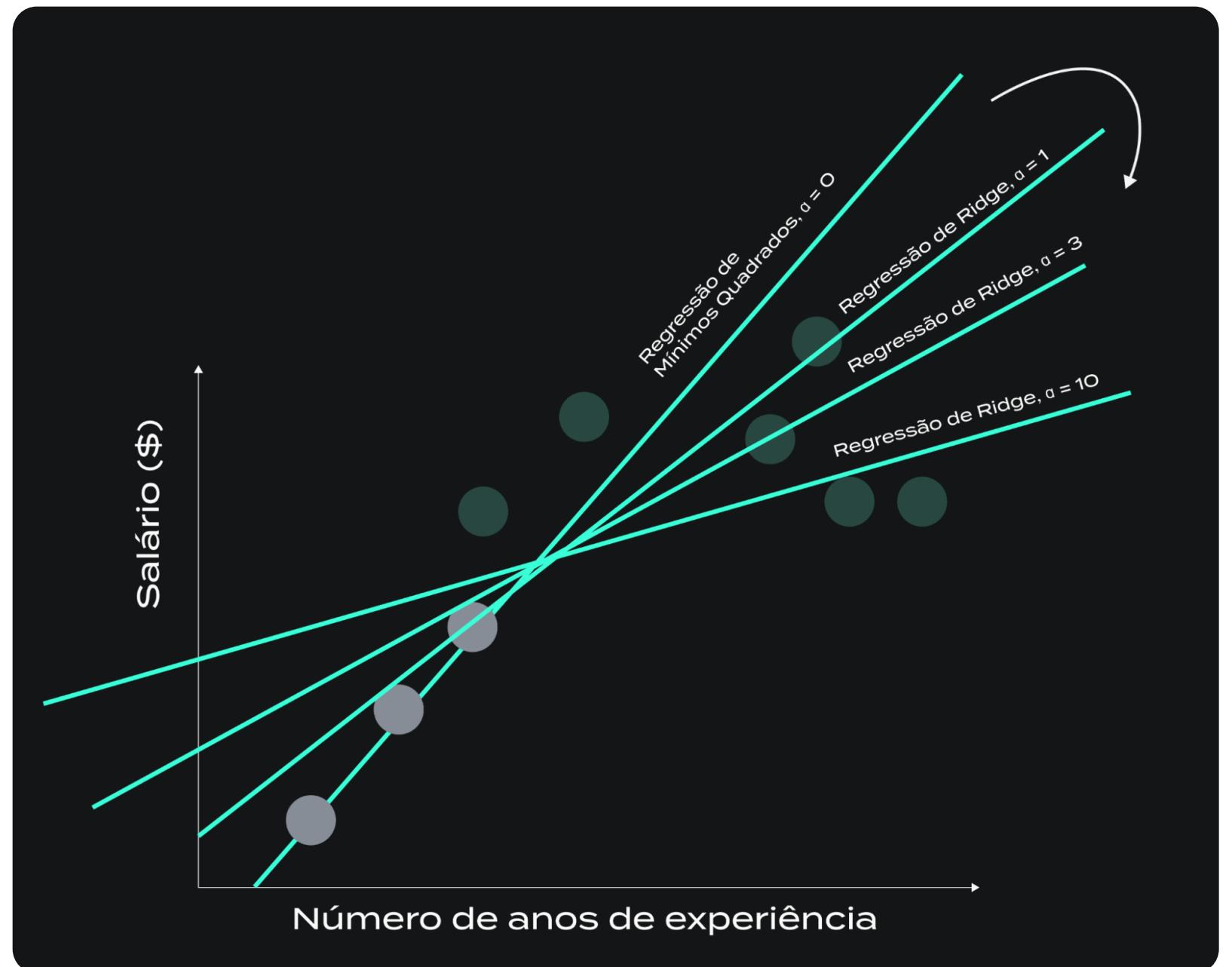
A regressão de ridge piora o fit em dados de treino para tentar pegar uma futura variância maior nos dados e ter uma projeção melhor nos dados de teste.



Isso vem através de um termo de penalização. Agora nós não estamos só minimizando o erro ao quadrado, estamos minimizando também esse termo que penaliza um super fit. (slope é a inclinação da curva).



Quanto maior nosso alfa, maior nossa inclinação. Isso pode ser es-
colhido por você.



A próxima aula é o projeto na prática.

Projeto 1, Mundo 5: Coletando os dados de cotação e separando entre dados de treino e teste.

Vamos pegar e organizar os dados

Para começar, escolha uma ação e pegue os dados de cotações da base de dados. Lembre de colocar o seu caminho do parquet.

```
cotacoes = pd.read_parquet(r' Seu_caminho_do_parquet')

acao_escolhida = 'MGLU3'

dados = cotacoes[cotacoes['ticker'] == acao_escolhida]
```

Pegue apenas duas colunas de dados, que são as duas únicas que vamos precisar.

```
datas = pd.to_datetime(dados['data'].iloc[:-1]).dt.date
dados = dados[['preco_fechamento_ajustado', 'volume_negociado']]
```

Criando o Y.

Iremos usar o preço da ação e o volume de um dia para prever a cotação do dia seguinte.

```
dados['cotacao_dia_seguinte'] = dados['preco_fechamento_ajustado'].shift(-1)
dados = dados.dropna()
```

Separar o data frame em 80/20 para fazer o teste.

```
tamanho_dados_treinamento = int(len(dados) * 0.8)
```

Aqui vale uma observação importante, treinar dados entre 0 e 1 tende a ser melhor. Por isso aqui é necessário transformar a escala dos dados para que fiquem dentro de 0 e 1. Sendo 1 a maior cotação e 0 a menor do período.

Para isso vai ser necessário também que escale individualmente, para não causar overfitting.

```
#Mas tem que escalar os dados de teste e treino individualmente!
escalador_treinamento = MinMaxScaler(feature_range=(0, 1))
escalador_teste = MinMaxScaler(feature_range=(0, 1))

dados_entre_0_e_1_treinamento = escalador_treinamento.fit_transform(dados.iloc[0: tamanho_dados_treinamento, :])

dados_entre_0_e_1_teste = escalador_teste.fit_transform(dados.iloc[tamanho_dados_treinamento: , :])
```

Separar o X e Y.

```
x_treinamento = dados_entre_0_e_1_treinamento[:,2:]
y_treinamento = dados_entre_0_e_1_treinamento[:,2:]

x_teste = dados_entre_0_e_1_teste[:,2:]
y_teste = dados_entre_0_e_1_teste[:,2:]
```

Projeto 1, Mundo 6: Treinando o modelo e estimando parâmetros para prever cotações.

Criando a primeira inteligência artificial com o modelo de regressão normal.

Usar .fit para treinar o modelo e colocar o x e y_treinamento para minimizar o erro desses dados.

```
regressao_linear = LinearRegression()
regressao_linear.fit(x_treinamento, y_treinamento)
```

Prever os preços utilizando os betas que foram estimados acima.

Otimizando o beta para chegar às cotações do dia seguinte o mais próximo possível.

```
score_treinamento = regressao_linear.score(x_treinamento, y_treinamento)
```

Crie um score de treinamento.

```
score_teste = regressao_linear.score(x_teste, y_teste)
```

Com os dados de teste e os dados preditos, concatenar o X e Y.

```
dados_teste = np.concatenate((x_teste, y_teste), axis=1)
dados_preditos = np.concatenate((x_teste, precos_preditos), axis=1)
```

Passe o inverse_transform para chegar nas cotações reais e retirar da escala de 0 e 1.

```
precos_teste_reais = escalador_teste.inverse_transform(dados_teste)
precos_teste_preditos = escalador_teste.inverse_transform(dados_preditos)
```

Faça um gráfico para visualizar a diferença entre os dados reais e preditos.

```
ax.plot(datas.iloc[tamanho_dados_treinamento:], precos_teste_reais[:, 2], label = 'Real')
ax.plot(datas.iloc[tamanho_dados_treinamento:], precos_teste_preditos[:, 2], label = 'Predito')
plt.legend()
```



Caso queira verificar a diferença em números entre os dois preços. No entanto, é preciso entender que os dados preditos nunca vão ser iguais ao valor da cotação, se apenas acertar se vai subir ou cair já é o suficiente.

```
np.around((precos_teste_reais[:, 2] - precos_teste_preditos[:, 2]), 2)
```

Projeto 1, Mundo 7: Fazendo previsões - A partir das cotações previstas é possível criar um modelo de investimento lucrativo? É possível acertar se a ação vai subir ou cair?

Crie um DataFrame com os preços, volume e preços preditos.

```
df = pd.DataFrame(precos_teste_preditos, index = datas.iloc[tamanho_dados_treinamento:])
df.columns = ['preco', 'volume', 'preco_predito_dia_seguinte']
```

Crie uma coluna de retorno e outra para mostrar se está comprado ou vendido.

```
df['retorno'] = df['preco'].pct_change()
df['comprado_vendido'] = pd.NA
df.loc[df['preco_predito_dia_seguinte'] < df['preco'], 'comprado_vendido'] = 'vendido'
```

Faça uma função para definir quando ficar comprado e quando ficar vendido.

```
df.loc[df['preco_predito_dia_seguinte'] > df['preco'], 'comprado_vendido'] = 'comprado'
df.loc[df['preco_predito_dia_seguinte'] < df['preco'], 'comprado_vendido'] = 'vendido'
```

Crie condições para acertos e erros.

```
df['acertos'] = pd.NA

df.loc[(df['comprado_vendido'] == 'comprado') & (df['retorno'] > 0), 'acertos'] = 1
df.loc[(df['comprado_vendido'] == 'comprado') & (df['retorno'] < 0), 'acertos'] = 0
df.loc[(df['comprado_vendido'] == 'vendido') & (df['retorno'] > 0), 'acertos'] = 0
df.loc[(df['comprado_vendido'] == 'vendido') & (df['retorno'] < 0), 'acertos'] = 1
df.loc[df['acertos'].isna(), 'acertos'] = 0

df = df.dropna()
```

Calcule qual a porcentagem de acertos.

```
acertou_o_lado = df['acertos'].sum()/len(df)
errou_o_lado = 1 - acertou_o_lado
```

Crie uma coluna com o retorno absoluto.

```
df['retorno_absoluto'] = df['retorno'].abs()
```

Calcule a média de lucros e perdas.

```
media_lucros_e_perdas = df.groupby('acertos')['retorno_absoluto'].mean()

media_lucros_e_perdas
```

Faça a expectativa matemática, que é a porcentagem das vezes que você acerta o lado vezes a média de lucros e perdas, menos a média de lucros e perdas

```
exp_mat_lucro = acertou_o_lado * media_lucros_e_perdas[1] - media_lucros_e_perdas[0] * errou_o_lado

exp_mat_lucro * 100
```

Calcule o retorno acumulado.

```
df['retorno_modelo'] = pd.NA

df.loc[df['acertos'] == True, 'retorno_modelo'] = df.loc[df['acertos'] == True]['retorno_absoluto']
df.loc[df['acertos'] == False, 'retorno_modelo'] = df.loc[df['acertos'] == False]['retorno_absoluto'] * -1

df['retorno_acum_modelo'] = (1 + df['retorno_modelo']).cumprod() - 1
df['retorno_acum_acao'] = (1 + df['retorno']).cumprod() - 1

retornos = df[['retorno_acum_modelo', 'retorno_acum_acao']]

retornos.plot()
display(retornos)
```

É extremamente importante entender alguns pontos:

- Principalmente em modelos apertados, com médias de ganhos e perdas bem parecidas, é preciso considerar o custo de corretagem, caso tenha.
- É preciso ter o preço de entrada certo, exatamente o preço que executou a ordem.
- Se a expectativa matemática for negativa, eventualmente você vai quebrar, mesmo que tenha um modelo em que o retorno acumulado esteja positivo. Confie na expectativa matemática.
- Sempre quanto mais dados melhor, evite pegar uma empresa nova com poucos dados.

Projeto 1, Mundo 8: Treinando e avaliando nossa IA que usa Ridge: As cotações previstas são melhores? O modelo é mais lucrativo?

Rode o mesmo modelo anterior mas agora usando Regressão Ridge. Escolha o Alpha.

```
regressao_ridge = Ridge(alpha = 3) #mudar o alpha  
regressao_ridge.fit(x_treinamento, y_treinamento)
```

O resto é exatamente o mesmo código da aula anterior, no entanto, agora usando os dados da Regressão Ridge. Apenas rode tudo novamente e veja como os valores finais mudam.

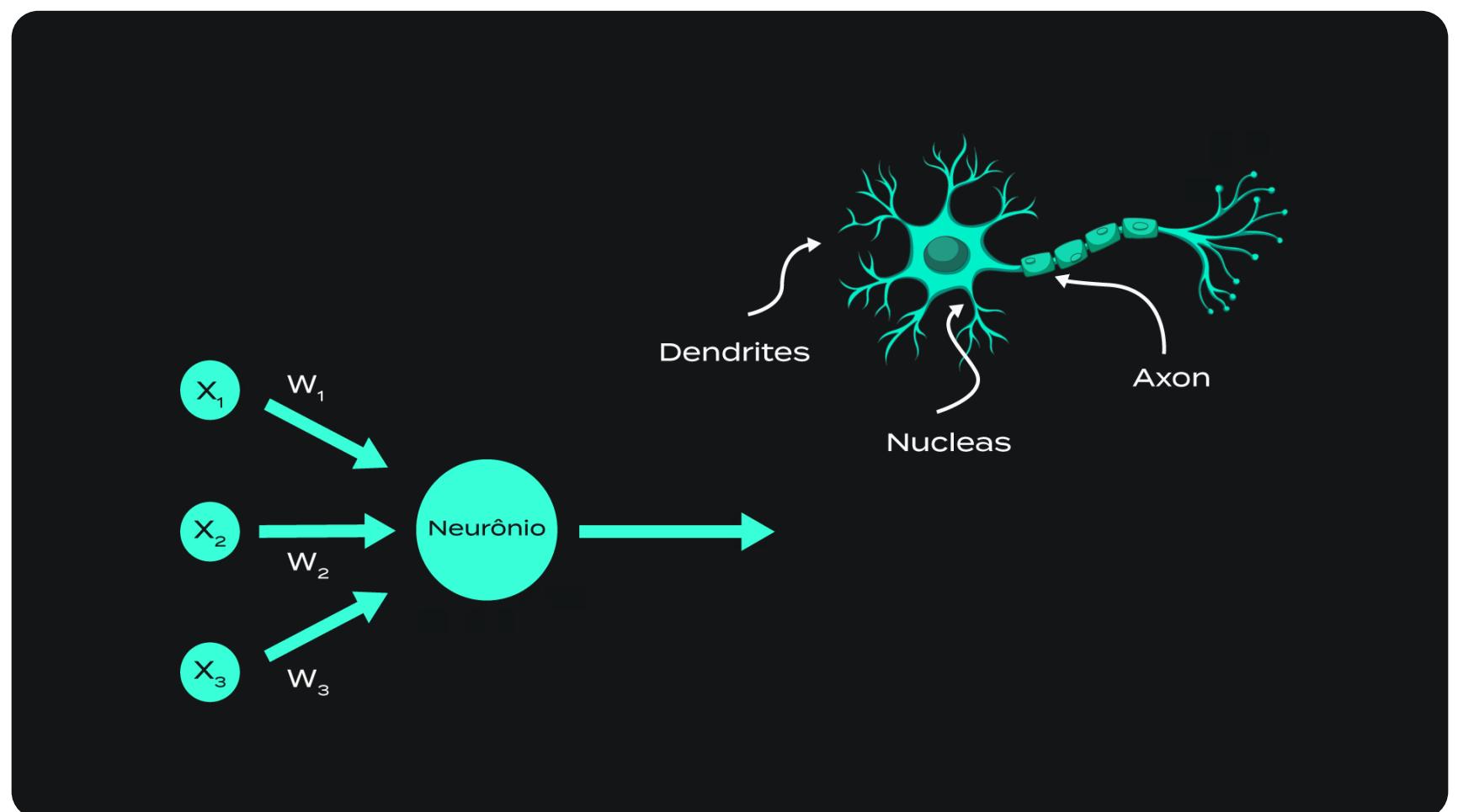
Nem sempre usar essa regressão pode gerar os melhores resultados, mas não deixa de ser uma ótima ferramenta para criar um modelo de Machine Learning.

