

Introdução a Machine Learning

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br

Aula 7 – Introdução ao Aprendizado Não Supervisionado

Introdução ao
aprendizado não
supervisionado.
Exemplos de
aplicações reais.

Conceitos iniciais
ao algoritmo de
agrupamento
Kmeans.

Aplicações
computacionais
com a linguagem
Python.

Recapitulando tudo o que construímos juntos

O que nós vimos até então?

- Em nossa jornada, aprendemos a ensinar máquinas a tomar decisões. **Partimos de um objetivo claro:** treinar modelos para acertar um "gabarito" que já tínhamos.
- Saímos do mundo de prever números ("Quanto?") e mergulhamos na tarefa de atribuir rótulos ("O quê?"). **Nosso desafio era ensinar o modelo a distinguir entre categorias como "Aprovado/Reprovado" ou "Spam/Não Spam".**
- **Além da Acurácia:** Vimos que existem outras métricas que complementam a acurácia e auxiliam no diagnóstico do modelo, como por exemplo Matriz de Confusão, precisão, recall, etc.
- **O Dilema Central:** Dominamos o trade-off entre Precision (não cometer falsos alarmes) e Recall (não deixar ninguém para trás), e aprendemos a equilibrá-los com o F1-Score e a AUC.

Recapitulando tudo o que construímos juntos

Quais Ferramentas Usamos?

- Regressão Logística: A calculadora de probabilidades.
- Árvore de Decisão: O fluxograma de regras "SE-ENTÃO".
- KNN: O voto da vizinhança.
- SVM: O especialista em encontrar padrões complexos.

Como Entendemos as Decisões? A Busca pela Interpretabilidade

- Não bastava ter um modelo preciso; precisávamos responder "Por quê?". Entramos no mundo da interpretabilidade para abrir a "Caixa Preta".
- Para modelos "Caixa de Vidro", extraímos regras claras e medimos o impacto exato de cada variável.

Recapitulando tudo o que construímos juntos

O que vamos fazer agora?

- Agora, a pergunta que muda tudo: **E se tirarmos essa bússola? O que acontece quando os dados não vêm com o rótulo?**
- Até agora, todos os nossos problemas tinham um **"gabarito"**.
- **Tínhamos dados com rótulos (variáveis explicativas e a variável alvo)** e nossa missão era ensinar o modelo a prever essa resposta. Ensinaamos a prever "Gato" ou "Não Gato" e também gerar uma estimativa do salário. Estávamos no mundo do Aprendizado Supervisionado.

Agora, vamos mudar as regras do jogo. E se não tivermos a resposta?

Aprendizado Não Supervisionado

Dados Sem Rótulos

- Imagine receber uma caixa gigante com milhares de peças de LEGO, de todos os tipos e cores, mas sem manual de instruções e sem a foto da caixa.
- Você não sabe o que construir. Sua única tarefa é encontrar padrões, separar as peças em grupos que fazem sentido.
- Isso é o **Aprendizado Não Supervisionado**: a arte de encontrar a estrutura escondida em dados "crus", sem um alvo pré-definido.

Mas para que ele serve?

Aprendizado Não Supervisionado

Procurar Padrões Para que?

- Mesmo sem rótulos, **é essencial saber o que se busca compreender**. O aprendizado não supervisionado não tem um alvo explícito, mas ele serve para revelar estruturas ocultas que podem orientar decisões ou hipóteses futuras.
- Você pode **querer descobrir grupos de clientes com comportamentos semelhantes, identificando padrões de compra, preferências ou perfis de uso**. Isso permite personalizar estratégias e entender melhor o público.
- Outra aplicação é **detectar anomalias ou outliers, encontrando elementos que fogem do padrão geral e que podem representar erros**, fraudes ou casos raros que merecem atenção especial.

Aprendizado Não Supervisionado

Diferença..



Aplicações

Motores de Recomendação (Netflix, Spotify, Amazon)

- Como sugerir o próximo filme ou produto para milhões de usuários diferentes?
- O algoritmo não sabe o que é um "filme de ação". Ele simplesmente agrupa usuários com históricos parecidos em "categorias de gosto". Quando um membro da sua categoria gosta de um filme novo, ele é recomendado para você.

Organização de Informação (Google Notícias)

- Como agrupar milhares de artigos de notícias sobre o mesmo evento, escritos por fontes diferentes?
- O algoritmo analisa o texto e agrupa documentos que usam palavras e contextos semelhantes, criando "tópicos" automaticamente, sem que um humano precise ler e categorizar cada um.

Aplicações

Segmentação de Clientes (Marketing e Vendas)

- Como uma empresa pode entender os diferentes perfis de seus clientes para criar campanhas mais eficazes?
- O algoritmo analisa dados de compra (frequência, valor, tipo de produto) e descobre "personas" de clientes que a empresa nem sabia que existiam, criando um mapa do tesouro para o marketing.

Mas como esse agrupamento funciona na prática?

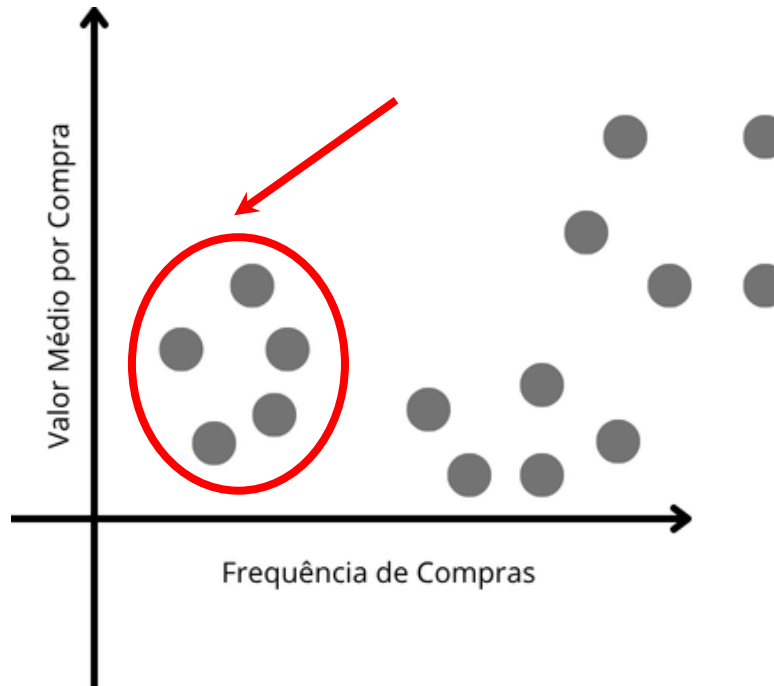
Aprendizado Não Supervisionado na Prática

Agrupamento

- Nossa primeira ferramenta para criar "personas" é o K-Means. Ele vai analisar o comportamento dos nossos clientes e agrupá-los automaticamente.
- Vamos usar dados de uma loja online, analisando duas variáveis simples:
 - X_1 : Frequência de Compras (Quantas vezes o cliente comprou no último ano).
 - X_2 : Valor Médio por Compra (Quanto ele gasta, em média, a cada vez que compra).
- Nossa missão é encontrar nossos perfis de clientes.

Aprendizado Não Supervisionado na Prática

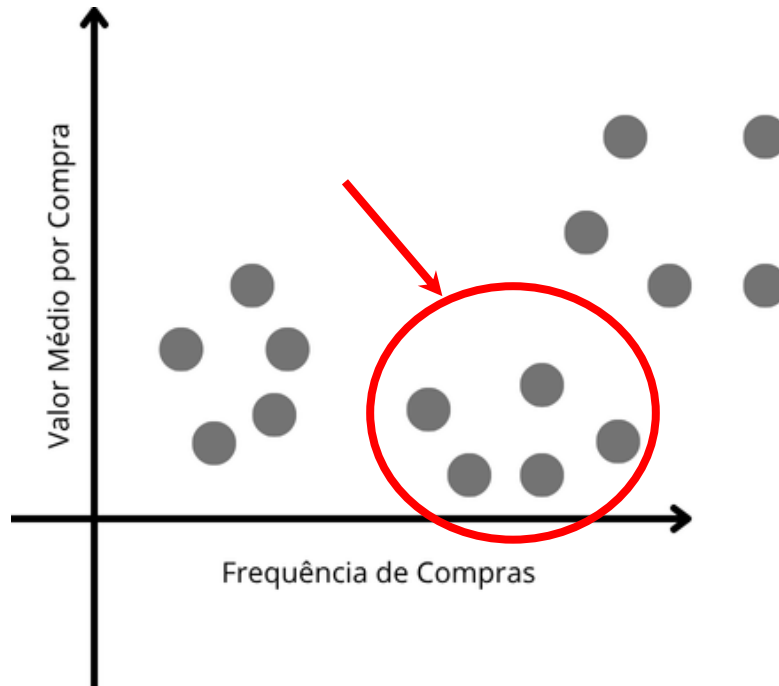
Agrupamento



- **Quantos perfis vocês veem aqui?**
- Perfil 1: Possui uma frequência de compras baixas e valor médio de compra baixo.

Aprendizado Não Supervisionado na Prática

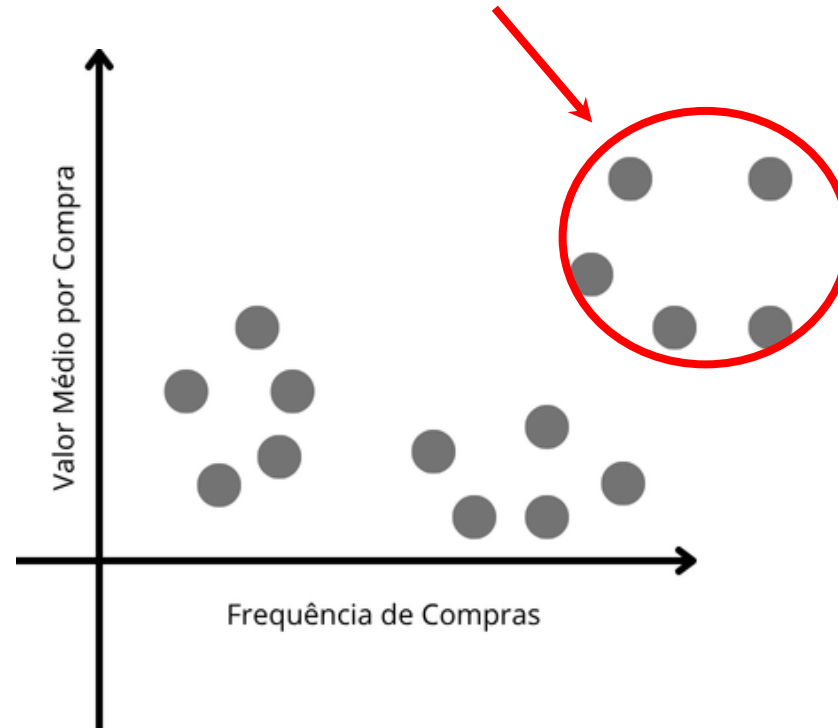
Agrupamento



- **Quantos perfis vocês veem aqui?**
- Perfil 2: Possui uma frequência de compras intermediário e valor médio de compra baixo.

Aprendizado Não Supervisionado na Prática

Agrupamento



- **Quantos perfis vocês veem aqui?**
- Perfil 3: Possui uma frequência de compras alta e valor médio de compra baixo.

Aprendizado Não Supervisionado na Prática

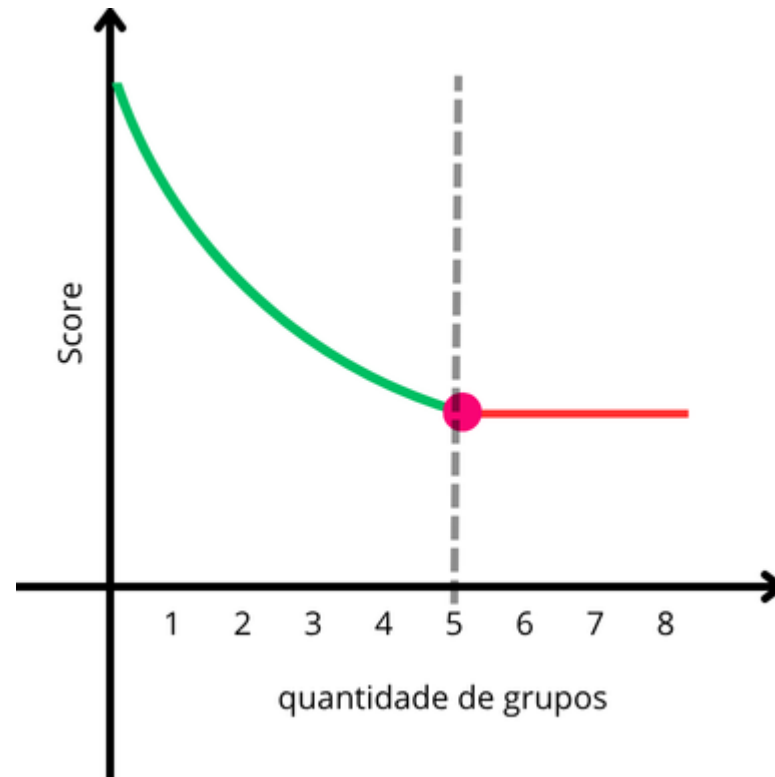
O que é o Kmeans?

- O K-means é um algoritmo de agrupamento (clustering) **que divide um conjunto de dados em K grupos distintos, chamados de clusters.**
- **O objetivo é que os pontos dentro de um mesmo grupo sejam semelhantes entre si e diferentes dos pontos em outros grupos.**
- **O primeiro passo do algoritmo Kmeans é determinar o número de categorias de perfis (clusters),** ou seja ele já pressupõe que você já sabe previamente a quantidade de categorias de perfis adequada para resolver o problema.
- **Nesse exemplo fica evidente que temos 3 perfis que podem ser gerados.** Mas e se não pudéssemos enxergar de maneira clara esse padrão? Como faríamos?

