



APOSTILA DO CURSO

# CIÊNCIA DE DADOS

## IMPRESSIONADOR

## MÓDULO 1 O QUE É CIÊNCIA DE DADOS?

---

1 - O que é Ciência de Dados?	32
2 - O que eu quero responder?	60
3 - Os Pilares da Ciência de Dados	65

## MÓDULO 2 INTRODUÇÃO A CIÊNCIA DE DADOS

---

1 - O que é ser um cientista?	85
2 - Um framework para Ciência de Dados	94
3 - Resumindo Ciência de Dados	107

## MÓDULO 3    PYTHON BÁSICO

---

1 - Explicando esse módulo	119
2 - Instalando o Python no Windows	120
3 - Problemas na Instalação - Resolvido	130
4 - Mac, Linux e Google Colab	139
5 - Criando seu Primeiro Programa	144
6 - Variáveis	152
7 - Tipos de Variáveis	156
8 - Estrutura do if – Condições no Python	160
9 - Elif	168
10 - Comparadores	174
11 - And e Or	181

## MÓDULO 3    PYTHON BÁSICO

---

12 - Listas em Python	190
13 - Índices em Lista, Consultando e Modificando Valores	192
14 - Estrutura de Repetição For	197
15 - For each - Percorrer cada item de uma lista	202
16 - For e if	204
17 - Estrutura While	209
18 - Loop infinito no While	212
19 - Tuplas	218
20 - Unpacking em Tuplas	223
21 - Dicionários em Python	228
22 - Pegar item Dicionário e Verificar Item Dicionário	233

## MÓDULO 3    PYTHON BÁSICO

---

23 - Range	238
24 - Functions no Python	244
25 - Retornar um valor na Function	246
26 - Argumentos e Parâmetros numa Function	249
27 - (Opcional) Aplicação em um Exemplo de argumento	250
28 - O que são módulos e qual a importância	252

## MÓDULO 4 PANDAS E NUMPY: AS BIBLIOTECAS BÁSICAS PARA CIÊNCIA DE DADOS

1 - As bibliotecas básicas para Ciência de Dados / Comparando Pandas e Excel	255
2 - Comparando Excel e Pandas	260
3 - Comparando Excel e Pandas na prática	261
4 - Introdução ao NumPy: A importância do NumPy	268
5 - Introdução ao NumPy: Propriedades de um array	272
6 - Introdução ao NumPy: Trabalhando com um array	276
7 - Introdução ao Pandas: Importando e visualizando uma base	284
8 - Introdução ao Pandas: DataFrame e Series	296
9 - Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas	305
10 - Introdução ao Pandas: informações estatísticas e filtros na base	315
11 - Introdução ao Pandas: criando gráficos	329

## MÓDULO 5 PROJETO 1 - ANALISANDO O ENGAJAMENTO DO INSTAGRAM

---

1 - Explicando o Projeto	332
2 - Importando e tratando a base com Pandas	334
3 - Tratando valores nulos da coluna Corrossel	340
4 - Analisando informações estatísticas e as 5 melhores / 5 piores publicações	346
5 - O group by (groupby) no pandas e a análise do engajamento	355
6 - Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)	371

## MÓDULO 6 INTRODUÇÃO A ESTATÍSTICA

---

1 - Introdução a Estatística e Estatística Descritiva	398
2 - Tabela de frequência e histograma	406
3 - Entendendo o conceito de média	414
4 - Mediana e sua relação com a média	423
5 - Usando Python para entender a relação entre média e mediana	428
6 - Média, mediana e moda	436
7 - Entendendo de forma prática a relação entre média, mediana e moda	444

## MÓDULO 7 MATPLOTLIB: CRIANDO GRÁFICOS EM PYTHON

1 - Apresentando o Matplotlib: Criando gráficos em Python	474
2 - Introdução ao Matplotlib	480
3 - Usando a documentação para criar nosso primeiro gráfico (gráfico de linha)	492
4 - (Opcional) Entendo a documentação do Matplotlib	495
5 - Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)	509
6 - Filtrando a base usando o contains (e fillna para tratar valores vazios)	521
7 - Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras	527
8 - Usando o annotate para adicionar rótulos de dados no gráfico	534
9 - Criando um scatter plot usando apenas a documentação	537
10 - (Opcional) Revisando o datetime e o astype	543
11 - (Opcional) Adicionando rótulo para cores de um scatter plot	547

## MÓDULO 8 BOAS PRÁTICAS PARA APRESENTAÇÃO DE DADOS

1 - Introdução aos conceitos básicos de apresentação de dados	559
2 - Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)	567
3 - Melhorando o seu visual (Proximidade e Similaridade)	581
4 - Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)	598
5 - Contraste e atributos pré-atentivos	616
6 - Visualização de dados no Python: Passo a passo para melhorar seus visuais no matplotlib	633
7 - Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras	638
8 - Visualização de dados no Python: Adicionando rótulo nos dados (annotate)	646
9 - Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado	655
10 - Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico	667
11 - Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha	678

## MÓDULO 8 BOAS PRÁTICAS PARA APRESENTAÇÃO DE DADOS

---

12 - Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

696

---

## MÓDULO 9 CRIANDO UMA APRESENTAÇÃO EXECUTIVA

1 - Apresentando o projeto	711
2 - Importando e analisando a base	712
3 - Tratando valores vazios	724
4 - Usando o datetime para tratar datas	731
5 - Criando um gráfico de barras no matplotlib	741
6 - Adicionando título no gráfico e ajustando o eixo x	747
7 - Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas	753
8 - Vendas por mês e transformando índices em colunas com o reset_index	770
9 - Entendendo o deslocamento das barras em um único gráfico de barras horizontais	777
10 - Adicionando todos os anos no gráfico de barras e colocando rótulo de dados	782
11 - Mudando os rótulos do eixo x e finalizando o visual de venda do mês	787

## MÓDULO 9 CRIANDO UMA APRESENTAÇÃO EXECUTIVA

---

12 - Respondendo qual foi a categoria mais vendida	791
13 - Criando um gráfico de barras horizontais para o top N itens	809
14 - Usando o merge para unir 2 bases no pandas	819
15 - Usando o merge para criar a relação de top N itens pelos anos	824
16 - Criando o gráfico de barras horizontais do top N itens pelos anos	829
17 - Concluindo o projeto e respondendo as informações do negócio	843
18 - Apresentando as informações em um PowerPoint	851
19 - Corrigindo o erro na transformação da data	856

## MÓDULO 10 INTRODUÇÃO AO APRENDIZADO DE MÁQUINAS

---

1 - O que é aprendizado de máquinas (Machine Learning)?	861
2 - Como funciona um modelo de aprendizado de máquinas?	864
3 - O aprendizado de máquinas no Python	871
4 - Regressão linear no Scikit-Learn	886
5 - Descrição estatística com Pandas	902

## MÓDULO 11 COMO AS MÁQUINAS APRENDEM?

---

1 - Tipos de aprendizado	923
2 - Tipos de aprendizado – Aprendizado supervisionado	924
3 - Tipos de aprendizado – Aprendizado não supervisionado	928
4 - Tipos de aprendizado – Aprendizado semi-supervisionado	936
5 - Tipos de aprendizado – Aprendizado por reforço	937
6 - Diferença entre aprender e decorar	940
7 - Considerações importantes	949

---

## MÓDULO 12 CRIANDO UM MODELO DE CLASSIFICAÇÃO

---

1 - Entendendo o dataset iris	955
2 - Utilizando o Pandas	958
3 - Visualizando os dados	961
4 - Criando uma reta capaz de separar os dados do modelo	964
5 - Criando uma função classificadora	967
6 - Perceptron	968

## MÓDULO 13 UTILIZANDO O APRENDIZADO DE MÁQUINAS

1 - Matriz de confusão	975
2 - Acurácia de um modelo de classificação	984
3 - Precisão de um modelo de classificação	986
4 - Recall de um modelo de classificação	988
5 - Conclusão de métricas de um modelo de classificação	991
6 - Separando os dados em treino e teste	992
7 - Avaliando nosso modelo	995
8 - O que é uma árvore de decisão?	999
9 - Reutilizando o dataset íris	1009
10 - Criando uma reta	1011
11 - Criando uma árvore de decisão	1012

## MÓDULO 13 UTILIZANDO O APRENDIZADO DE MÁQUINAS

---

12 - Comparando os modelos

1013

## MÓDULO 14 ANÁLISE EXPLORATÓRIA DE DADOS

---

1 - Entendendo o dataset Titanic	1016
2 - Analisando as informações da base	1018
3 - Criando visualizações	1024
4 - Correlação entre colunas	1038
5 - Tratando valores ausentes e outliers	1047
6 - Pandas Profiling	1052
7 - Finalizando a análise e apresentando os resultados	1061

## MÓDULO 15 O SCIKIT-LEARN

---

1 - Conhecendo o Scikit-Learn	1063
2 - A base de dados Íris	1065
3 - Entendendo o Perceptron	1066
4 - Entendendo árvore de decisão	1071
5 - Comparando os modelos com mais uma classe	1073
6 - Entendendo o parâmetro average	1080
7 - Separando os dados em treino e teste e avaliando os modelos	1085
8 - Adicionando um novo modelo – Regressão logística	1089
9 - Entendendo regressão linear	1092
10 - Regressão linear no mercado de ações	1095

---

## MÓDULO 16 CRIANDO UM ALGORITMO DE REGRESSÃO

1 - Conhecendo a base de casas da Califórnia	1107
2 - Visualizando os dados	1111
3 - Verificando valores duplicados e outliers	1118
4 - Separando a base em treino e teste e usando regressão linear simples	1124
5 - Entendendo o coeficiente de determinação	1127
6 - Métricas de erro para regressão	1130
7 - Regressão linear múltipla	1139
8 - Comparando modelos	1142

## MÓDULO 17 CONCEITOS BÁSICOS DE SQL PARA CIÊNCIA DE DADOS

---

1 - O que é SQL?	1146
2 - Usando SQL com o Pandas	1147
3 - Selecionando colunas específicas com SELECT	1150
4 - Criando uma função para nossos estudos de SQL com Pandas	1152
5 - Lidando com valores repetidos com DISTINCT	1155
6 - Filtros com WHERE e os operadores OR, AND, e NOT	1158
7 - Funções de agregação	1166
8 - Ordenando registros com ORDER BY	1170
9 - Limitando a base com LIMIT	1171
10 - Filtros com HAVING	1172
11 - Condicionais com CASE	1175

## MÓDULO 17 CONCEITOS BÁSICOS DE SQL PARA CIÊNCIA DE DADOS

---

12 - Subquery	1180
13 - Outros filtros - IN	1182
14 - Outros filtros - LIKE	1183
15 - Unindo bases com JOIN	1186
16 - O UNION e o FULL JOIN no SQL	1195

# SUMÁRIO

## MÓDULO 18 TÉCNICAS DE STORYTELLING COM DADOS: UTILIZANDO O SQL COM DADOS REAIS DE VENDA

1 - Apresentando a base de dados que vamos utilizar nesse módulo	1204
2 - Buscando os arquivos de nossa base	1206
3 - Passando a base para o Pandas	1207
4 - Analisando ordens, itens e pagamentos	1212
5 - Pivot	1218
6 - Analisando pagamentos, vendedores e avaliações	1222
7 - Criando um banco de dados com sqlite3	1229
8 - Atualizando e deletando registros	1238
9 - Criando um banco de dados para nosso estudo	1240
10 - Fazendo uma análise	1246
11 - Cenário de estudos	1248

# SUMÁRIO

## MÓDULO 18 TÉCNICAS DE STORYTELLING COM DADOS: UTILIZANDO O SQL COM DADOS REAIS DE VENDA

---

12 - Storytelling com dados	1249
13 - Preparando nosso ambiente	1250
14 - Validando o problema	1251
15 - Relação entre atraso e nota de avaliação	1260
16 - Avaliando os comentários dos pedidos atrasados	1272

## MÓDULO 19 CRIANDO UM MODELO DE IDENTIFICAÇÃO DE FRAUDE

1 - Conhecendo nossa base de dados	1279
2 - Verificando o desbalanceamento da base	1285
3 - Estratégias para tratar o desbalanceamento	1288
4 - Visualizando as diferentes estratégias	1292
5 - Aplicando o RandomUnderSampler em nossa base de dados	1303
6 - Aplicando o RandomOverSampler em nossa base de dados	1307
7 - Aplicando o ClusterCentroids e NearMiss em nossa base de dados	1308
8 - Aplicando outras estratégias de oversampling em nossa base de dados	1309
9 - Combinando over e undersampling	1310
10 - Curva ROC	1311
11 - Curva PRC	1320

## MÓDULO 19 CRIANDO UM MODELO DE IDENTIFICAÇÃO DE FRAUDE

---

12 - Comparando diferentes modelos em nossa base de dados	1321
13 - Normalizando dados	1325
14 - Testando diferentes hiperparâmetros	1327
15 - Seleção de hiperparâmetros com GridSearchCV	1330
16 - Comparando modelos e avaliando novos hiperparâmetros	1336

## MÓDULO 20 SUBINDO SEU MODELO PARA PRODUÇÃO (DEPLOY)

---

1 - O que faremos neste modulo?	1342
2 - Criando um modelo de regressão linear	1343
3 - Persistindo o modelo com joblib	1346
4 - Utilizando o modelo criado em dados de produção	1347
5 - Modelo em produção com Jupyter Notebook	1350
6 - Modelo em produção com arquivo Python	1351
7 - Modelo em produção com arquivo executável	1352
8 - Modelo em produção com Streamlit	1355

## MÓDULO 21 AJUSTANDO OS DADOS PARA O MODELO (DATA CLEANING)

1 - Explicando a importância da limpeza dos dados e importando a base	1363
2 - Buscando na base por valores nulos e linhas duplicadas	1365
3 - Procurando na base alguns problemas que podem ter sido gerados por erros humanos	1371
4 - Tratando valores vazios e linhas duplicadas	1377
5 - Tratando valores digitados errados (erros humanos)	1380
6 - Limpeza de Dados - Exercício	1384
7 - Entendendo a base e respondendo as perguntas sem fazer o tratamento dos dados	1386
8 - Contando a quantidade de alunos que responderam o questionário	1389
9 - Eliminando valores duplicados e discutindo sobre o tratamento do ID_aluno	1395
10 - Somando a matrícula dos alunos que responderam (visualizando e tratando outliers)	1402
11 - Verificando o tamanho da blusa para todos os alunos	1407

## MÓDULO 21 AJUSTANDO OS DADOS PARA O MODELO (DATA CLEANING)

12 - Descobrindo quantos alunos vão participar da formatura	1413
13 - Estimando a altura de um aluno usando média e mediana dos dados	1417
14 - Aprofundando no tratamento de dados: Entendendo a base de notas de português	1426
15 - Usando o drop_duplicates para retirar valores duplicados da base	1429
16 - Analisando o describe e o boxplot e tratando outliers nos dados	1432
17 - Criando a função para transformar as notas dadas em conceitos (textos) em números de 1 a 10	1435
18 - Otimizando a função criada, unindo duas bases e calculando a média final dos alunos	1440
19 - Apresentando a base de cadastro dos alunos e tratando e-mails escritos errados	1446
20 - Tratando a data e ajustando as colunas de texto no cadastro dos alunos	1451
21 - Exercício: limpeza dos dados no dataset do titanic	1460
22 - Tratando as informações de embarque vazias e usando a mediana para as idades	1464

## MÓDULO 21 AJUSTANDO OS DADOS PARA O MODELO (DATA CLEANING)

---

23 - Analisando a média das idades pela classe, gênero e pelo título extraído do nome	1468
24 - Usando o transform para substituir as idades vazias pelo resultado do groupby e eliminando colunas desnecessárias	1474
25 - Analisando outliers, cardinalidade e eliminando colunas desnecessárias	1479

## MÓDULO 1

# O que é Ciência de Dados?



## Módulo 1 – O que é Ciência de Dados?

Ciência de dados é o **processo** de **exploração, manipulação e análise** dos dados para a **descoberta e previsão** através da criação **de hipóteses, testes e validação** com o objetivo de **responder perguntas do negócio e/ou fazer recomendações** capazes de serem diferenciais de negócio.

Todo esse processo precisa ter um forte **embasamento estatístico e matemático** e ser **diretamente ligado ao negócio**, além de poder ser feito de forma **escalável e replicável**.

# Módulo 1 – O que é Ciência de Dados?

Objetivos da Ciência de Dados:

- Usar dados históricos para fazer previsões sobre o futuro e tentar ser cada vez mais assertivo com a estratégia futura da empresa;
- Descobrir diferenciais para o negócio;
- Encontrar oportunidades ocultas;
- Buscar padrões que não estavam perceptíveis ao olho humano;
- Responder perguntas do negócio e/ou fazer recomendações capazes de serem diferenciais para o negócio.

# Módulo 1 – O que é Ciência de Dados?

Todo esse processo precisa ter um forte embasamento estatístico e matemático e ser diretamente ligado ao negócio, além de poder ser feito de forma escalável e replicável.

Na ciência de dados nós temos 3 pilares principais:



## Módulo 1 – O que é Ciência de Dados?

Toda essa parte de teste, validação, de fazer as suas próprias perguntas e validar se aquelas perguntas realmente estão acontecendo, isso faz parte do **processo científico**.

Existe um método científico! Observação, hipóteses, testes e validação, análises, monitoramento.



## Módulo 1 – O que é Ciência de Dados?

Quando se fala em **dados**, logicamente quer dizer que é necessário a existência deles e que estejam armazenados ou que pelo menos comecem a ser armazenados a partir do início do projeto.

Ciência de  
Dados

Precisamos que existam dados armazenados (ou pelo menos começar esse armazenamento).  
Armazenamento, processamento, visualização.

# Módulo 1 – O que é Ciência de Dados?

Então para entregar um bom projeto de *Data Science* é necessário todo um conjunto de **tecnologia** e de **conhecimento**.

Existe um método científico! Observação, hipóteses, testes e validação, análises, monitoramento.



Precisamos que existam dados armazenados (ou pelo menos começar esse armazenamento). Armazenamento, processamento, visualização.

# Módulo 1 – O que é Ciência de Dados?

Você com certeza já ouviu falar que o dado é o nosso ativo mais valioso ou que é o **novo petróleo**.

Isso porque quando as empresas sabem usar o dado e usam de forma correta, isso vira um diferencial para o negócio.



# Módulo 1 – O que é Ciência de Dados?

Atualmente é necessário encantar e fidelizar o cliente, porque você não precisa mais ir em uma loja, existem sites que vendem o mesmo produto.

Por isso a importância de **conhecer o cliente**, saber o que, como e quando oferecer. Às vezes o cliente nem queria comprar um produto e decide comprar porque em um momento estratégico você colocou uma propaganda para ele.



# Módulo 1 – O que é Ciência de Dados?

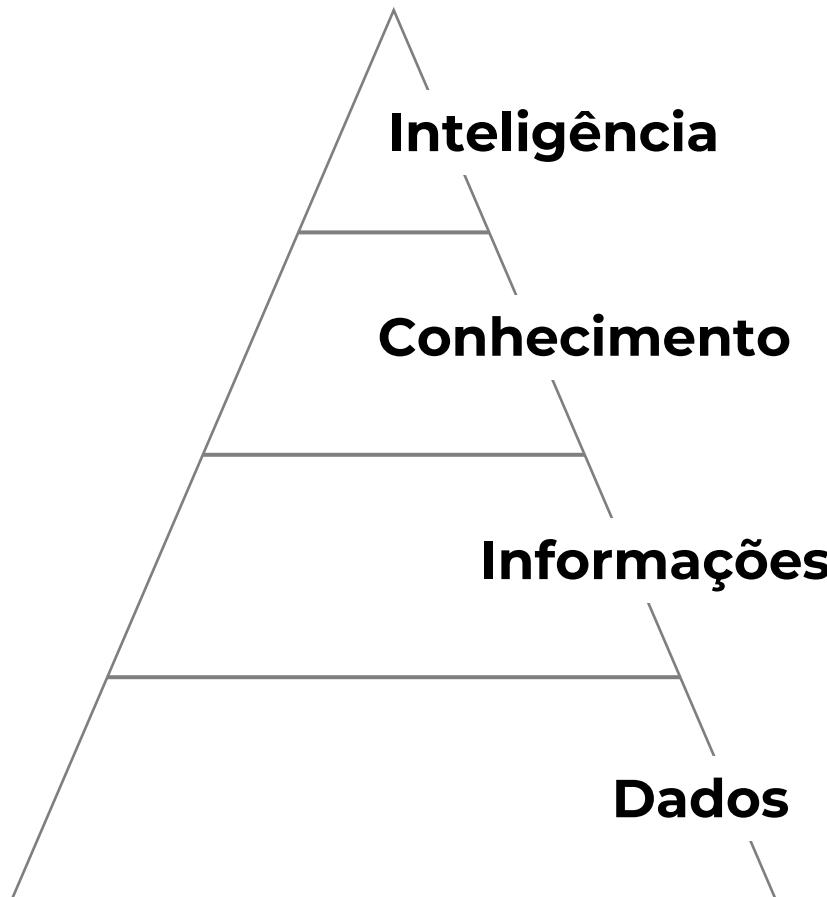
Um exemplo do dia a dia é quando você passa do lado de uma academia e o Instagram mostra uma promoção desse local.