Projeto 2, Mundo 1: Usando redes neurais para prever o retorno de uma ação - O passo a passo de um modelo que usa uma rede neural para estimar o retorno de uma empresa no futuro.

Nesse projeto vamos fazer basicamente a mesma coisa que fizemos no projeto 1, na qual conseguimos prever os preços das ações, no entanto, agora vamos criar Redes Neurais e comparar os resultados no final e descobrir se o modelo de Rede Neural é melhor que o modelo de Regressão.

Projeto 2, Mundo 1: Usando redes neurais para prever o retorno de uma ação - O passo a passo de um modelo que usa uma rede neural para estimar o retorno de uma empresa no futuro.

Através das redes neurais nós tentamos simular, matematicamente, como funcionam os neurônios do nosso cérebro.



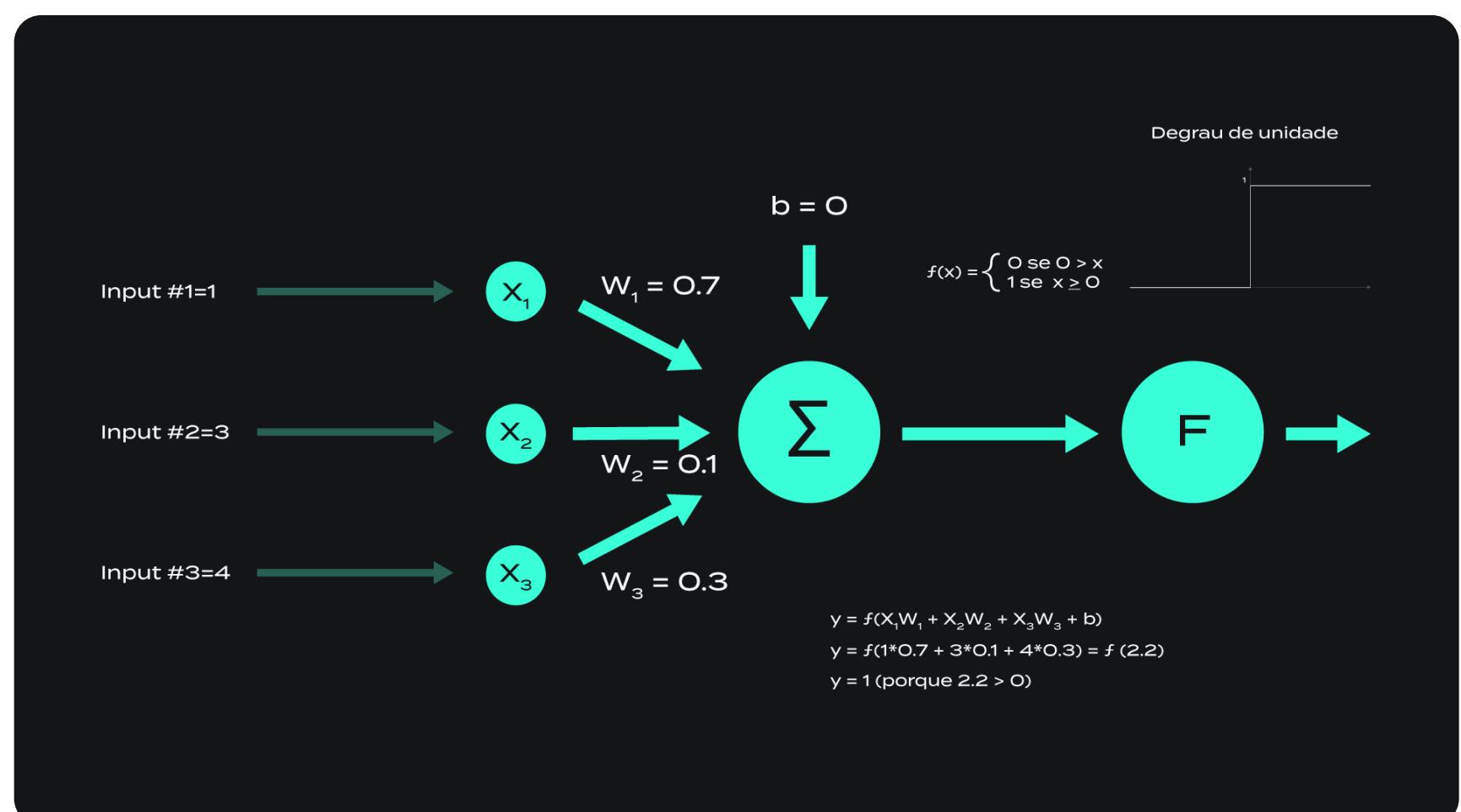
Inputs $Xs \rightarrow$ pesos ótimos \rightarrow soma tudo.

- Se for maior do que 1, ativa o neurônio. Menor do que 1, desativa.

Absolutamente tudo são pesos. Você vai otimizar os pesos para que, caso aconteça X, saia Y.

É uma grande regressão linear. São só pesos em vetores. A soma vai retornar a resposta.

O computador funciona de forma binária, a complexidade ta em usar milhares de neurônios.



Só que nesse caso, são dezenas de milhares de neurônios que se comunicam entre si, gerando respostas complexas através da ativação de x neurônios e uma matriz de 0 ou 1.

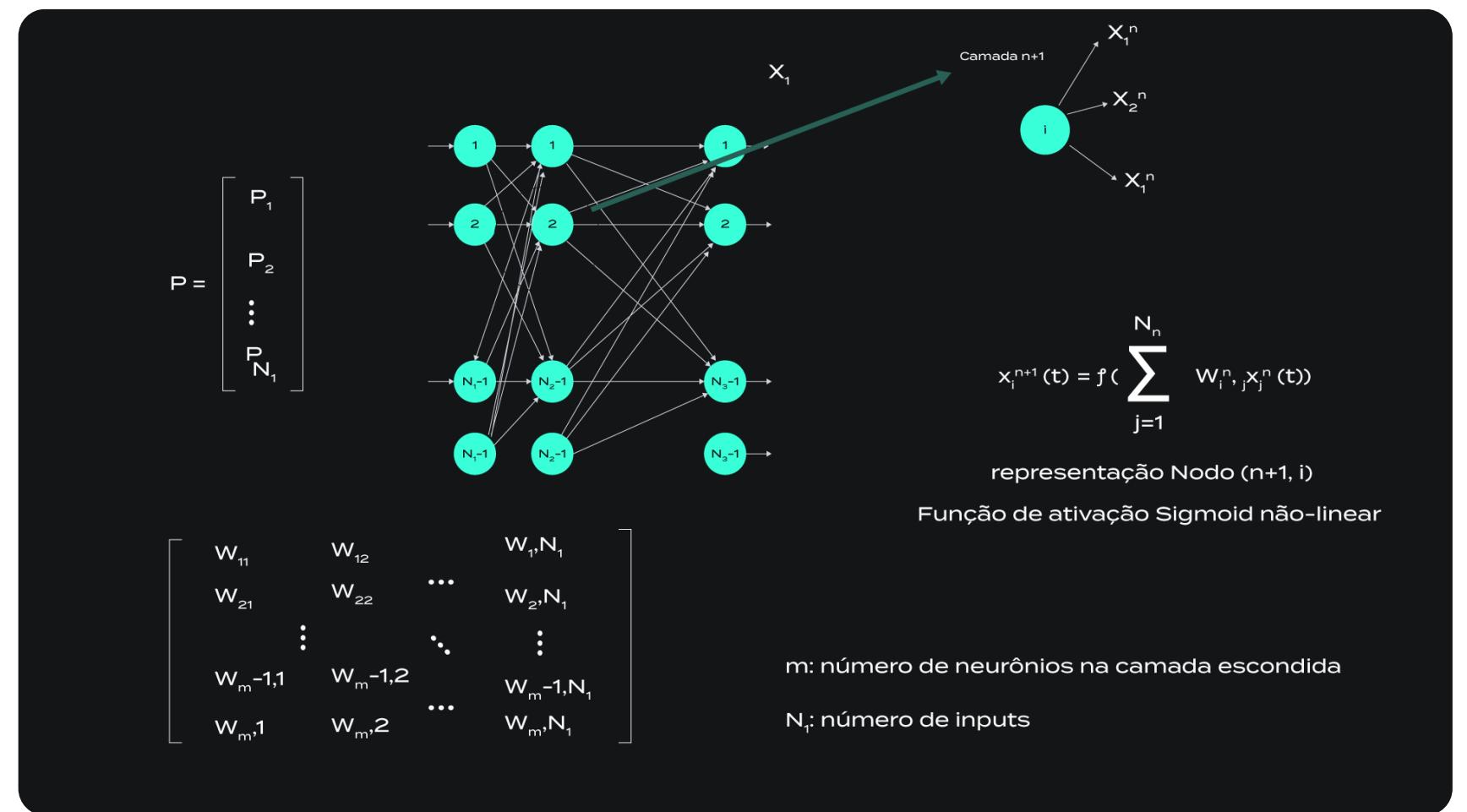
Tudo se resume a números, vetores e matemática!

Por isso chatGPT trabalha com tokens → cada palavra tem uma correspondência numérica. Ele sabe como completar o texto por que, dados os "Xs" do seu input, ele vai ativar a rede neural, multiplicar as palavras anteriores pelos pesos pré treinados e ativar X milhões de neurônios, chegando ao número 2056, que corresponde a uma palavra.

Por isso é literalmente impossível uma IA sair do controle ou ser realmente inteligente. Ela não consegue fazer nada que não foi treinada, é só um monte de peso pré pronto que multiplica os inputs e faz um somatório. Não tem magia. É uma otimização monstruosa que não cai em overfitting.

Ela não tem como gerar respostas para situações às quais ela não foi treinada, pois ela não sabe os pesos exatos para essa situação (input).

Perceba que quando você encara IA como uma simples otimização de pesos em uma matriz, a “magia” some, e você também entende porque IA existe desde o início do século XX mas não era posto em prática: precisava de um computador potente para fazer todas as contas e otimizar tantos pesos para gerar sistemas capazes de, a partir de diferentes inputs, gerar diferentes respostas certas e “inteligentes”.



Essa rede neural é uma ANN e ainda não tem memória. Todo output depende do input.