Passo 1: Identificando o número de Clusters

Gráfico de Cotovelo (Elbow Plot)

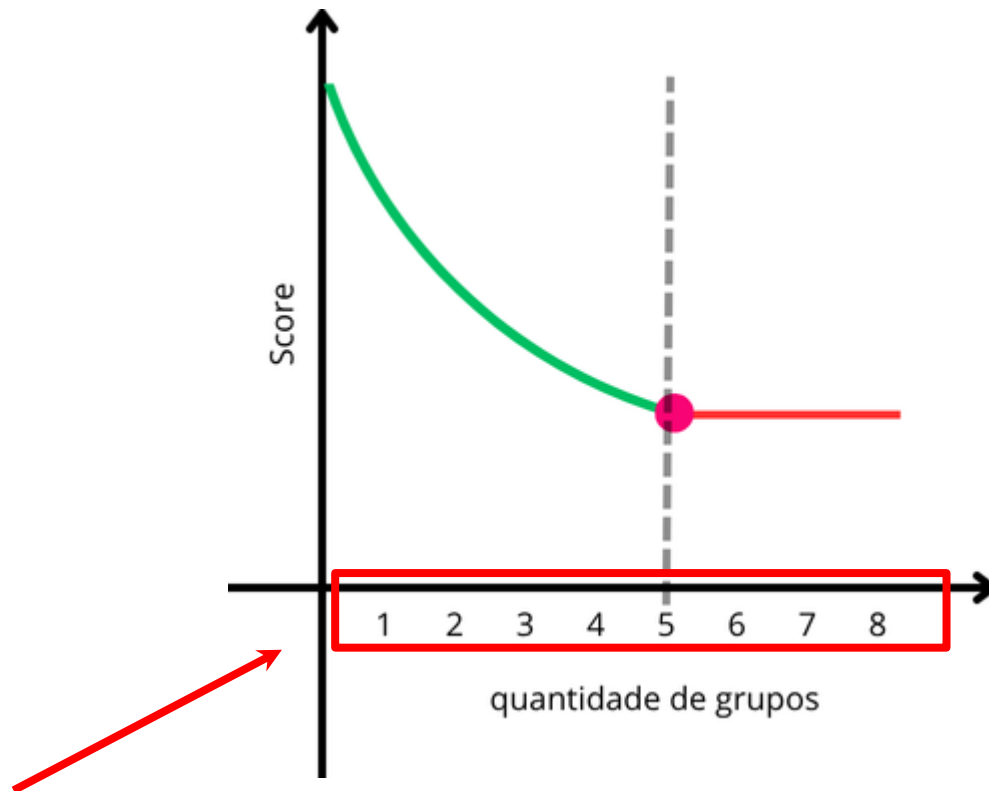
- Para tomar a decisão da quantidade de categorias a escolher, utilizamos uma técnica visual denominada **gráfico de “cotovelo”**.



Passo 1: Identificando o número de Clusters

Gráfico de Cotovelo (Elbow Plot)

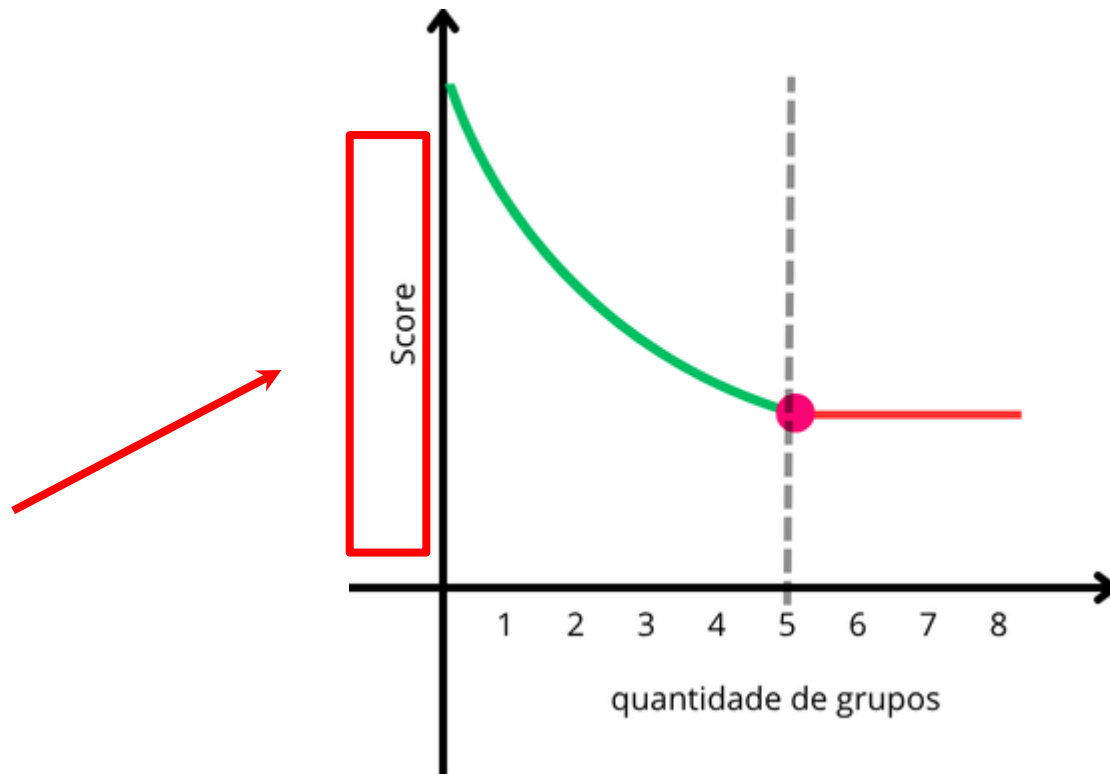
- **Eixo X (Quantidade de Grupos):** É o número de categorias (K) que estamos testando. Para gerar este gráfico, rodamos o algoritmo K-Means várias vezes: uma vez com $K=1$, depois com $K=2$, $K=3$, e assim por diante. Note que é você quem escolhe a quantidade de grupos.



Passo 1: Identificando o número de Clusters

Gráfico de Cotovelo (Elbow Plot)

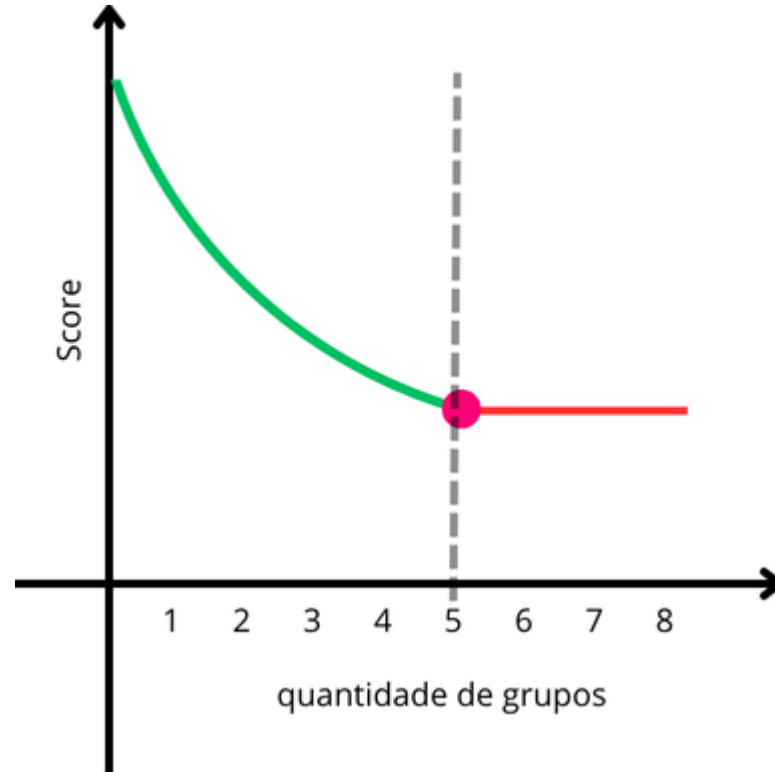
- **Eixo Y (Score / Inércia):** Esta é a métrica chave. Ela mede o quão compactos estão os nossos clusters (categorias). Tecnicamente, é a soma das distâncias ao quadrado de cada ponto de dado até o centro do seu próprio cluster (centróide). Essa métrica é calculada automaticamente.



Passo 1: Identificando o número de Clusters

Gráfico de Cotovelo (Elbow Plot)

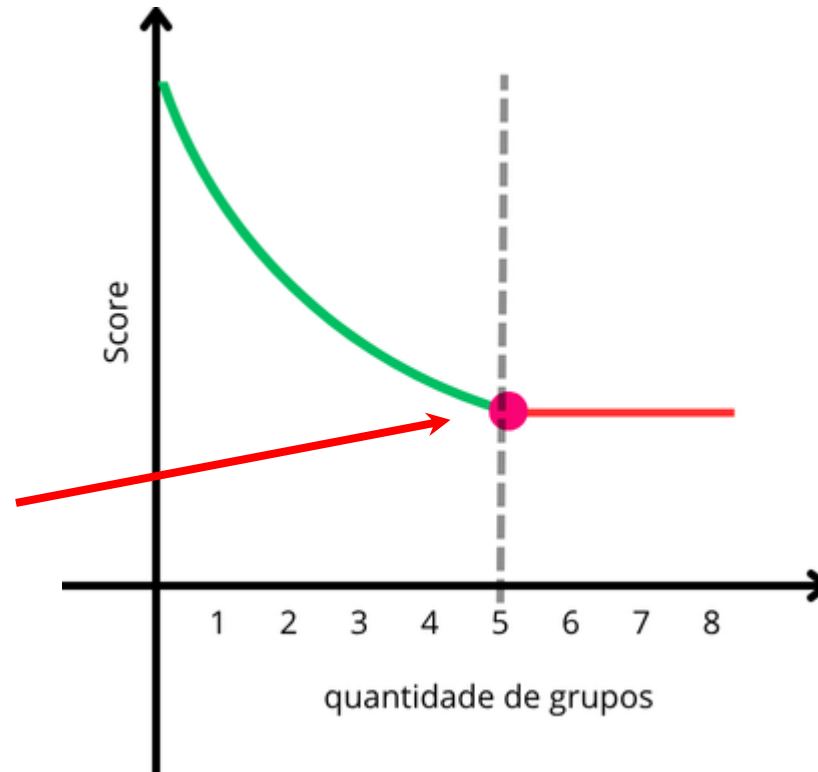
- **A regra é simples:** quanto menor o score, isto é quanto mais próximo de 0, mais agrupados e bem definidos estão os clusters.



Passo 1: Identificando o número de Clusters

Gráfico de Cotovelo (Elbow Plot)

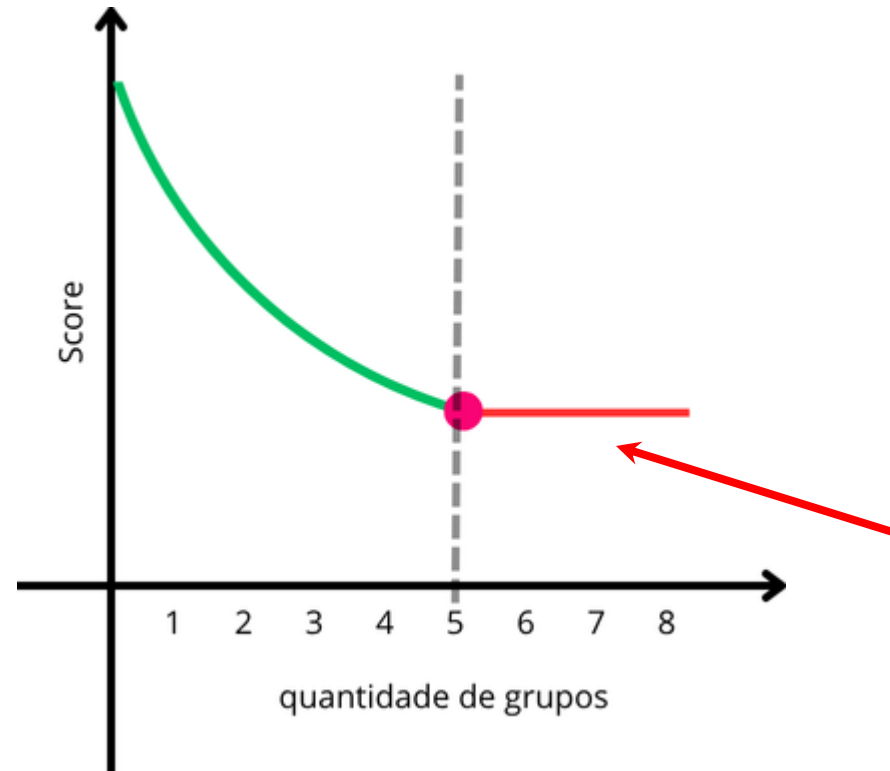
- **A Curva Verde (De K=1 até K=5):** Observe a queda acentuada. Cada novo cluster que adicionamos nesta fase causa uma redução drástica no "Score". Isso significa que cada grupo adicional está capturando uma parte significativa da estrutura dos dados, melhorando muito a organização geral. O ganho em adicionar um novo cluster é alto.



Passo 1: Identificando o número de Clusters

Gráfico de Cotovelo (Elbow Plot)

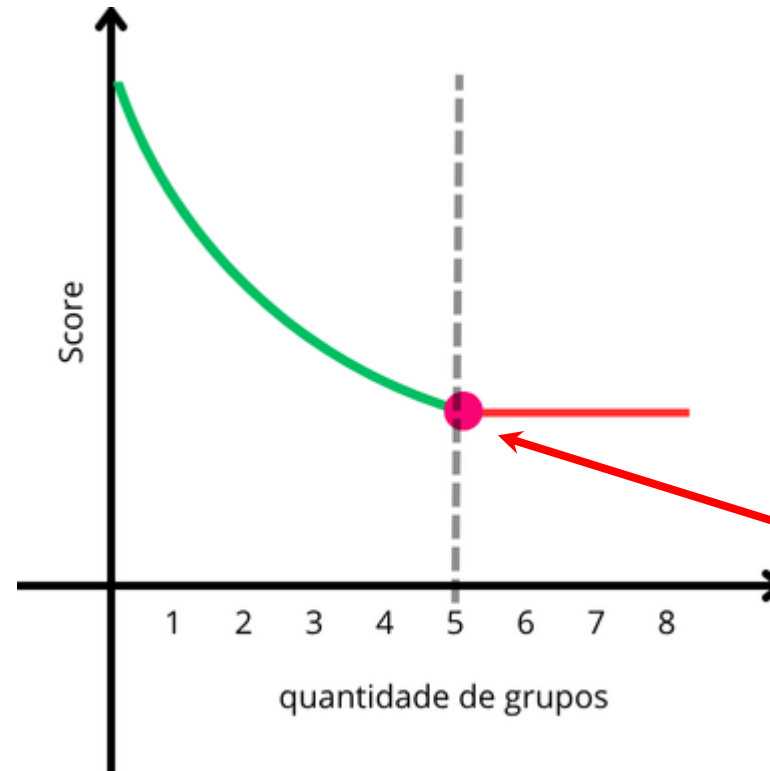
- **A Curva Vermelha (De K=5 em diante):** Veja como a curva se torna praticamente horizontal. Adicionar mais clusters (6, 7, 8) não traz um benefício relevante. O "Score" quase não diminui mais. Isso indica que estamos no ponto de retorno decrescente: estamos apenas dividindo grupos que já estavam bem definidos, sem adicionar nova informação útil.