Aí surge a pergunta “**como eles sabem disso?**”.

A resposta é simples, isso é um dado e é assim que ele está sendo usado hoje em dia pelas empresas. Se a sua empresa ainda não usa, ela pode ficar para trás.



# Módulo 1 – O que é Ciência de Dados?

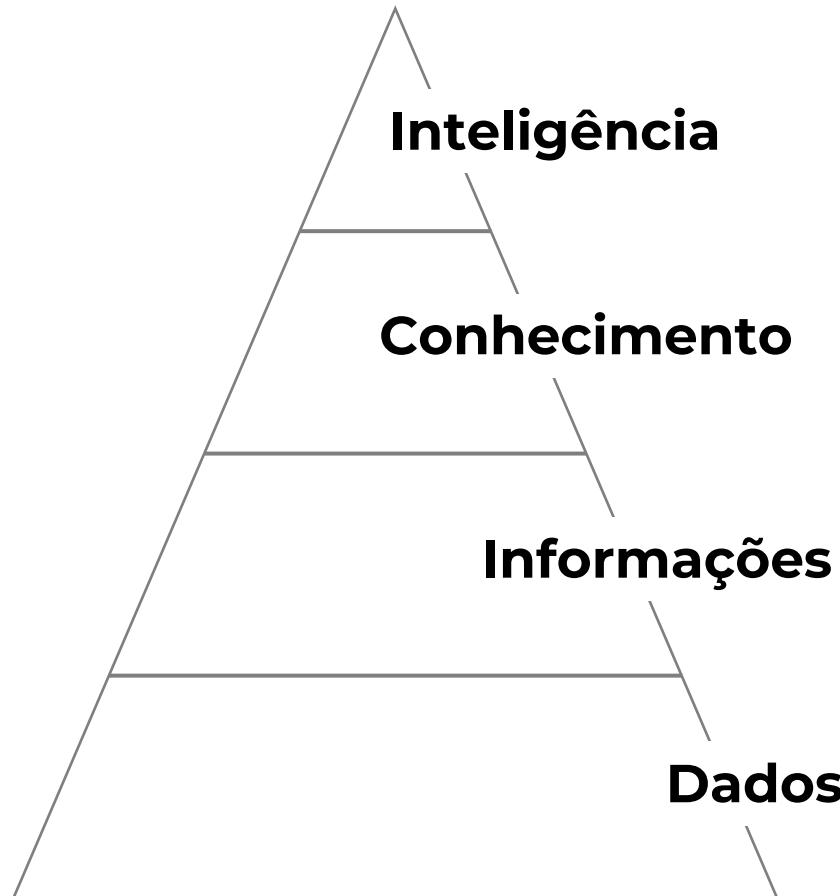


A pirâmide ao lado mostra a **relação entre dados e inteligência**, porque dado qualquer empresa pode ter, às vezes está em uma planilha no Excel ou em papel.

Porém, a maneira como se usa, organiza e transforma esse dado, isso sim é o diferencial.

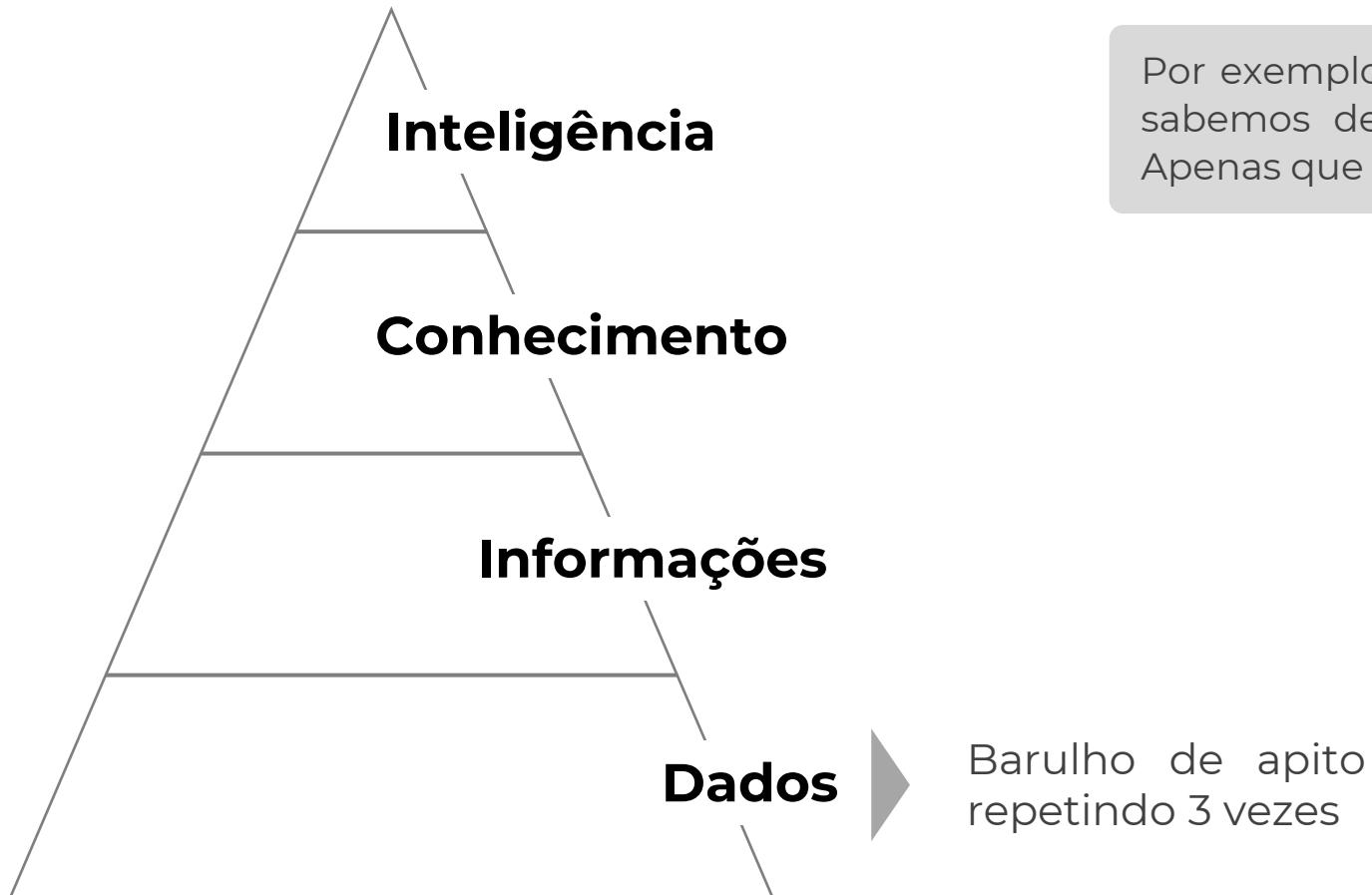
A intenção máxima é **transformar os dados em inteligência** e para isso é necessário ter estatística e um bom conhecimento do negócio, mas isso só é possível se for feito junto com a tecnologia.

# Módulo 1 – O que é Ciência de Dados?



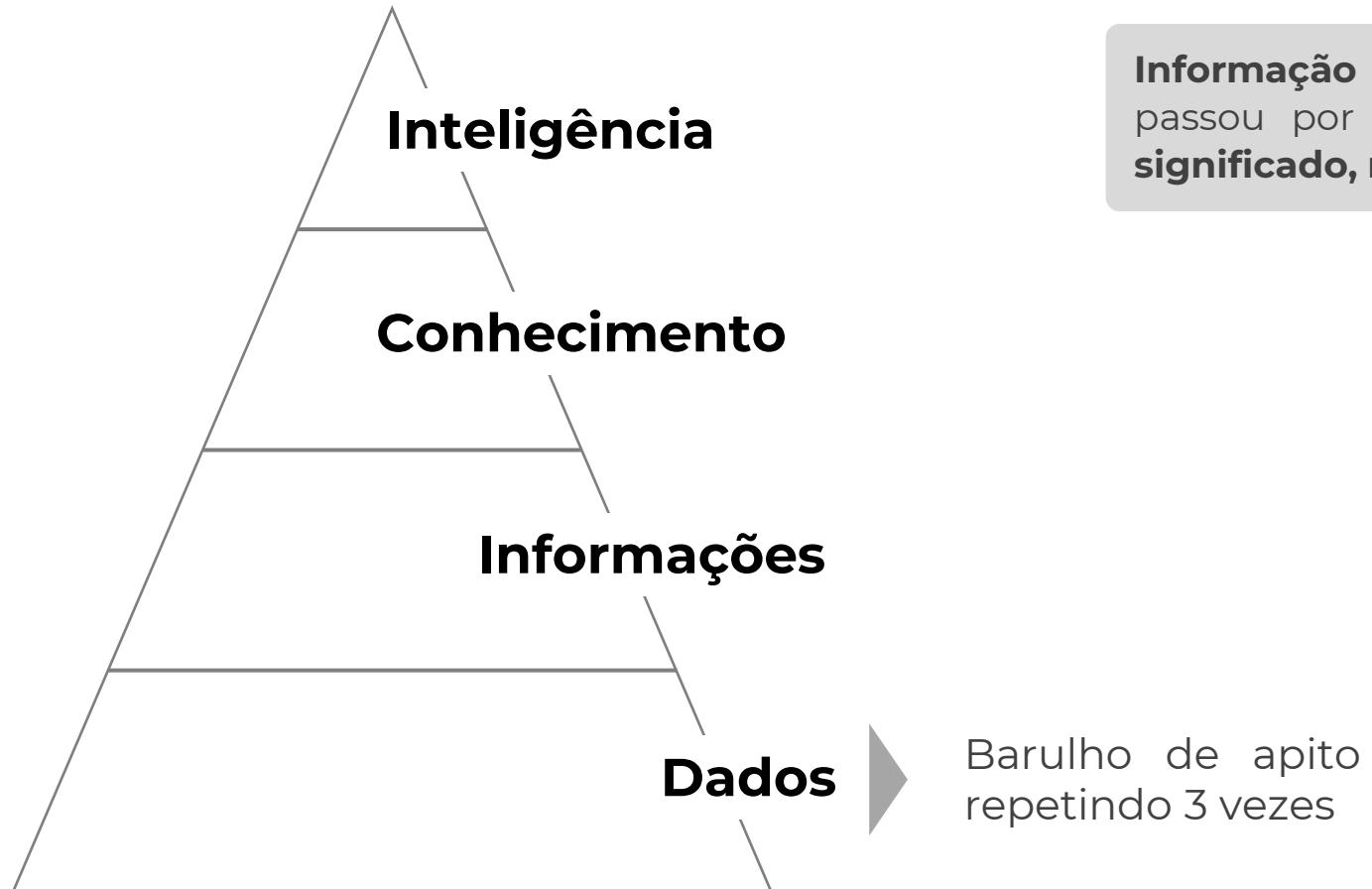
Dado é a **informação bruta**, é o registro de que algo aconteceu.

# Módulo 1 – O que é Ciência de Dados?



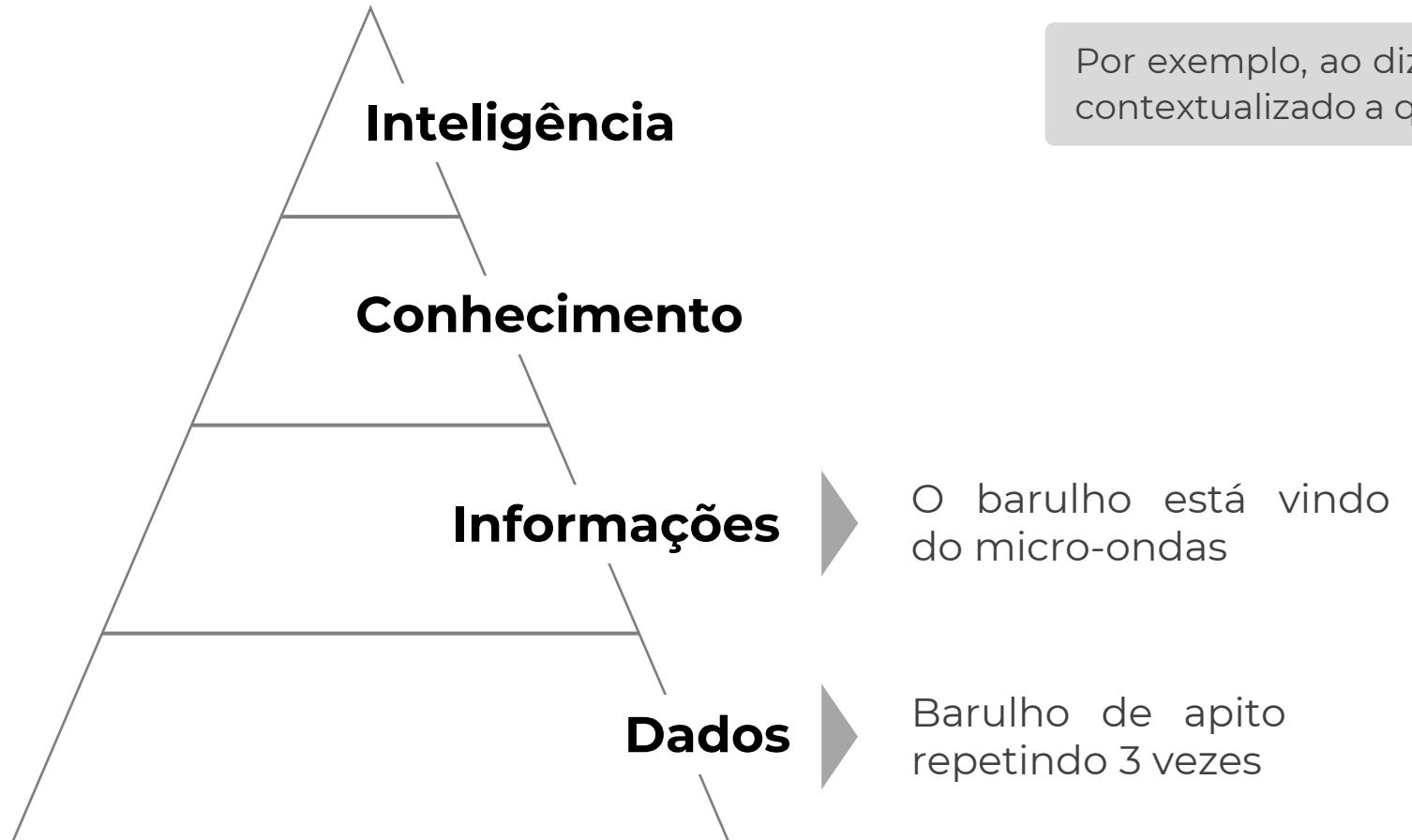
Por exemplo, quando se escuta um barulho apitando 3 vezes, não sabemos de onde está vindo o barulho, nem a que se refere. Apenas que se observou o barulho apitando 3 vezes.

# Módulo 1 – O que é Ciência de Dados?



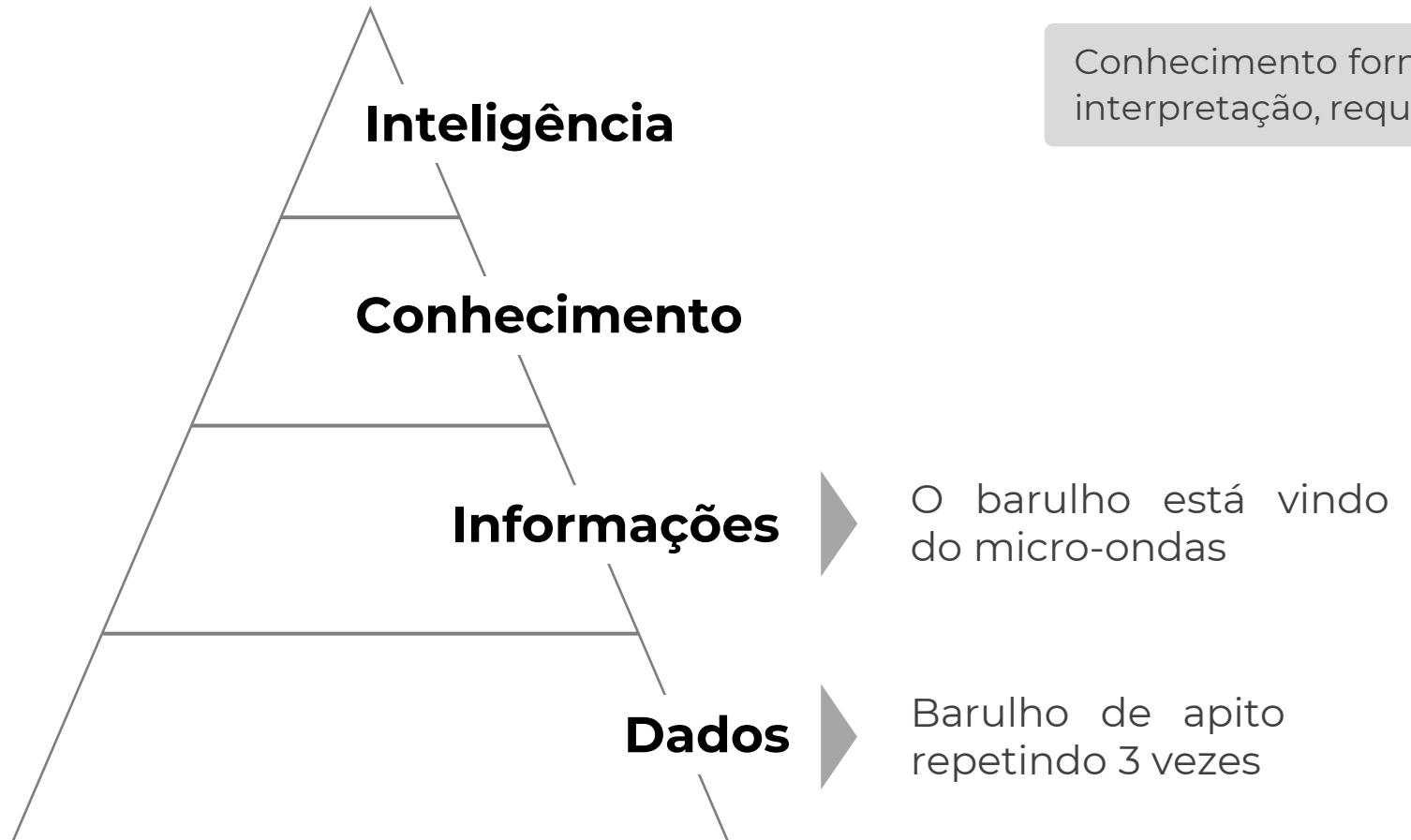
**Informação** é a contextualização do dado, é o dado depois que passou por algum processamento, ou seja, começa a ter um **significado, relevância e propósito**.