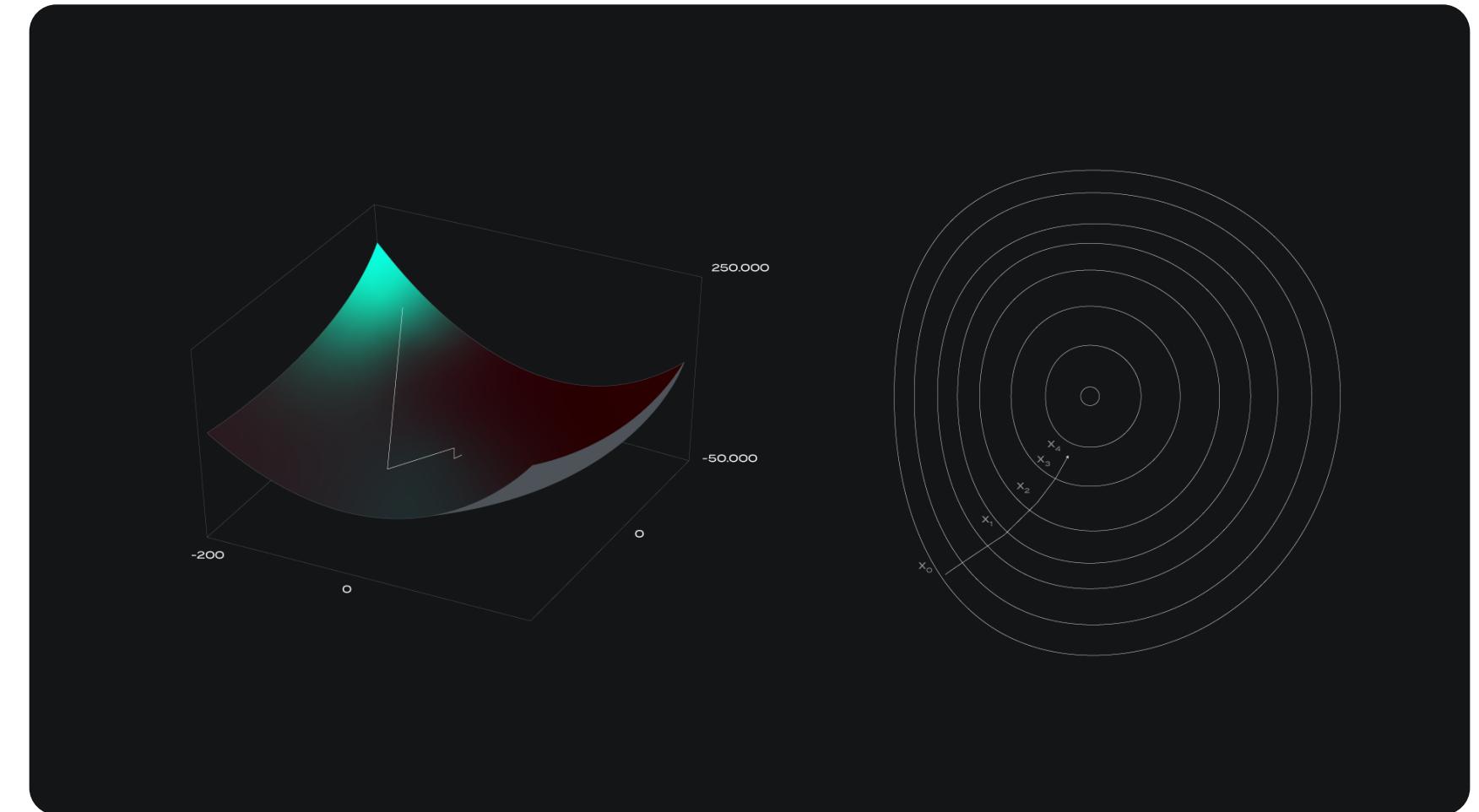
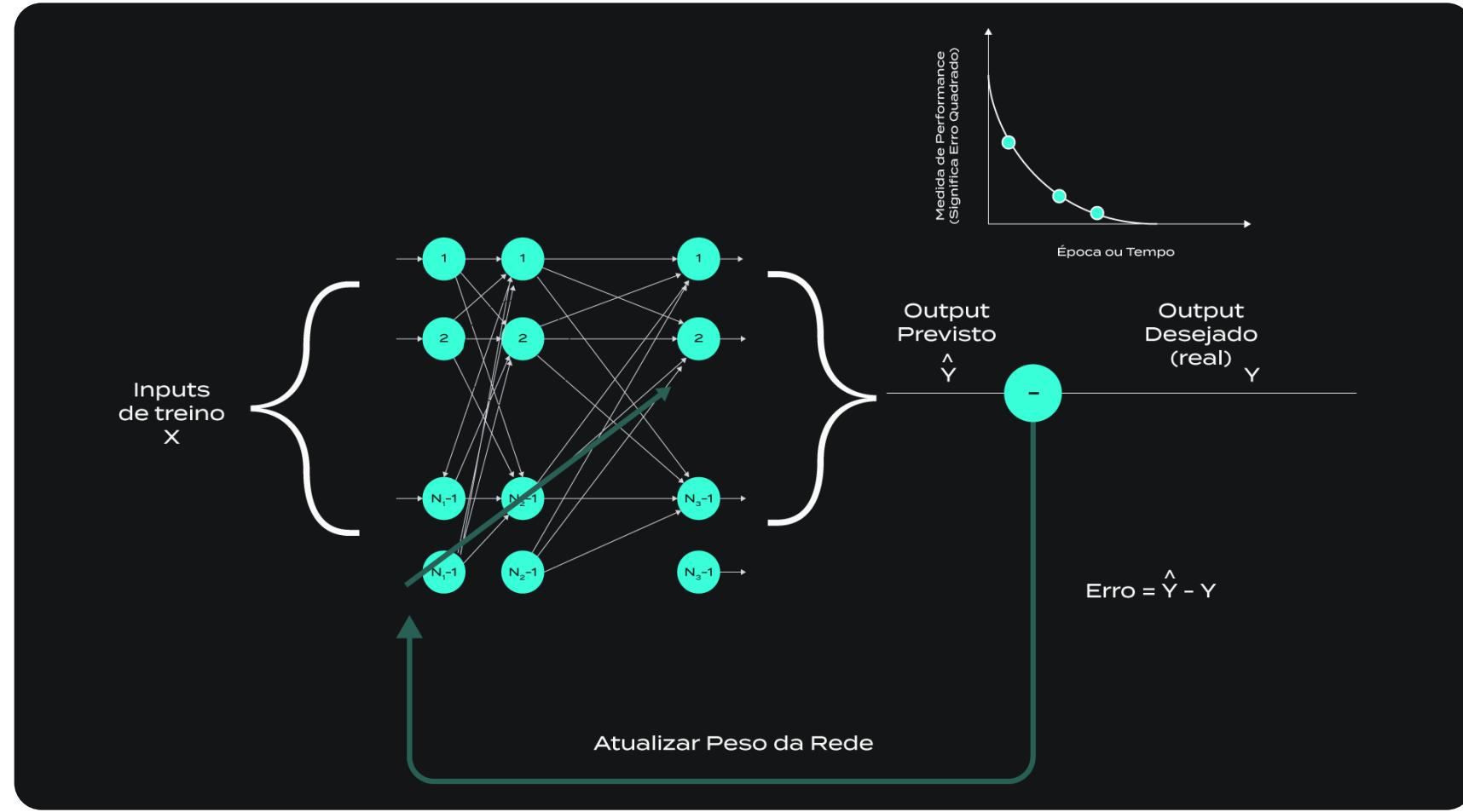
Link do Tensorflow para treinar redes neurais e entender como funcionam.

[Tensorflow – Neural Network Playground](#)

Projeto 2, Mundo 3: Como funciona a otimização dos pesos e o treinamento de uma rede.

O mesmo processo de uma regressão. Só que agora, invés de uma única equação, nós temos uma rede de pesos e somas de 0 ou 1.

Dado o X, eu gero um Y e meus pesos finais são os pesos que minimizam o erro pro Y real nos dados de treinamento.



Mas como chegar nos pesos ideais?

Basicamente tentativa e erro. Por isso o modelo começa a ser péssimo.

Conforme você muda os pesos, você vai minimizando o erro para o Y real nos dados de teste. Ela vai chutando pesos que chegam em mínimos locais e depois em mínimos globais.

Essa curva de aprendizado pode ser agressiva ou não.

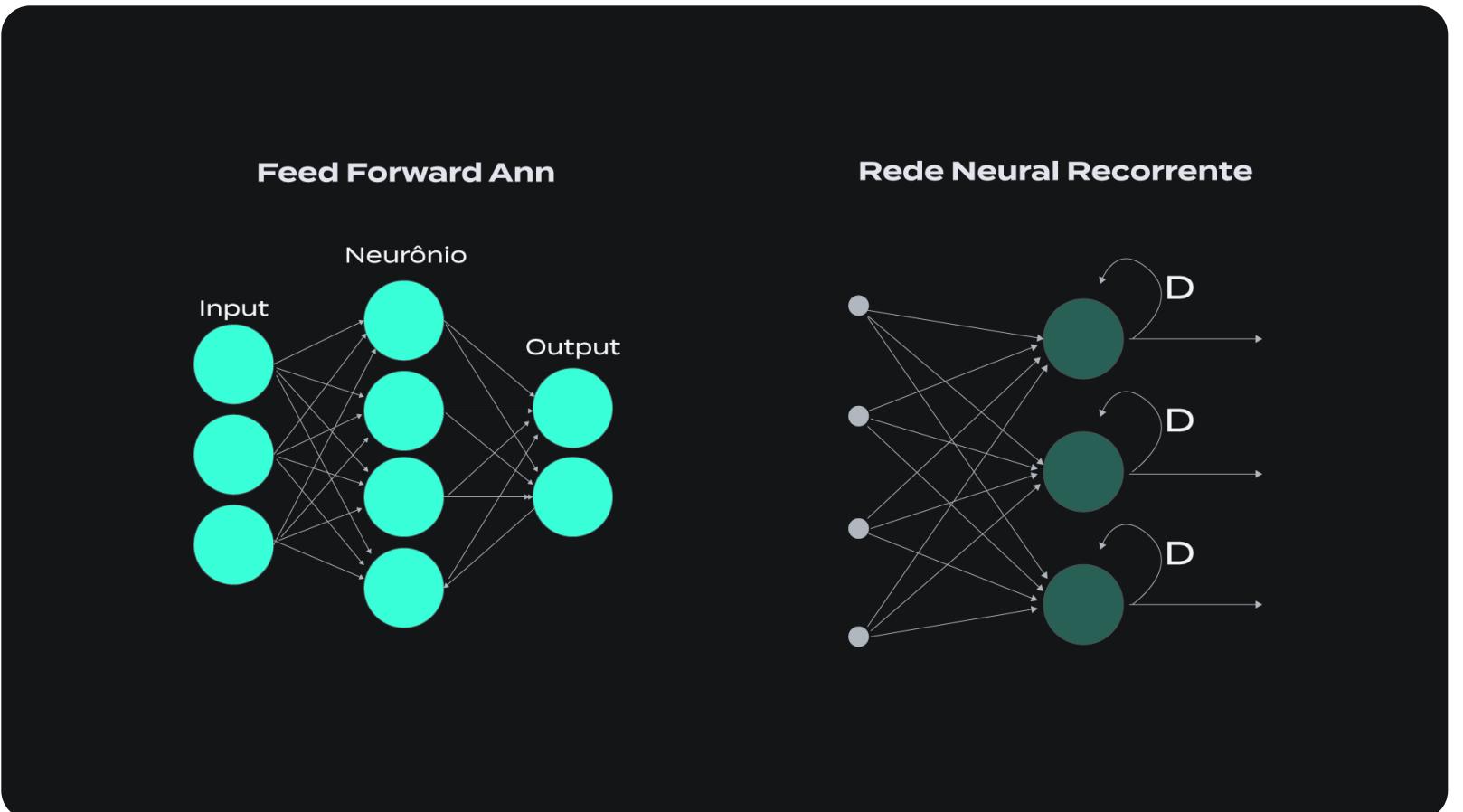
Ao mudar a "curva de aprendizado" para cada vez mais agressiva os valores vão mudar drasticamente de um teste pro outro, chegando mais rápido ao valor ótimo. Entretanto, você pode dar "overshoot" no valor ideal, ficando preso a um mínimo local por exemplo ou não conseguindo chegar exatamente no melhor valor. Com uma curva mais lenta você vai chegar mais devagar, mas tem menos chance de ficar preso em mínimos locais e você consegue chegar exatamente no valor ótico.

Projeto 2, Mundo 4: Como funciona a RNN - Agora temos memória.

ANN não tem memória. O output depende única e exclusivamente do input: obviamente nós usamos o passado para treinar os pesos, mas ele não tem mais influência sobre novos resultados. É uma relação direta: Input → pesos → Output.

Já a RNN tem memória e, além do input você vai ter também o próprio output do modelo como input! O output que aconteceu em $t-1$ também é levado em conta.

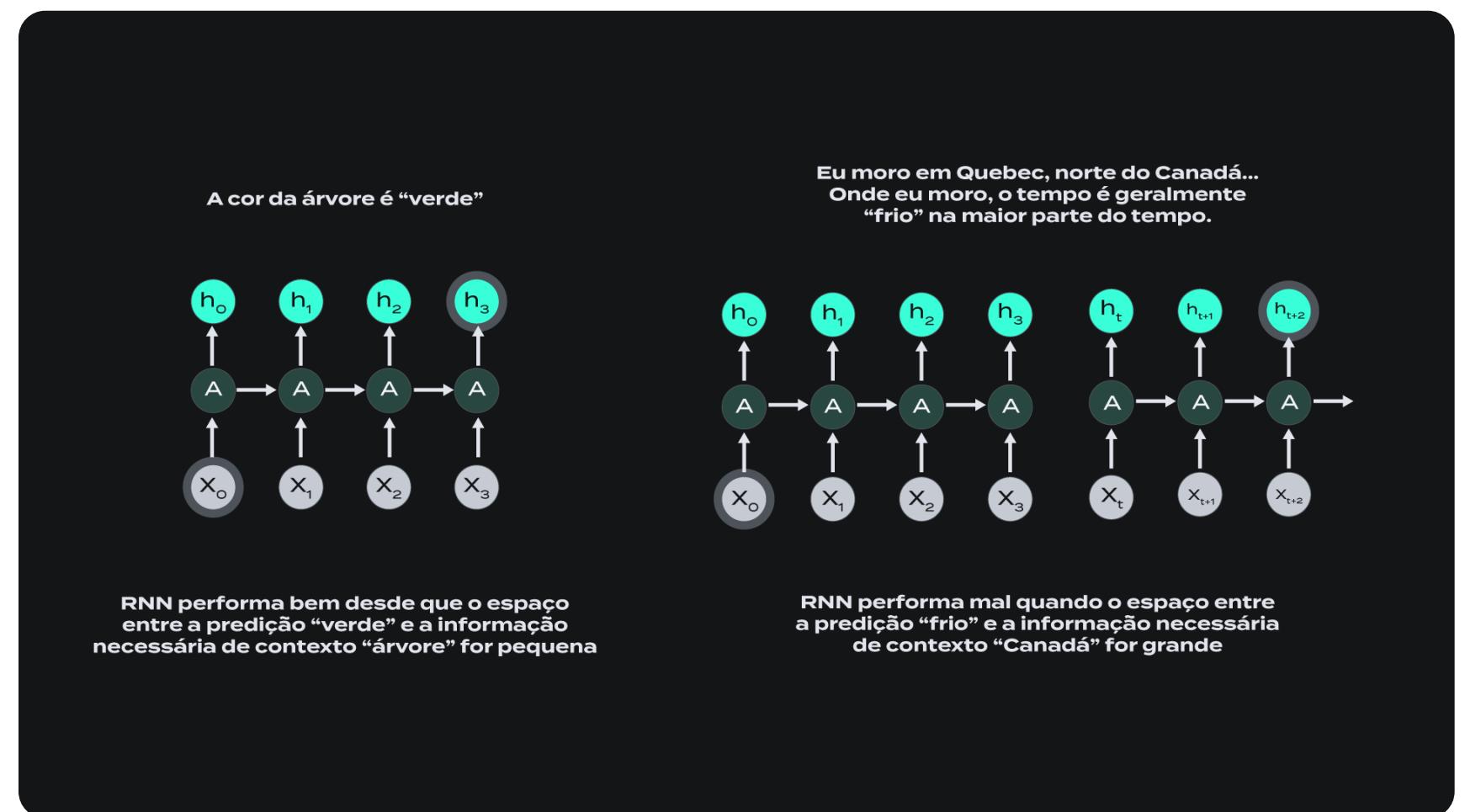
O ser humano também age assim: ele tem aprendido de outros momentos, mas cada situação é uma situação. Você leva os dois em consideração na hora de tomar decisões.



Só que tem 2 problemas em introduzir memória.

- Caso os primeiros outputs estejam errados, o erro pode se perpetuar dentro do modelo. Erros sistemáticos.
- Uma vez que novos pesos no treinamento são gerados através de um peso anterior e de um output anterior (memória), cada período que passa vai ficando mais difícil treinar a rede, os pesos praticamente não mudam pq já tem muito output influenciando. É basicamente um velho preso ao passado.

Devido a esses dois problemas, quando o output está muito longe do input, a rede tem problema, pois muita informação irrelevante é usada como input do próprio modelo e ele tem dificuldade em dar o veredito.



Para corrigir esse problema, surgiu a LSTM, um subtipo de RNN que optimiza até a memória.

Projeto 2, Mundo 5: O treinamento de uma RNN LSTM - Otimizando a memória.

Até agora nós otimizamos os pesos.

Depois, introduzimos a memória: nada é completamente novo, o passado pode ter interferência no que fazemos agora.