Passo 1: Identificando o número de Clusters

Gráfico de Cotovelo (Elbow Plot)

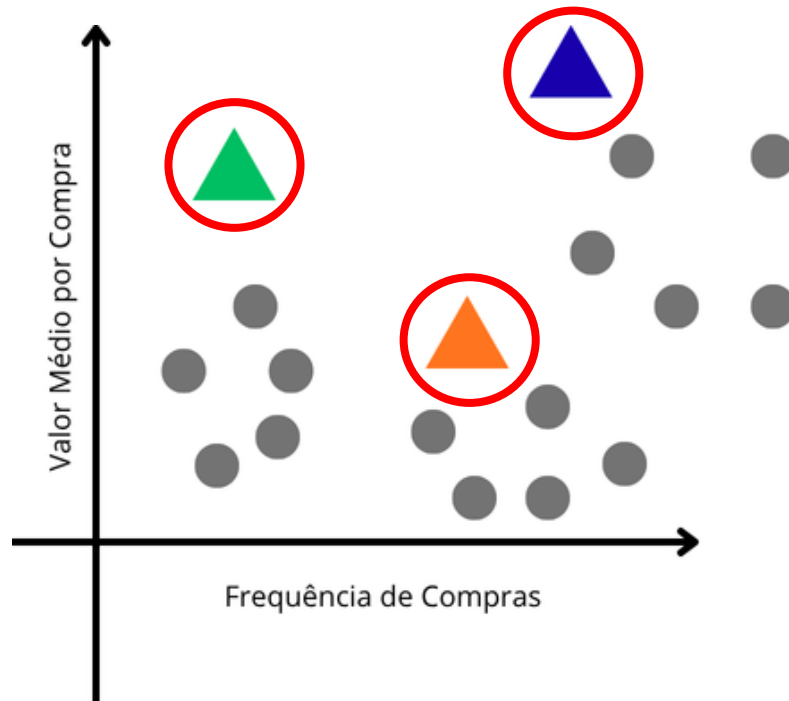
- Com base neste gráfico, o número ideal de grupos para segmentar nossos dados é 5. Agora conseguimos identificar de maneira eficiente a quantidade de clusters a serem utilizados em qualquer problema de agrupamento (clusterização).



Passo 2: Inicialização

Kmeans na prática:

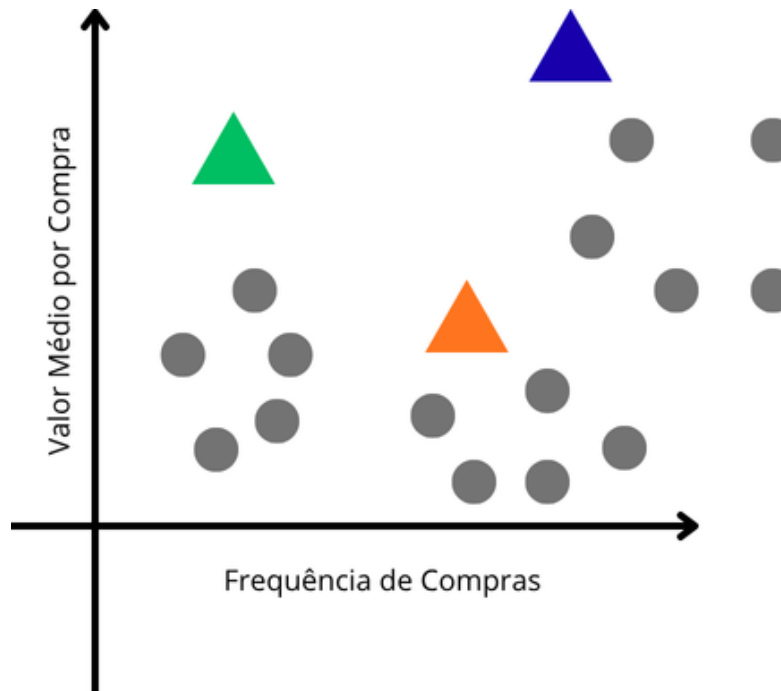
- No exemplo anterior, vimos que temos 3 perfis aparentes então nosso número de cluster **será igual a $k=3$** . Note que podemos também identificar pelo gráfico de cotovelo, mas nesse cenário a abordagem visual pelo gráfico de dispersão é mais simples.
- **A próxima etapa será chamada de inicialização.** Posicionamento Aleatório: O algoritmo coloca 3 pontos, chamados centróides, em locais aleatórios no nosso gráfico. Eles podem ou não cair em cima de um cliente existente, lembre-se que nossa missão é encontrar os perfis dos clientes.



Passo 2: Inicialização

Kmeans na prática:

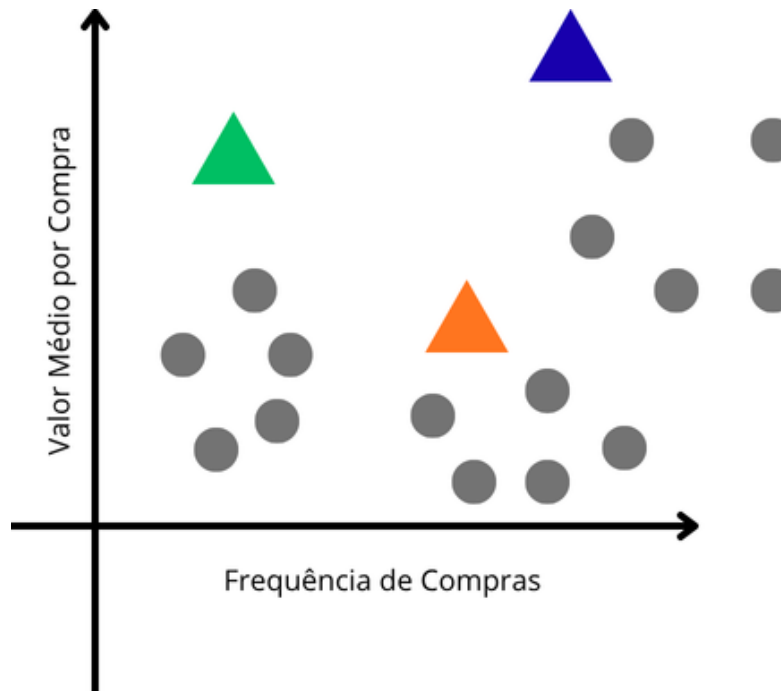
- É crucial entender que esta primeira etapa é um "chute", o algoritmo que aleatoriza e não o pesquisador. **O algoritmo ainda não tem ideia de onde os grupos realmente estão.** A localização inicial é arbitrária e serve apenas como uma referência inicial.



Passo 2: Inicialização

Kmeans na prática:

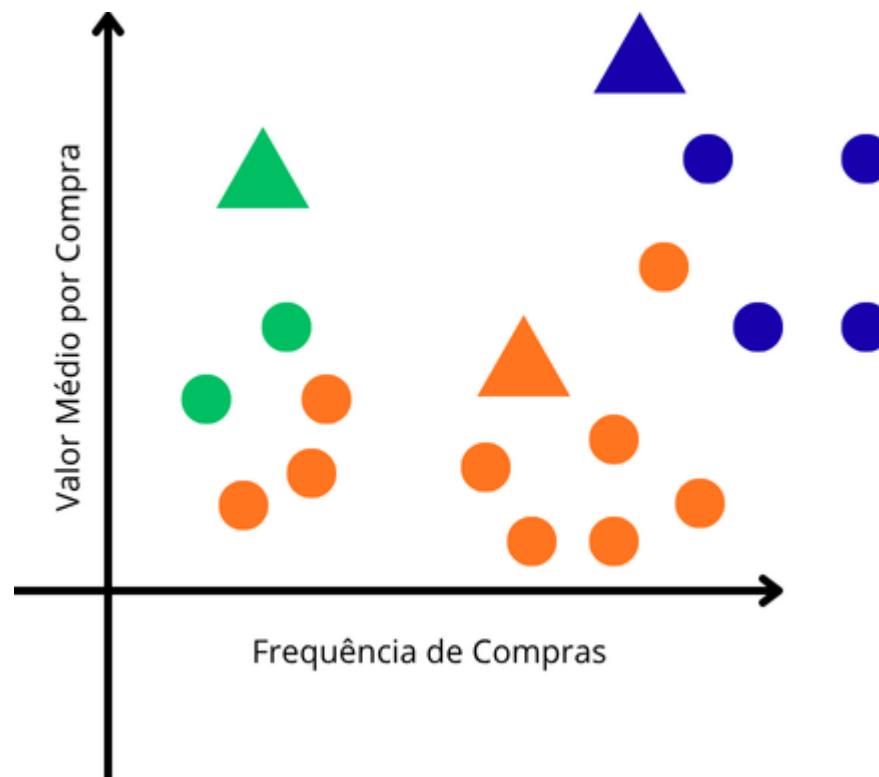
- **Candidatos a Centro:** Cada um desses centróides coloridos é um candidato a se tornar o "coração" (o centro de gravidade) de um dos nossos segmentos de clientes. Por enquanto, eles são apenas hipóteses.



Passo 3: Medição da Distância

Kmeans na prática:

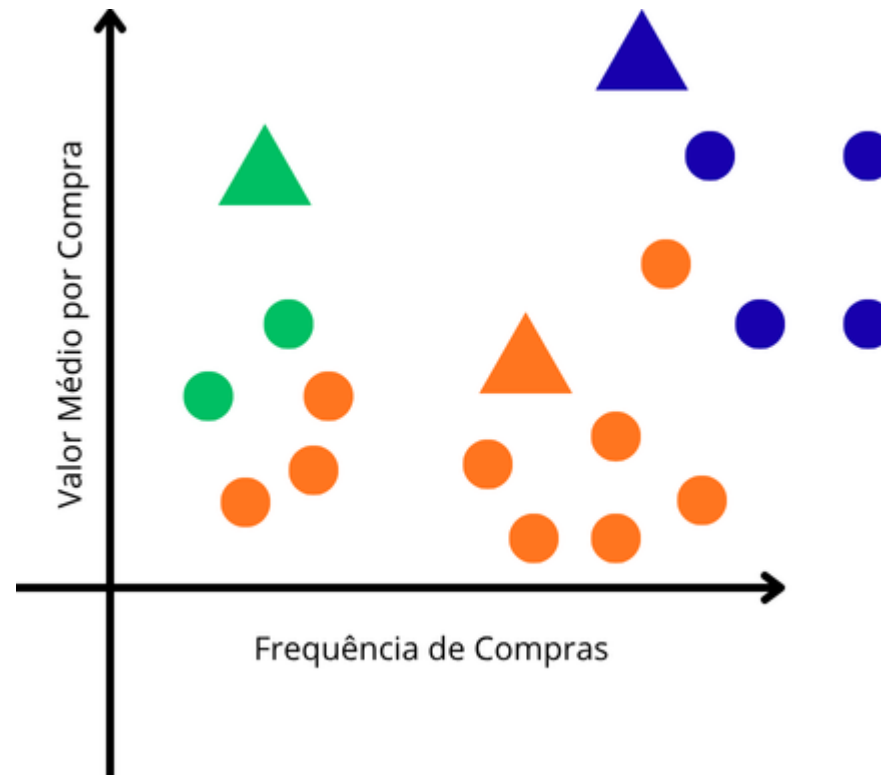
- **Medir a Distância:** Para cada cliente (bolinha), o algoritmo calcula a distância em linha reta até o centróide verde, até o centróide Azul e até o centróide laranja.



Passo 3: Medição da Distância

Kmeans na prática:

- **Colorir o Ponto:** O ponto, que antes era cinza, é então pintado com a cor do seu centróide vencedor, isto é o que estiver mais próximo do centróide.

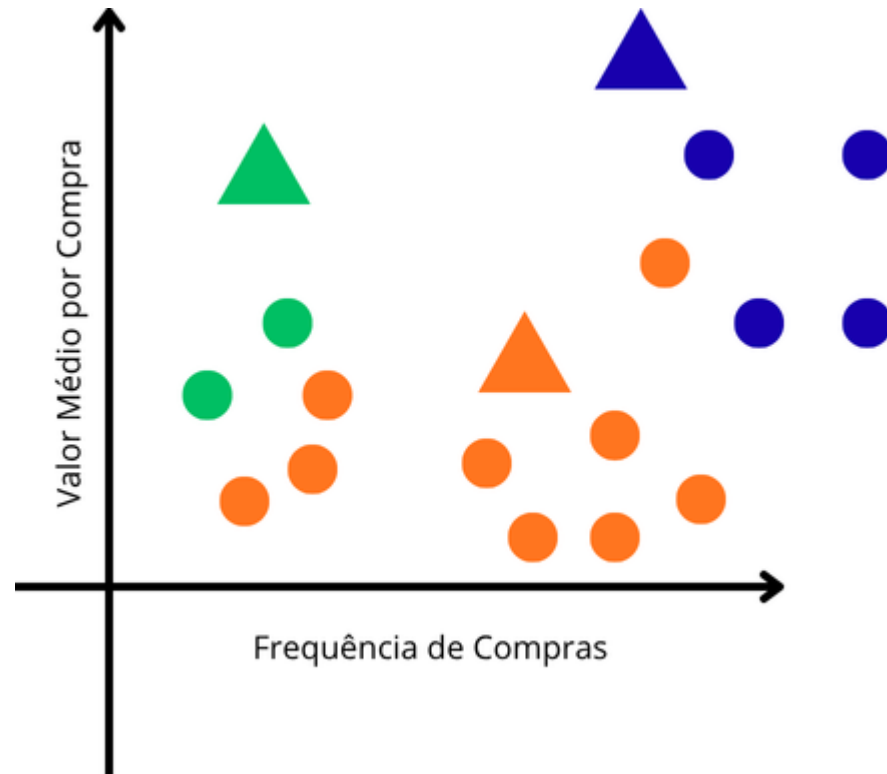


- **Mas há um problema fundamental!**

Passo 3: Medição da Distância

Kmeans na prática:

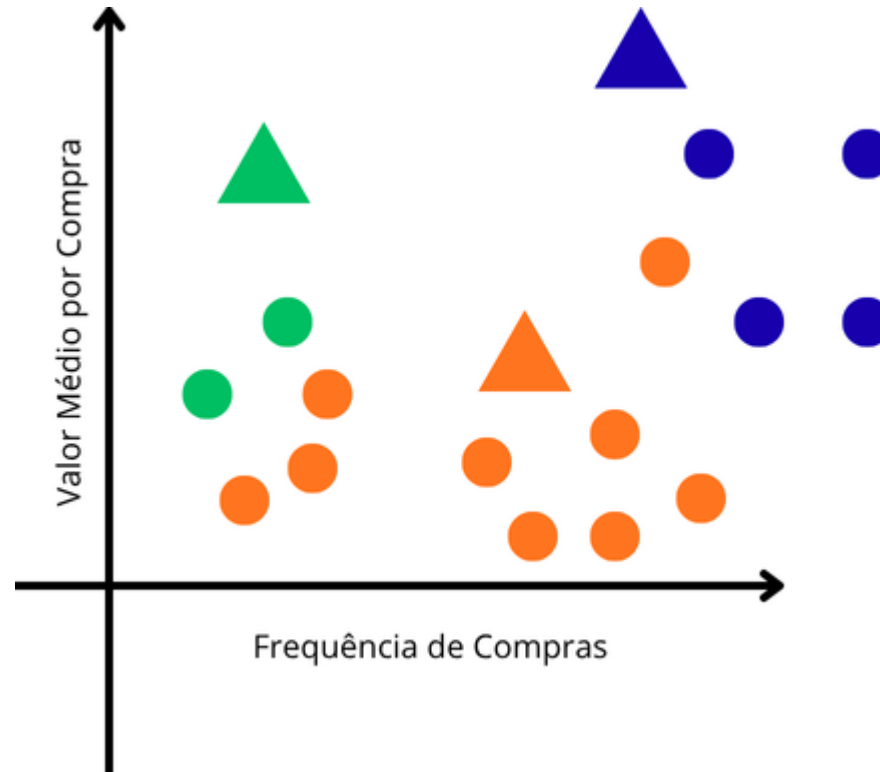
- Estes grupos foram formados com base nos nossos centróides aleatórios, que eram apenas um chute inicial. Os centros que usamos não representam a "média" real de seus membros. Eles ainda estão no lugar errado.