# Módulo 1 – O que é Ciência de Dados?

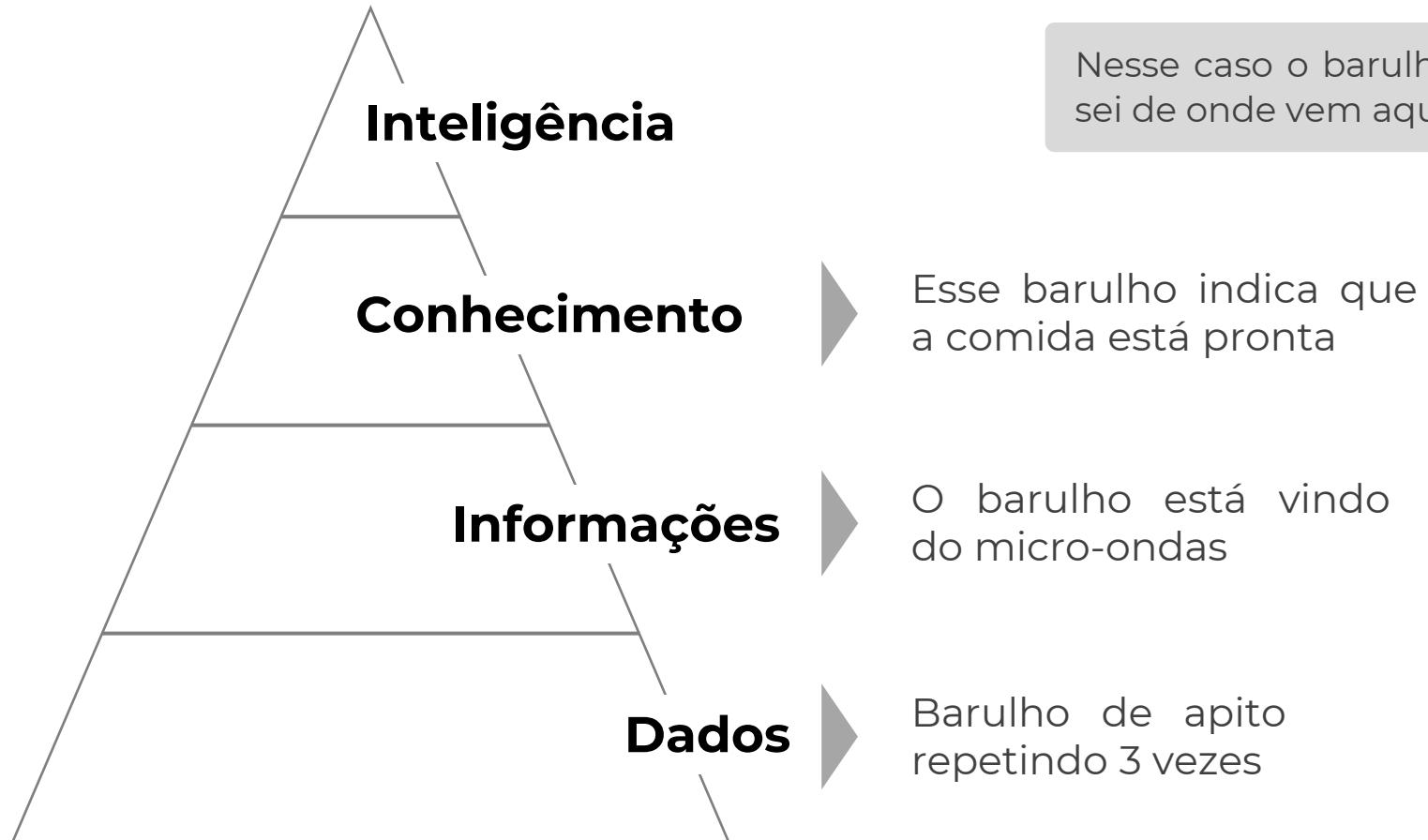


Por exemplo, ao dizer que o barulho está vindo do micro-ondas, foi contextualizado a que se refere o dado.

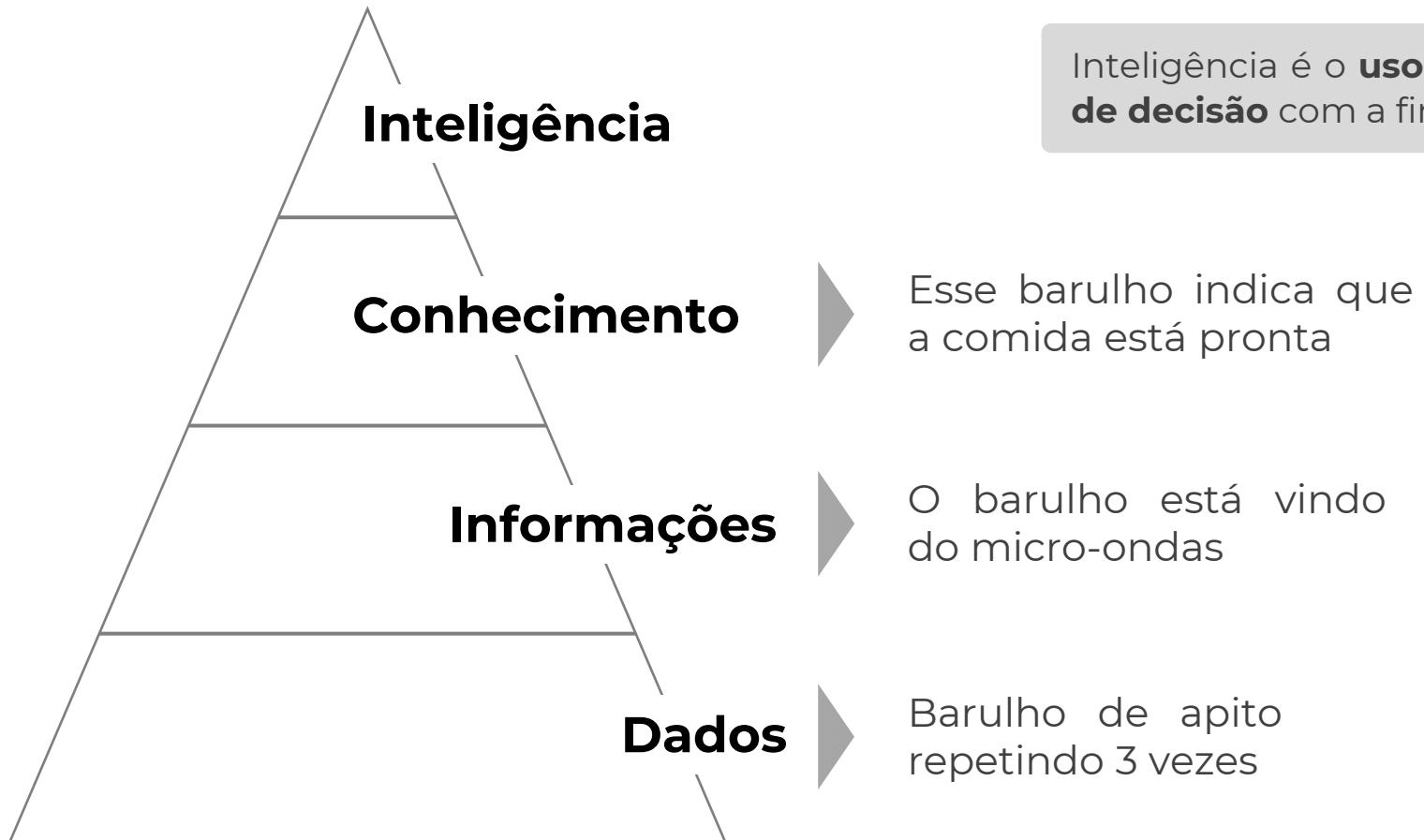
# Módulo 1 – O que é Ciência de Dados?



# Módulo 1 – O que é Ciência de Dados?

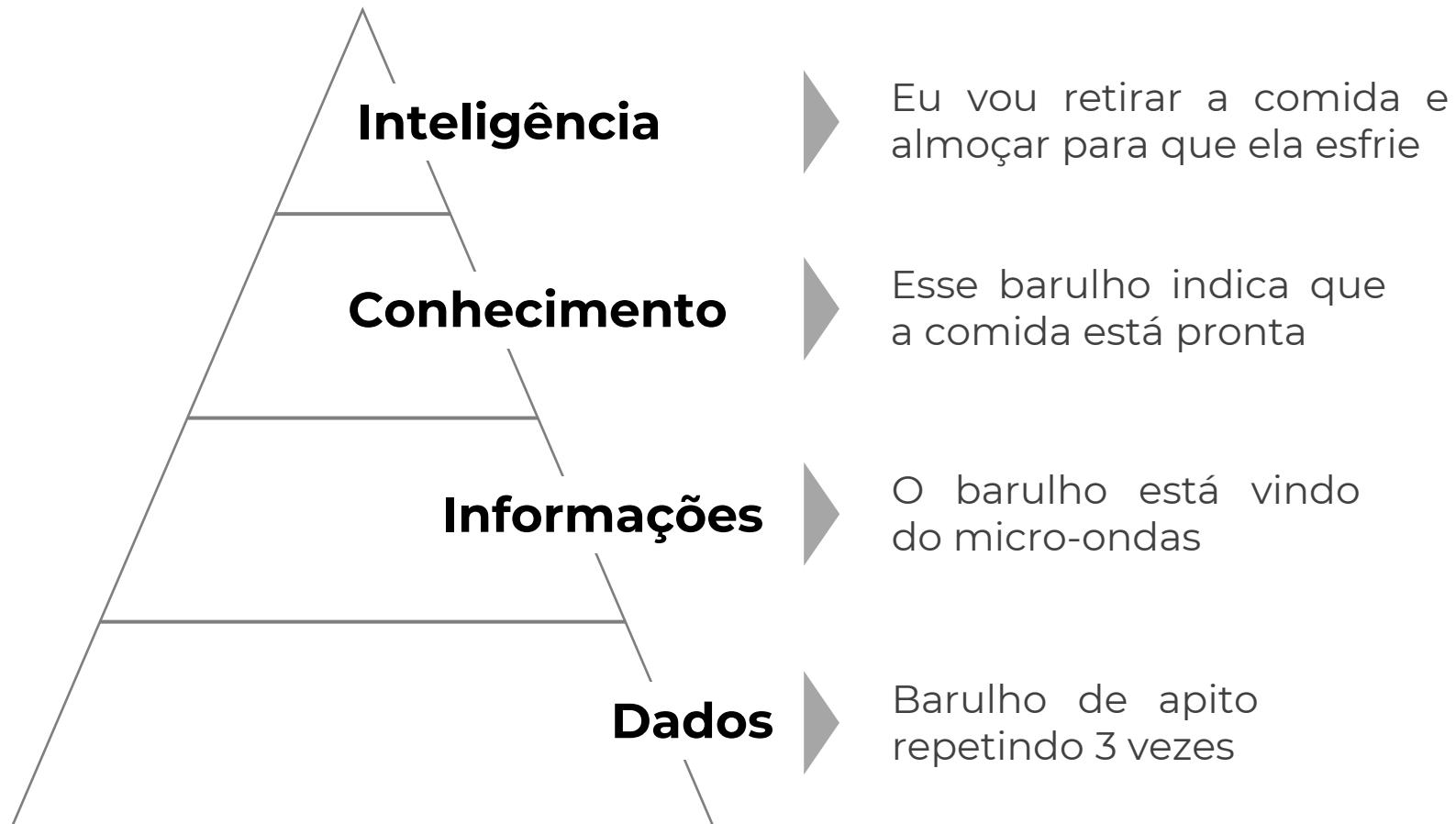


# Módulo 1 – O que é Ciência de Dados?



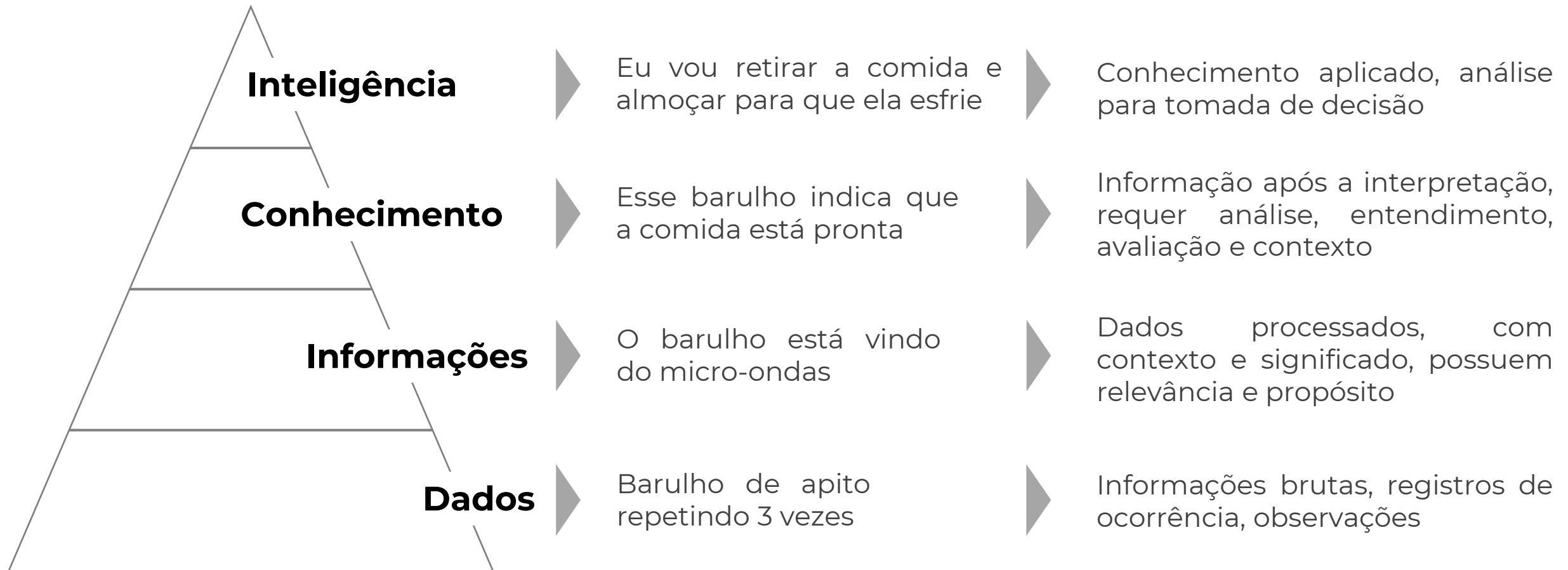
Inteligência é o **uso do conhecimento** aplicado para uma **tomada de decisão** com a finalidade de trazer uma vantagem.

# Módulo 1 – O que é Ciência de Dados?



Nesse exemplo, a tomada de decisão é retirar a comida para que ela não esfrie.

# Módulo 1 – O que é Ciência de Dados?

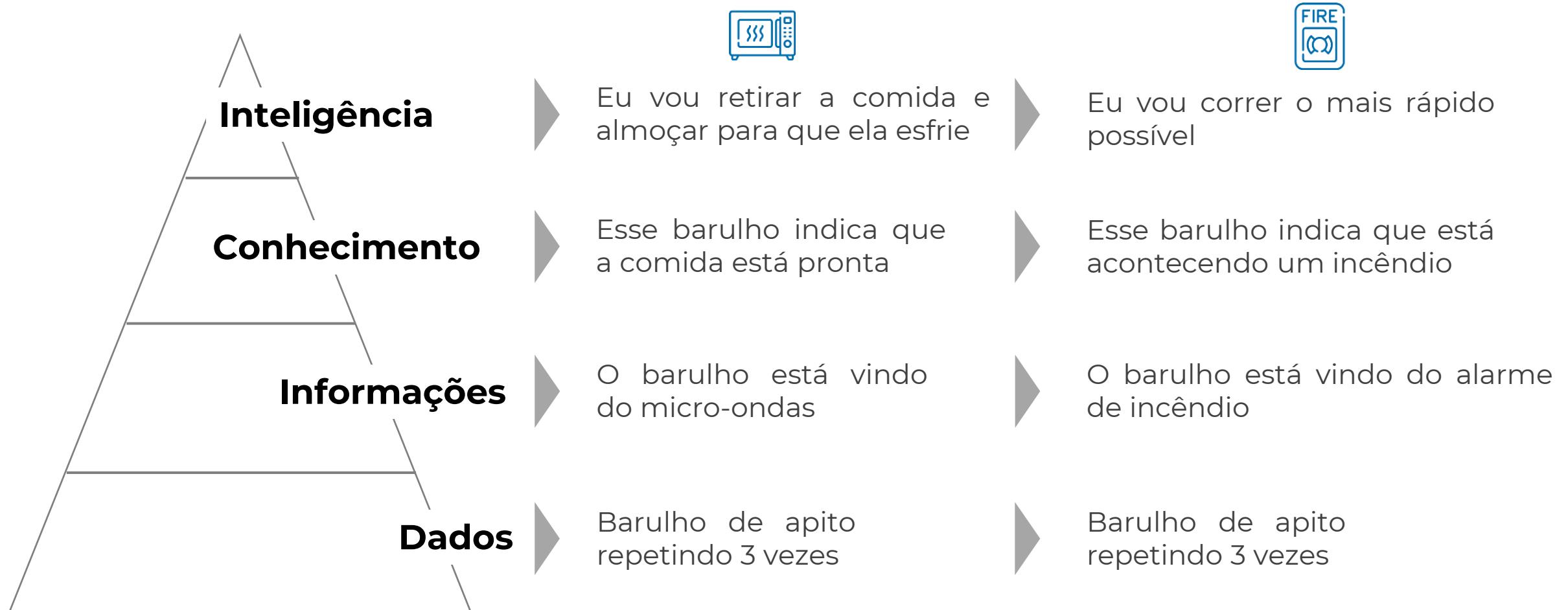


## Módulo 1 – O que é Ciência de Dados?

Um mesmo dado pode ter contextos diferentes que vão levar a uma tomada de decisão diferente.

Vamos comparar a seguir a tomada de decisão para um mesmo dado, utilizando como exemplo o alarme de incêndio e o micro-ondas.

# Módulo 1 – O que é Ciência de Dados?



## Módulo 1 – O que é Ciência de Dados?

Então, para todo dado é necessário **avaliar esse conhecimento** para depois aplicar uma **tomada de decisão**.

No micro-ondas o conhecimento é que a minha comida está pronta e a decisão é a retirada da comida para almoçar antes que ela esfrie.

Enquanto no alarme de incêndio, o meu conhecimento é identificar que está acontecendo um incêndio e a minha inteligência será a decisão de correr o mais rápido possível, porque nesse caso é algo muito mais urgente do que a comida esfriar.

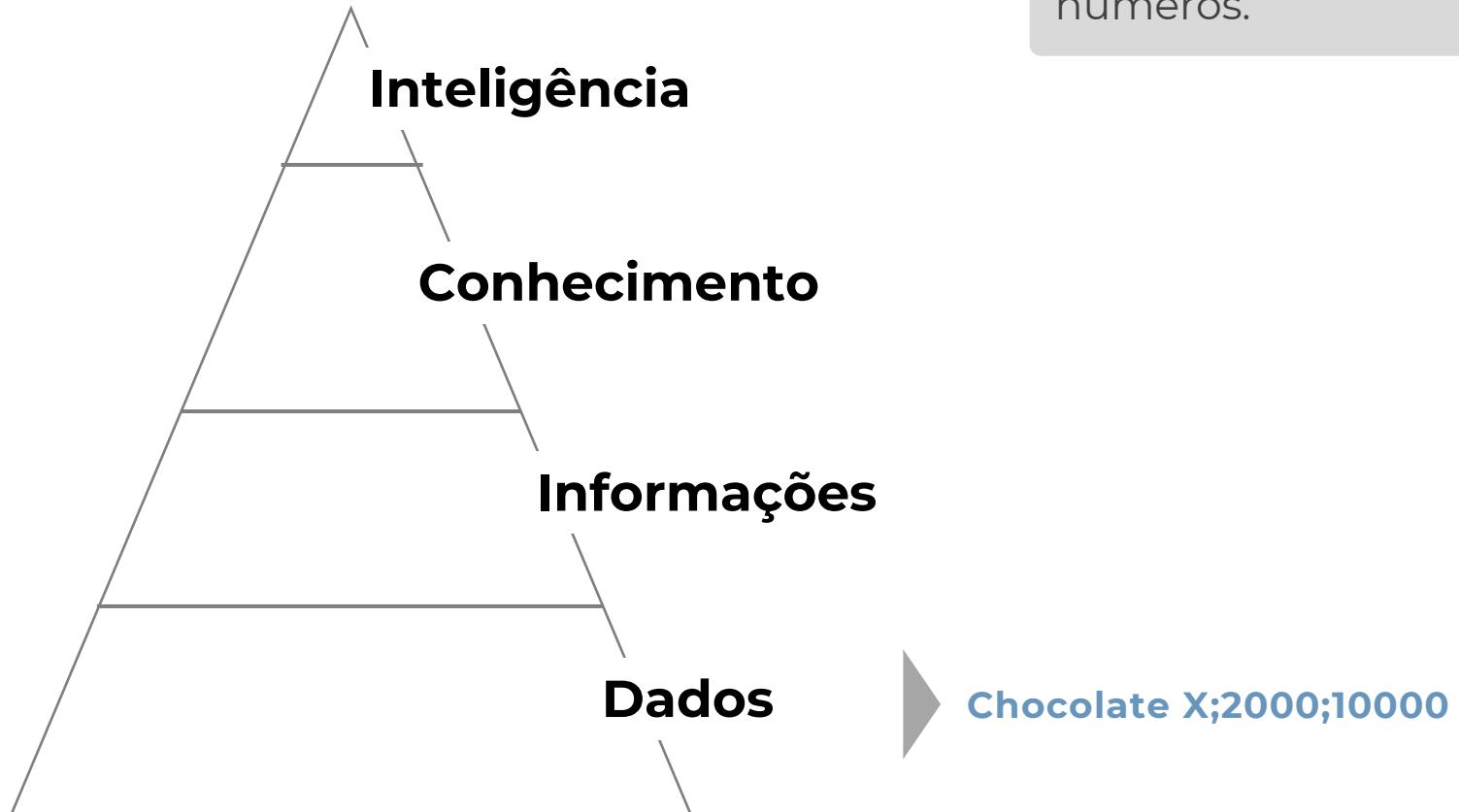
## Módulo 1 – O que é Ciência de Dados?