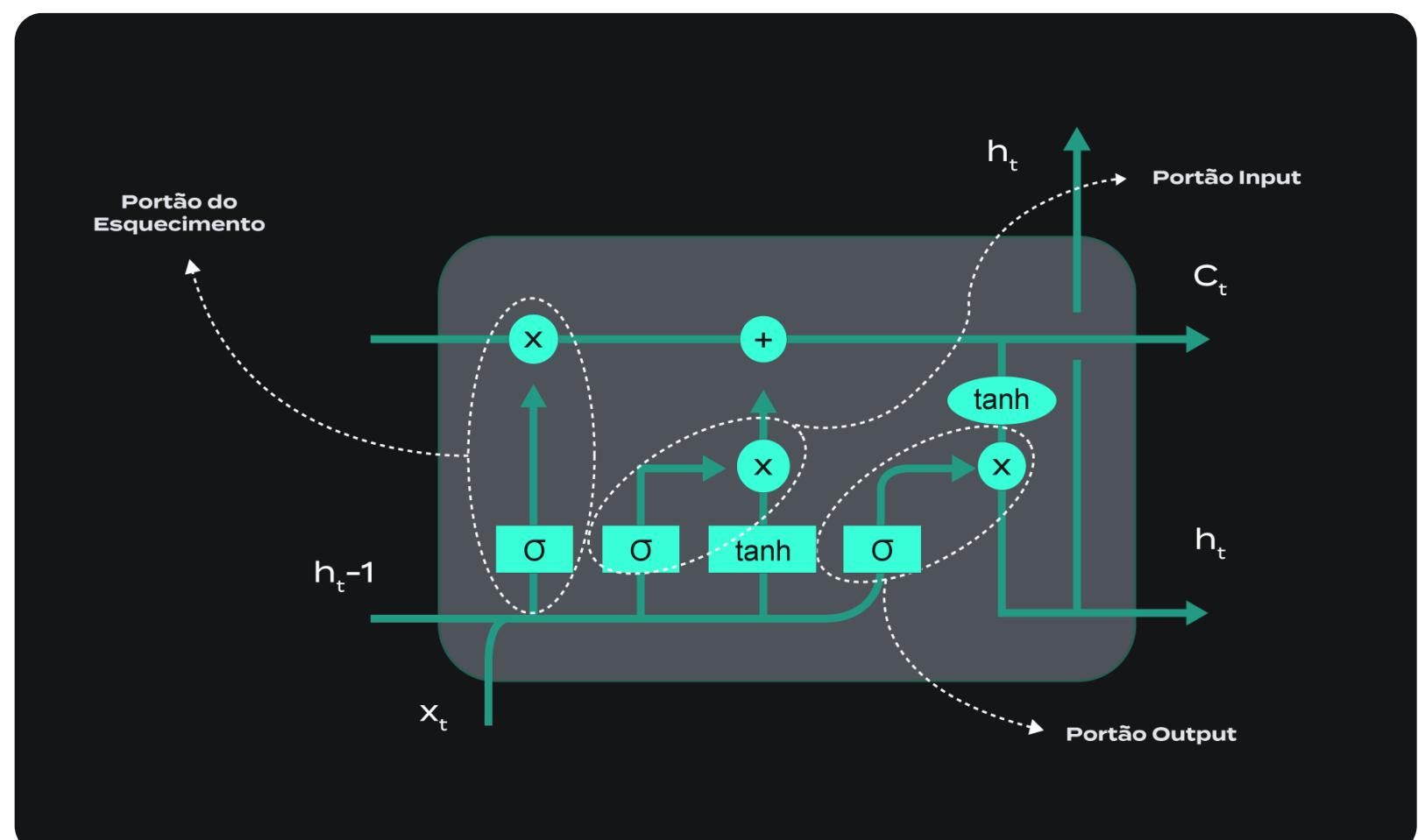
Só que nós precisamos controlar o quanto de informação nós vamos usar.

- Nós dirigimos no automático, sempre virando pra direita todos os dias em uma curva.
- Só que certo dia, esse caminho estava fechado e eu precisei virar para esquerda.
- No cenário 1, o passado teve mais peso. No cenário 2, os inputs.

É isso que a LSTM faz: usa o histórico mas sabe o “momento certo” e quanto utilizar para não perpetuar erros.

O modelo controla, através dos parâmetros:

- Quanto de informação histórica passa.
- Quanto de informação do input passa.
- Quanto de output é gerado pelo sistema.



Projeto 2, Mundo 6: Coletando os dados de cotação e separando entre dados de treino e teste.

Para esse projeto vamos usar a biblioteca do tensorflow para criar as redes neurais.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow import ...
import keras
```

Mesmo processo do projeto 1, mas dessa vez vamos pegar somente o preço de fechamento ajustado.

```
cotacoes = pd.read_parquet(r'C:\Users\lsiqu\dev\base_dados_br\cotacoes.parquet')

acao_escolhida = 'MGLU3'

dados = cotacoes[cotacoes['ticker'] == acao_escolhida]

datas = pd.to_datetime(dados['data'].iloc[:-1]).dt.date
dados = dados[['preco_fechamento_ajustado']]
```

Retorne os dados da cotação do dia seguinte.

```
dados['cotacao_dia_seguinte'] = dados['preco_fechamento_ajustado'].shift(-1)
dados = dados.dropna()
dados
```

Defina os dados de treinamento, aqui colocamos 0.8

```
tamanho_dados_treinamento = int(len(dados) * 0.8)
```

Fazer o mesmo processo anterior de escalar os dados entre 0 e 1. Lembrando de escalar os dados de teste e treino individualmente!

```
escalador_treinamento = MinMaxScaler(feature_range=(0, 1))
escalador_teste = MinMaxScaler(feature_range=(0, 1))

dados_entre_0_e_1_treinamento = escalador_treinamento.fit_transform(dados.iloc[0: tamanho_dados_treinamento, :])

dados_entre_0_e_1_teste = escalador_teste.fit_transform(dados.iloc[tamanho_dados_treinamento: , :])
```

Gere o resultado do x_treinamento, que também já foi mostrado anteriormente.

```
x_treinamento = dados_entre_0_e_1_treinamento[:,0]
y_treinamento = dados_entre_0_e_1_treinamento[:,1]
x_teste = dados_entre_0_e_1_teste[:,0]
y_teste = dados_entre_0_e_1_teste[:,1]
x_treinamento
```

Transforme o vetor em um vetor de três dimensões. É preciso fazer isso para criar as redes neurais

```
x_treinamento = x_treinamento.reshape(x_treinamento.shape[0], 1, 1)
y_treinamento = y_treinamento.reshape(y_treinamento.shape[0], 1, 1)
x_teste = x_teste.reshape(x_teste.shape[0], 1, 1)
y_teste = y_teste.reshape(y_teste.shape[0], 1, 1)
x_treinamento.shape
x_treinamento
```

Projeto 2, Mundo 7: O que são Epochs, Batch Size e validation_split.

Epochs

Em machine learning, "epochs" (épocas) referem-se às iterações completas de treinamento de um modelo em todo o conjunto de dados fornecido. Durante cada época, o algoritmo de treinamento passa por todo o conjunto de dados, ajusta os parâmetros do modelo com base nas informações presentes nos dados de treinamento e calcula a perda (ou erro) associada às previsões feitas pelo modelo em relação às saídas reais.

Cada época é composta por várias iterações, onde uma iteração geralmente consiste em alimentar um lote (subconjunto) dos dados de treinamento para o modelo, calcular o erro, atualizar os pesos do modelo por meio de um algoritmo de otimização (como o gradiente descendente) e repetir esse processo até que todos os lotes tenham sido usados.

Batch size: a divisão de lotes dentro de um epoch.

O "batch size" (tamanho do lote) em machine learning refere-se ao número de exemplos de treinamento que são usados em uma única iteração de treinamento. Durante o treinamento de um modelo, os dados de treinamento são divididos em lotes para processamento eficiente. Cada lote contém um número definido de exemplos de treinamento.

Exemplo:

1 época e 4 batch size → otimizar 4 vezes os pesos.

2 épocas e 4 batch size → otimizar 8 vezes os pesos.

O tamanho do lote tem um impacto na velocidade e eficiência do treinamento, bem como na dinâmica da otimização do modelo.

Atenção:

Consumo de memória: O tamanho do lote afeta o consumo de memória durante o treinamento. Lotes maiores podem exigir mais memória, portanto, é importante considerar os recursos disponíveis no seu sistema.

Validation Split:

Durante o treinamento, o modelo é ajustado nos dados de treinamento e avaliado nos dados de validação a cada época.

Isso ajuda a gerar métricas reais a cada epoch e evitar o overfitting.

Projeto 2, Mundo 8: Construindo e treinando nossa Rede Neural para prever os preços das ações com Deep Learning.

Criando a rede neural.

Assim como foi explicado antes, teremos os inputs, as redes neurais (x) e os outputs.

O primeiro valor em $x_{treinamento}.shape$ é a quantidade de x que vai ter no programa. Como passamos só um x , o valor será 1. Já o segundo valor se refere a quantos outputs devem ser gerados a partir desse x , que nesse caso também será 1.

Após isso, é preciso utilizar layers para construir a LSTM, a rede neural com memória otimizada, e colocar os neurônios de forma sequencial.

Na rede_neural coloque loss="mse", para ajudar a minimizar o erro quadrado das projeções

```
inputs = keras.layers.Input(shape=(x_treinamento.shape[1], 1))
x = keras.layers.LSTM(150, return_sequences= True)(inputs)
x = keras.layers.LSTM(150, return_sequences=True)(x)
x = keras.layers.LSTM(150, return_sequences=True)(x)
outputs = keras.layers.Dense(1, activation='linear')(x)

rede_neural = keras.Model(inputs=inputs, outputs=outputs)
rede_neural.compile(optimizer='adam', loss="mse")
rede_neural.summary()
```

Agora de fato, treine a rede neural.

```
rede_neural.fit(
    x_treinamento, y_treinamento,
    epochs = 2,
    batch_size = 32,
    validation_split = 0.2
)
```

Crie os preços preditos.

```
precos_preditos = rede_neural.predict(x_teste)
```

A partir daqui, é igual ao modelo de regressão.

```
precos_preditos = precos_preditos.reshape(precos_preditos.shape[0], 1)
x_treinamento = x_treinamento.reshape(x_treinamento.shape[0], 1)
y_treinamento = y_treinamento.reshape(y_treinamento.shape[0], 1)
x_teste = x_teste.reshape(x_teste.shape[0], 1)
y_teste = y_teste.reshape(y_teste.shape[0], 1)
```

Faça os dados de teste e dados preditos.

```
dados_teste = np.concatenate((x_teste, y_teste),axis=1)
dados_preditos = np.concatenate((x_teste, precos_preditos),axis=1)
```

Tire a escala dos dados.

```
precos_teste_reais = escalador_teste.inverse_transform(dados_teste)
precos_teste_preditos = escalador_teste.inverse_transform(dados_preditos)
```

Crie o DataFrame.

```
pd.DataFrame(precos_teste_preditos)
```

Plot o gráfico.

```
ax.plot(datas.iloc[tamanho_dados_treinamento:], precos_teste_reais[:, 1], label = 'Real')
ax.plot(datas.iloc[tamanho_dados_treinamento:], precos_teste_preditos[:, 1], label = 'Modelo')

plt.legend()
```

Projeto 2, Mundo 9: Comparando cotações reais x cotações previstas, avaliando resultados do modelo e descobrindo se ele é lucrativo.

Para começar, faça o mesmo processo do projeto passado,

```
df = pd.DataFrame(precos_teste_preditos, index = datas.iloc[tamanho_dados_treinamento:])

df.columns = ['preco', 'preco_predito_dia_seguinte']

df['retorno'] = df['preco'].pct_change()

df['comprado_vendido'] = pd.NA

df.loc[df['preco_predito_dia_seguinte'] > df['preco'], 'comprado_vendido'] = 'comprado'
df.loc[df['preco_predito_dia_seguinte'] < df['preco'], 'comprado_vendido'] = 'vendido'

df['acertos'] = pd.NA

df.loc[(df['comprado_vendido'] == 'comprado') & (df['retorno'] > 0), 'acertos'] = 1
df.loc[(df['comprado_vendido'] == 'comprado') & (df['retorno'] < 0), 'acertos'] = 0
df.loc[(df['comprado_vendido'] == 'vendido') & (df['retorno'] > 0), 'acertos'] = 0
df.loc[(df['comprado_vendido'] == 'vendido') & (df['retorno'] < 0), 'acertos'] = 1
df.loc[df['acertos'].isna(), 'acertos'] = 0

df = df.dropna()

df
```

Veja quantas vezes acertou e quantas errou o lado.

```
acertou_o_lado = df['acertos'].sum()/len(df)
errou_o_lado = 1 - acertou_o_lado
```

Faça o retorno absoluto.

```
df['retorno_absoluto'] = df['retorno'].abs()
```

Médias de lucros e perdas

```
media_lucros_e_perdas = df.groupby('acertos')['retorno_absoluto'].mean()

media_lucros_e_perdas
```

Crie a expectativa matemática do lucro.

```
exp_mat_lucro = acertou_o_lado * media_lucros_e_perdas[1] - media_lucros_e_perdas[0] * errou_o_lado

exp_mat_lucro * 100
```

Crie o gráfico com o retorno do modelo.