Passo 3: Medição da Distância

Kmeans na prática:

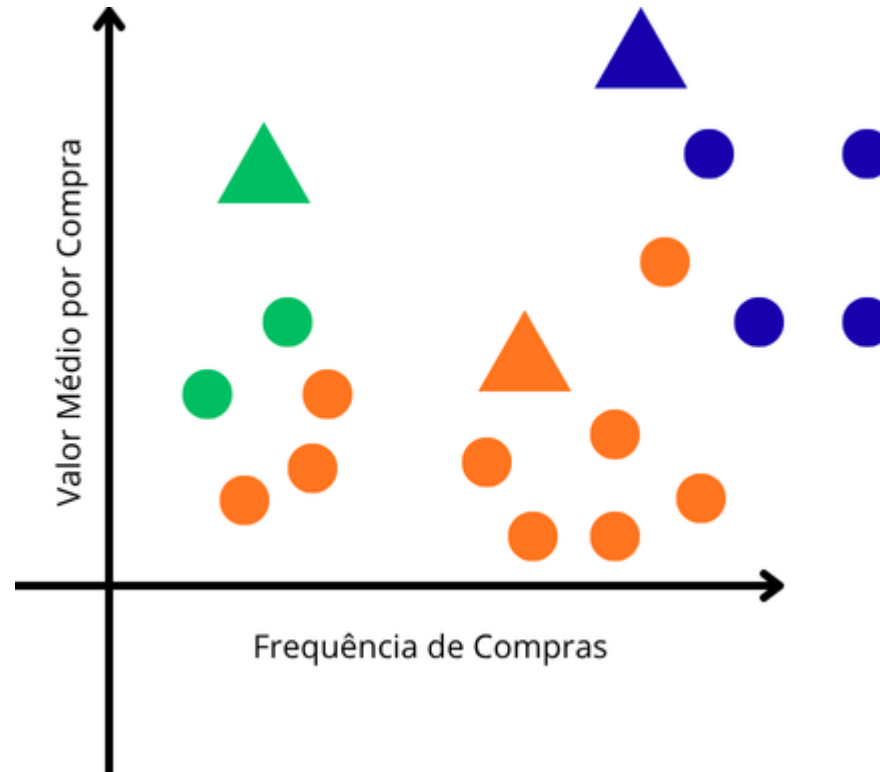
- No passo anterior, formamos nossos primeiros grupos, mas eles foram baseados em um chute. Os centróides ainda estão nos lugares aleatórios onde começaram. Agora, vamos corrigir isso e **movê-los para onde eles realmente deveriam estar**: o "coração" de cada segmento.



Passo 3: Medição da Distância

Kmeans na prática:

- O que é o "coração" ou "centro de gravidade" de um grupo? **É simplesmente o ponto médio, ou a média, de todos os clientes que pertencem à ele. O deslocamento dos centróides acontece depois de já termos calculado (e pintado) cada bolinha (cliente).**



Passo 4: Calculando a Média dos Grupos

Kmeans na prática:

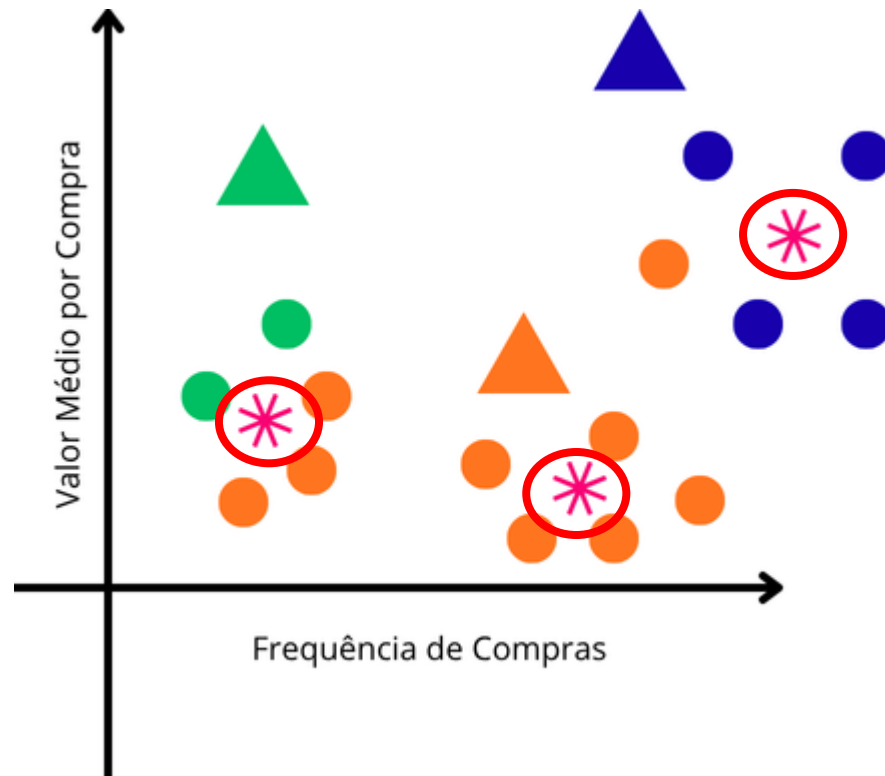
- Para cada grupo de cor, o algoritmo executa um cálculo simples:
 - Foco no Grupo: Pega todos os clientes para cada grupo.
 - Calcula a média de "Frequência de Compras" de todos os pontos por grupo.
 - Calcula a média de "Valor Médio por Compra" de todos os pontos por grupo.
- Move o Centróide: A nova coordenada de cada centróide são exatamente os pontos médios em relação a cada grupo

Vamos visualizar!

Passo 4: Calculando a Média dos Grupos

Kmeans na prática:

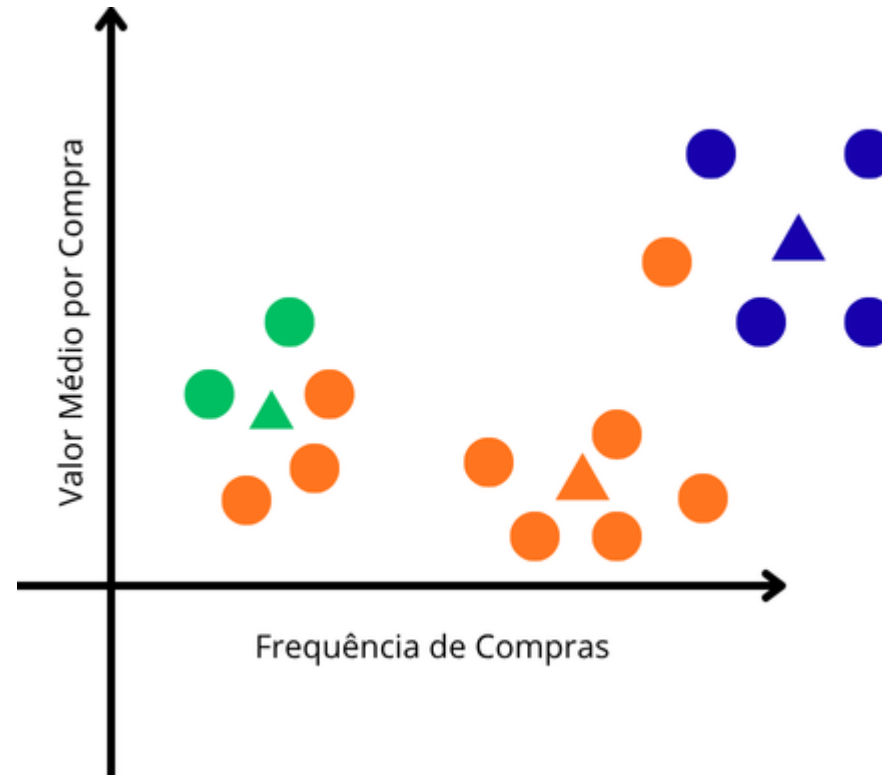
- Cada “estrela” é o ponto médio do grupo.



Passo 4: Calculando a Média dos Grupos

Kmeans na prática:

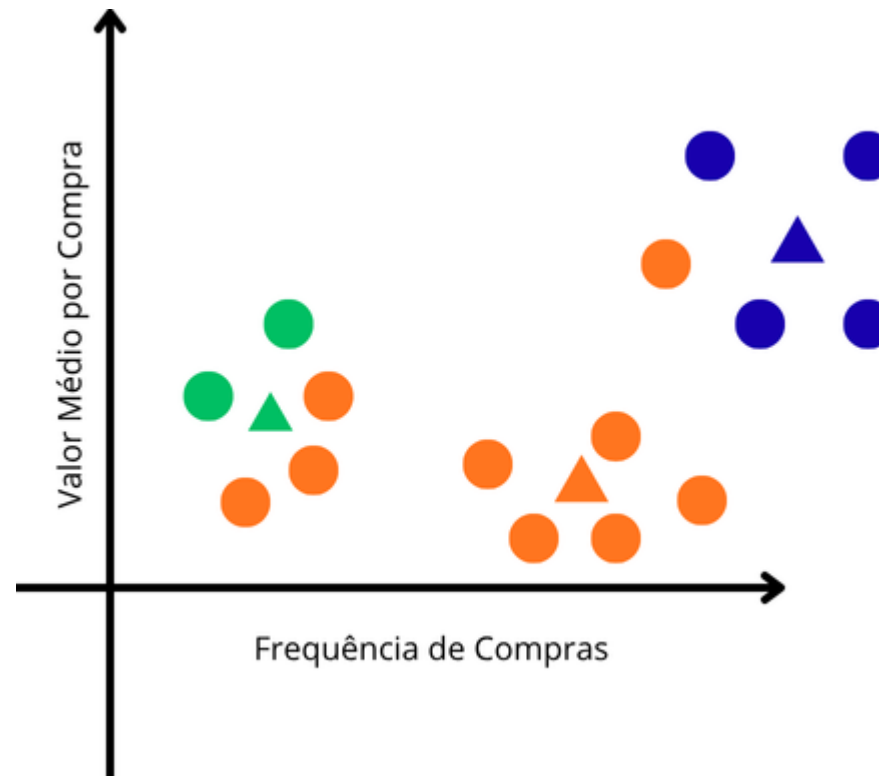
- Agora cada um dos centróides irá se mover para o ponto médio. Assim que for feito esse deslocamento, o algoritmo irá refazer os passos de **Calcular a Distância entre os clientes (slide 25)** e o **Cálculo do ponto médio (slide 30)**.



Passo 4: Calculando a Média dos Grupos

Kmeans na prática:

- O algoritmo é iterativo, ou seja ele sempre vai refinar o agrupamento com base nos passos de **Calcular a Distância entre os clientes (slide 25)** e o **Cálculo do ponto médio (slide 30)**.

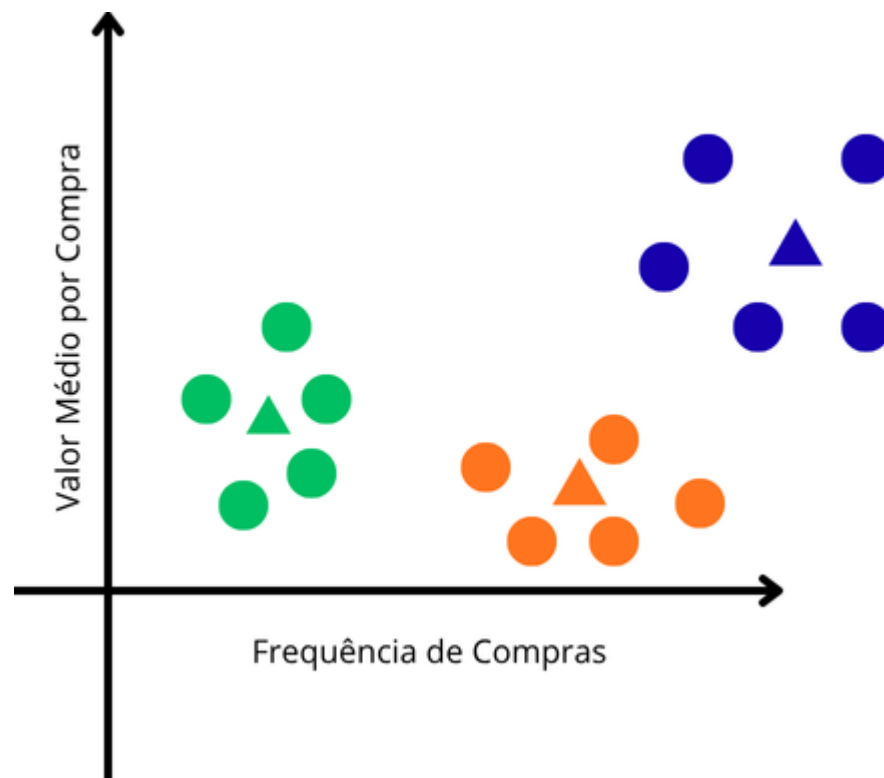


E quando ele para?