Analogamente, imagine um produto que não vende muito. Se em determinado mês a venda dele for baixa, não é necessário ficar preocupado.

Porém, se a venda do meu produto principal for baixa, eu preciso entender para agir o mais rápido possível.

# Módulo 1 – O que é Ciência de Dados?

Agora um exemplo prático:

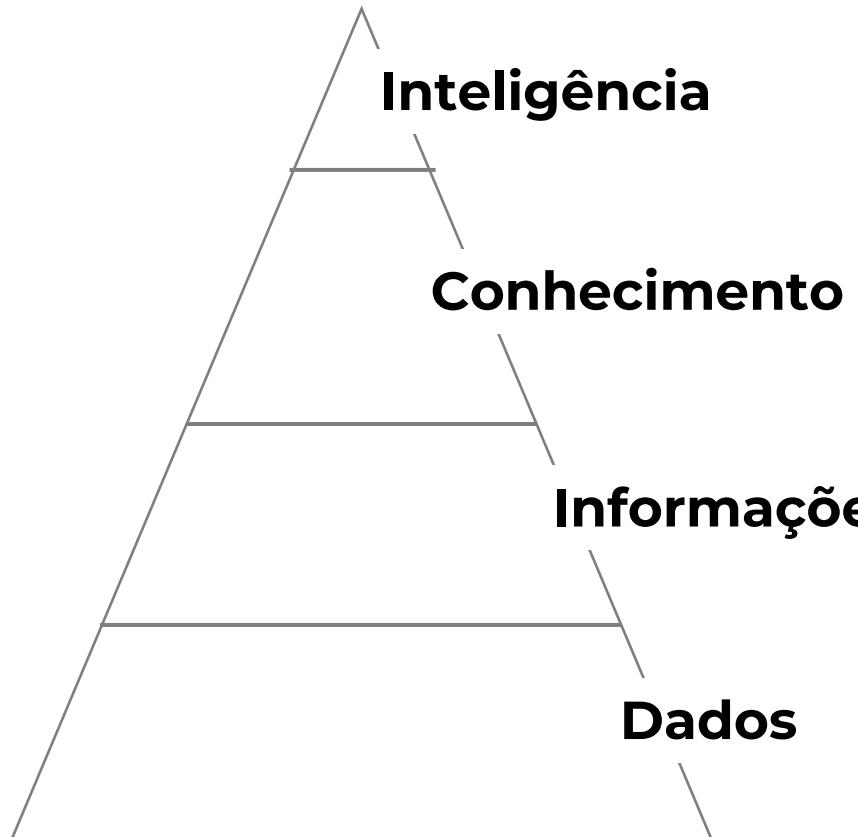


Apenas com o dado não sabemos o significado desses números.

Informações brutas, registros de ocorrência, observações

# Módulo 1 – O que é Ciência de Dados?

Agora um exemplo prático:



Agora com a informação foi colocado um rótulo do que cada coluna está dizendo. Porém ainda não podemos concluir se esse estoque é bom.

Produto: Chocolate X,  
Estoque: 2000, Venda  
Mensal: 10000

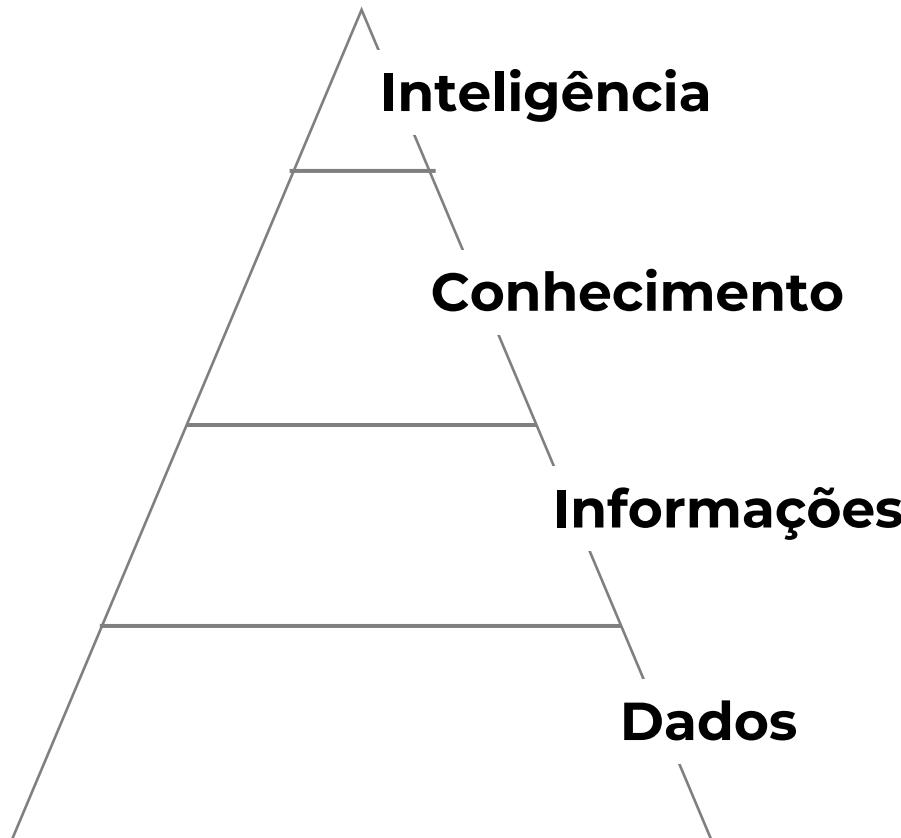
**Chocolate X;2000;10000**

→ Informações brutas, registros de ocorrência, observações

→ Informações brutas, registros de ocorrência, observações

# Módulo 1 – O que é Ciência de Dados?

Agora um exemplo prático:



No conhecimento vamos fazer a interpretação, ou seja, fazer a comparação do estoque com a venda mensal.

Existe o risco de falta de estoque (ruptura) desse chocolate na loja

Produto: Chocolate X,  
Estoque: 2000, Venda  
Mensal: 10000

Chocolate X;2000;10000

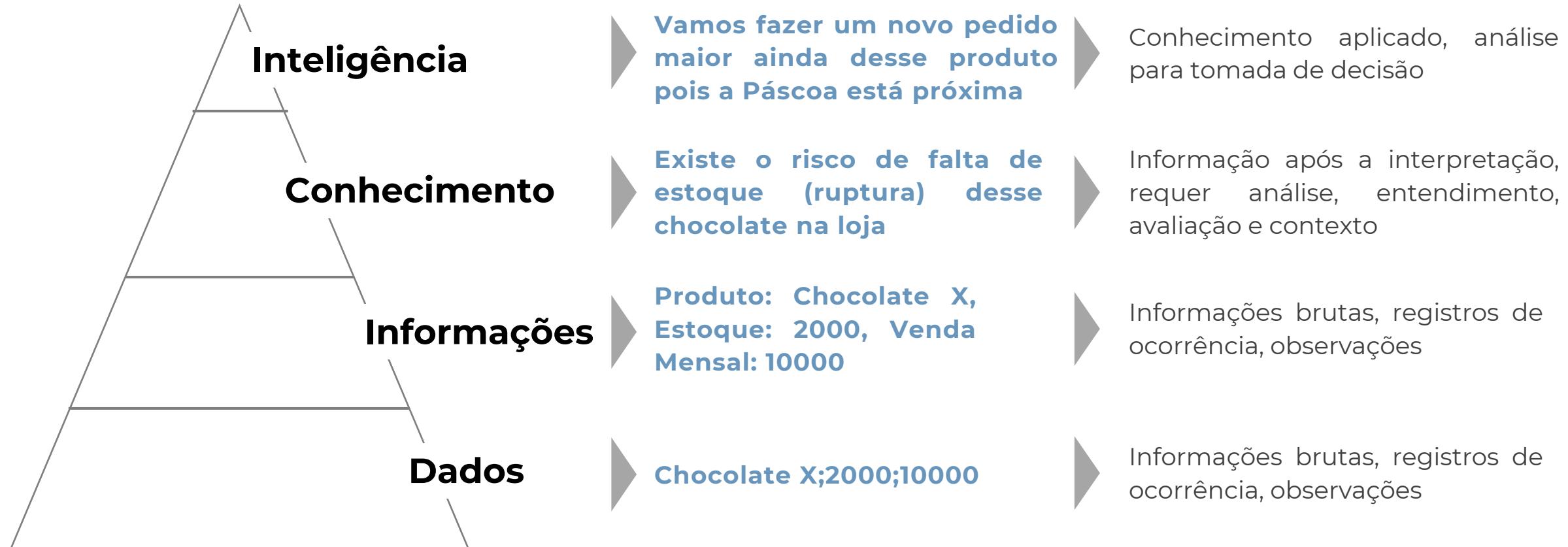
Informação após a interpretação, requer análise, entendimento, avaliação e contexto

Informações brutas, registros de ocorrência, observações

Informações brutas, registros de ocorrência, observações

# Módulo 1 – O que é Ciência de Dados?

Agora um exemplo prático:



## Módulo 1 – O que é Ciência de Dados?

Então uma informação que antes não tinha sentido como “Chocolate X;2000;10000” nos proporcionou o *insight* de enviar mais chocolate à loja para que não falte pois a páscoa está próxima.

Esse *insight* é o que nós vamos fazer em escalas maiores, como por exemplo, colocar um produto do lado de outro porque a probabilidade é maior desses produtos venderem juntos ou oferecer um produto para um cliente porque a chance dele comprar é alta.

## Módulo 1 – O que eu quero responder?

São necessárias muitas perguntas e observações para fazer as nossas hipóteses, mas existe uma pergunta fundamental que garante o nosso sucesso na Ciência de Dados:



## Módulo 1 – O que eu quero responder?

Na maioria das vezes em projeto de dados não trabalhamos sozinho, trabalhamos para um cliente ou para um gestor. Sempre vai ter alguém querendo saber alguma coisa.

Por que a venda foi ruim?

Por que eu não consegui vender o meu produto?

Em geral, essas perguntas são muito abertas e não nos dizem o que eles realmente querem saber. É por isso que vamos precisar investigar e procurar um pouco mais.

Por exemplo: a venda geral dos produtos foi ruim ou apenas a venda de um produto em específico?

# Módulo 1 – O que eu quero responder?



Se a conclusão apresentada para o chefe foi de que a venda foi ruim em todas as empresas, mas ele queria saber se aquele produto vendeu nas outras empresas, a resposta que ele precisava não foi dita.

A pergunta estava muito clara para ele, mas não respondemos. Por isso, vamos trabalhar no nosso framework de data Science que é: como fazer as perguntas, como mapear o que exatamente a gente precisa descobrir, para depois começar a entrar nos dados.

## Módulo 1 – O que eu quero responder?

Um exemplo para responder a pergunta “O que eu quero responder” é o livro “O guia do mochileiro das galáxias”.

No livro querem encontrar a resposta para a pergunta “Qual a resposta da vida, do universo e tudo mais?” e a resposta dada pela máquina construída é 42.

As pessoas falaram que estava errado e a máquina respondeu que não era a resposta que estava errada, e sim que eles não souberam fazer a pergunta.



## Módulo 1 – O que eu quero responder?

Ou seja, se você não sabe o que perguntar qualquer resposta que lhe for entregue vai servir para você e se não servir você deve reformular a pergunta.

Então, garantam que vocês vão saber o que responder para fazer um projeto de data Science. Pensem muito bem no que vocês querem responder antes de começar.

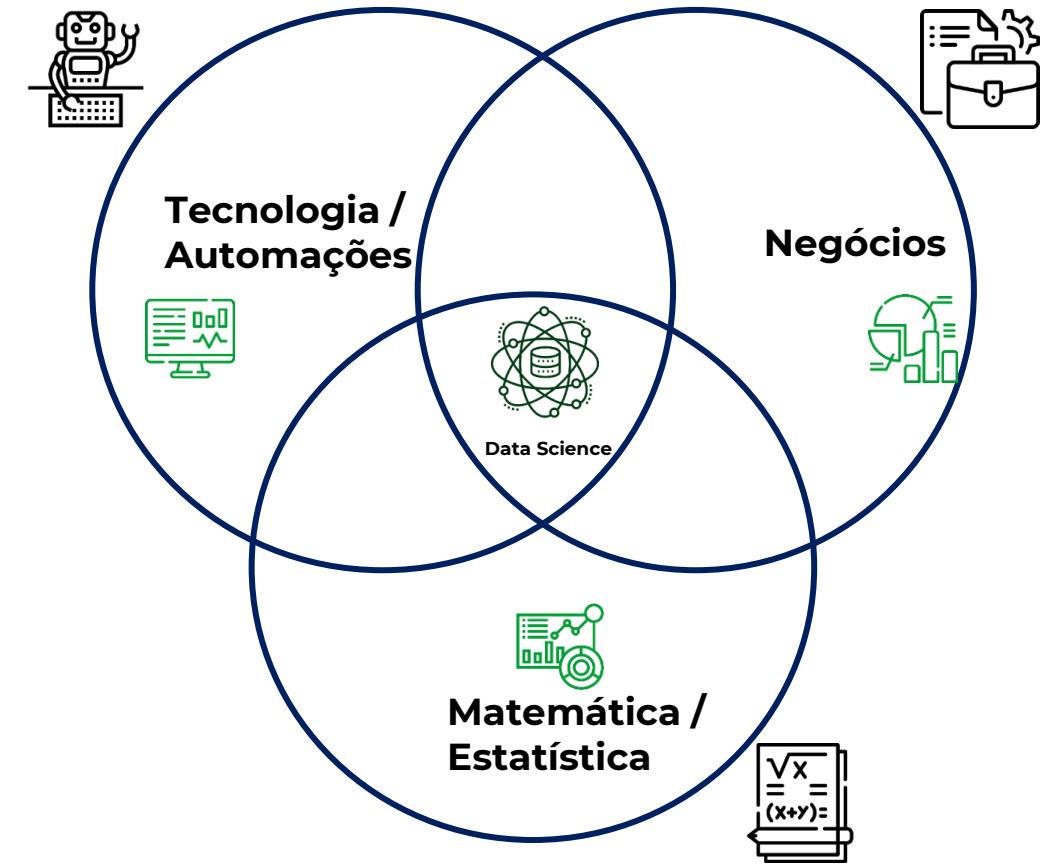


# Módulo 1 – Os Pilares da Ciência de Dados

A ciência de dados tem **3 pilares**: tecnologia, negócio e matemática.

O cientista de dados consegue relacionar essas 3 áreas, mas para isso é necessário ter uma base estatística, tecnológica (Python) e de negócios.

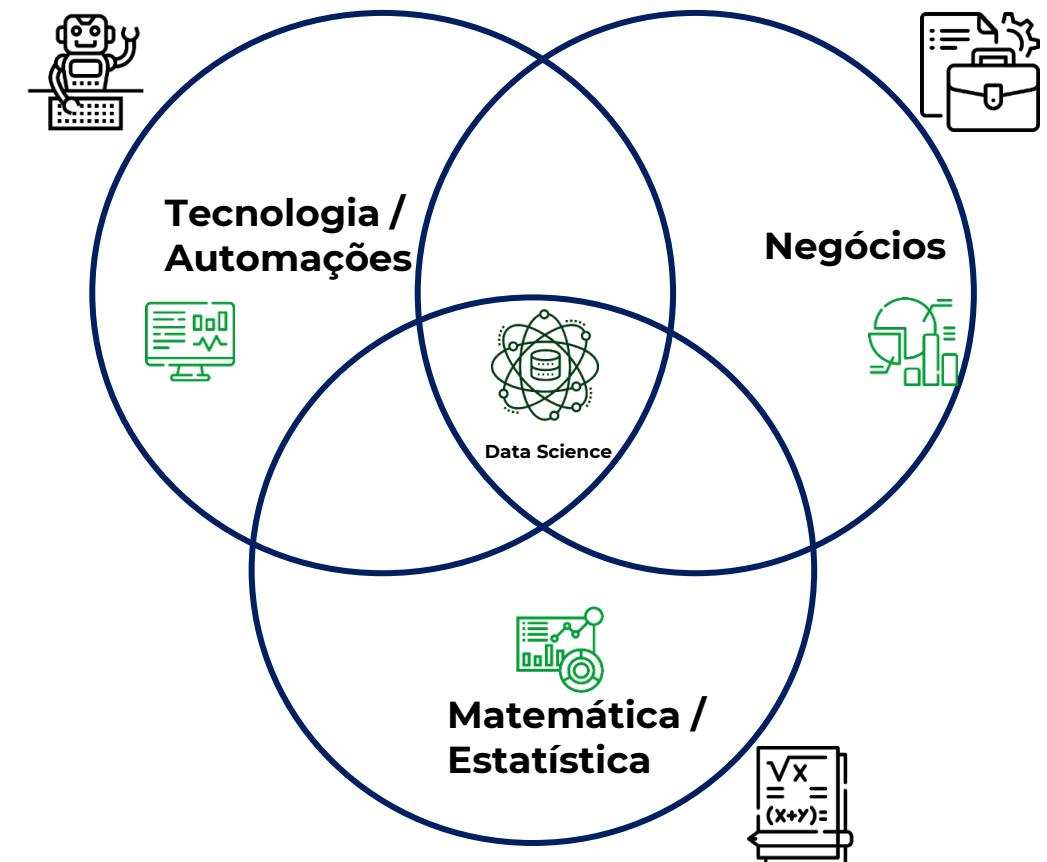
Diferente de um projeto de automação o data Science lida muito com o cliente e com expectativas que não estão claras para ele.



# Módulo 1 – Os Pilares da Ciência de Dados

Esse é um projeto muito mais complexo que lida diretamente com o cliente ou chefe.

Às vezes podemos achar um resultado certo, mas que pode não ter validade estatística ou não fazer sentido dentro do dia a dia do negócio.