```
df['retorno_modelo'] = pd.NA

df.loc[df['acertos'] == True, 'retorno_modelo'] = df.loc[df['acertos'] == True]['retorno_absoluto']
df.loc[df['acertos'] == False, 'retorno_modelo'] = df.loc[df['acertos'] == False]['retorno_absoluto'] * -1

df['retorno_acum_modelo'] = (1 + df['retorno_modelo']).cumprod() - 1
df['retorno_acum_acao'] = (1 + df['retorno']).cumprod() - 1

retornos = df[['retorno_acum_modelo', 'retorno_acum_acao']]

retornos.plot()
display(retornos)
```

Às vezes a diferença entre usar rede neural e regressão linear vão ser mínimas, mas isso é normal, principalmente para casos mais “simples”.

No entanto, normalmente até mesmo nesses casos, a rede neural tende a ser superior.

Projeto 3, Mundo 1: Classificação K-MEANS: Criando uma IA que pode segmentar clientes em grupos.

Para esse projeto vamos pegar os dados do Kaggle. A maior comunidade de machine learning do mundo.

Utilizaremos dados bancários de 8 mil clientes.

Modelo não supervisionado.

O modelo K-MEANS tenta agrupar observações calculando a distância entre os pontos em um plano euclidiano. Se forem 2 variáveis, por exemplo, ele vai calcular a distância entre os pontos para tentar agrupá-los.

Projeto 3, Mundo 2: Passo a Passo de um algoritmo K-MEANS - Como otimizar a criação de grupos.

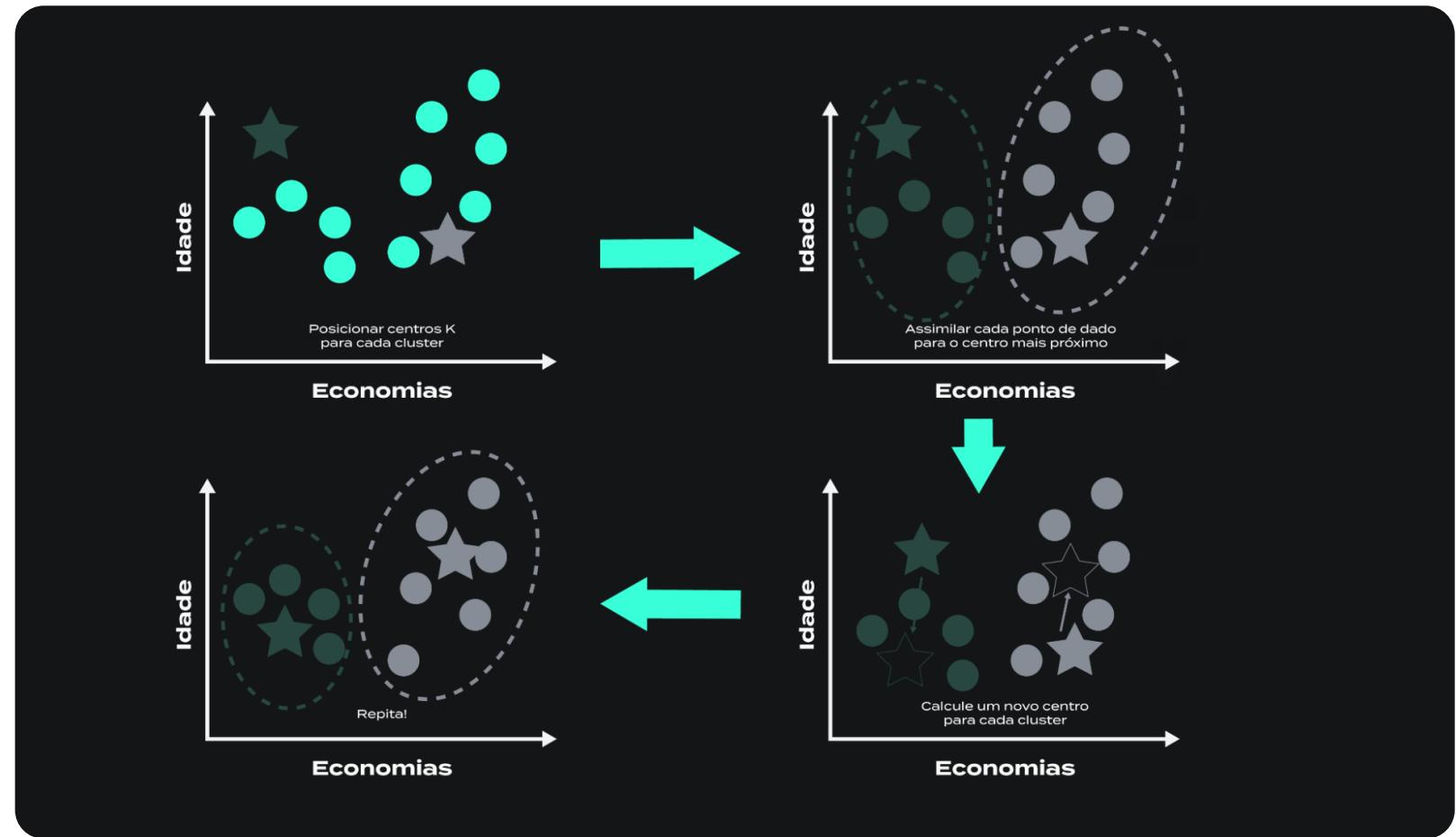
Os grupos se chamam "clusters"

A partir de dados objetivos e quantificáveis, nós podemos traçá-los em um plano.

- Vamos imaginar um plano 2D.
- A partir da plotagem, podemos calcular a distância entre os pontos através de vetores.

Passos do algoritmo K-means

1. Escolha o número do cluster de "K".
2. Selecione pontos K aleatórios que vão ser os centros de cada cluster.
3. Assimilar cada ponto de dado com o centro mais próximo, fazendo isso, vai nos permitir criar um "K" número de cluster.
4. Calcular o novo centro de cada cluster.
5. Reassimilar cada ponto de dado para o novo centro mais próximo
6. Vá para o passo 4 e repita.



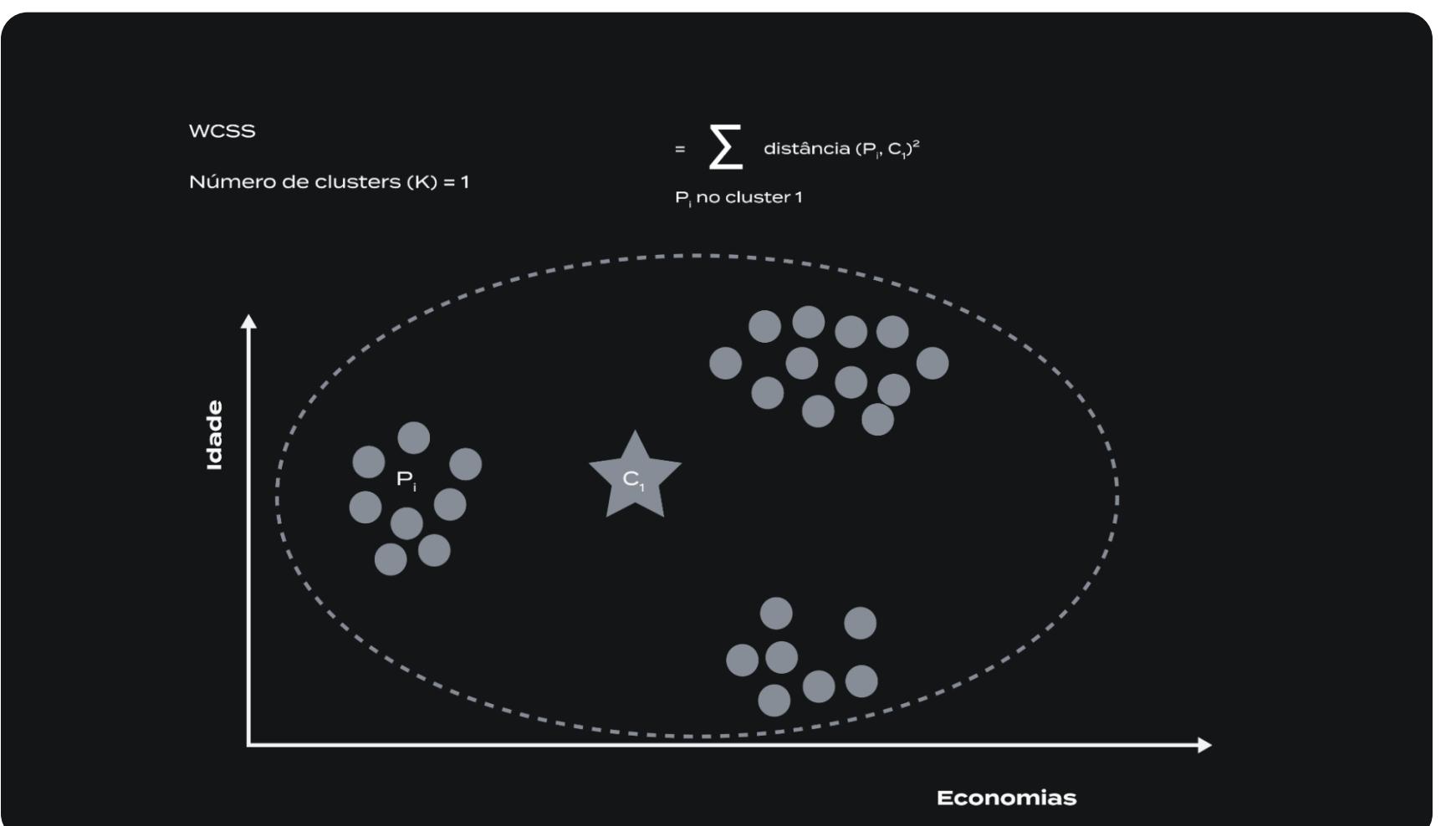
Perceba que não tem supervisão ou orientação de certo ou errado, ele só otimiza o processo para tentar chegar a uma conclusão sozinho. Não tem “certo” ou “errado” no processo, é tudo matemática e distância no plano euclidiano.

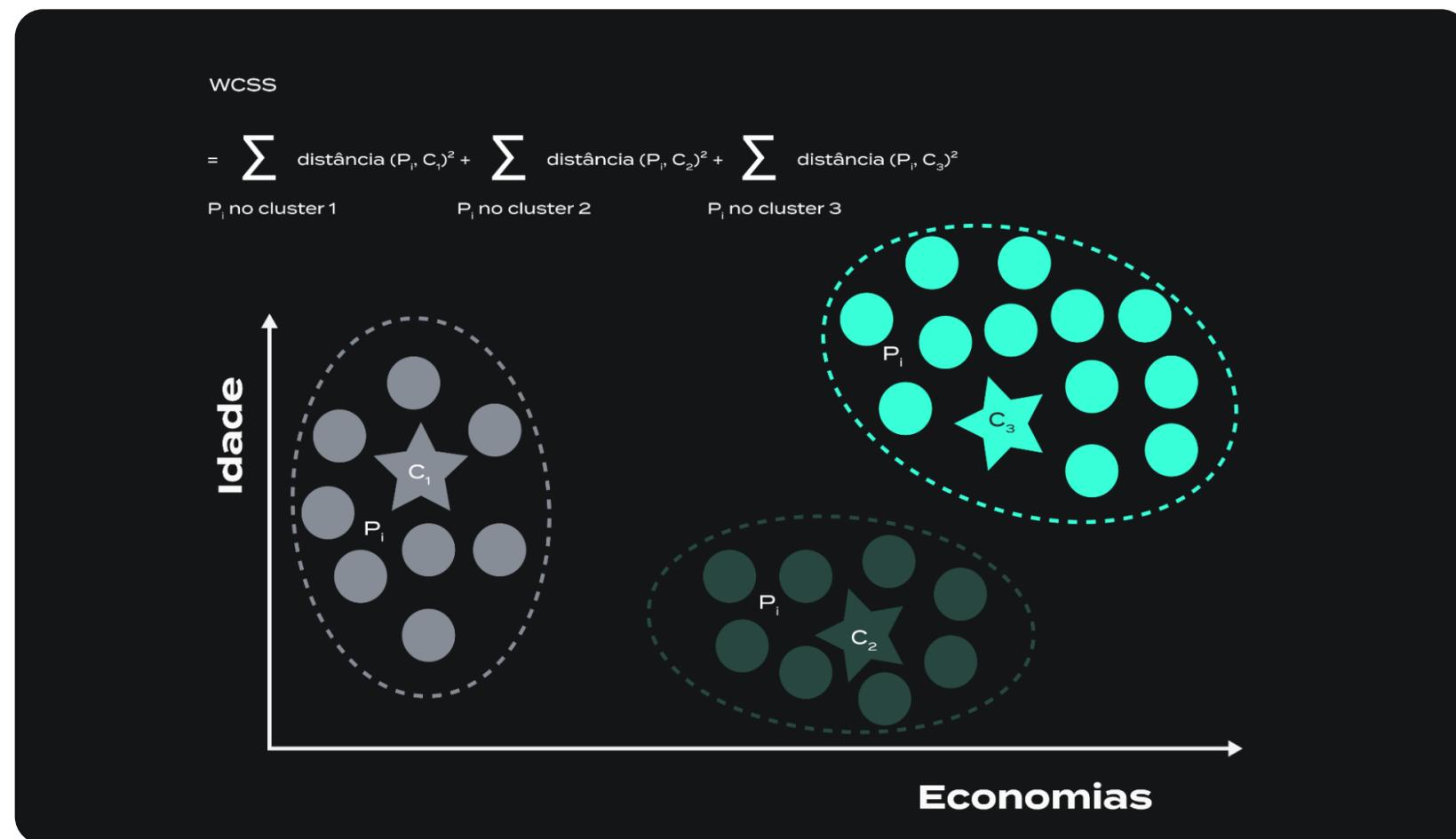
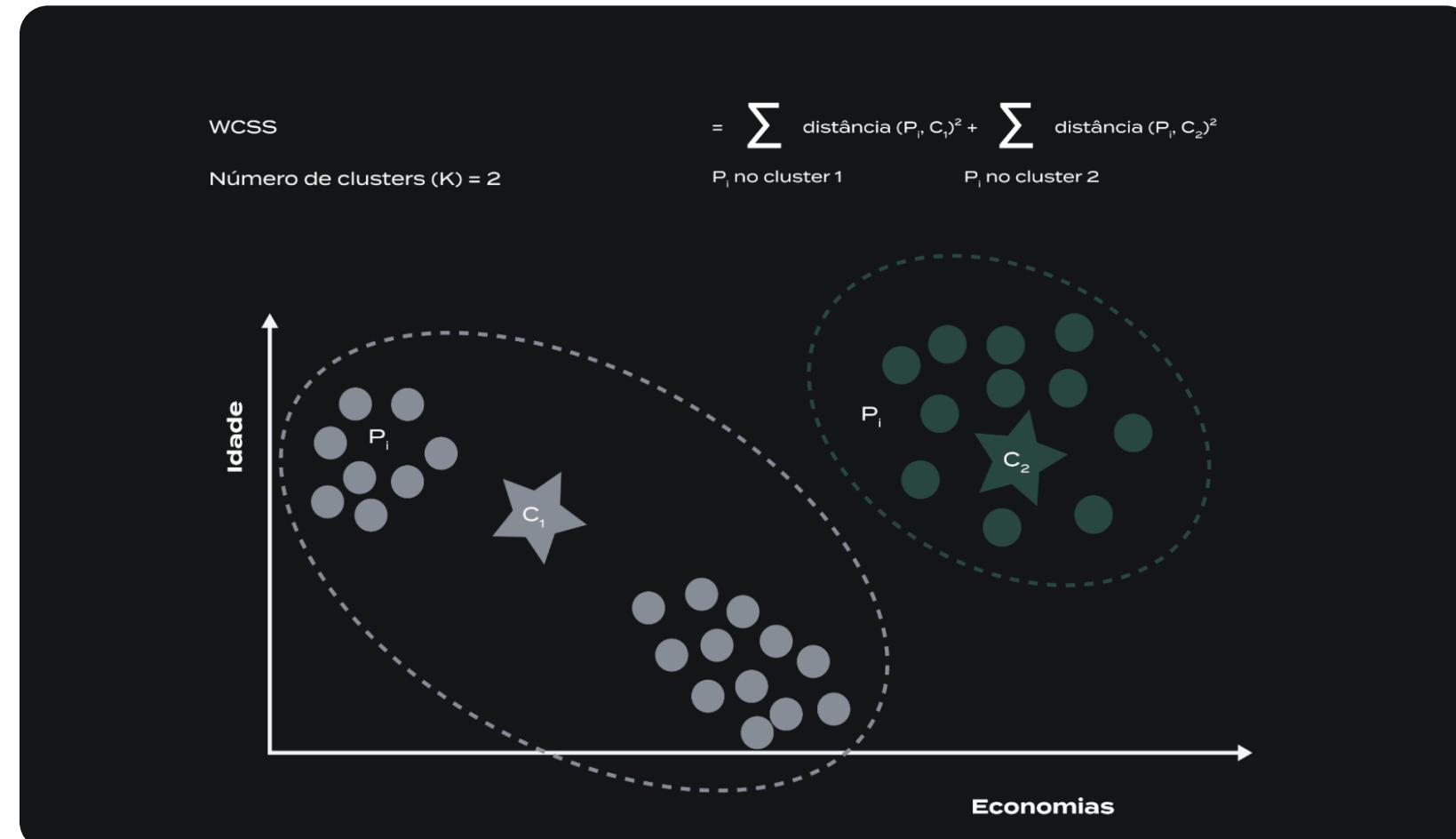
Projeto 3, Mundo 3: Como otimizar a quantidade de grupos presente nos dados - ELBOW METHOD e WCSS.

O Elbow method permite chegarmos no número ótimo de grupos.

A métrica chave é a WCSS. Ela calcula a soma da distância ao quadrado dos pontos ao centro do Cluster, sendo o melhor valor sendo zero.

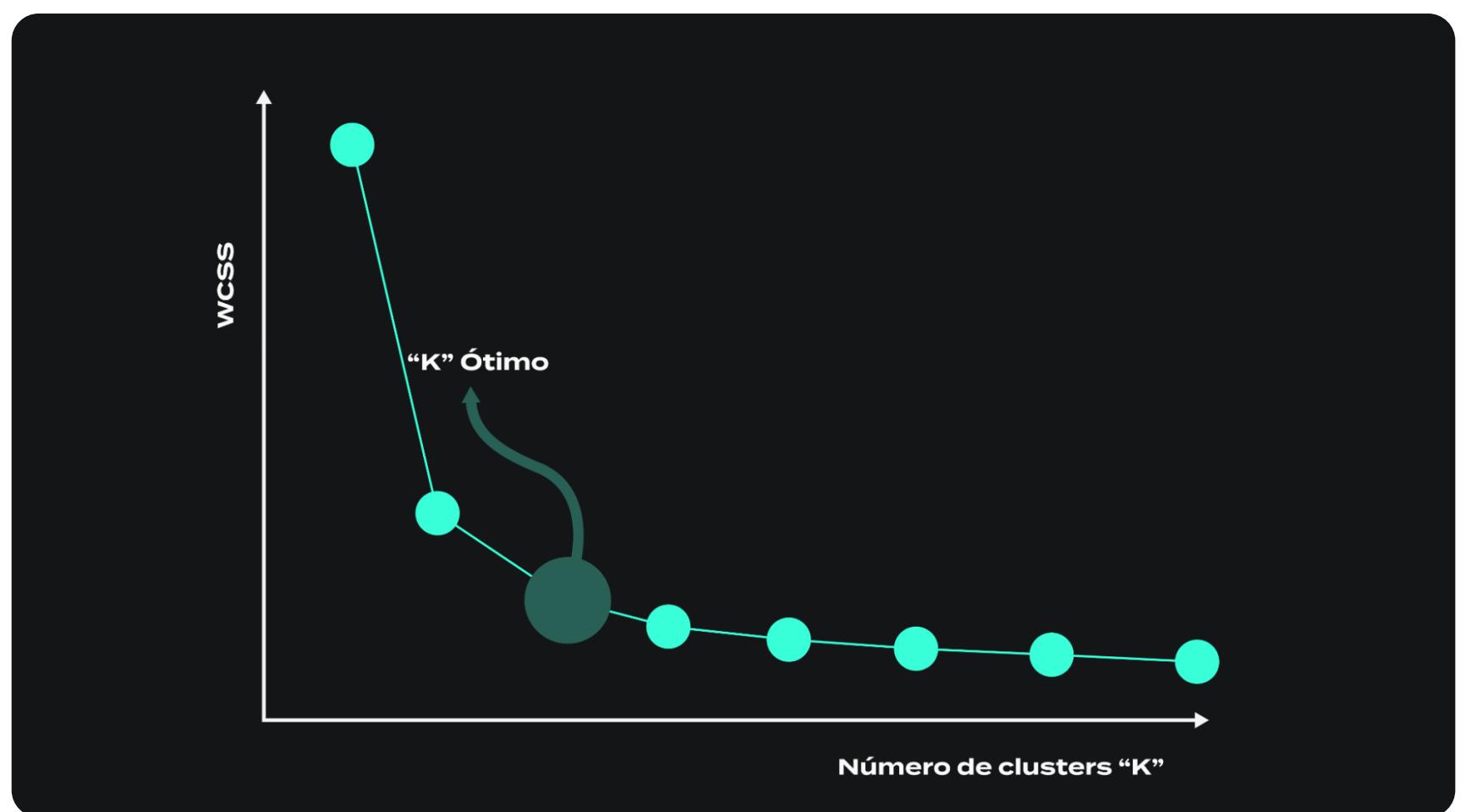
Obviamente, quanto mais grupos existirem, menor será esse indicador.





Existe um tradeoff: quanto mais grupos, melhor será a divisão, mas menos conclusivo é ter centenas de grupos.

Você deve traçar o gráfico da WCSS e decidir, por conta própria, qual número a vantagem marginal de adicionar um grupo é nula.



Projeto 3, Mundo 4: Puxando e tratando dados de clientes.

Importe as bibliotecas necessárias.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

Segue uma explicação caso tenha dúvida do que possa significar alguma coluna dos dados:

- CUSTID: Identificação do Titular do Cartão de Crédito
- SALDO: Saldo restante na conta do cliente para efetuar compras
- BALANCE_FREQUENCY: Com que frequência o Saldo é atualizado, pontuação entre 0 e 1 (1 = atualizado com frequência, 0 = não atualizado com frequência)
- COMPRAS: Valor de compras feitas na conta
- ONEOFFPURCHASES: Valor máximo de compra feito de uma só vez
- INSTALLMENTS_PURCHASES: Valor da compra parcelada
- CASH_ADVANCE: Dinheiro adiantado dado pelo usuário
- PURCHASES_FREQUENCY: Com que frequência as Compras estão sendo feitas, pontue entre 0 e 1 (1 = compra frequentemente, 0 = não compra com frequência)
- ONEOFF_PURCHASES_FREQUENCY: Com que frequência as compras acontecem de uma só vez (1 = comprado com frequência, 0 = não comprado com frequência)
- PURCHASES_INSTALLMENTS_FREQUENCY: Com que frequência as compras parceladas estão sendo feitas (1 = feito com frequência, 0 = não feito com frequência)
- CASH_ADVANCE_FREQUENCY: Com que frequência o dinheiro adiantado está sendo pago
- CASH_ADVANCE_TRX: Quantidade de Transações feitas com "Dinheiro Adiantado"
- PURCHASES_TRX: Número de transações de compra realizadas
- CREDIT_LIMIT: Limite de Cartão de Crédito por usuário
- PAGAMENTOS: Valor do pagamento feito pelo usuário
- MINIMUM_PAYMENTS: Valor mínimo de pagamentos feitos pelo usuário
- PRC_FULL_PAYMENT: Porcentagem do pagamento integral pago pelo usuário
- TENURE: Posse do serviço de cartão de crédito para o usuário

Pegue a base de dados.