Quando o Algoritmo Para

Kmeans na prática:

- O critério de parada é muito simples: O ciclo termina quando, após as etapas de **Calcular a Distância entre os clientes (slide 25)** e o **Cálculo do ponto médio (slide 30)**, NENHUM cliente muda de grupo.



Kmeans Para Mais de Duas Variáveis

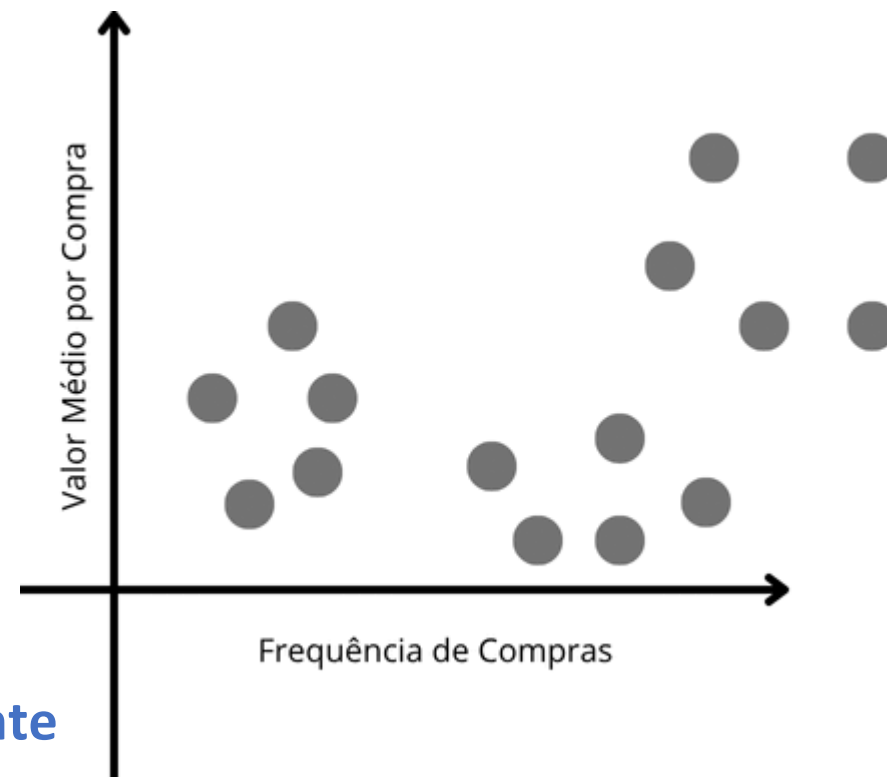
O que acontece?

- Vimos que o **exemplo só tem duas variáveis**: Frequência de Compras e Valor Médio das Compras.
- **Mas o que acontece se tivéssemos mais de duas variáveis?** Por exemplo, Número de Itens por Compra, Valor Total Gasto, etc. Como poderíamos fazer o Kmeans?
- A boa notícia é que **o Kmeans pode ser usado para mais de duas variáveis**. Porém, perderíamos a capacidade de visualizar por meio do gráfico de dispersão, já que só enxergamos no máximo até três dimensões.
- Mas para sanar essa problemática, **podemos fazer o uso do gráfico de Cotovelo**, assim teríamos a noção de quantos clusters utilizar na nossa base de dados sem pormenores relacionados a dimensão.

Recapitulando o Kmeans

Recapitulando o Kmeans

- **1- Definição do Hiperparâmetro (K):** Análise prévia, como o Método do Cotovelo, para determinar o número ótimo de clusters.
 - **2 - Inicialização:** Estabelecimento das posições iniciais dos K centróides no espaço de features.
 - **3 - Atribuição:** Cada ponto de dado é associado ao centróide mais próximo.
 - **4 - Atualização:** A posição de cada centróide é recalculada como a média dos pontos de seu cluster.
- Repita os passos 3 e 4 até que **NENHUM** cliente muda de grupo.



Limitações

Limitação Essencial: A Incompatibilidade com Variáveis Categóricas

- O K-Means é um algoritmo, cuja mecânica interna depende de dois conceitos matemáticos essenciais: distância e média. Essa dependência impõe uma restrição crítica sobre os tipos de dados que ele pode processar diretamente.

A Barreira Conceitual

- As operações centrais do K-Means são:
 - Atribuição: Alocar um ponto ao centróide mais próximo, o que exige o cálculo de uma distância.
 - Atualização: Reposicionar o centróide no ponto médio de todos os seus membros.
- Essas operações são matematicamente indefinidas para variáveis categóricas nominais.

Limitações

A Barreira Conceitual

- As variáveis categóricas nominais **são aquelas que não possuem algum tipo de hierarquia ou ordem**. Não existe “maior” ou “menor”, apenas tipos diferentes (Ex: Cor dos olhos, Tipo de crime).
- Como **o algoritmo se baseia em cálculos de distância**, seria matematicamente inviável calcular a distância entre a Cor azul e a Cor vermelha.
- Assim o **Kmean é usado apenas para variáveis contínuas**, por exemplo Renda, valor médio de compras, frequência de compras, etc.
- Se tiver alguma variável categórica, ela **não deve ser utilizada para a abordagem de agrupamento via kmeans**.

Etapas de Machine Learning para Aprendizado Não Supervisionado

O Pré-Processamento no Aprendizado Não Supervisionado

- No Aprendizado Supervisionado, o pipeline de pré-processamento é rigorosamente estruturado para evitar o vazamento de dados (data leakage) e permitir uma avaliação imparcial da capacidade de generalização do modelo.
- A divisão dos dados em conjuntos de treino e teste (**train_test_split**) é, portanto, uma etapa indispensável.
- No Aprendizado Não Supervisionado, o objetivo fundamental é outro. Não buscamos treinar um modelo para prever uma variável-alvo em dados futuros, mas sim descobrir a estrutura inerente e os padrões latentes no conjunto de dados completo que possuímos.

Limitações

O Fim da Divisão Treino-Teste

- Como o objetivo é a descoberta de padrões na totalidade dos dados, a separação em treino e teste perde seu propósito principal.
- A avaliação é feita sobre a qualidade da estrutura encontrada (ex: coesão dos clusters), e não sobre previsões corretas.

Quais Etapas de Pré-Processamento Permanecem Críticas?

- Embora a divisão de dados seja descartada, outras etapas de preparação não apenas permanecem relevantes, como se tornam ainda mais cruciais para o sucesso do modelo.
- A imputação de valores faltantes e a transformação dos dados são necessárias, já que o Kmeans não foi construído para lidar com dados faltantes e com unidades de medidas diferentes.

Outras Abordagens de Aprendizado Não Supervisionado

- O aprendizado não supervisionado **não se limita apenas agrupar os indivíduos** de característica semelhante, ele possui outras abordagens extremamente úteis.

PCA (Análise de Componentes Principais)

- **Reduz a dimensionalidade dos dados**, transformando variáveis correlacionadas em um conjunto menor de componentes independentes.
- É muito usada para visualização, compressão e preparação de dados antes de aplicar outros modelos.
- O PCA é **muito útil para redução de dimensionalidade, eliminação de redundância** e identificação de padrões latentes.

Outras Abordagens de Aprendizado Não Supervisionado

Análise de Associação

- A Análise de Associação tem como objetivo **descobrir relações frequentes entre variáveis ou itens em grandes bases de dados**.
- Em vez de prever uma variável alvo, **ela busca padrões de coocorrência**, revelando quais elementos costumam aparecer juntos em um mesmo registro ou transação.
- O exemplo clássico é a **análise de cestas de mercado**, onde o algoritmo identifica combinações de produtos comprados simultaneamente.
- A partir disso, podem surgir regras de associação como: **“Clientes que compram pão e manteiga também tendem a comprar leite”**.

Vamos a Obra

Desafio!

Vamos observar cenários fictícios onde foi implementado o modelo Kmeans. O objetivo principal é caracterizar cada grupo de acordo com as análises descritivas.

Cenário 1 – Segmentação de Clientes em uma Academia

- Contexto: A academia quer entender melhor o perfil dos seus alunos para personalizar os planos de treino e promoções.

Vamos a Obra

Cenário 1 – Segmentação de Clientes em uma Academia

- **Contexto:** A academia quer entender melhor o perfil dos seus alunos para personalizar os planos de treino e promoções.
- Foi realizada uma coleta de dados com 500 clientes, contendo as variáveis:
 - **Idade (anos)**
 - **Frequência semanal de visitas (dias/semana)**
 - **Tempo médio de treino (minutos por sessão)**

Vamos a Obra

Cenário 1 – Segmentação de Clientes em uma Academia

Grupo	Idade Média	Frequência Semanal	Tempo Médio de Treino
1	22 anos	2,1 dias	45 min
2	35 anos	4,5 dias	70 min
3	48 anos	3,2 dias	55 min

Quais são as características de cada grupo?

- **Grupo 1:** Jovens iniciantes, com presença irregular e treinos curtos.

Vamos a Obra

Cenário 1 – Segmentação de Clientes em uma Academia

Grupo	Idade Média	Frequência Semanal	Tempo Médio de Treino
1	22 anos	2,1 dias	45 min
2	35 anos	4,5 dias	70 min
3	48 anos	3,2 dias	55 min

Quais são as características de cada grupo?

- **Grupo 2:** Adultos disciplinados, frequentam a academia quase todos os dias.

Vamos a Obra

Cenário 1 – Segmentação de Clientes em uma Academia

Grupo	Idade Média	Frequência Semanal	Tempo Médio de Treino
1	22 anos	2,1 dias	45 min
2	35 anos	4,5 dias	70 min
3	48 anos	3,2 dias	55 min

Quais são as características de cada grupo?

- **Grupo 3:** Clientes maduros, foco em manutenção da saúde e bem-estar. Frequência moderada, treino equilibrado.

Vamos a Obra

Cenário 2 – Padrões de Compra em um E-commerce

- **Contexto:** Uma loja online de produtos eletrônicos chamada TechBuy quer entender seus clientes para definir estratégias de marketing segmentado.
- A base de dados contém 2.000 clientes e inclui:
 - Valor médio por compra (R\$)
 - Número médio de itens por pedido
 - Frequência média de compras por mês

Vamos a Obra

Cenário 2 – Padrões de Compra em um E-commerce

Grupo	Valor Médio (R\$)	Itens por Pedido	Compras/Mês
1	75	3	1,2
2	220	6	3,5
3	480	9	2,0
4	40	2	0,5

Quais são as características de cada grupo?

- **Grupo 1:** Compradores ocasionais, gastam pouco e compram raramente.

Vamos a Obra

Cenário 2 – Padrões de Compra em um E-commerce

Grupo	Valor Médio (R\$)	Itens por Pedido	Compras/Mês
1	75	3	1,2
2	220	6	3,5
3	480	9	2,0
4	40	2	0,5

Quais são as características de cada grupo?

- **Grupo 2:** Clientes frequentes e ativos, compram com regularidade e fazem pedidos médios.

Vamos a Obra

Cenário 2 – Padrões de Compra em um E-commerce

Grupo	Valor Médio (R\$)	Itens por Pedido	Compras/Mês
1	75	3	1,2
2	220	6	3,5
3	480	9	2,0
4	40	2	0,5

Quais são as características de cada grupo?

- **Grupo 3:** Compradores premium, fazem poucas compras, mas de alto valor e maior quantidade por pedido.

Vamos a Obra

Cenário 2 – Padrões de Compra em um E-commerce


Grupo	Valor Médio (R\$)	Itens por Pedido	Compras/Mês
1	75	3	1,2
2	220	6	3,5
3	480	9	2,0
4	40	2	0,5

Quais são as características de cada grupo?

- **Grupo 4:** Clientes pouco engajados e com baixo gasto. Compras esporádicas, podem ser novos clientes ou de baixo interesse.

Recapitulando os Dados aula_01_exemplo_01

- Vamos utilizar o kmeans no nosso exemplo de custos médicos. Mas antes vamos selecionar apenas as bibliotecas críticas.



Código

```
[1]: import pandas as pd # manipulação de tabelas (DataFrames)

from sklearn.impute import SimpleImputer # preenchimento de valores faltantes
from sklearn.preprocessing import StandardScaler # padronização numérica
from sklearn.pipeline import Pipeline # encadear pré-processamento + modelo
import numpy as np # operações numéricas e matrizes
import matplotlib.pyplot as plt # criação de gráficos
from sklearn.cluster import KMeans # modelo kmeans

# Carregando dataset
df = pd.read_csv("aula_01_exemplo_01.csv") # lê o arquivo aula_01_exemplo_01.csv em um DataFrame
```

- Note que reduzimos a quantidade de bibliotecas que vamos utilizar e adicionamos o comando para criar o modelo do Kmeans.

Esse código não retorna nada!

Recapitulando os Dados aula_01_exemplo_01

- Vamos usar o comando `.info()` para identificar quais são as colunas numéricas para utilizarmos no Kmeans.

[2]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype  
---  -
0   age         1338 non-null   int64  
1   sex         1338 non-null   object  
2   bmi         1338 non-null   float64 
3   children    1338 non-null   int64  
4   smoker      1338 non-null   object  
5   region      1338 non-null   object  
6   charges     1338 non-null   float64 
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

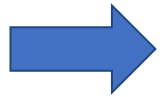
Relembrando:

- **int64:** Representam números inteiros (sem casas decimais)
- **object:** Representa dados de texto
- **float64:** Representa números com casas decimais (contínuos)

- Perceba que vamos utilizar a variável age(idade), bmi(IMC), children (quantidade de dependentes) e charges (custos médicos), isto é, vamos retirar as variáveis que são tipo **object**.

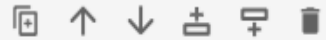
Recapitulando os Dados aula_01_exemplo_01

- Agora vamos criar uma base apenas com as variáveis numéricas.



Código

```
[3]: X = df[["age", "bmi", "children", "charges"]] # selecionando as variáveis numéricas  
X.head()
```



```
[3]:
```

	age	bmi	children	charges
0	19	27.900	0	16884.92400
1	18	33.770	1	1725.55230
2	28	33.000	3	4449.46200
3	33	22.705	0	21984.47061
4	32	28.880	0	3866.85520



Saída

- Acima vemos que selecionamos apenas as variáveis numéricas.

Recapitulando os Dados aula_01_exemplo_01

- Como estamos lidando apenas com as variáveis numéricas, deixaremos apenas o pipeline numérico.

[4]: # Pré-processamento

escalador = StandardScaler() # Escalonador, ele vai padronizar as variáveis numéricas

imputador_numerico = SimpleImputer(strategy="median") # imputador numérico usando mediana



- Vamos novamente criar o escalonador e o imputador para os dados numéricos.