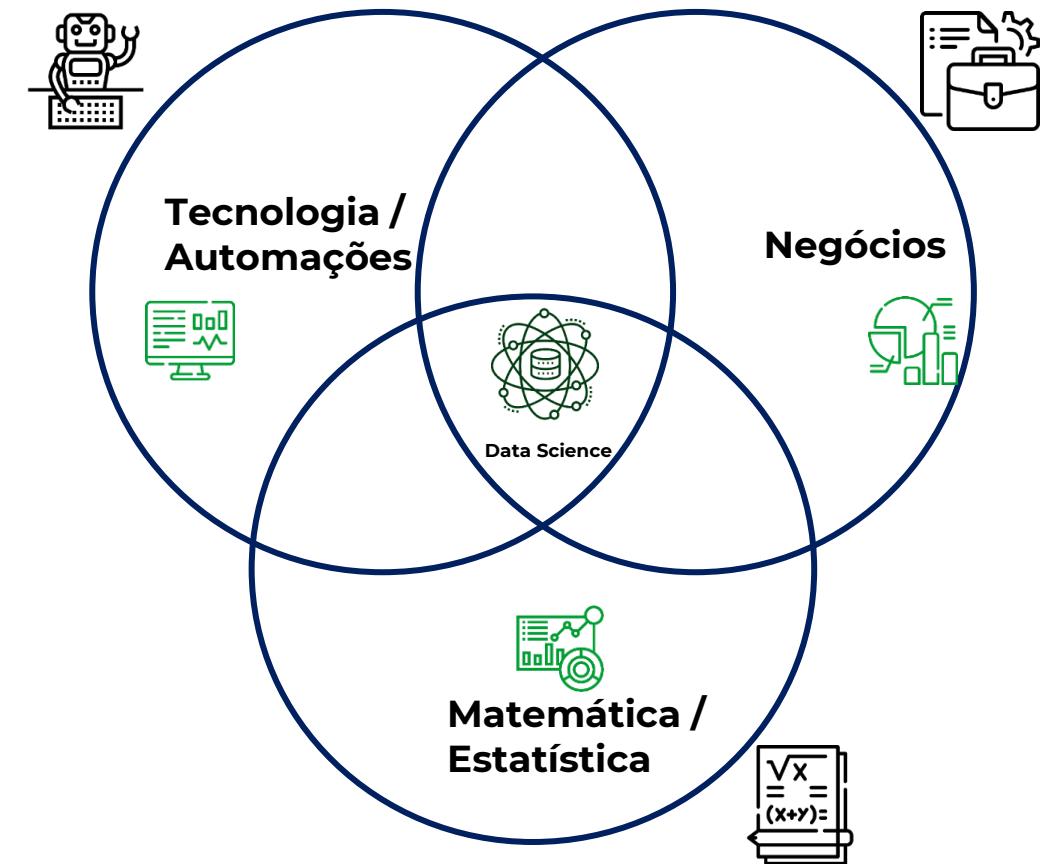


# Módulo 1 – Os Pilares da Ciência de Dados

O cientista de dados é um profissional multidisciplinar, que consegue navegar e entender o que as áreas precisam.

Ele possui uma forte base estatística, para verificar se faz sentido o que os modelos estão gerando.

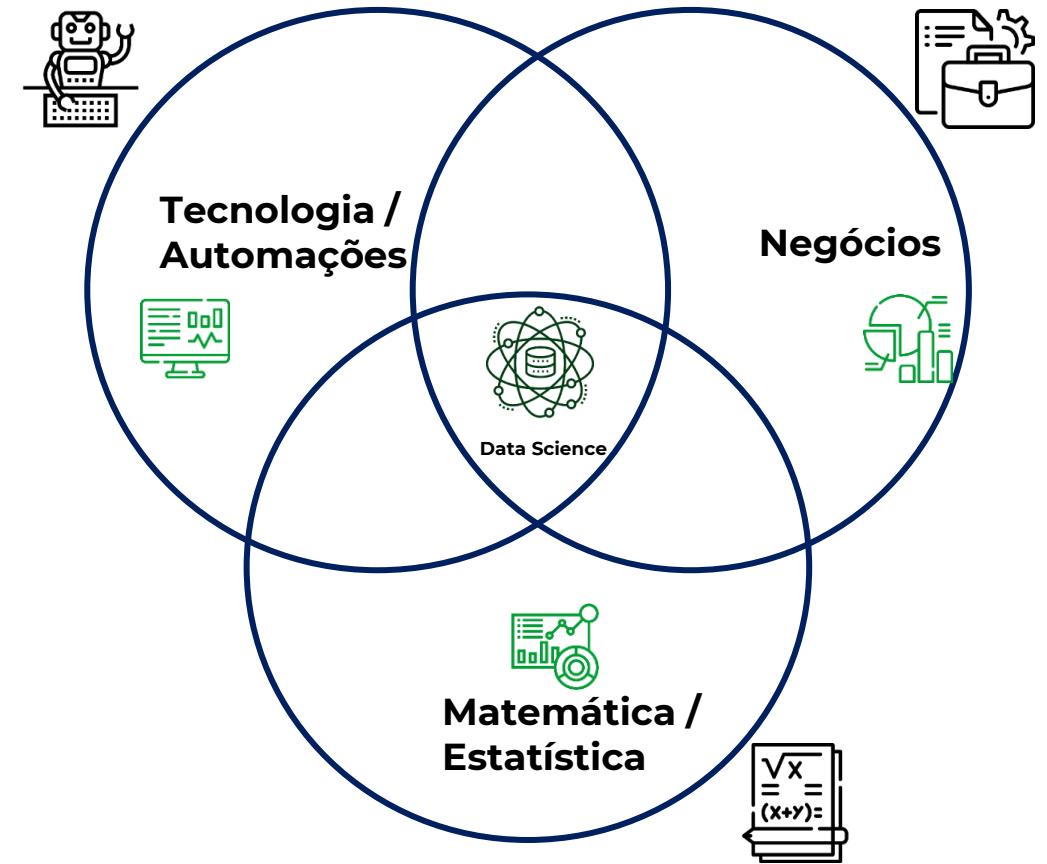
Além de conhecer muito sobre tecnologia, seja para fazer o upload dos dados ou tratar os que já estão disponíveis.



# Módulo 1 – Os Pilares da Ciência de Dados

Você pode estar se perguntando: “ok, mas para que serve um cientista de dados se já existem softwares que fazem isso e as empresas já usam a estatística tradicional?”

A ciência de dados é a junção da tecnologia, negócio e matemática, para calcular e analisar um resultado com base nesses 3 pilares.

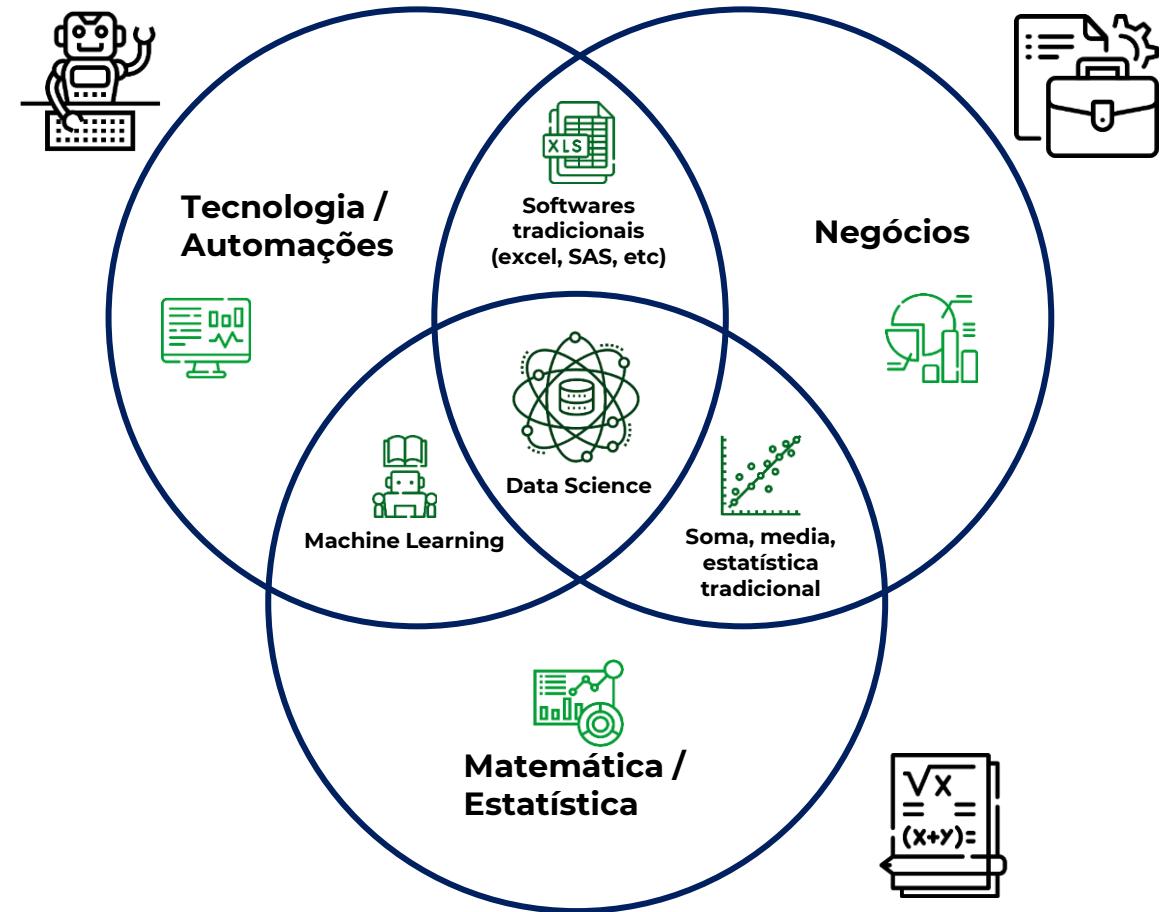


# Módulo 1 – Os Pilares da Ciência de Dados

Quando se fala de tecnologia e negócio existem diversas ferramentas e softwares de apoio, como o Excel.

Na matemática e tecnologia temos o modelo *Machine Learning*. Principalmente quando se fala em matemática, estatística e negócio estamos falando de soma, média e estatística tradicional que as empresas já fazem.

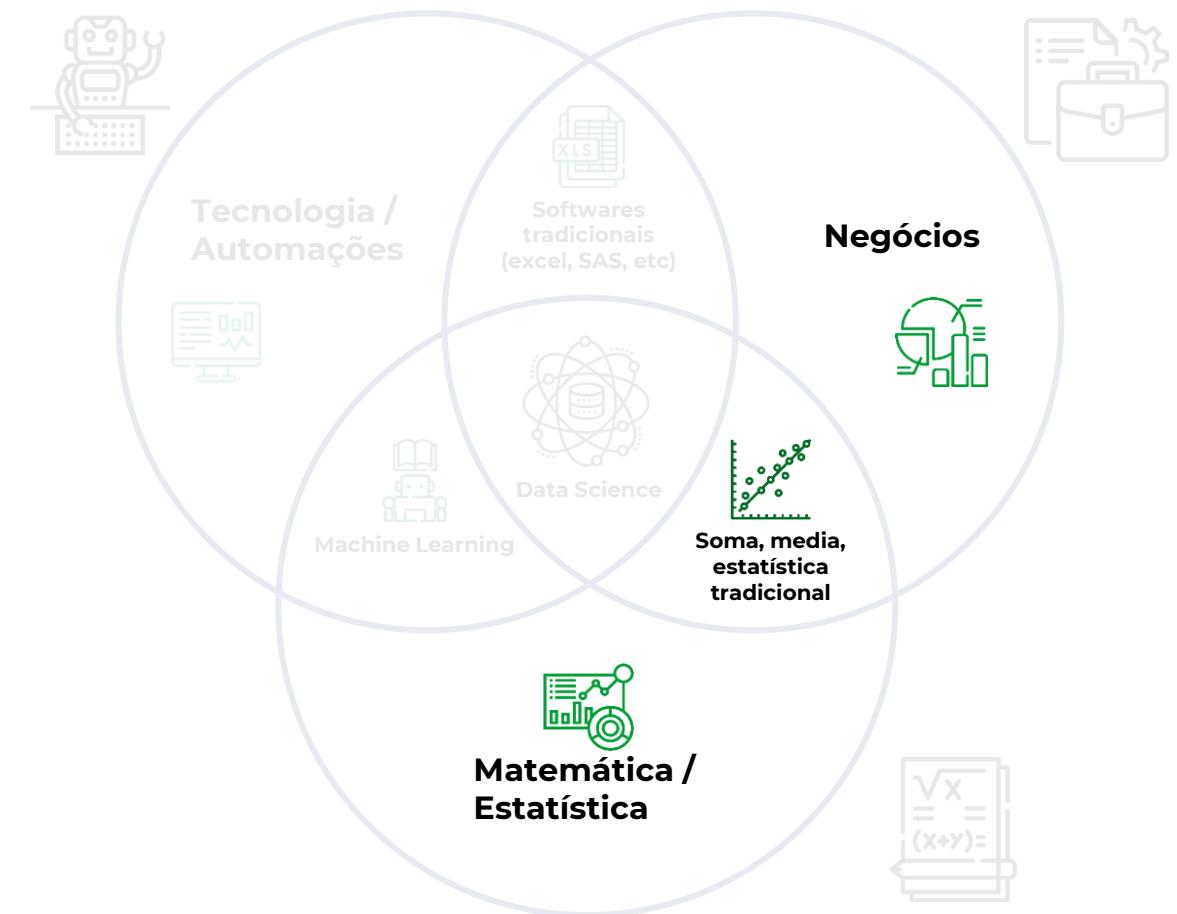
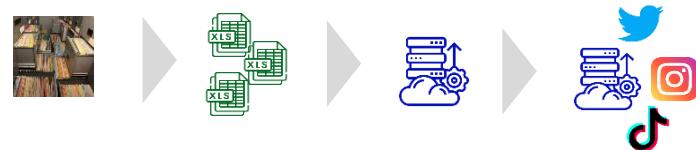
Porém, muitas vezes essas médias não traduzem a realidade, por isso, a importância da área de *Data Science*.



# Módulo 1 – Os Pilares da Ciência de Dados

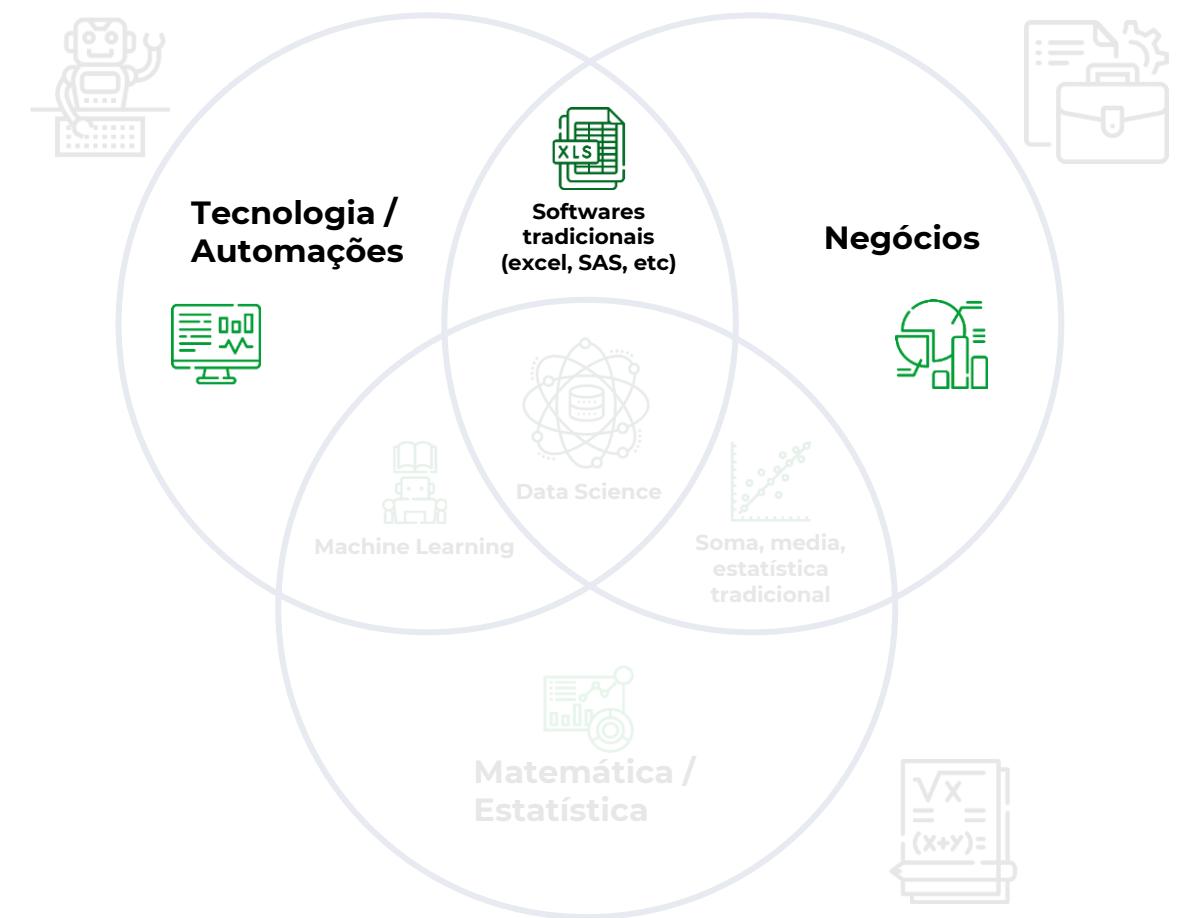
Por que é tão perigoso a gente não usar esses 3 pilares não relacionados?

Sem a tecnologia atrelada não é possível fazer a empresa continuar funcionando na quantidade de dados que temos hoje em dia.



# Módulo 1 – Os Pilares da Ciência de Dados

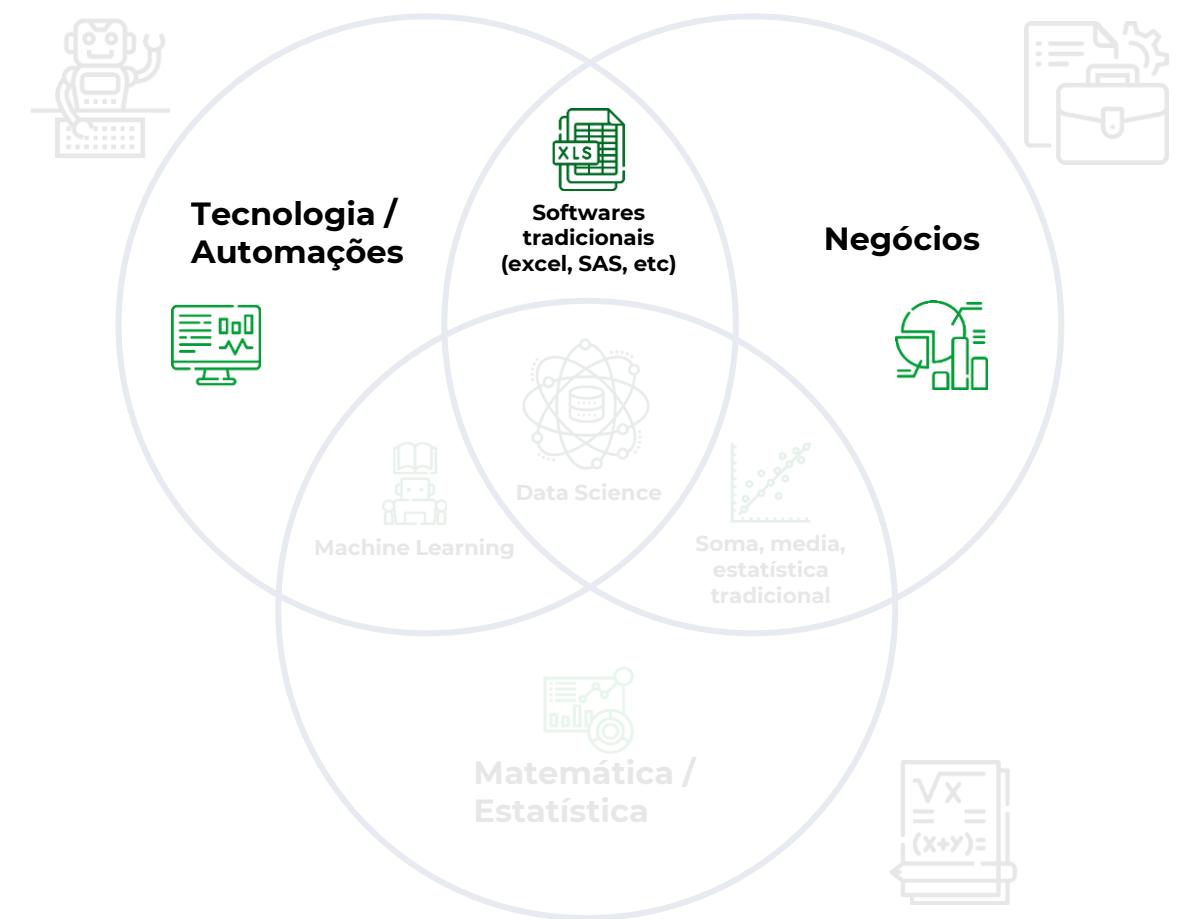
Agora, por exemplo, se só considerarmos os softwares tradicionais, podemos ter conclusões corretas, só que pode ter pouca validade, ou até nenhuma validade estatística e assim também pode não ter tanto significado para o negócio.