```
dados_clientes = pd.read_csv('dados_clientes.csv')
```

Faça um drop no CUST_ID para deixar o DataFrame apenas com números.

```
dados_clientes = dados_clientes.drop('CUST_ID', axis = 1)
```

Por último, um dropna e reset_index para terminar a formatação.

```
dados_clientes = dados_clientes.dropna()
dados_clientes = dados_clientes.reset_index(drop = True)

dados_clientes
```

Projeto 3, Mundo 5: Achando o número ótimo de grupos dos nossos dados - Aplicando Elbow Method na prática.

Pegue os dados escalados com a função fit_transform para transformar em um array.

```
escalador = StandardScaler()
dados_clientes_escalados = escalador.fit_transform(dados_clientes)
dados_clientes_escalados
```

Caso queira verificar o número de linhas e colunas.

```
dados_clientes_escalados.shape
```

Aqui vamos precisar aplicar o elbow antes de rodar de vez os grupos finais.

Escolha o número de grupos. Nesse caso colocamos 50.

```
for i in range_de_grupos:  
    kmeans = KMeans(n_clusters = i)  
    kmeans.fit(dados_clientes_escalados)  
    lista_WCSS.append(kmeans.inertia_)  
  
lista_WCSS
```

Faça o plot para encontrar o melhor número de clusters. Nesse caso, um bom número foi 10, de acordo com o que estudamos anteriormente

```
plt.plot(lista_WCSS)  
plt.title('Achando o número ótimo de clusters')  
plt.xlabel('Clusters')  
plt.ylabel('WCSS')  
plt.show()
```

Projeto 3, Mundo 6: Treinando o modelo K-MEANS, classificando nossos clientes em diferentes grupos e observando insights nos resultados que jamais seriam possíveis sem uma IA.

Utilizando o número de clusters como 10, essa função aqui retorna o agrupamento de todas as linhas do DataFrame anterior.

```
kmeans = KMeans(10)  
kmeans.fit(dados_clientes_escalados)  
grupos = kmeans.labels_  
grupos
```

Crie essa função para retornar o centro dos dados de cada grupo em cada informação. Além de inverter os dados para não ficarem escalados.

```
centro_dos_dados_por_cluster = kmeans.cluster_centers_  
centro_dos_dados_por_cluster =  
escalador.inverse_transform(centro_dos_dados_por_cluster)
```

Transforme em DataFrame que retorna os 10 grupos com todos os centros de cada grupo e de cada informação.

```
centro_dos_dados_por_cluster = pd.DataFrame(data =
centro_dos_dados_por_cluster, columns = [dados_clientes.columns])
centro_dos_dados_por_cluster
```

Juntando os grupos no Dataframe original.

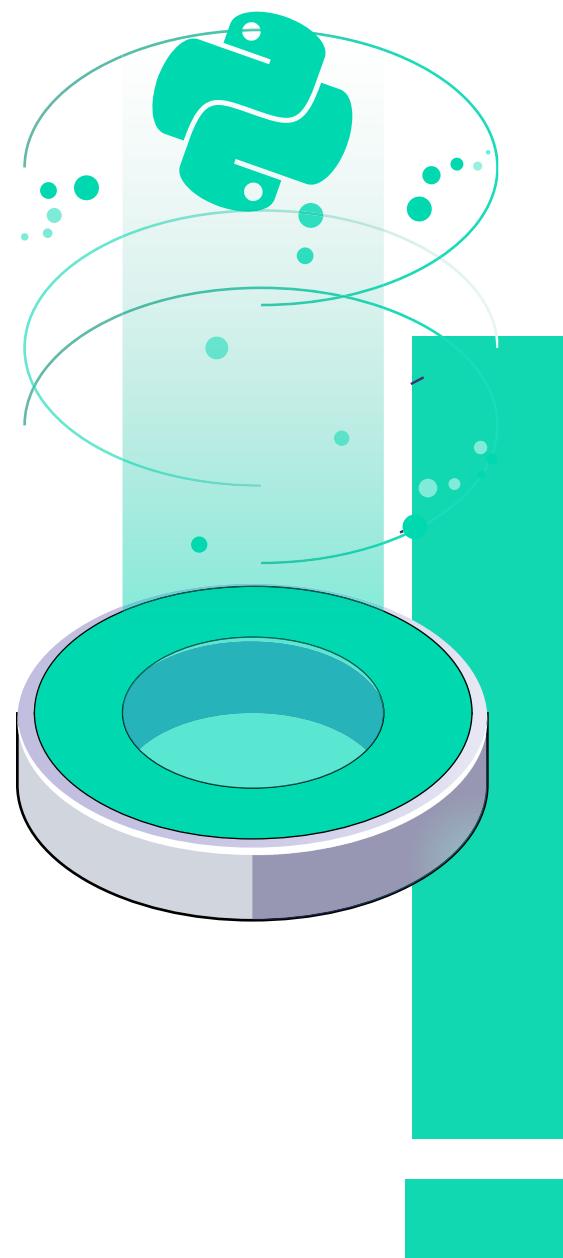
```
dados_clientes_com_grupos = pd.concat([dados_clientes,
pd.DataFrame({'grupo':grupos})], axis = 1)
dados_clientes_com_grupos
```

Plot de visualização dos dados por cluster.

```
for i in dados_clientes_com_grupos.columns:
plt.figure(figsize = (35, 5))
for j in range(10):
    plt.subplot(1,10,j+1)
    grupo = dados_clientes_com_grupos[dados_clientes_com_grupos['grupo'] == j]
    grupo[i].hist(bins = 20)
    plt.title(f'{i} \n Grupo {j} ')
plt.show()
```

Essa é uma ferramenta poderosa e extremamente útil para classificação de grupos, a inteligência artificial consegue separar os clientes de uma forma que nós humanos jamais conseguiríamos fazer a olho nu.

Na próxima aula vamos utilizar outra inteligência artificial para verificar se essa classificação está correta, se faz sentido e avaliar tudo isso.

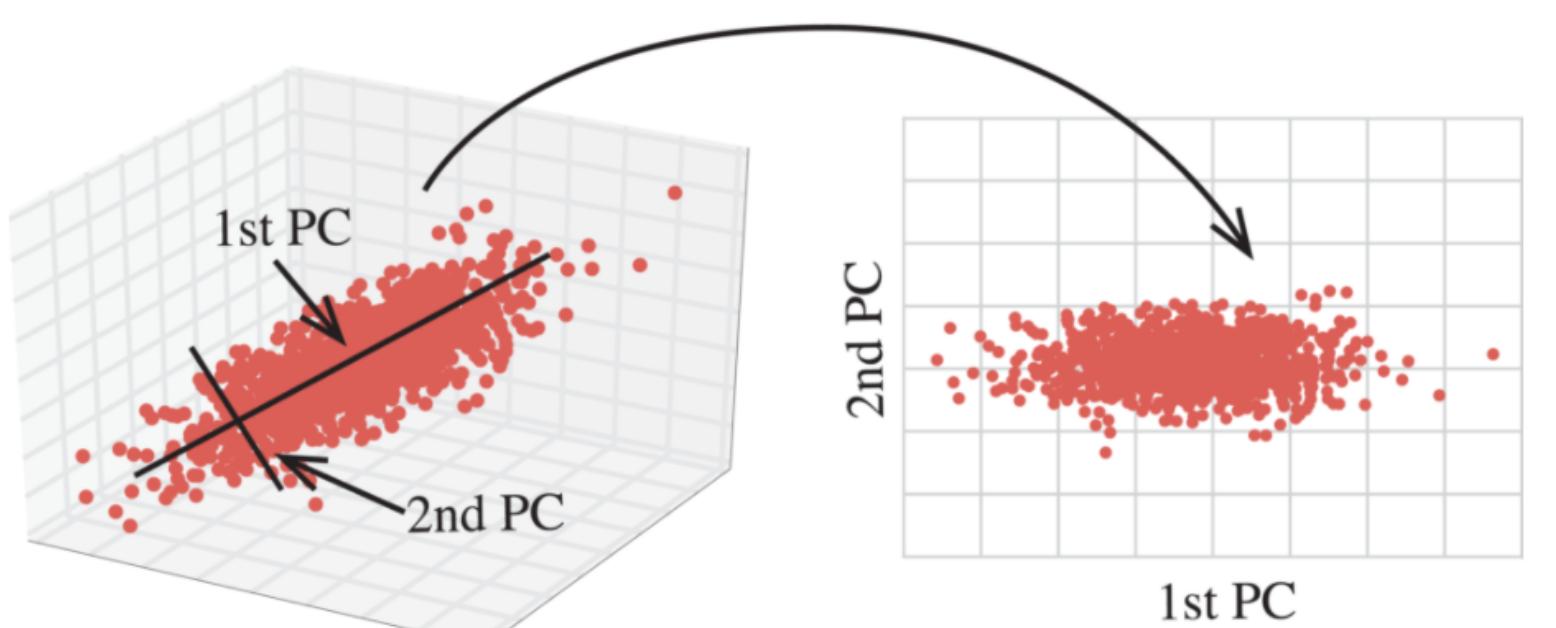


Projeto 3, Mundo 7: Como funciona a inteligência artificial PCA - Diminuindo a dimensão dos nossos dados.

Machine learning PCA: Redução da dimensionalidade de um dado.

Para um modelo de PCA funcionar, as variáveis devem ser correlacionadas.

Isso é ótimo para visualizar se o K-MEANS funcionou e agrupou as coisas de forma minimamente decente.



Projeto 3, Mundo 8: Aplicando PCA para visualizar e avaliar os resultados do nosso modelo K-MEANS - Nossos clientes foram bem classificados?

In_components=2 significa que o gráfico vai ter duas dimensões, escolha o número que desejar.