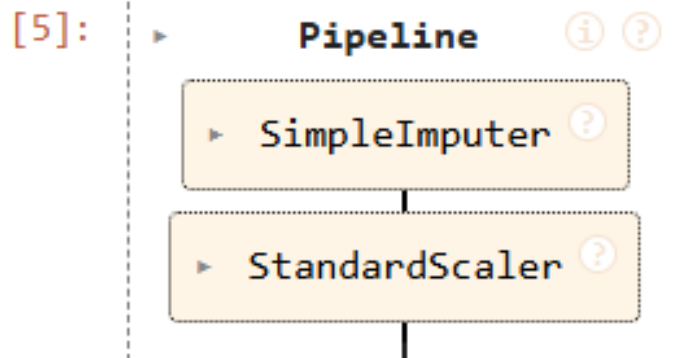
Esse código não retorna nada!

Imputando e Transformando os Dados

- Criando o pipeline numérico, temos:

➡ Código

```
[5]: # Etapas de transformação das variáveis numéricas
etapas_numericas = Pipeline(
    [
        ("imputer", imputador_numerico),    # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela mediana)
        ("scaler", escalonador)             # "scaler" é o nome da etapa → aplica padronização
    ]
)
etapas_numericas
```



- Novamente, criamos nossa etapa numérica.

Imputando e Transformando os Dados

- Agora vamos fazer o `.fit_transform()` para todos os nossos dados. Lembre-se que o `fit_transform()` vai realizar o processo de imputação e transformação dos dados de maneira automática.



```
[6]: x_transformado = etapas_numericas.fit_transform(X) # aplicando a imputação e transformação nos dados
```



Código

- Agora podemos seguir com a modelagem dos dados.

Esse código não retorna nada!

Criando o Gráfico de Cotovelo

- Vamos fazer o primeiro passo e determinar o número de clusters.

➔
Código

```
[7]: score = [] # Cria uma lista vazia para armazenar os valores de Score para cada k
    K_range = range(1, 11) # Define o intervalo de k (número de clusters) de 1 até 10 (esse número é o pesquisador que decide)

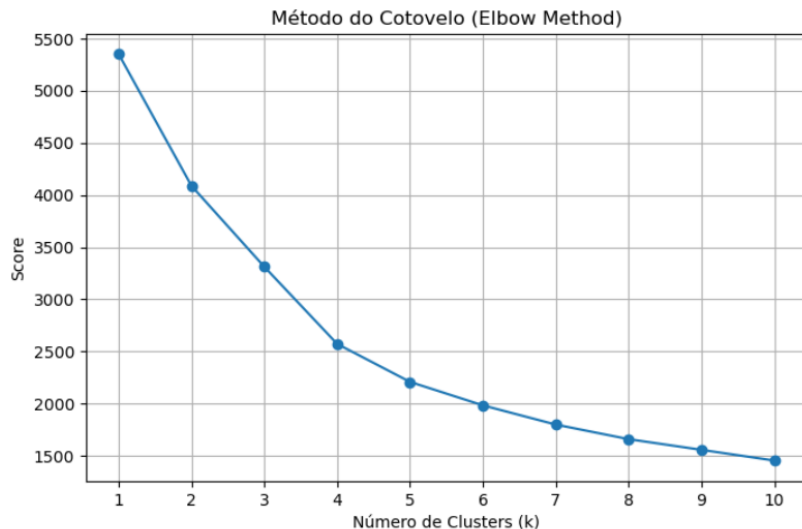
    # Loop para testar diferentes valores de k
    for k in K_range:
        kmeans = KMeans(n_clusters=k, n_init=10, random_state=42) # Cria o modelo KMeans com k clusters, 10 inicializações
        kmeans.fit(x_transformado) # Ajusta o modelo KMeans aos dados transformados
        score.append(kmeans.inertia_) # Armazena o score em uma lista

    # Criação do gráfico do método do cotovelo
    plt.figure(figsize=(8, 5)) # Define o tamanho da figura do gráfico (largura, altura), o pesquisador define o tamanho do gráfico

    # K_range vai colocar no eixo X a quantidade de clusters e o score eixo y, o markers = "o" cria bolinhas no gráfico
    plt.plot(K_range, score, marker='o')

    plt.title('Método do Cotovelo (Elbow Method)') # Define o título do gráfico
    plt.xlabel('Número de Clusters (k)') # Define o rótulo do eixo x
    plt.ylabel('Score') # Define o rótulo do eixo y
    plt.xticks(K_range) # Define os valores do eixo x para mostrar todos os k de 1 a 10
    plt.grid(True) # Ativa a grade no gráfico para melhor visualização
    plt.show() # Exibe o gráfico na tela
```

➔
Saída



- Primeiramente podemos definir um intervalo de possíveis valores de clusters. Isso o pesquisador quem vai decidir, mas geralmente colocamos por convenção valores entre 1 e 10.

Criando o Gráfico de Cotovelo

- Vamos fazer o primeiro passo e determinar o número de clusters.

➡
Código

```
[7]: score = [] # Cria uma lista vazia para armazenar os valores de Score para cada k
    K_range = range(1, 11) # Define o intervalo de k (número de clusters) de 1 até 10 (esse número é o pesquisador que decide)

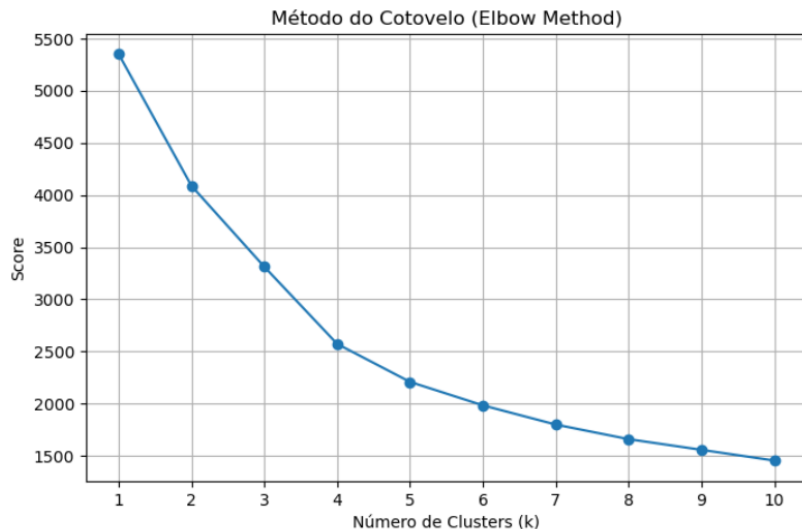
    # Loop para testar diferentes valores de k
    for k in K_range:
        kmeans = KMeans(n_clusters=k, n_init=10, random_state=42) # Cria o modelo KMeans com k clusters, 10 inicializações
        kmeans.fit(x_transformado) # Ajusta o modelo KMeans aos dados transformados
        score.append(kmeans.inertia_) # Armazena o score em uma lista

    # Criação do gráfico do método do cotovelo
    plt.figure(figsize=(8, 5)) # Define o tamanho da figura do gráfico (largura, altura), o pesquisador define o tamanho do gráfico

    # K_range vai colocar no eixo X a quantidade de clusters e o score eixo y, o markers = "o" cria bolinhas no gráfico
    plt.plot(K_range, score, marker='o')

    plt.title('Método do Cotovelo (Elbow Method)') # Define o título do gráfico
    plt.xlabel('Número de Clusters (k)') # Define o rótulo do eixo x
    plt.ylabel('Score') # Define o rótulo do eixo y
    plt.xticks(K_range) # Define os valores do eixo x para mostrar todos os k de 1 a 10
    plt.grid(True) # Ativa a grade no gráfico para melhor visualização
    plt.show() # Exibe o gráfico na tela
```

➡
Saída



- Precisamos criar um loop **for** que irá calcular o Score para cada número de cluster e no fim irá mostrar pra gente o gráfico.

Criando o Gráfico de Cotovelo

- Vamos fazer o primeiro passo e determinar o número de clusters.

➡
Código

```
[7]: score = [] # Cria uma lista vazia para armazenar os valores de Score para cada k
K_range = range(1, 11) # Define o intervalo de k (número de clusters) de 1 até 10 (esse número é o pesquisador que decide)

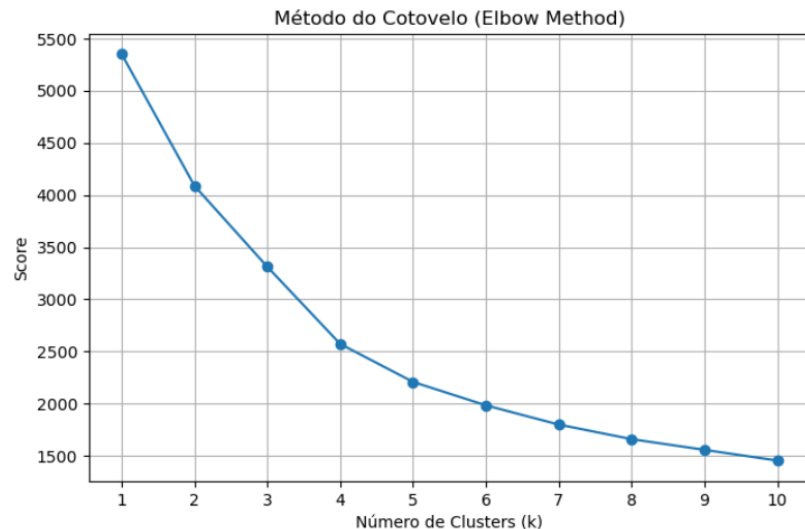
# Loop para testar diferentes valores de k
for k in K_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42) # Cria o modelo KMeans com k clusters, 10 inicializações
    kmeans.fit(X_transformado) # Ajusta o modelo KMeans aos dados transformados
    score.append(kmeans.inertia_) # Armazena o score em uma lista

# Criação do gráfico do método do cotovelo
plt.figure(figsize=(8, 5)) # Define o tamanho da figura do gráfico (largura, altura), o pesquisador define o tamanho do gráfico

# K_range vai colocar no eixo X a quantidade de clusters e o score eixo y, o markers = "o" cria bolinhas no gráfico
plt.plot(K_range, score, marker='o')

plt.title('Método do Cotovelo (Elbow Method)') # Define o título do gráfico
plt.xlabel('Número de Clusters (k)') # Define o rótulo do eixo x
plt.ylabel('Score') # Define o rótulo do eixo y
plt.xticks(K_range) # Define os valores do eixo x para mostrar todos os k de 1 a 10
plt.grid(True) # Ativa a grade no gráfico para melhor visualização
plt.show() # Exibe o gráfico na tela
```

➡
Saída



- Para calcular o Kmeans, é necessário dizer a quantidade de clusters (**n_clusters**) e a quantidade de inicializações(**n_init**).
- O **n_init=10** vai rodar internamente 10 vezes o algoritmo para então selecionar a melhor solução.

Criando o Gráfico de Cotovelo

- Vamos fazer o primeiro passo e determinar o número de clusters.

➡
Código

```
[7]: score = [] # Cria uma lista vazia para armazenar os valores de Score para cada k
K_range = range(1, 11) # Define o intervalo de k (número de clusters) de 1 até 10 (esse número é o pesquisador que decide)

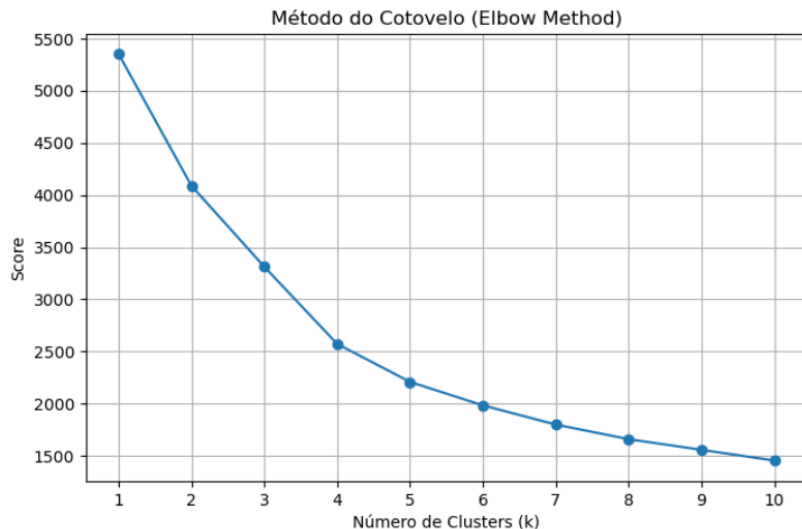
# Loop para testar diferentes valores de k
for k in K_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42) # Cria o modelo KMeans com k clusters, 10 inicializações
    kmeans.fit(X_transformado) # Ajusta o modelo KMeans aos dados transformados
    score.append(kmeans.inertia_) # Armazena o score em uma lista

# Criação do gráfico do método do cotovelo
plt.figure(figsize=(8, 5)) # Define o tamanho da figura do gráfico (largura, altura), o pesquisador define o tamanho do gráfico

# K_range vai colocar no eixo X a quantidade de clusters e o score eixo y, o markers = "o" cria bolinhas no gráfico
plt.plot(K_range, score, marker='o')

plt.title('Método do Cotovelo (Elbow Method)') # Define o título do gráfico
plt.xlabel('Número de Clusters (k)') # Define o rótulo do eixo x
plt.ylabel('Score') # Define o rótulo do eixo y
plt.xticks(K_range) # Define os valores do eixo x para mostrar todos os k de 1 a 10
plt.grid(True) # Ativa a grade no gráfico para melhor visualização
plt.show() # Exibe o gráfico na tela
```

➡
Saída



- Isso é feito para evitar problemas de não convergência do algoritmo do Kmeans.
- Não convergência seria quando o Kmeans não consegue plenamente agrupar os grupos. O `n_init` serve para corrigir isso.

Criando o Gráfico de Cotovelo

- Vamos fazer o primeiro passo e determinar o número de clusters.