# Módulo 1 – Os Pilares da Ciência de Dados

Por exemplo: se a média de vendas desse ano foi melhor que a do ano passado, porém se eu tive muito mais lojas ou muito mais vendedores, essa não é uma informação que tem tanta validade.

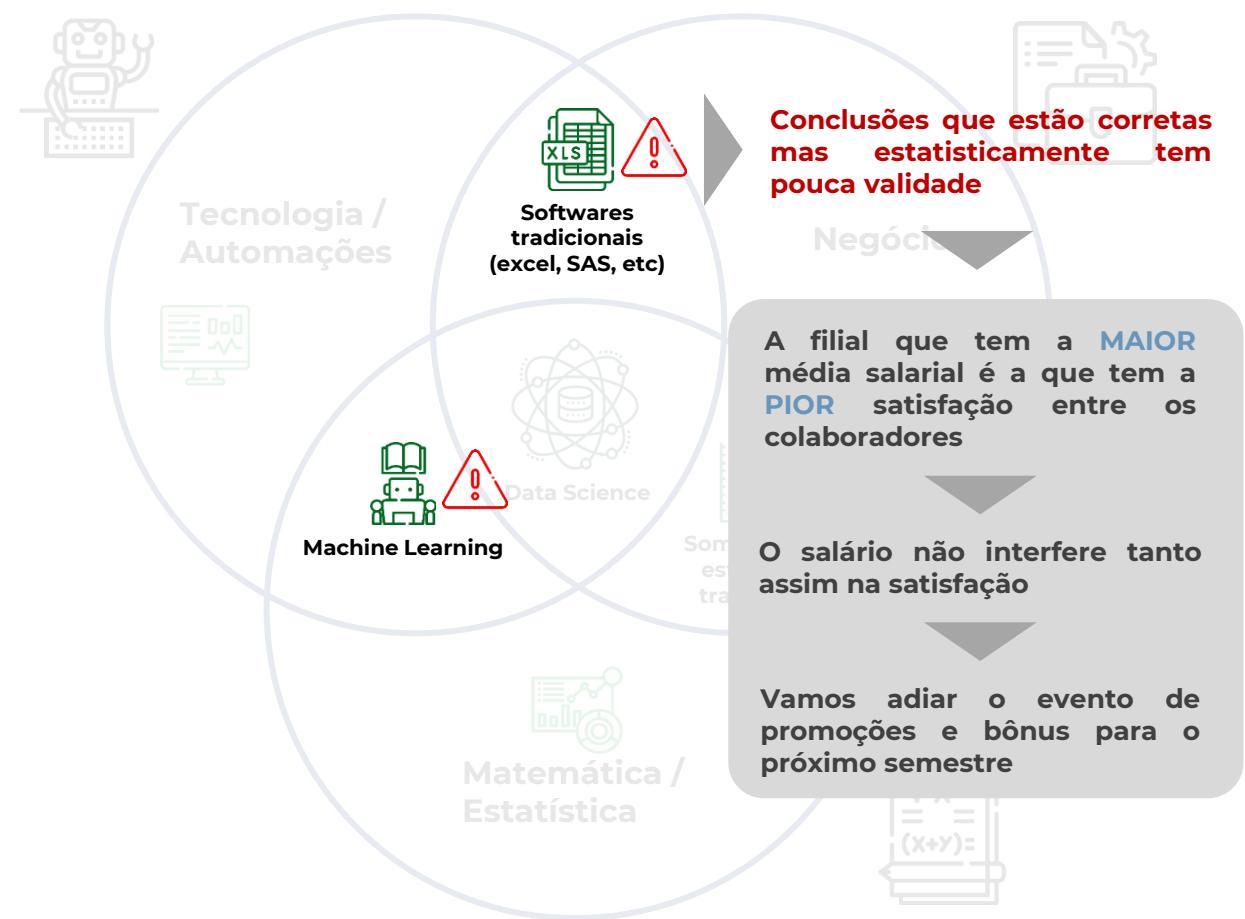
Por isso é necessário entender estatisticamente se aquele resultado faz sentido.



# Módulo 1 – Os Pilares da Ciência de Dados

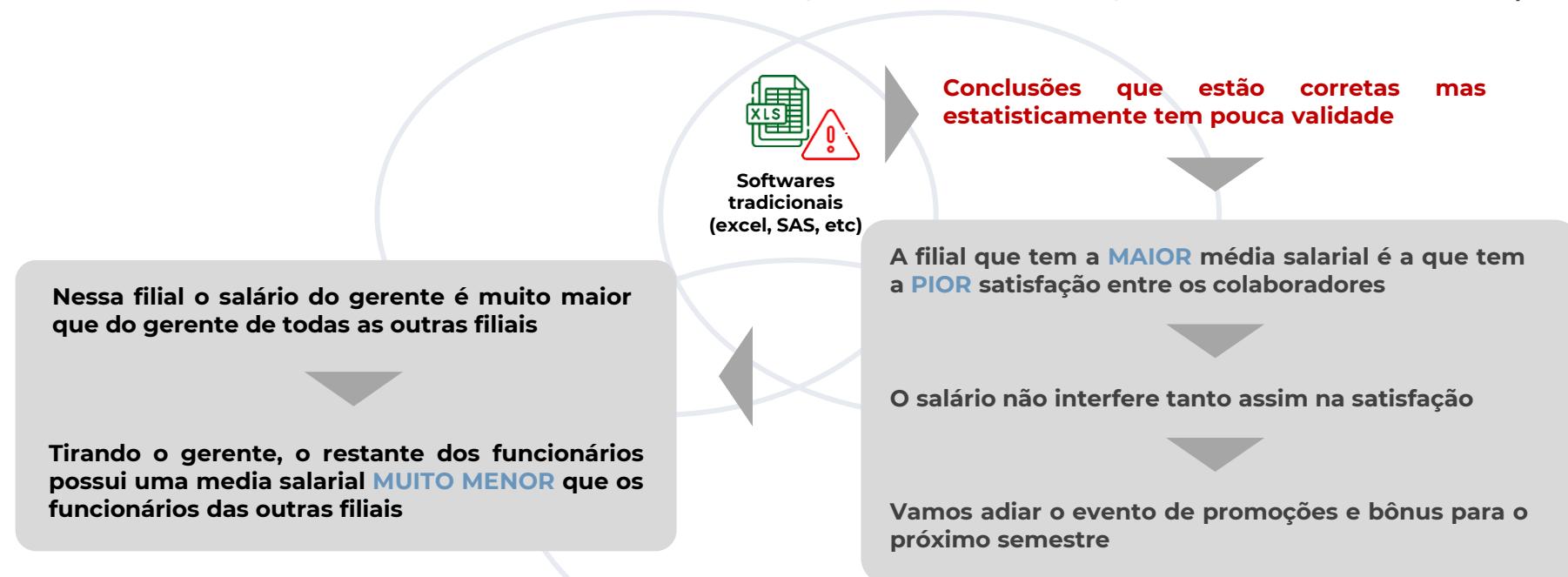
Agora vamos ver um exemplo simples:

Realizar uma análise de filiais porque o índice de insatisfação está alto. Pela análise da média salarial é possível notar que a filial com pior satisfação é a que possui maior média, com isso, conclui-se que o salário não é o problema e adia-se o evento de promoções.



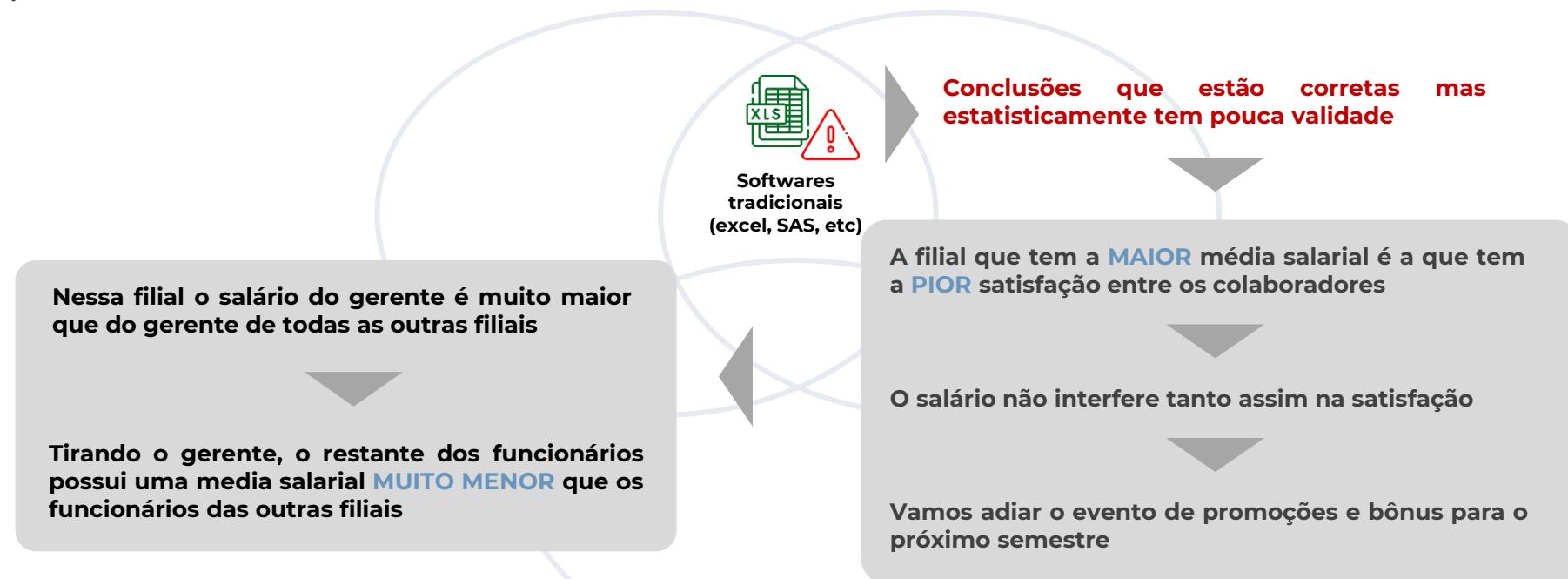
# Módulo 1 – Os Pilares da Ciência de Dados

Quando se analisa a estatística dessa informação, nota-se que na verdade nessa filial de maior média salarial, o salário do gerente é muito maior que o das outras filiais. Então, essa média é maior por causa do salário do gerente, porém o salário de todos os outros funcionários é muito menor, isso acontece pois a média pode ser afetada pelos outliers.



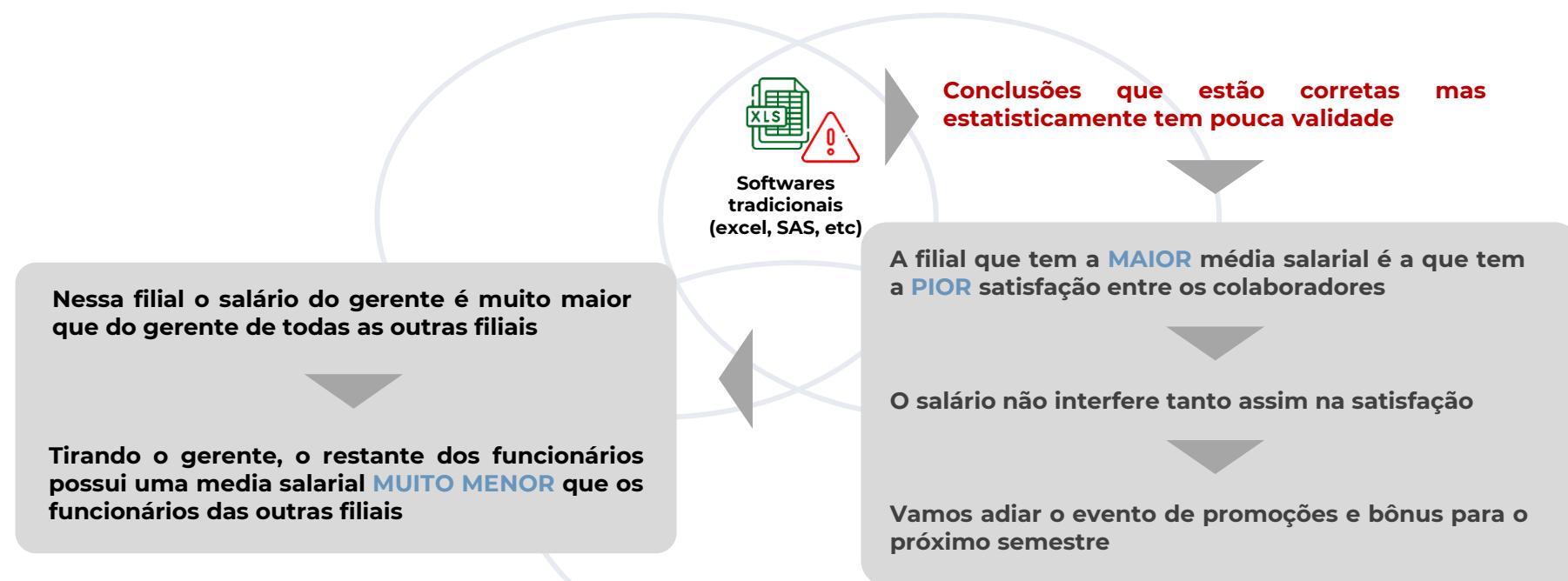
# Módulo 1 – Os Pilares da Ciência de Dados

Com a conclusão equivocada da fonte do problema, as promoções foram adiadas e vários funcionários pediram demissão. Nesse caso eu gerei uma conclusão que estava certa (a média salarial é maior), só que estatisticamente não foi levado em consideração o desvio padrão.



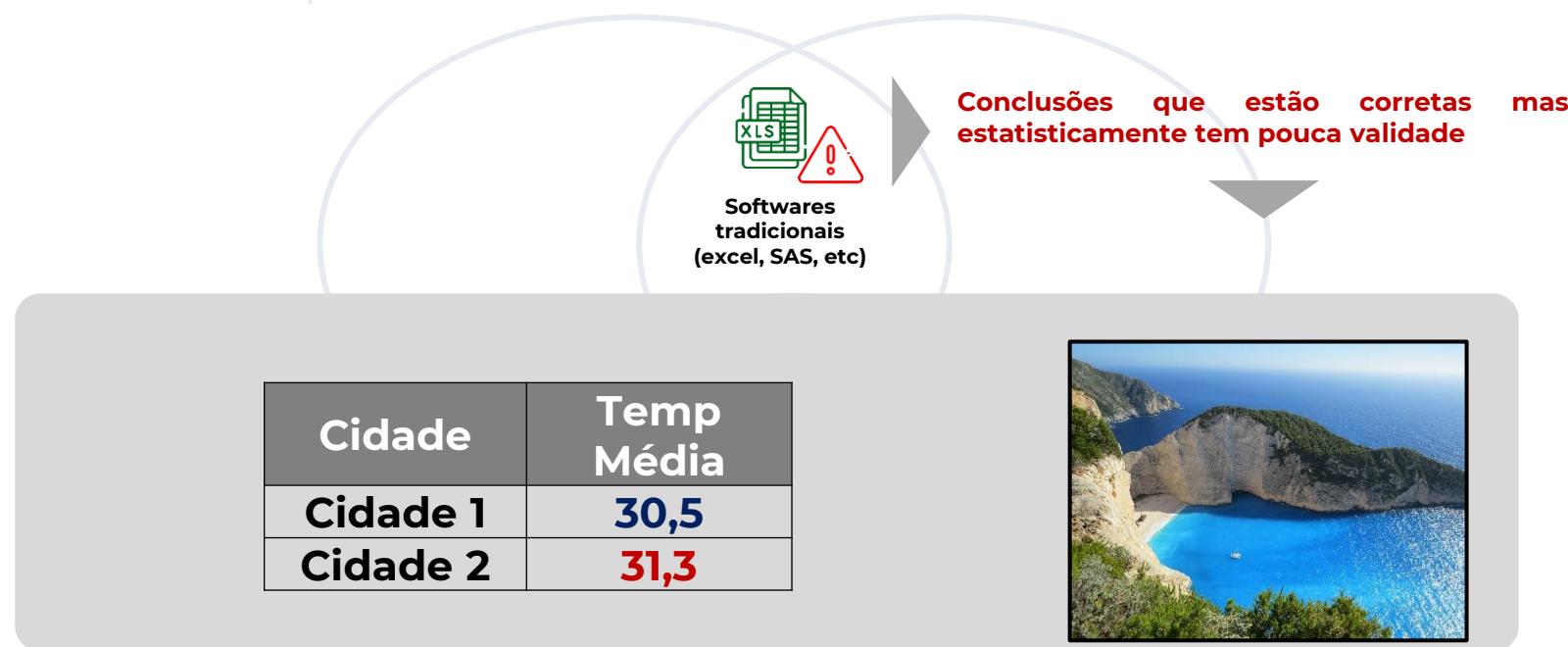
# Módulo 1 – Os Pilares da Ciência de Dados

A decisão foi tomada, mas não tinha validade estatística e a decisão prejudicou a empresa.



# Módulo 1 – Os Pilares da Ciência de Dados

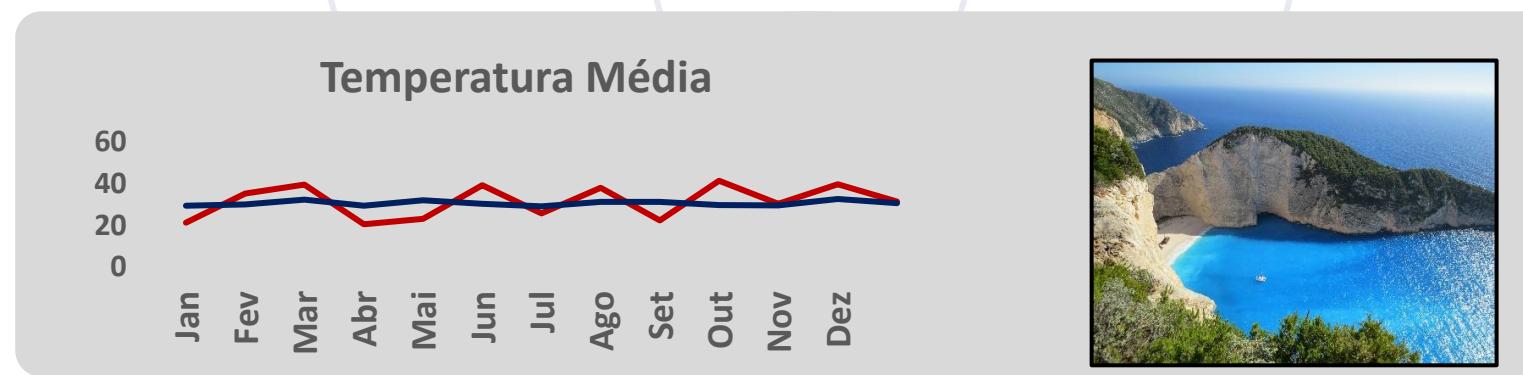
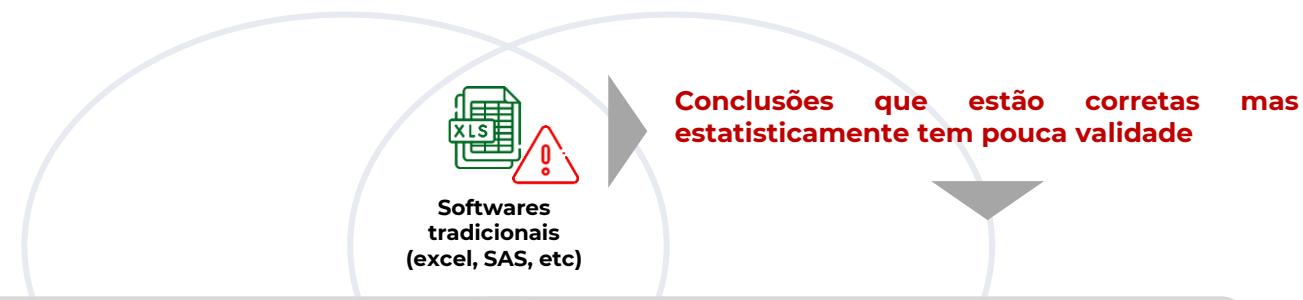
Um outro exemplo, se estivéssemos na dúvida entre uma viagem para essas 2 cidades e o nosso objetivo fosse ir à praia. Observamos que a Cidade 2 tem uma temperatura média maior que a Cidade 1, como o objetivo é ir à praia eu vou escolher a Cidade 2 que tem uma média de temperatura maior.