```
pca = PCA(n_components=2)
dados_clientes_pca = pca.fit_transform(dados_clientes_escalados)
dados_clientes_pca
```

Novamente transforme em DataFrame.

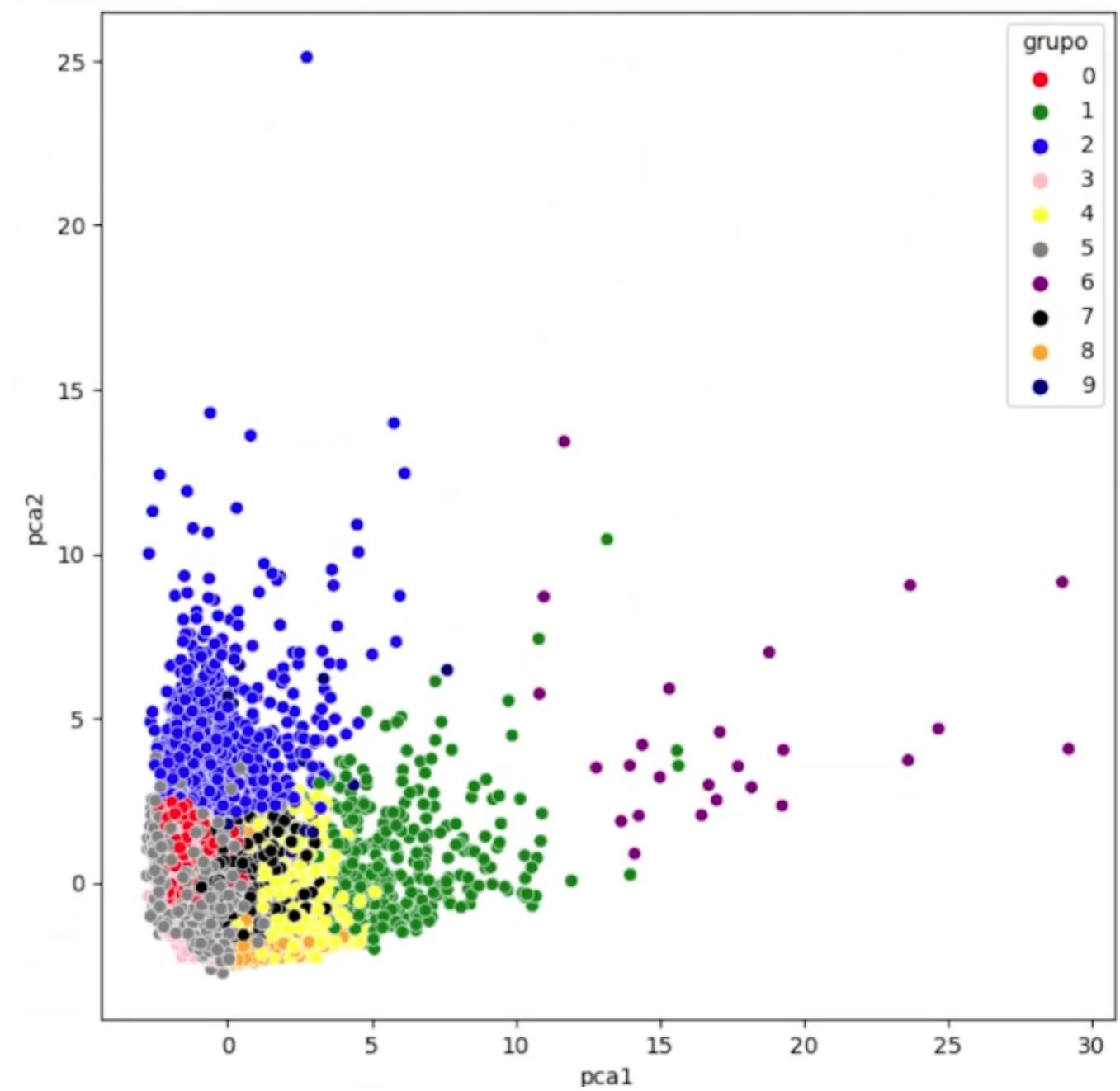
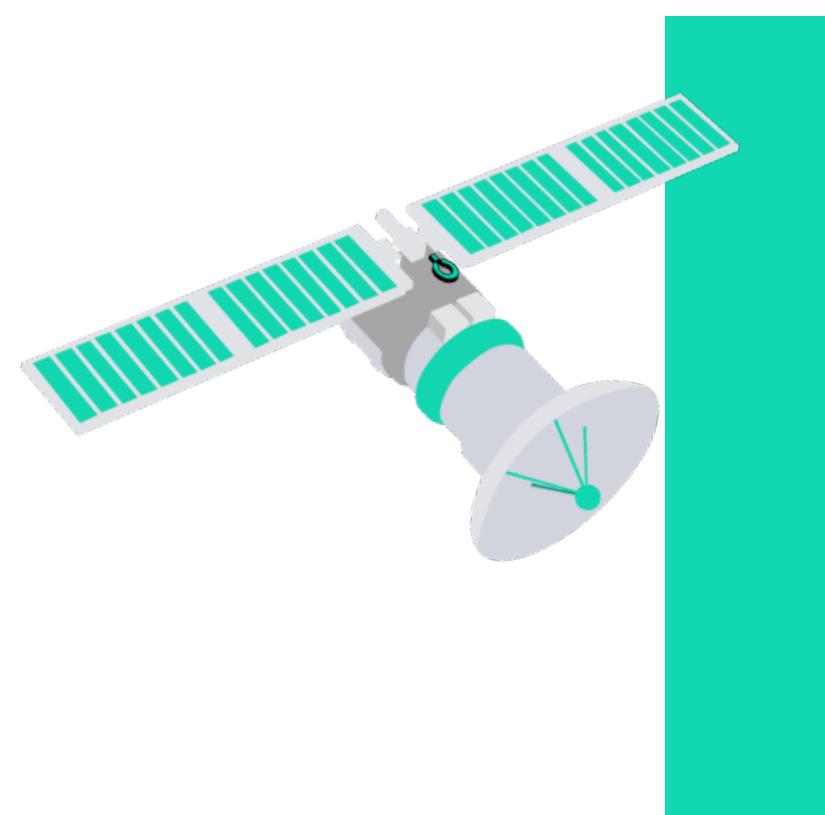
```
pca_df = pd.DataFrame(data = dados_clientes_pca, columns = 
['pca1', 'pca2'])
pca_df
```

Faça um pd.concat com os grupos para visualizar qual grupo cada dado pertence.

```
pca_df = pd.concat([pca_df,pd.DataFrame({'grupo':grupos}], axis = 1)
pca_df
```

Construa o gráfico que mostra os grupos separados por cores para visualizar como funciona na prática.

```
plt.figure(figsize=(8,8))
ax = sns.scatterplot(x="pca1", y="pca2", hue = "grupo",
data = pca_df, palette =['red','green','blue','pink','yellow',
'gray','purple', 'black', 'orange', 'navy'])
plt.show()
```



O potencial desse projeto é gigante, podendo ser aplicado a vários contextos, desde marketing de direcionamento de campanha até classificação de empresas por setores no mercado financeiro.

Projeto 4, Mundo 1: Prevendo o Default de clientes no cartão de crédito com uma IA.

Para esse último projeto também vamos pegar os dados do Kaggle.

Dados de crédito de 30 mil clientes.

Modelo de regressão, supervisão e de classificação.

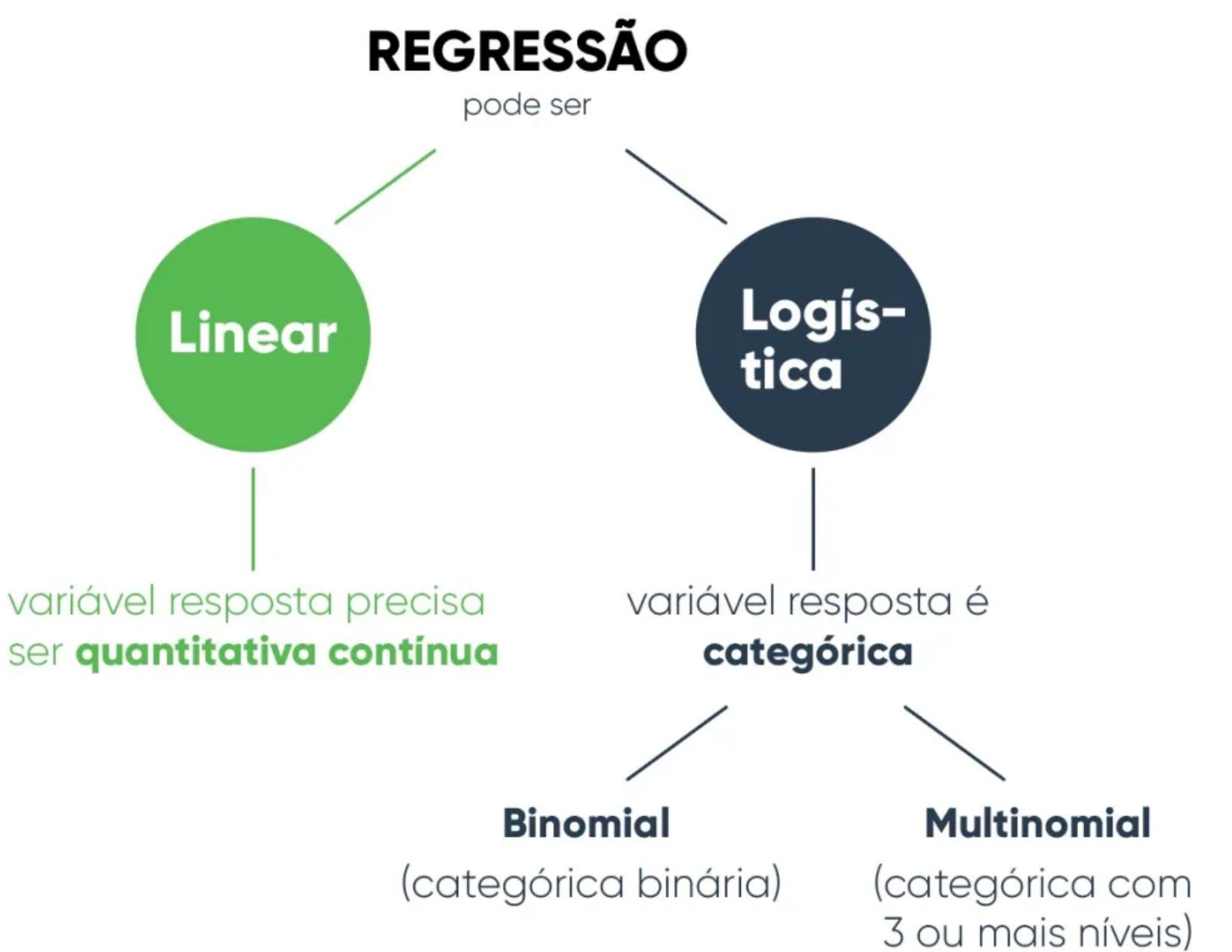
Regressão logística.

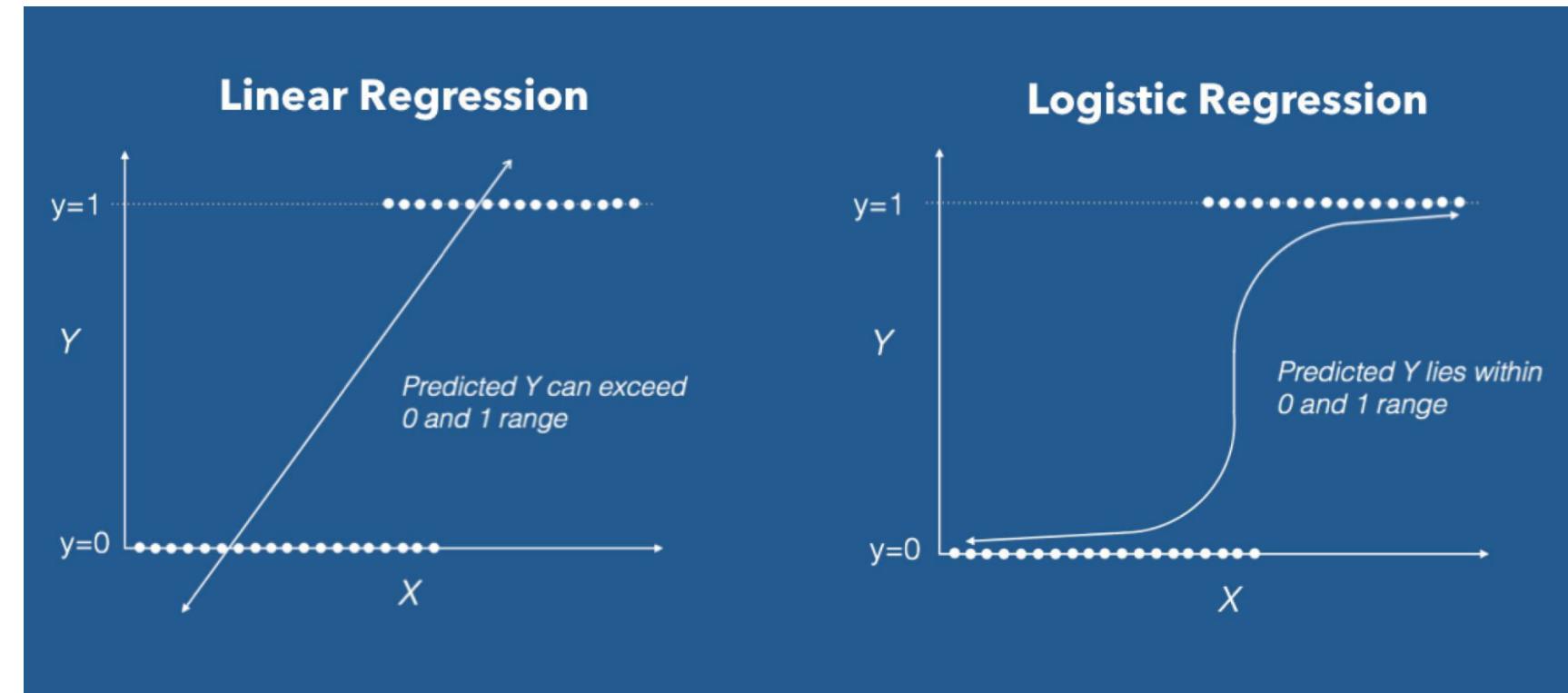
Vamos treinar um modelo que verifica se o cliente vai dar default no próximo mês baseado se pessoas parecidas com ele deram default nos dados de treino.

É um algoritmo parecido com classificar se um e-mail é SPAM ou não.

Projeto 4, Mundo 2: O que é uma regressão logística?

A regressão logística é igual a regressão linear, só que ao invés de gerar uma variável Y que pode ir de menos infinito a mais infinito, vai gerar uma variável categórica em um número finito de possibilidades, sendo, normalmente, 0 ou 1.





Projeto 4: Mundo 3: Pegando os dados bancários de 30 mil pessoas para treinar nossa IA.

Importe as seguintes bibliotecas.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import seaborn as sns
```