➡
Código

```
[7]: score = [] # Cria uma lista vazia para armazenar os valores de Score para cada k
K_range = range(1, 11) # Define o intervalo de k (número de clusters) de 1 até 10 (esse número é o pesquisador que decide)

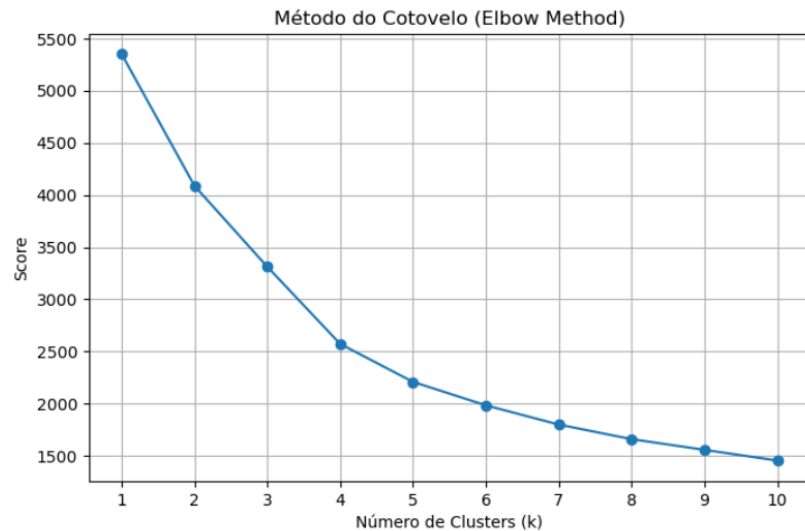
# Loop para testar diferentes valores de k
for k in K_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42) # Cria o modelo KMeans com k clusters, 10 inicializações
    kmeans.fit(x_transformado) # Ajusta o modelo KMeans aos dados transformados
    score.append(kmeans.inertia_) # Armazena o score em uma lista

# Criação do gráfico do método do cotovelo
plt.figure(figsize=(8, 5)) # Define o tamanho da figura do gráfico (largura, altura), o pesquisador define o tamanho do gráfico

# K_range vai colocar no eixo X a quantidade de clusters e o score eixo y, o markers = "o" cria bolinhas no gráfico
plt.plot(K_range, score, marker='o')

plt.title('Método do Cotovelo (Elbow Method)') # Define o título do gráfico
plt.xlabel('Número de Clusters (k)') # Define o rótulo do eixo x
plt.ylabel('Score') # Define o rótulo do eixo y
plt.xticks(K_range) # Define os valores do eixo x para mostrar todos os k de 1 a 10
plt.grid(True) # Ativa a grade no gráfico para melhor visualização
plt.show() # Exibe o gráfico na tela
```

➡
Saída



- Com o gráfico criado, qual número de clusters definir?

Criando o Gráfico de Cotovelo

- Vamos fazer o primeiro passo e determinar o número de clusters.

➡
Código

```
[7]: score = [] # Cria uma lista vazia para armazenar os valores de Score para cada k
    K_range = range(1, 11) # Define o intervalo de k (número de clusters) de 1 até 10 (esse número é o pesquisador que decide)

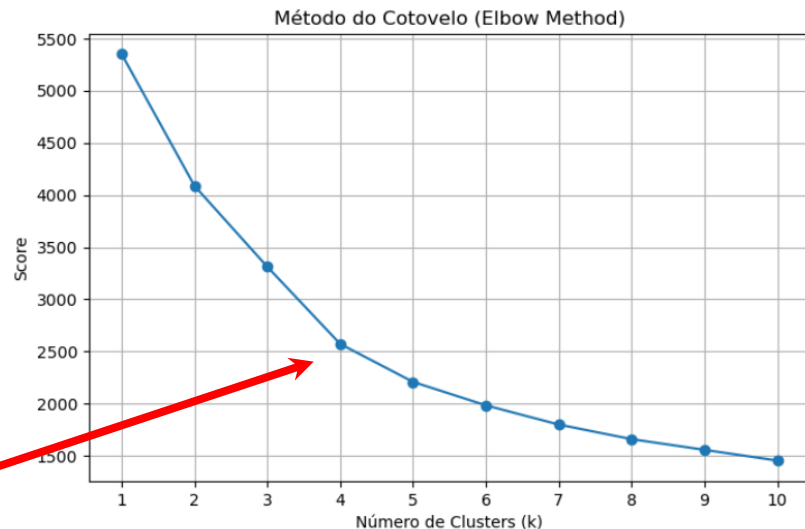
    # Loop para testar diferentes valores de k
    for k in K_range:
        kmeans = KMeans(n_clusters=k, n_init=10, random_state=42) # Cria o modelo KMeans com k clusters, 10 inicializações
        kmeans.fit(x_transformado) # Ajusta o modelo KMeans aos dados transformados
        score.append(kmeans.inertia_) # Armazena o score em uma lista

    # Criação do gráfico do método do cotovelo
    plt.figure(figsize=(8, 5)) # Define o tamanho da figura do gráfico (largura, altura), o pesquisador define o tamanho do gráfico

    # K_range vai colocar no eixo X a quantidade de clusters e o score eixo y, o markers = "o" cria bolinhas no gráfico
    plt.plot(K_range, score, marker='o')

    plt.title('Método do Cotovelo (Elbow Method)') # Define o título do gráfico
    plt.xlabel('Número de Clusters (k)') # Define o rótulo do eixo x
    plt.ylabel('Score') # Define o rótulo do eixo y
    plt.xticks(K_range) # Define os valores do eixo x para mostrar todos os k de 1 a 10
    plt.grid(True) # Ativa a grade no gráfico para melhor visualização
    plt.show() # Exibe o gráfico na tela
```

➡
Saída



- Sempre olhe para esses pontos de inflexão do gráfico que parecem de fato um “cotovelo”.

Criando o Gráfico de Cotovelo

- Vamos fazer o primeiro passo e determinar o número de clusters.

➡
Código

```
[7]: score = [] # Cria uma lista vazia para armazenar os valores de Score para cada k
K_range = range(1, 11) # Define o intervalo de k (número de clusters) de 1 até 10 (esse número é o pesquisador que decide)

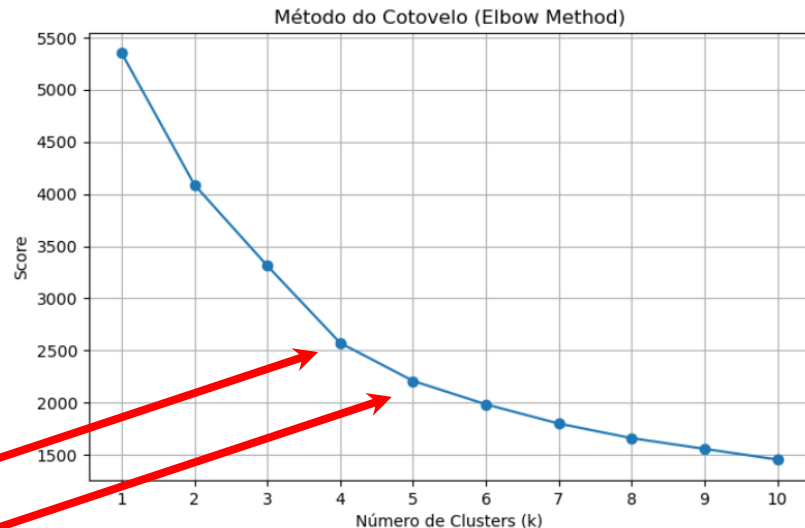
# Loop para testar diferentes valores de k
for k in K_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42) # Cria o modelo KMeans com k clusters, 10 inicializações
    kmeans.fit(x_transformado) # Ajusta o modelo KMeans aos dados transformados
    score.append(kmeans.inertia_) # Armazena o score em uma lista

# Criação do gráfico do método do cotovelo
plt.figure(figsize=(8, 5)) # Define o tamanho da figura do gráfico (largura, altura), o pesquisador define o tamanho do gráfico

# K_range vai colocar no eixo X a quantidade de clusters e o score eixo y, o markers = "o" cria bolinhas no gráfico
plt.plot(K_range, score, marker='o')

plt.title('Método do Cotovelo (Elbow Method)') # Define o título do gráfico
plt.xlabel('Número de Clusters (k)') # Define o rótulo do eixo x
plt.ylabel('Score') # Define o rótulo do eixo y
plt.xticks(K_range) # Define os valores do eixo x para mostrar todos os k de 1 a 10
plt.grid(True) # Ativa a grade no gráfico para melhor visualização
plt.show() # Exibe o gráfico na tela
```

➡
Saída



- Nesse caso temos duas possíveis escolhas:
 - $K = 4$
 - $K = 5$
- Você poderia escolher qualquer uma, mas por questões de simplicidade, escolheremos $k = 4$.

Criando o Modelo Kmeans

- Vamos implementar o Kmeans.



Código

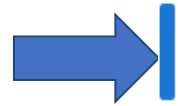
```
[8]: kmeans = KMeans(n_clusters=4, n_init=10, random_state=42) # modelo do Kmeans com k = 4 clusters, rodando 10 vezes o algoritmo internamente (n_init)
      kmeans.fit(x_transformado) # inserindo os dados para extrair os grupos
```

- Agora podemos extrair os grupos gerados pelo Kmeans e entender o perfil dos indivíduos.

Esse código não retorna nada!

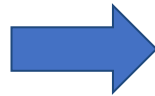
Extraindo os Grupos

- Vamos extrair os grupos que o Kmeans identificou e armazenaremos eles na nossa base de dados original.



Código

```
[9]: df['cluster'] = kmeans.labels_ # extraindo os grupos  
df.head() # visualizando as primeiras 5 linhas
```



Saída

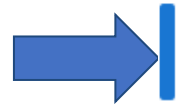
```
[9]:
```

	age	sex	bmi	children	smoker	region	charges	cluster
0	19	female	27.900	0	yes	southwest	16884.92400	3
1	18	male	33.770	1	no	southeast	1725.55230	3
2	28	male	33.000	3	no	southeast	4449.46200	2
3	33	male	22.705	0	no	northwest	21984.47061	3
4	32	male	28.880	0	no	northwest	3866.85520	3

- Com o comando `df['cluster'] = kmeans.labels_`, estamos armazenando nossos grupos em uma nova coluna chamada 'cluster' dentro da nossa base de dados original.

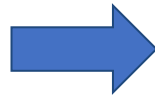
Extraindo os Grupos

- Vamos extrair os grupos que o Kmeans identificou e armazenaremos eles na nossa base de dados original.



Código

```
[9]: df['cluster'] = kmeans.labels_ # extraindo os grupos  
df.head() # visualizando as primeiras 5 linhas
```



Saída

```
[9]:
```

	age	sex	bmi	children	smoker	region	charges	cluster
0	19	female	27.900	0	yes	southwest	16884.92400	3
1	18	male	33.770	1	no	southeast	1725.55230	3
2	28	male	33.000	3	no	southeast	4449.46200	2
3	33	male	22.705	0	no	northwest	21984.47061	3
4	32	male	28.880	0	no	northwest	3866.85520	3

- Assim podemos realizar análises descritivas com o objetivo de entender melhor esses grupos. Pela saída vemos que foi criado a coluna de cluster.

Caracterizando os Grupos Criados

- Usando o `.groupby()` podemos agrupar por cluster e calcular a descritiva para cada variável

→
Código

```
[10]: # Descritivas numéricas por cluster para a variável de idade (age)
desc_age = df.groupby('cluster')['age'].describe() # agrupando o cluster segundo a variável de idade(age) e computando a descritiva para cada cluster

print("=== Age ===") # identificado que é para a variável de idade (Age)
desc_age
```

→
Saída

```
=== Age ===
[10]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	52.514706	7.338082	36.0	47.0	53.0	58.0	64.0
1	165.0	39.872727	14.288902	18.0	27.0	41.0	52.0	64.0
2	346.0	39.939306	10.562680	18.0	31.0	40.0	48.0	64.0
3	419.0	25.381862	6.458351	18.0	19.0	24.0	31.0	41.0

- **Cluster 0 (Média: 52.5 | Mín: 36.0 | Máx: 64.0):** Grupo Mais Velho/Sênior, Concentrado na faixa de clientes maduros a idosos. A média alta, próxima do limite superior, indica uma população mais velha.

Caracterizando os Grupos Criados

- Usando o `.groupby()` podemos agrupar por cluster e calcular a descritiva para cada variável

→
Código

```
[10]: # Descritivas numéricas por cluster para a variável de idade (age)
desc_age = df.groupby('cluster')['age'].describe() # agrupando o cluster segundo a variável de idade(age) e computando a descritiva para cada cluster

print("=== Age ===") # identificado que é para a variável de idade (Age)
desc_age
```

→
Saída

```
=== Age ===
[10]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	52.514706	7.338082	36.0	47.0	53.0	58.0	64.0
1	165.0	39.872727	14.288902	18.0	27.0	41.0	52.0	64.0
2	346.0	39.939306	10.562680	18.0	31.0	40.0	48.0	64.0
3	419.0	25.381862	6.458351	18.0	19.0	24.0	31.0	41.0

- Cluster 1 (Média: 39.9 | Mín: 18.0 | Máx: 64.0):** Possui uma alta variância (14.29). Abrange a maior amplitude etária. A característica-chave é a mistura de todas as idades, sendo provável que outra variável, e não a idade, defina este grupo em específico.

Caracterizando os Grupos Criados

- Usando o `.groupby()` podemos agrupar por cluster e calcular a descritiva para cada variável

→
Código

```
[10]: # Descritivas numéricas por cluster para a variável de idade (age)
desc_age = df.groupby('cluster')['age'].describe() # agrupando o cluster segundo a variável de idade(age) e computando a descritiva para cada cluster

print("=== Age ===") # identificado que é para a variável de idade (Age)
desc_age
```

→
Saída

```
=== Age ===
[10]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	52.514706	7.338082	36.0	47.0	53.0	58.0	64.0
1	165.0	39.872727	14.288902	18.0	27.0	41.0	52.0	64.0
2	346.0	39.939306	10.562680	18.0	31.0	40.0	48.0	64.0
3	419.0	25.381862	6.458351	18.0	19.0	24.0	31.0	41.0

- Cluster 2 (Média: 39.9 | Mín: 18.0 | Máx: 64.0):** Adultos Intermediários. Idade média de adulto, com amplitude máxima total. Distingue-se do Cluster 1 por ter uma concentração muito maior de indivíduos em torno da média (grupo típico de adultos).

Caracterizando os Grupos Criados

- Usando o `.groupby()` podemos agrupar por cluster e calcular a descritiva para cada variável

→
Código

```
[10]: # Descritivas numéricas por cluster para a variável de idade (age)
desc_age = df.groupby('cluster')['age'].describe() # agrupando o cluster segundo a variável de idade(age) e computando a descritiva para cada cluster

print("=== Age ===") # identificado que é para a variável de idade (Age)
desc_age
```

→
Saída

```
=== Age ===
[10]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	52.514706	7.338082	36.0	47.0	53.0	58.0	64.0
1	165.0	39.872727	14.288902	18.0	27.0	41.0	52.0	64.0
2	346.0	39.939306	10.562680	18.0	31.0	40.0	48.0	64.0
3	419.0	25.381862	6.458351	18.0	19.0	24.0	31.0	41.0

- Cluster 3 (Média: 25.4 | Mín: 18.0 | Máx: 41.0):** Jovens Adultos/Jovens. Grupo mais jovem. A média baixa e o limite máximo restrito (41.0) mostram que é um cluster focado em indivíduos na fase inicial da vida adulta.

Caracterizando os Grupos Criados

- Para a variável de IMC.