# Módulo 1 – Os Pilares da Ciência de Dados

Ao analisar o histórico da temperatura, percebe-se que o histórico da cidade 2 tem uma variação de temperatura alta e por mais que a média seja maior, tem chance de a temperatura estar 20°C, então eu vou acabar sentindo frio na praia.

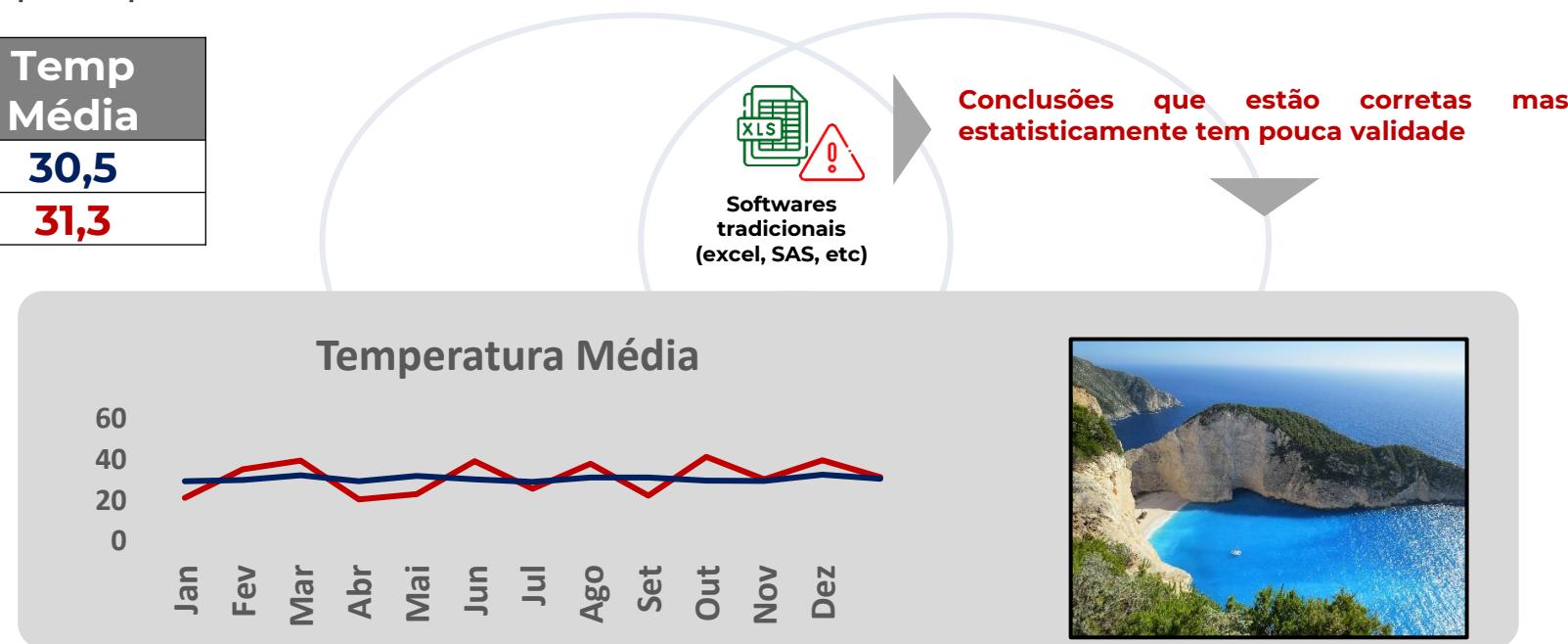
Cidade	Temp Média
Cidade 1	30,5
Cidade 2	31,3



# Módulo 1 – Os Pilares da Ciência de Dados

Embora a Cidade 1 tenha uma média menor, a sua temperatura é mais constante. Ou seja, se eu quiser uma segurança de viajar com uma temperatura agradável eu deveria escolher a Cidade 1. Nesse caso, iríamos escolher um lugar que tinha a chance de sentir frio na praia, porque não entendemos esses conceitos básicos de estatística.

Cidade	Temp Média
<b>Cidade 1</b>	<b>30,5</b>
<b>Cidade 2</b>	<b>31,3</b>

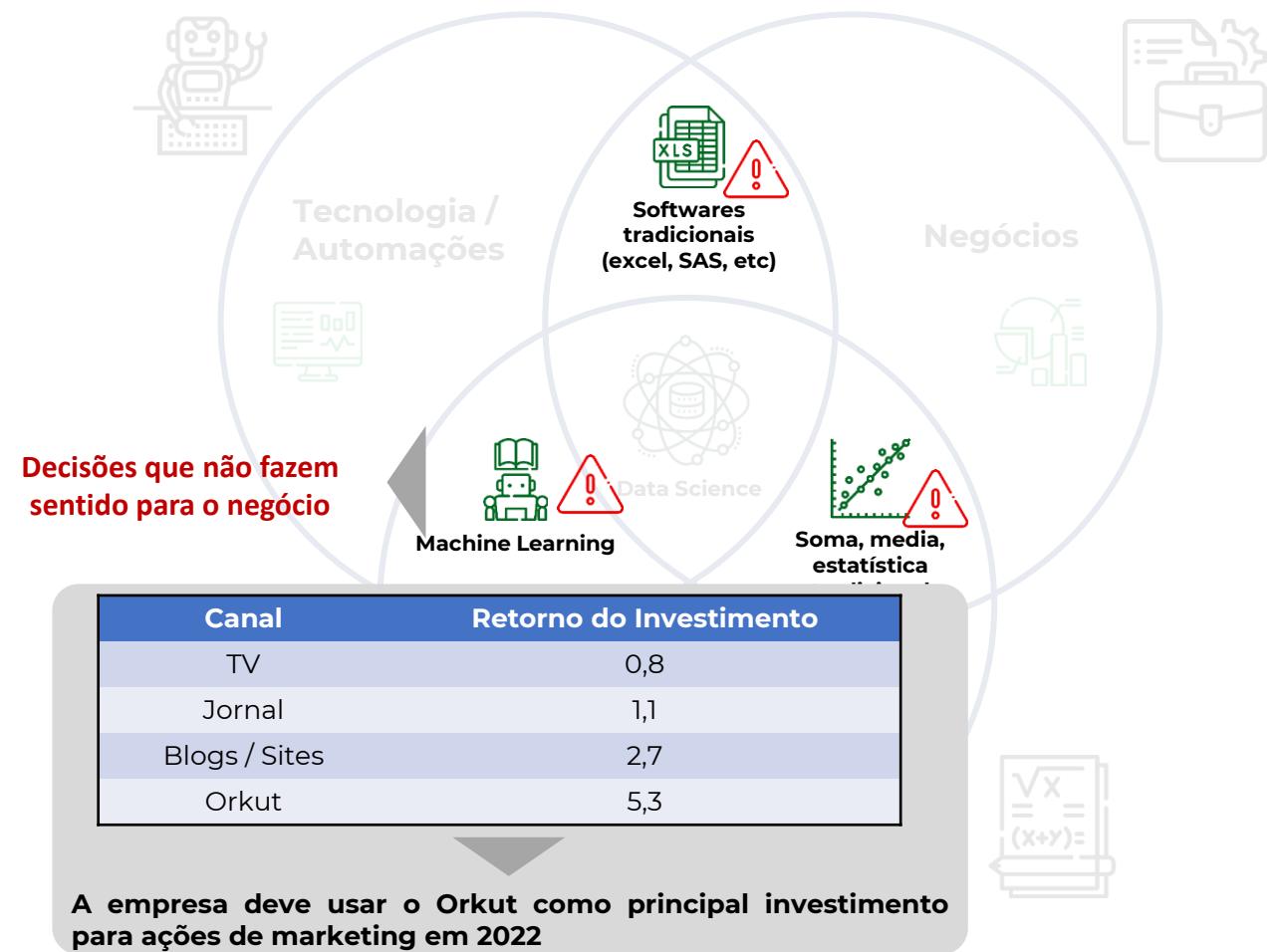


# Módulo 1 – Os Pilares da Ciência de Dados

Quando falamos de *Machine Learning* podemos encontrar decisões que não fazem sentido para o negócio.

Por exemplo: análise do canal e o retorno do investimento.

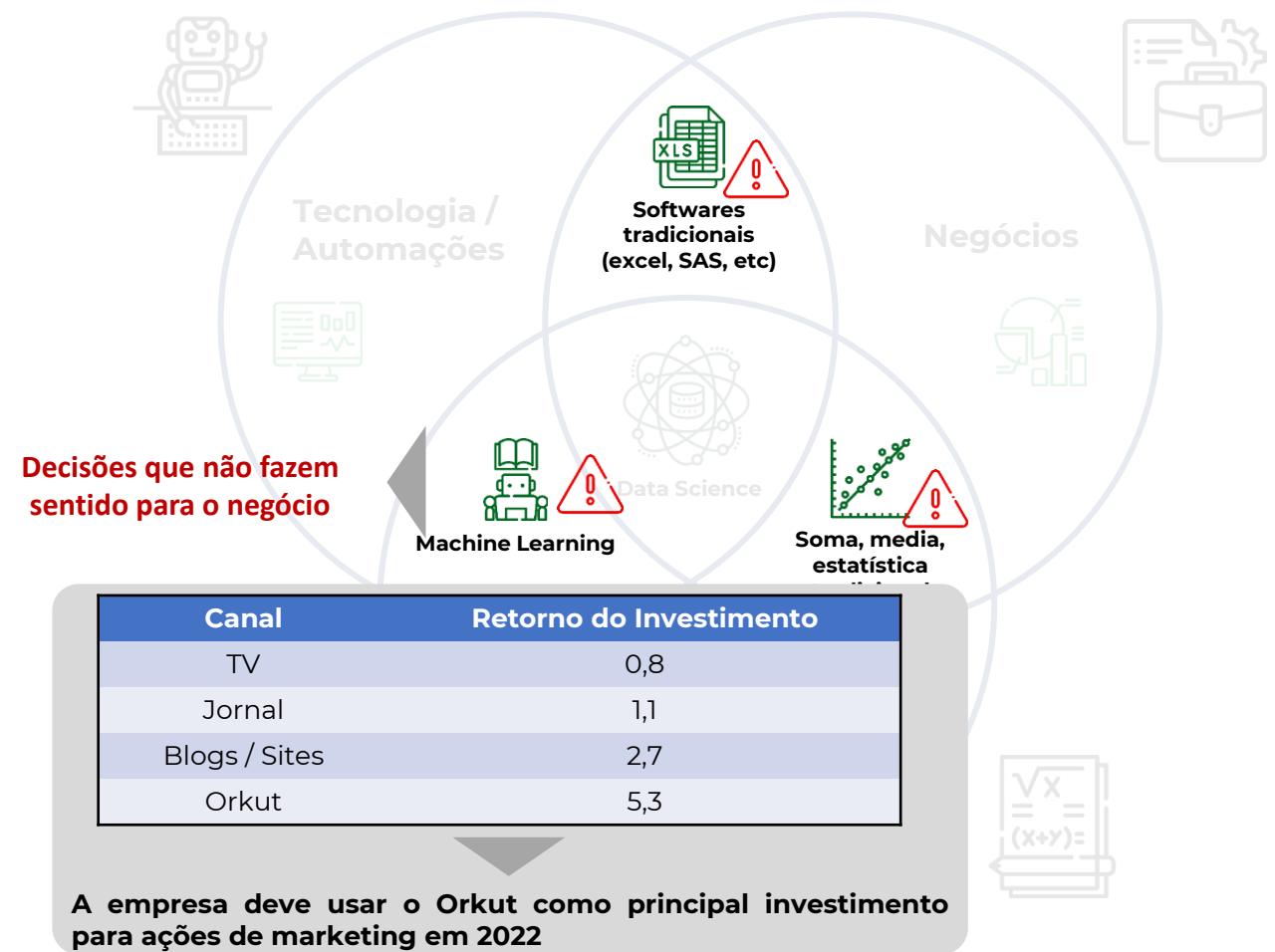
Pelo modelo de *Machine Learning*, a empresa deve usar como principal investimento o Orkut, porém é uma plataforma que não existe mais.



# Módulo 1 – Os Pilares da Ciência de Dados

O modelo vai usar os dados que ele possui. Então, se foi feito um investimento no Orkut, que em algum momento teve um retorno muito bom, o modelo pode indicar que esse é o melhor investimento.

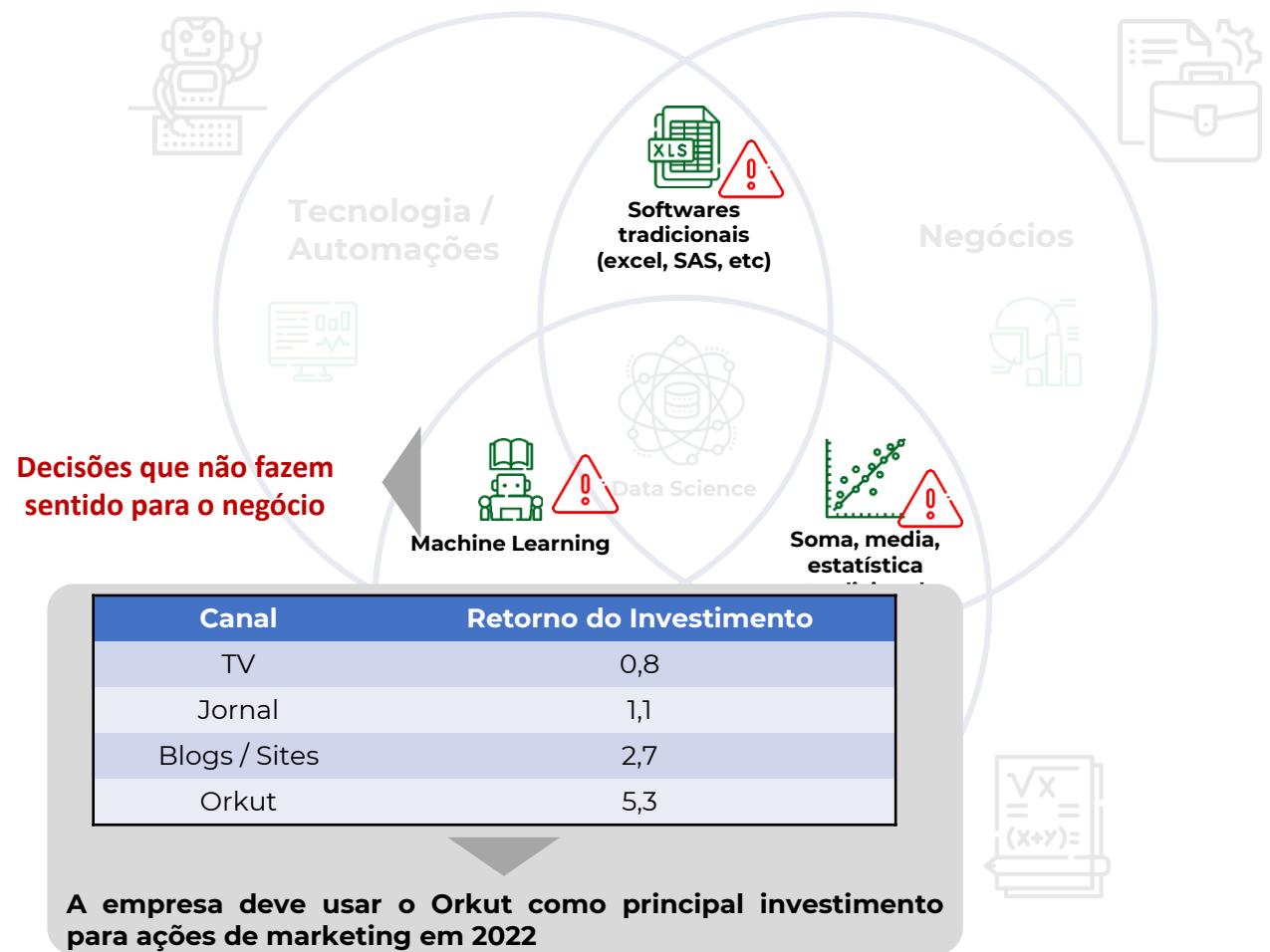
Então, é necessário ter muito cuidado para garantir que resultado do modelo faça sentido dentro do negócio.



# Módulo 1 – Os Pilares da Ciência de Dados

Muitas empresas tomam decisões baseado em resultados do Machine Learning sem entender realmente todo o contexto dentro da empresa e pode gerar muito prejuízo.

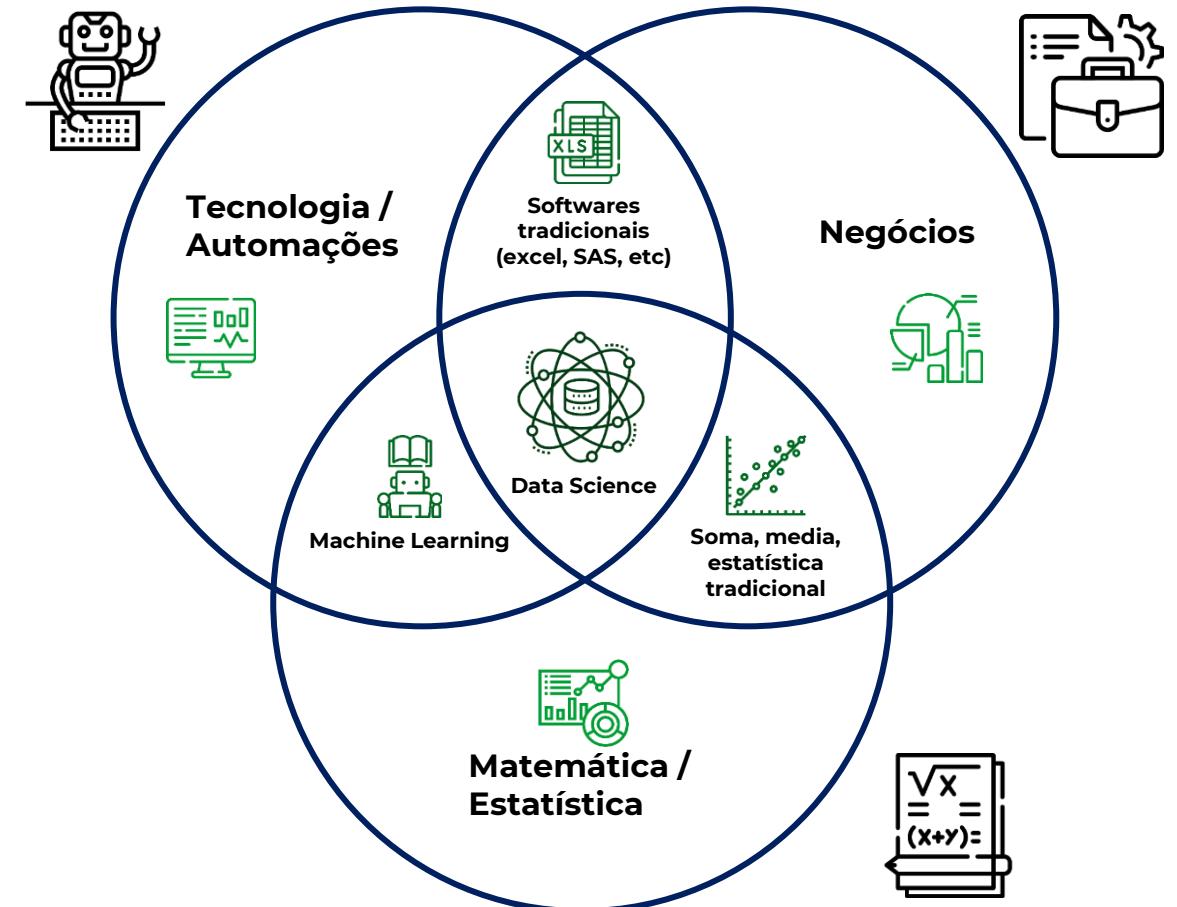
Nesse exemplo, poderíamos ter evitado essa conclusão criando uma tabela com a informação se a plataforma está ativa.