Assim como no projeto anterior, aqui vai uma breve descrição dos dados que vamos utilizar:

- ID: ID de cada cliente
- LIMIT_BAL: Valor do crédito concedido em NT dólares (inclui crédito individual e familiar/suplementar)
- SEXO: Gênero (1=masculino, 2=feminino)
- EDUCAÇÃO: (1=pós-graduação, 2=universidade, 3=ensino médio, 4=outros, 5=desconhecido, 6=desconhecido)
- CASAMENTO: Estado civil (1=casado, 2=solteiro, 3=outros)
- IDADE: idade em anos

- PAY_0: Status do reembolso em setembro de 2005 (-1=pagamento devido, 1=atraso no pagamento por um mês, 2=atraso no pagamento por dois meses, ... 8=atraso no pagamento por oito meses, 9=atraso no pagamento por nove meses e acima)
- PAY_2: Status do reembolso em agosto de 2005 (escala igual à acima)
- PAY_3: Status do reembolso em julho de 2005 (escala igual à acima)
- PAY_4: Status do reembolso em junho de 2005 (escala igual à acima)
- PAY_5: Status do reembolso em maio de 2005 (escala igual à acima)
- PAY_6: Status do reembolso em abril de 2005 (escala igual à acima)
- BILL_AMT1: Valor do extrato da conta em setembro de 2005 (NT dólar)
- BILL_AMT2: Valor do extrato da conta em agosto de 2005 (NT dólar)

- BILL_AMT3: Valor do extrato da conta em julho de 2005 (NT dólar)
- BILL_AMT4: Valor do extrato da conta em junho de 2005 (NT dólar)
- BILL_AMT5: Valor do extrato da conta em maio de 2005 (NT dólar)
- BILL_AMT6: Valor do extrato da conta em abril de 2005 (NT dólar)
- PAY_AMT1: Valor do pagamento anterior em setembro de 2005 (NT dólares)
- PAY_AMT2: Valor do pagamento anterior em agosto de 2005 (NT dólar)
- PAY_AMT3: Valor do pagamento anterior em julho de 2005 (NT dólar)
- PAY_AMT4: Valor do pagamento anterior em junho de 2005 (NT dólar)
- PAY_AMT5: Valor do pagamento anterior em maio de 2005 (NT dólar)
- PAY_AMT6: Valor do pagamento anterior em abril de 2005 (NT dólar)
- default.payment.next.month: Pagamento padrão (1=sim, 0=não)

Pegue a base de dados.

```
dados_bancarios = pd.read_csv('dados_cartao_credito.csv')
dados_bancarios
```

Faça um drop no ID para deixar o DataFrame apenas com números.

```
dados_bancarios = dados_bancarios.drop('ID', axis = 1)
```

Dummies

Como vamos trabalhar com dados de classificação em uma regressão, precisamos pegar dummies pros dados categóricos.

Assim, conseguimos inserir um beta que vai multiplicar o valor da dummy e adicionar um valor à nossa equação caso a pessoa em questão se enquadre em alguma classificação.

Com dummies nós conseguimos medir exatamente a sensibilidade de adicionar uma variável ou não.

Exemplo:

$$Y = S + EF * DEF + EM * DEM + FA * DFA$$

$$Y = 1250 + 250 * 1 + 750 * 1 + 1000 * 1$$

$$\text{Salário} = 1250$$

$$EF = \text{Ensino fundamental} = 1500$$

$$EM = \text{Ensino Médio} = 2250$$

$$FA = \text{Faculdade} = 3250$$

```
dicionario_colunas_dummies = {
    'EDUCATION': 'EDU',
    'MARRIAGE': 'MAR'
}
```

Como exemplo, dummies de educação e casamento.

Para cada coluna, pegue os dummies e de um pd.concat e depois um drop nos dados originais para ficar apenas com os dados dummies.

```
for column, prefix in dicionario_colunas_dummies.items():
    dummies = pd.get_dummies(dados_bancarios[column], prefix=prefix)
    dados_bancarios = pd.concat([dados_bancarios, dummies], axis=1)
    dados_bancarios = dados_bancarios.drop(column, axis=1)

dados_bancarios
```

Projeto 4: Mundo 4: Separando nossos dados entre teste e treino.

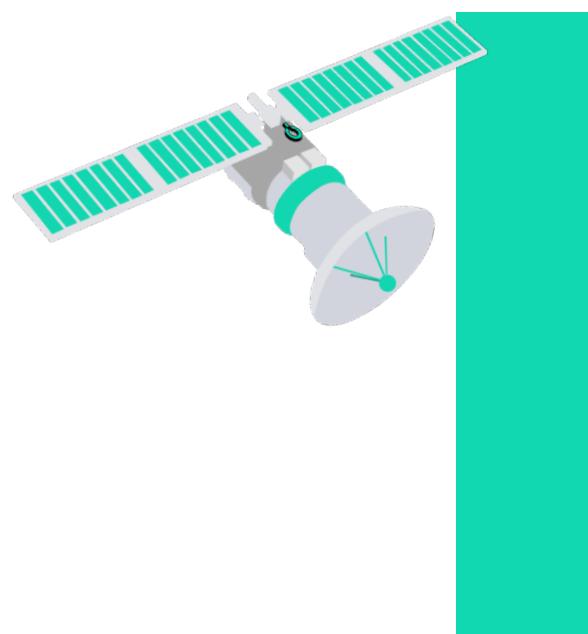
Vamos treinar o nosso modelo, separar os dados para ver se as pessoas vão pagar ou não as dívidas.

Agora que já temos as dummies vamos pegar os nosso X e Y.

```
Y = dados_bancarios['default.payment.next.month'].copy()
X = dados_bancarios.drop('default.payment.next.month', axis=1).copy()
```

Separar em dados de treino e dados de testes. De uma forma mais simplificada dessa vez.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.8)
```



Escale o treino e o teste. Nesse caso só vamos precisar escalar o X porque o Y já é 0 ou 1. Ou a pessoa pagou ou não pagou.

```
escalador_treino = StandardScaler()
escalador_teste = StandardScaler()

X_train = pd.DataFrame(escalador_treino.fit_transform(X_train),
columns=X.columns) #só precisamos escalar o X pq o Y é 0 ou 1.
X_test = pd.DataFrame(escalador_teste.fit_transform (X_test),
columns=X.columns)

X_train
```

Treinando o modelo.

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

Projeto 4, Mundo 5: O que é uma matriz de confusão? E o score?

A melhor coisa que você pode avaliar em um modelo de classificação é a matriz de confusão.

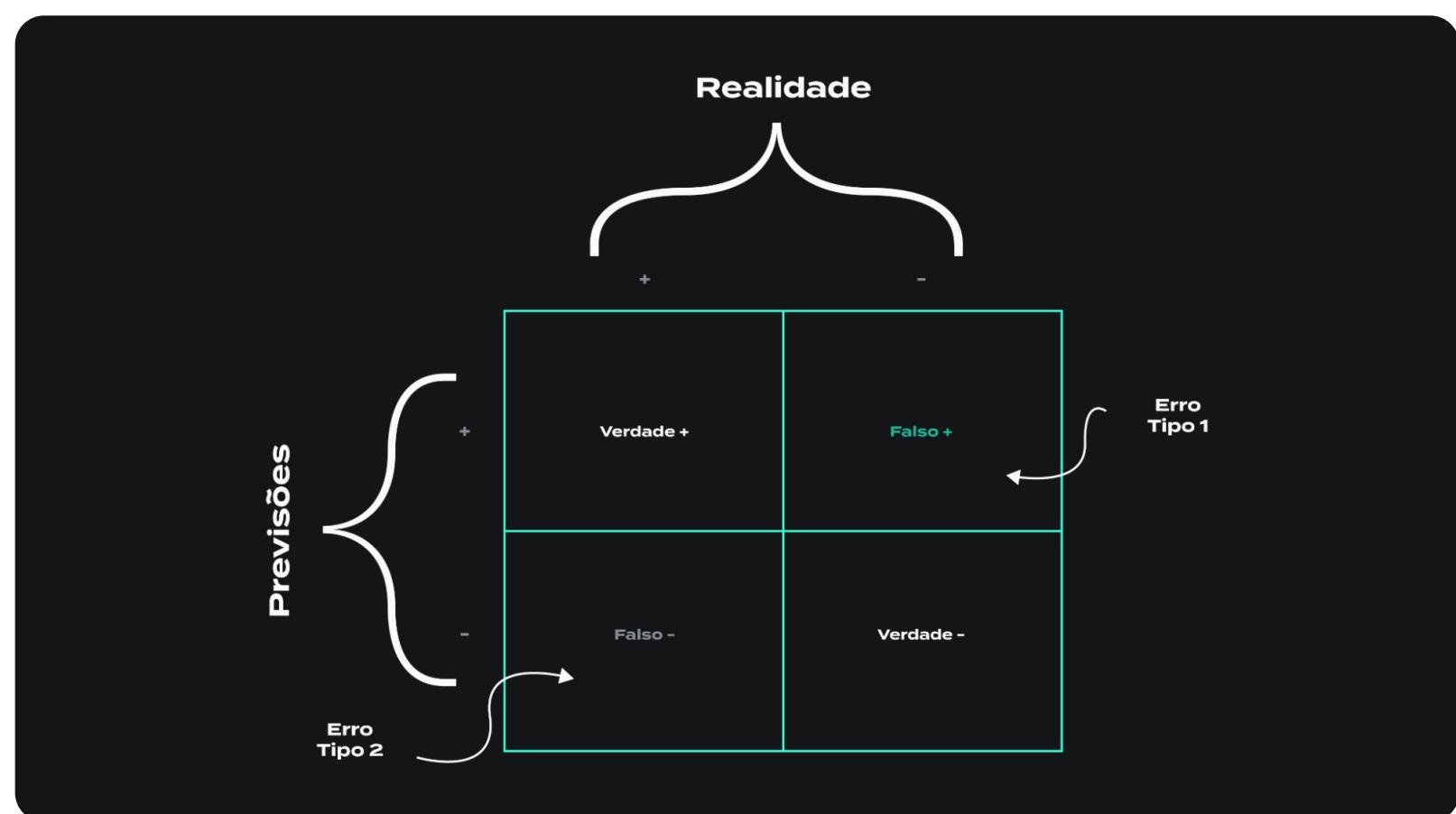
É um cálculo econômico e um trade-off entre falsos positivos e falsos negativos. O que é pior?

O score de um modelo é a soma dos acertos dividido pelo total. Só que essa medida pode enganar em modelos que a maioria dos dados são 0.

Exemplo: doença rara que afeta 0,001% da população. Qualquer algoritmo que você rode vai ter um score de 99,9%, nem por isso é bom. Você precisa olhar pros erros ao invés dos acertos e comparar com um humano.

Só é possível minimizar um tipo de erro. Como médico, ou você fala que uma pessoa está doente e ela descobre que não está ou não fala e no fim ela fica doente, na minha opinião, o segundo erro é pior (falso negativo). No entanto, depende do caso, em casos bancários é muito pior o falso positivo, um bancário dar um crédito para um cliente e esse acabar dando calote, do que não dar crédito para clientes que não daria calote.

Com isso se minimiza o erro. Ignorando as vezes que vai deixar de ganhar dinheiro e vai minimizar as pessoas que não vai dar crédito que não vão pagar.



Projeto 4: Mundo 6: Treinando e avaliando o modelo através da matriz de confusão - Uma IA que consegue prever se uma pessoa vai dar calote no cartão de crédito.

Fazer previsões de default.

Em x% dos casos ele disse que ia dar default e realmente deu e ele disse que não ia dar e realmente não deu.

Calcule o score.

```
"Acertos: {:.2f}%".format(modelo.score(X_test, y_test) * 100)
```

Fazer o `modelo.score` é a mesma coisa que dar um `predict`.

```
prevendo_defaults = modelo.predict(X_test)
```

```
prevendo_defaults
```

Chegando ao mesmo valor calculado com o .score.

```
from sklearn.metrics import accuracy_score

acertos = accuracy_score(y_test.values, prevendo_defaults)
acertos
```

Matriz de confusão. Gere a matriz com o y_test sobre o que realmente aconteceu e as nossas previsões.

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test.values, prevendo_defaults)
cm
```

Resultado:

0/0 - 0/1

1/0 - 1/1

Sendo 0/0 as vezes em que o modelo previu que o cliente ia pagar e ele realmente pagou.

0/1 as vezes em que o modelo previu que o cliente ia pagar e ele não pagou.

1/0 as vezes modelo previu que o cliente não ia pagar e ele pagou.

E 1/1 as vezes que o modelo previu que o cliente não ia pagar e ele realmente não pagou.

Análise que você sempre deve fazer:

- Qual erro é pior pra você?
- Otimizar o modelo.
- ML é uma otimização de processo: o algoritmo é melhor do que existe hoje? Ele erra menos ou mais?
- No caso dos bancos, o erro tipo 1 é crucial. Deixar de dar crédito pra alguém que ia pagar é menos pior do que tomar default.
- Deixamos de ganhar dinheiro em 1040 casos e perdemos em 143. Calcular o custo disso tudo e ver se isso é melhor do que está sendo praticado hoje.