Código

```
[11]: # Descritivas numéricas por cluster para a variável de IMC(BMI)
desc_bmi = df.groupby('cluster')['bmi'].describe() # agrupando o cluster segundo a variável de IMC(BMI) e computando a descritiva para cada cluster

print("=== BMI ===") # identificado que é para a variável de IMC (BMI)
desc_bmi
```



Saída

```
=== BMI ===
[11]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	30.927292	6.024227	18.050	26.410	30.4000	35.21125	49.06
1	165.0	35.302879	4.315540	22.895	32.200	34.9600	37.07000	52.58
2	346.0	29.963382	6.057864	16.815	25.475	29.7175	33.60625	48.07
3	419.0	29.157482	5.896811	15.960	25.175	28.7850	33.00000	53.13

- **Cluster 0:** A média está no nível de Obesidade (30.9). No entanto, o mínimo (18.05) mostra a presença de indivíduos com peso normal. A maioria do grupo (mediana 30.4) está no limite entre sobrepeso e obesidade.

Caracterizando os Grupos Criados

- Para a variável de IMC.

Código

```
[11]: # Descritivas numéricas por cluster para a variável de IMC(BMI)
desc_bmi = df.groupby('cluster')['bmi'].describe() # agrupando o cluster segundo a variável de IMC(BMI) e computando a descritiva para cada cluster

print("=== BMI ===") # identificado que é para a variável de IMC (BMI)
desc_bmi
```

Saída

```
=== BMI ===
[11]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	30.927292	6.024227	18.050	26.410	30.4000	35.21125	49.06
1	165.0	35.302879	4.315540	22.895	32.200	34.9600	37.07000	52.58
2	346.0	29.963382	6.057864	16.815	25.475	29.7175	33.60625	48.07
3	419.0	29.157482	5.896811	15.960	25.175	28.7850	33.00000	53.13

- **Cluster 1:** Este é, de longe, o cluster com o maior IMC médio (35.3). O mínimo já está na faixa de sobrepeso. Representa um grupo com um grau de obesidade mais significativo e menos variação em comparação com outros.

Caracterizando os Grupos Criados

- Para a variável de IMC.

Código

```
[11]: # Descritivas numéricas por cluster para a variável de IMC(BMI)
desc_bmi = df.groupby('cluster')['bmi'].describe() # agrupando o cluster segundo a variável de IMC(BMI) e computando a descritiva para cada cluster

print("=== BMI ===") # identificado que é para a variável de IMC (BMI)
desc_bmi
```

Saída

```
=== BMI ===
[11]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	30.927292	6.024227	18.050	26.410	30.4000	35.21125	49.06
1	165.0	35.302879	4.315540	22.895	32.200	34.9600	37.07000	52.58
2	346.0	29.963382	6.057864	16.815	25.475	29.7175	33.60625	48.07
3	419.0	29.157482	5.896811	15.960	25.175	28.7850	33.00000	53.13

- **Cluster 2:** A média está exatamente no limiar de Obesidade. Apresenta o menor mínimo (16.81), mas o máximo ainda é alto. A maioria do grupo (mediana 29.7) está no limiar entre sobrepeso e obesidade.

Caracterizando os Grupos Criados

- Para a variável de IMC.

Código

```
[11]: # Descritivas numéricas por cluster para a variável de IMC(BMI)
desc_bmi = df.groupby('cluster')['bmi'].describe() # agrupando o cluster segundo a variável de IMC(BMI) e computando a descritiva para cada cluster

print("=== BMI ===") # identificado que é para a variável de IMC (BMI)
desc_bmi
```

Saída

```
=== BMI ===
[11]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	30.927292	6.024227	18.050	26.410	30.4000	35.21125	49.06
1	165.0	35.302879	4.315540	22.895	32.200	34.9600	37.07000	52.58
2	346.0	29.963382	6.057864	16.815	25.475	29.7175	33.60625	48.07
3	419.0	29.157482	5.896811	15.960	25.175	28.7850	33.00000	53.13

- **Cluster 3:** Este cluster tem o menor IMC médio (29.2), Apresenta o menor mínimo geral e, curiosamente, o maior máximo geral (53.13), mostrando grande dispersão.

Caracterizando os Grupos Criados

- Para a variável de Charges.



Código

```
[12]: # Descritivas numéricas por cluster para a variável de Custos médicos (charges)
desc_charges = df.groupby('cluster')['charges'].describe()#agrupando o cluster segundo a variável de charges e computando a descritiva para cada cluster

print("=== Charges ===") # identificado que é para a variável de Custos (Charges)
desc_charges
```



Saída

```
=== Charges ===
[12]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	12592.953037	5643.456498	5397.61670	8794.562000	11349.3733	13568.095863	30259.99556
1	165.0	40308.320033	6778.084898	18963.17192	36149.483500	39774.2763	44423.803000	63770.42801
2	346.0	10709.606982	6366.134224	3443.06400	6200.039287	8600.3256	12568.223588	32787.45859
3	419.0	5397.382650	5478.202121	1121.87390	2135.891875	3227.1211	5242.995950	26125.67477

- **Cluster 0:** Custo médico intermediário-alto, sendo o segundo maior em número de membros (408). Com uma média de cobranças de aproximadamente 12.592. A distribuição dos custos varia de cerca de 5.397 a 30.259, mostrando uma variabilidade considerável, mas concentrando-se em uma faixa de custo bem definida e acima da média geral.

Caracterizando os Grupos Criados

- Para a variável de Charges.



Código

```
[12]: # Descritivas numéricas por cluster para a variável de Custos médicos (charges)
desc_charges = df.groupby('cluster')['charges'].describe()#agrupando o cluster segundo a variável de charges e computando a descritiva para cada cluster

print("=== Charges ===") # identificado que é para a variável de Custos (Charges)
desc_charges
```



Saída

```
=== Charges ===
[12]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	12592.953037	5643.456498	5397.61670	8794.562000	11349.3733	13568.095863	30259.99556
1	165.0	40308.320033	6778.084898	18963.17192	36149.483500	39774.2763	44423.803000	63770.42801
2	346.0	10709.606982	6366.134224	3443.06400	6200.039287	8600.3256	12568.223588	32787.45859
3	419.0	5397.382650	5478.202121	1121.87390	2135.891875	3227.1211	5242.995950	26125.67477

- **Cluster 1:** A média de cobranças é de aproximadamente 40.308, um valor drasticamente superior aos outros grupos. É importante notar que até o valor mínimo de cobrança neste grupo (18.963) é superior ao custo de 75% dos membros dos outros clusters, indicando que todos os indivíduos aqui representam um altíssimo custo.

Caracterizando os Grupos Criados

- Para a variável de Charges.



Código

```
[12]: # Descritivas numéricas por cluster para a variável de Custos médicos (charges)
desc_charges = df.groupby('cluster')['charges'].describe()#agrupando o cluster segundo a variável de charges e computando a descritiva para cada cluster

print("=== Charges ===") # identificado que é para a variável de Custos (Charges)
desc_charges
```



Saída

```
=== Charges ===
[12]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	12592.953037	5643.456498	5397.61670	8794.562000	11349.3733	13568.095863	30259.99556
1	165.0	40308.320033	6778.084898	18963.17192	36149.483500	39774.2763	44423.803000	63770.42801
2	346.0	10709.606982	6366.134224	3443.06400	6200.039287	8600.3256	12568.223588	32787.45859
3	419.0	5397.382650	5478.202121	1121.87390	2135.891875	3227.1211	5242.995950	26125.67477

- **Cluster 2:** A principal característica aqui é a alta variabilidade (desvio padrão de 6.366), o que significa que, embora a média seja moderada, existem tanto indivíduos com custos baixos quanto alguns com custos bem elevados dentro do mesmo grupo.

Caracterizando os Grupos Criados

- Para a variável de Charges.



Código

```
[12]: # Descritivas numéricas por cluster para a variável de Custos médicos (charges)
desc_charges = df.groupby('cluster')['charges'].describe()#agrupando o cluster segundo a variável de charges e computando a descritiva para cada cluster

print("=== Charges ===") # identificado que é para a variável de Custos (Charges)
desc_charges
```



Saída

```
=== Charges ===
[12]:
```

	count	mean	std	min	25%	50%	75%	max
cluster								
0	408.0	12592.953037	5643.456498	5397.61670	8794.562000	11349.3733	13568.095863	30259.99556
1	165.0	40308.320033	6778.084898	18963.17192	36149.483500	39774.2763	44423.803000	63770.42801
2	346.0	10709.606982	6366.134224	3443.06400	6200.039287	8600.3256	12568.223588	32787.45859
3	419.0	5397.382650	5478.202121	1121.87390	2135.891875	3227.1211	5242.995950	26125.67477

- **Cluster 3:** A média de custos é de apenas 5.397, e a mediana é ainda menor, em 3.227, o que indica que a grande maioria dos membros deste grupo possui despesas médicas muito baixas.

Conclusão

Cluster 0: "Idosos com Custo Moderado"

- **Perfil:** Este é o segundo maior grupo (408 membros) e é definido principalmente pela idade. É o cluster com a média de idade mais alta (52.5 anos). Seu IMC médio é de 30.9, indicando obesidade.
- **Consequência:** Este grupo possui custos médicos intermediários-altos, com uma média de R\$ 12.592. São mais caros que os grupos mais jovens, mas significativamente mais baratos que o grupo de alto IMC (Cluster 1).

Conclusão

Cluster 1: "Alto Risco, Alto Custo"

- **Perfil:** Este é o menor grupo (165 membros), mas o de maior impacto financeiro. É caracterizado por indivíduos com o IMC mais elevado (média de 35.3, indicando obesidade), apesar de não serem os mais velhos (idade média de 39.8 anos).
- **Consequência:** A combinação desses fatores, principalmente o IMC muito alto, resulta nos custos médicos mais altos de forma disparada, com uma média de R\$ 40.308. Este grupo representa o perfil de maior risco e maior custo para o sistema de saúde.

Conclusão

Cluster 2: "Adultos com Custo Controlado"

- **Perfil:** Este grupo (346 membros) tem uma idade média semelhante à do cluster de alto custo (39.9 anos), mas com um IMC mais baixo (média de 29.9, na faixa de sobrepeso/obesidade leve).
- **Consequência:** Eles representam um perfil de custo intermediário-baixo, com uma média de R\$ 10.709. A comparação direta com o Cluster 1 sugere que, para a mesma faixa etária, um IMC menor está fortemente associado a custos médicos consideravelmente mais baixos.

Conclusão

Cluster 3: "Jovens e Saudáveis"

- **Perfil:** Este é o maior grupo (419 membros) e representa o perfil de menor custo. É composto pelos indivíduos mais jovens (média de 25.3 anos), com o menor IMC médio entre os grupos (29.1, na faixa de sobrepeso).
- **Consequência:** Devido à juventude e ao IMC relativamente controlado, este cluster apresenta os custos médicos mais baixos, com uma média de apenas R\$ 5.397. É o grupo de menor risco e menor custo.

Considerações Finais

O que a análise nos trouxe:

- A análise de clusterização revelou quatro perfis de clientes distintos e acionáveis, permitindo a transição de uma visão genérica para uma estratégia de negócio personalizada.
- Assim, conseguimos mostrar que a análise de cluster é poderosa em capturar indivíduos em situações problemáticas de risco e cuidado.
- Mas quais são as implicações legais e morais de se utilizar modelos de Machine Learning? Na próxima aula nos desbruçaremos nos debates quanto ao Fairness em Machine Learning.

Disclaimer: propriedade intelectual

Este material foi criado pela professora Roberta Moreira Wichmann e é de sua propriedade intelectual.

É destinado exclusivamente ao uso dos alunos para fins educacionais no contexto das aulas.

Qualquer reprodução, distribuição ou utilização deste material, no todo ou em parte, sem a expressa autorização prévia da autora, é estritamente proibida.

O não cumprimento destas condições poderá resultar em medidas legais.

Referências Bibliográficas

- ABU-MOSTAFA, Yaser S.; MAGDON-ISMAIL, Malik; LIN, Hsuan-Tien. Learning from Data: A Short Course. Pasadena: California Institute of Technology (AMLBook), 2012.
- DEMÉTRIO, Clarice Garcia Borges; ZOCCHI, Sílvio Sandoval. Modelos de regressão. Piracicaba: Departamento de Ciências Exatas, ESALQ/USP, 2011. Disponível em: https://www.researchgate.net/publication/266233241_Modelos_de_Regressao. Acesso em: 23 set. 202
- GERON, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2. ed. Sebastopol, CA: O'Reilly Media, 2019.
- HARRIS, C. R. et al. Array programming with NumPy. Nature, v. 585, p. 357–362, 2020. DOI: 10.1038/s41586-020-2649-2.
- HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. The elements of statistical learning: data mining, inference, and prediction. 2. ed. New York: Springer, 2009.
- KAPOOR, Sayash; NARAYANAN, Arvind. Leakage and the reproducibility crisis in machine-learning-based science. Patterns, v. 4, n. 9, 2023.

Referências Bibliográficas

- IZBICKI, Rafael; DOS SANTOS, Tiago Mendonça. Aprendizado de máquina: uma abordagem estatística. Rafael Izbicki, 2020.
- MORETTIN, Pedro Alberto; SINGER, Júlio da Motta. Estatística e ciência de dados. 2. ed. Rio de Janeiro: LTC, 2022.
- PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, v. 12, p. 2825–2830, 2011.
- PONCE-BOBADILLA, Ana Victoria et al. Practical guide to SHAP analysis: Explaining supervised machine learning model predictions in drug development. Clinical and Translational Science, v. 17, n. 11, p. e70056, nov. 2024. DOI: 10.1111/cts.70056.
- PYTHON SOFTWARE FOUNDATION. Python Language Reference. Disponível em: <https://docs.python.org/3/reference/index.html>. Acesso em: 10 set. 2025.
- THE PANDAS DEVELOPMENT TEAM. pandas-dev/pandas: Pandas. Zenodo, 2024. Disponível em: <https://doi.org/10.5281/zenodo.10537285>. Acesso em: 10 set. 2025.

Referências Bibliográficas

- VON LUXBURG, Ulrike; SCHÖLKOPF, Bernhard. Statistical Learning Theory: Models, Concepts, and Results. In: GABBAY, D. M.; HARTMANN, S.; WOODS, J. H. (eds.). Handbook of the History of Logic, vol. 10: Inductive Logic. Amsterdam: Elsevier North Holland, 2011. p. 651–706. DOI: 10.1016/B978-0-444-52936-7.50016-1.

Introdução ao Aprendizado Não Supervisionado

Obrigada!

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br



INSTITUTO BRASILEIRO DE ENSINO,
DESENVOLVIMENTO E PESQUISA