# Módulo 1 – Os Pilares da Ciência de Dados

O *Data Science* está exatamente na interseção entre essas 3 áreas, não existe uma área que é mais prioritária ou um pilar que é mais importante.

Precisamos entender esses **3 pilares** e como eles trabalham juntos, esse é o grande diferencial no Cientista de Dados.



## MÓDULO 2

# Introdução a Ciência de Dados

## Módulo 2 – O que é ser um cientista?

Ciência é ter um **método científico**, fazer observações e hipóteses.

A primeira coisa do cientista é ele entender que ele não está buscando coisas novas, que ele busca coisas em artigos que já existem, programas que outras pessoas já escreveram.

## Módulo 2 – O que é ser um cientista?

O **método científico** exige que o nosso estudo consiga ser replicado, ou seja, não adianta fazer um modelo ótimo, nível de certeza alto se aquele modelo não consegue ser replicado.

Existe um método científico! Observação, hipóteses, testes e validação, análises, monitoramento.



Precisamos que existam dados armazenados (ou pelo menos começar esse armazenamento). Armazenamento, processamento, visualização.

## Módulo 2 – O que é ser um cientista?

Quando falamos em método científico falamos de observação, ou seja, é necessário observar o que está acontecendo na empresa.

O cientista de dados precisa ter muito conhecimento de negócio, precisa ver se o que é observado realmente faz sentido, ou se foi uma coisa muito pontual que não é o padrão.



Observação

## Módulo 2 – O que é ser um cientista?

É necessário criar hipóteses, pois não podemos fazer várias análises sem realmente pensar em o que queremos analisar.

Então primeiro pensamos em qual hipótese queremos testar e depois vamos analisar essa hipótese, ela pode ser verdadeira ou não.

Se a hipótese não for verdadeira vamos ter que testar outras hipóteses.



Observação

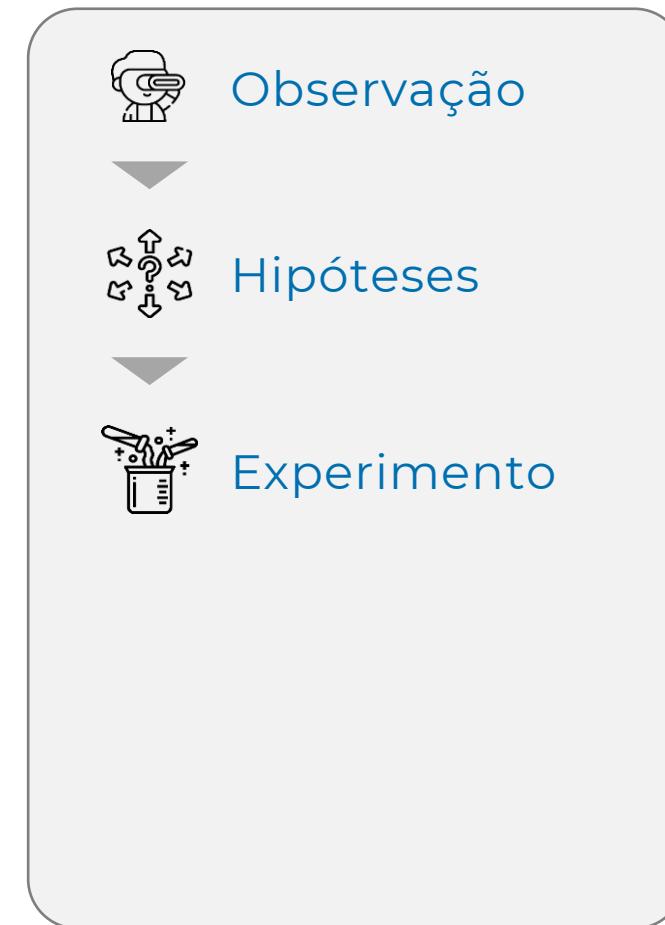


Hipóteses

## Módulo 2 – O que é ser um cientista?

Uma das etapas é fazer experimentos, testar se aquilo realmente funciona ou se vai ser necessário buscar modelos diferentes.

Nessa etapa vamos testar modelos e variáveis.



## Módulo 2 – O que é ser um cientista?

Depois de testar é necessário analisar e observar quais foram os resultados gerados.

Para então chegar a uma conclusão.



## Módulo 2 – O que é ser um cientista?

Só que esse processo não é tão linear, nem sempre do resultado vamos direto para a conclusão.

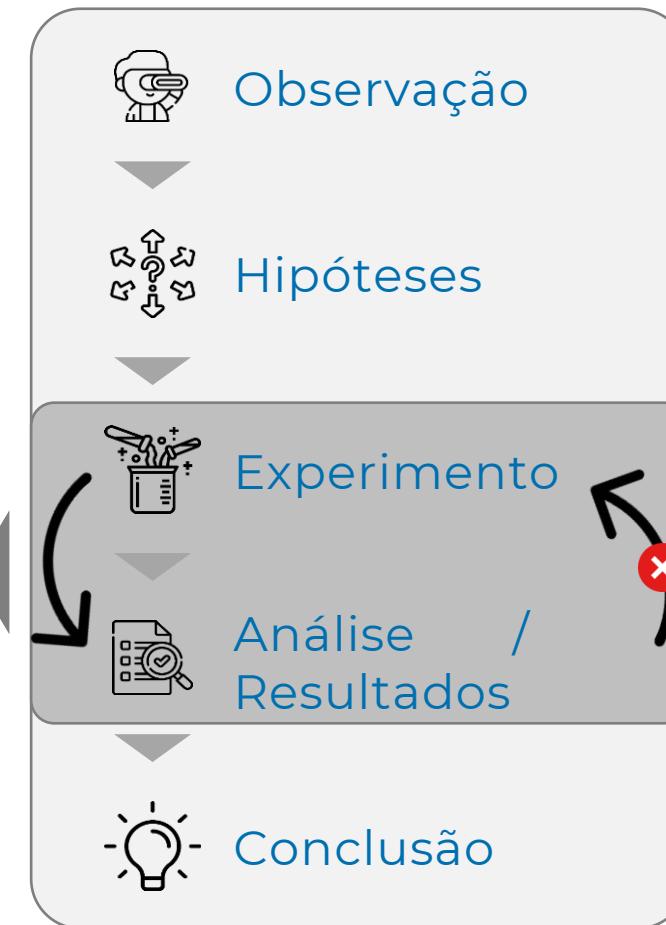
Muitas vezes vamos ficar muito tempo nesse processo de experimentar e analisar. Se o experimento não deu certo, mudamos o modelo ou a variável.



## Módulo 2 – O que é ser um cientista?

Esse processo gera uma complexidade muito grande quando falamos de dados, por isso a tecnologia é tão importante para esse processo científico.

Atualmente temos muitos dados por isso precisamos de tecnologia, que no nosso caso será o Python.



## Módulo 2 – O que é ser um cientista?

Vamos experimentar muitas coisas usando o modelo de *Machine Learning*, vamos buscar coisas que já existem e bibliotecas já existentes.

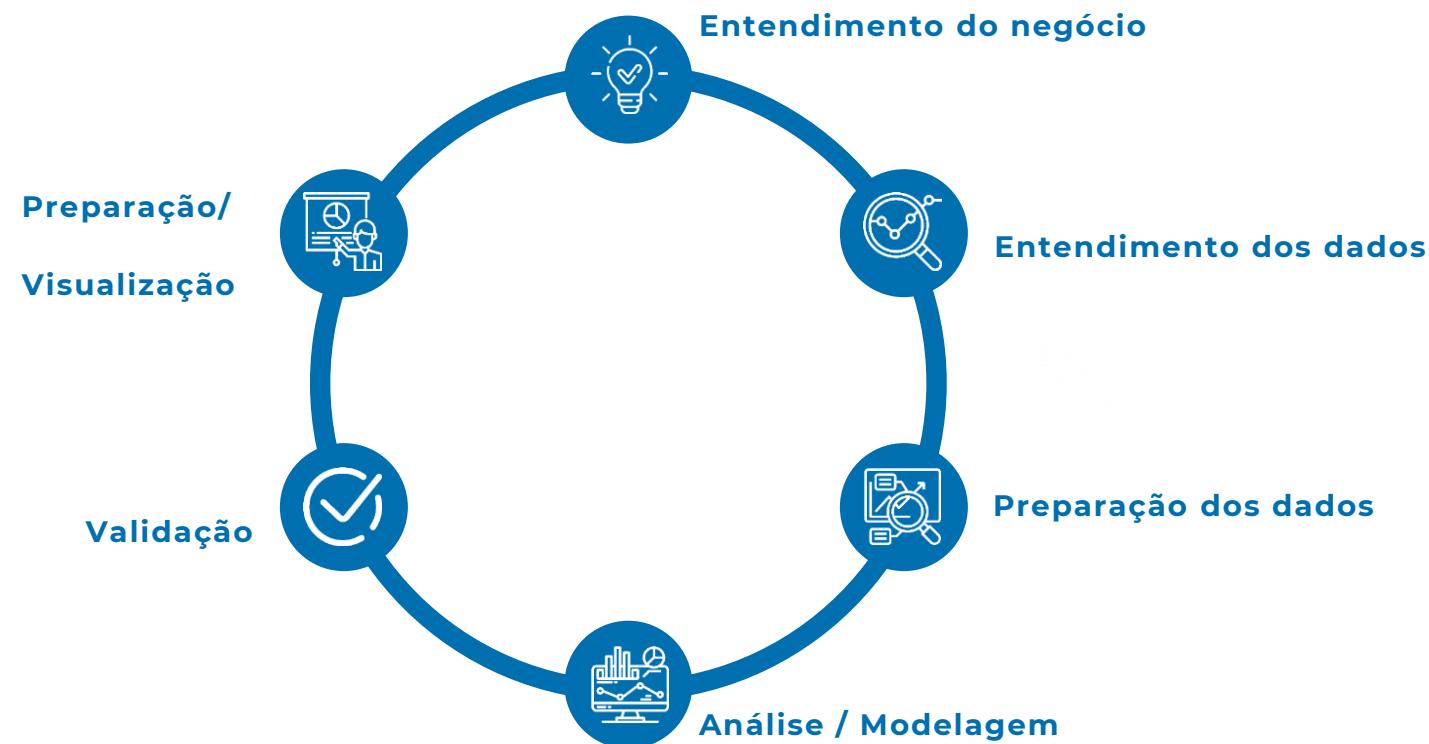
Além disso, vamos buscar também um processo que outra pessoa já fez, por exemplo: problema de ruptura na empresa.

Nós vamos buscar em fontes que já tiveram esse mesmo problema, para saber o que a empresa fez, o quê e como ela analisou.



## Módulo 2 – Um framework para Ciência de Dados

Podemos usar o CRISP-DM (*CRoss Industry Standard Process for Data Mining*) que é um processo existente como base para o nosso framework.



## Módulo 2 – Um framework para Ciência de Dados

O entendimento do negócio é dividido em 2 etapas:

- Definição do problema: entender qual pergunta o seu chefe quer que você responda
- Alinhamento das expectativas: alinhe a precisão para não ter quebra de expectativa.



## Módulo 2 – Um framework para Ciência de Dados

Entre o entendimento e a preparação dos dados é necessário fazer a engenharia dos dados, para garantir que a informação vai estar redonda antes de colocar no modelo.

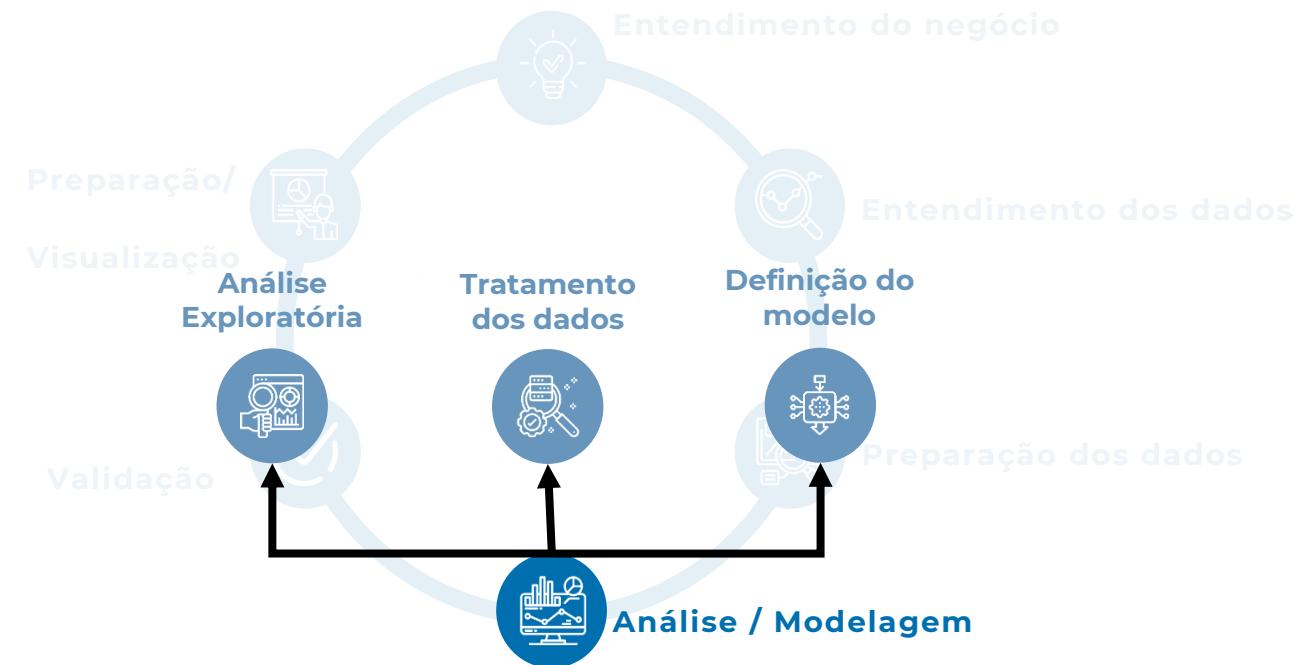
Em seguida tem a preparação de dados para fazer um filtro e diminuir o trabalho no tratamento dos dados.



## Módulo 2 – Um framework para Ciência de Dados

A análise e modelagem são divididas em 3 etapas:

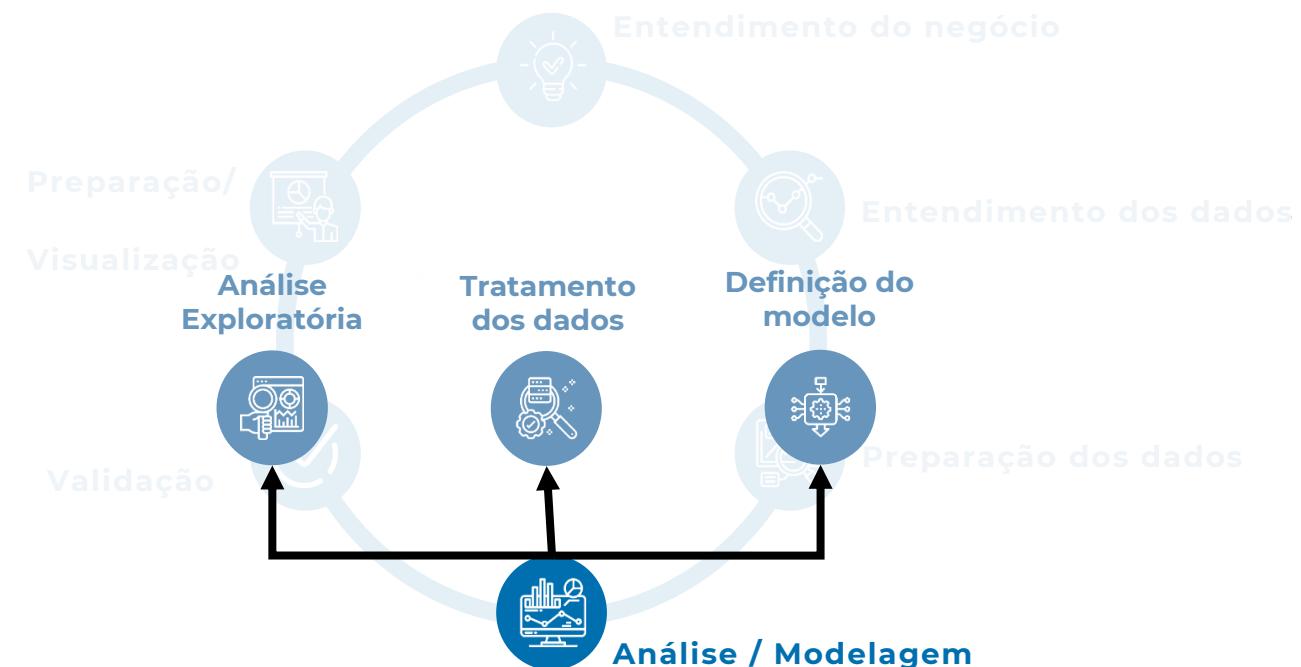
- Análise exploratória
- Tratamento de dados
- Definição do modelo



## Módulo 2 – Um framework para Ciência de Dados

Na parte de análise exploratória já conversamos muito com o contratante do projeto.

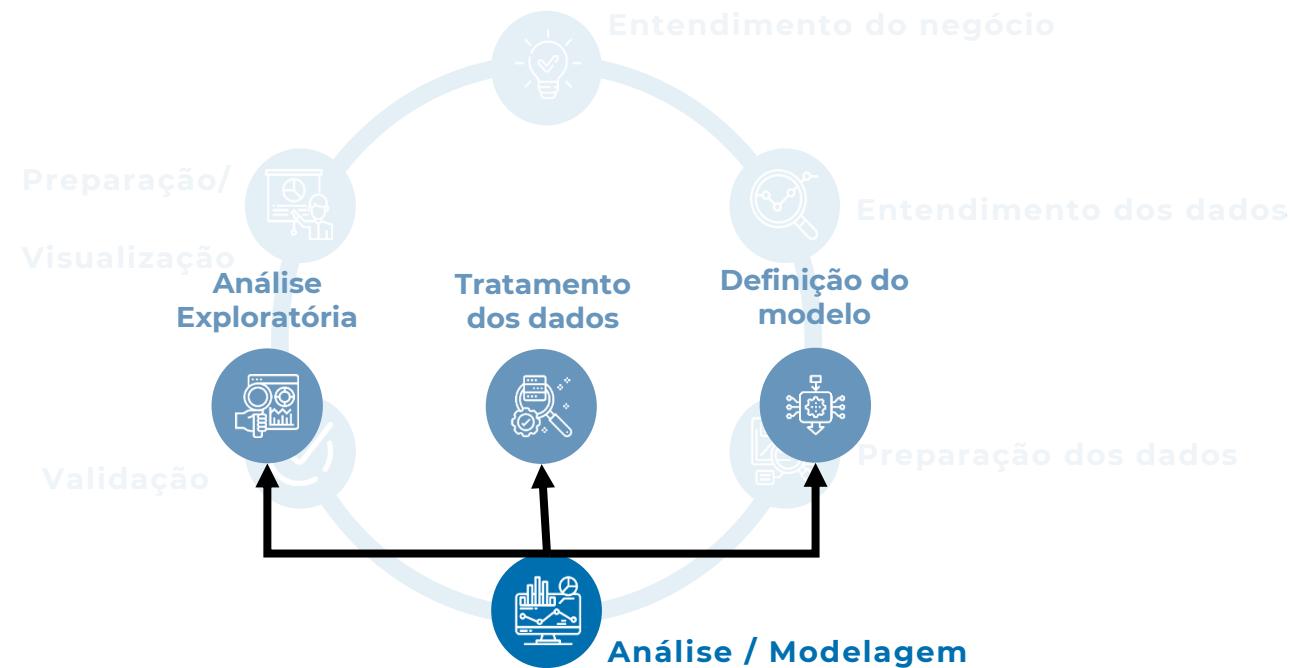
- Essa parte é para entender os dados e o negócio.
- A análise exploratória tem capacidade de começar a gerar valor, por exemplo: descobrir o produto que mais vende.



## Módulo 2 – Um framework para Ciência de Dados

Tratamento dos dados é a etapa mais importante, porque se colocarmos dados ruins no modelo, ele vai gerar um resultado ruim.

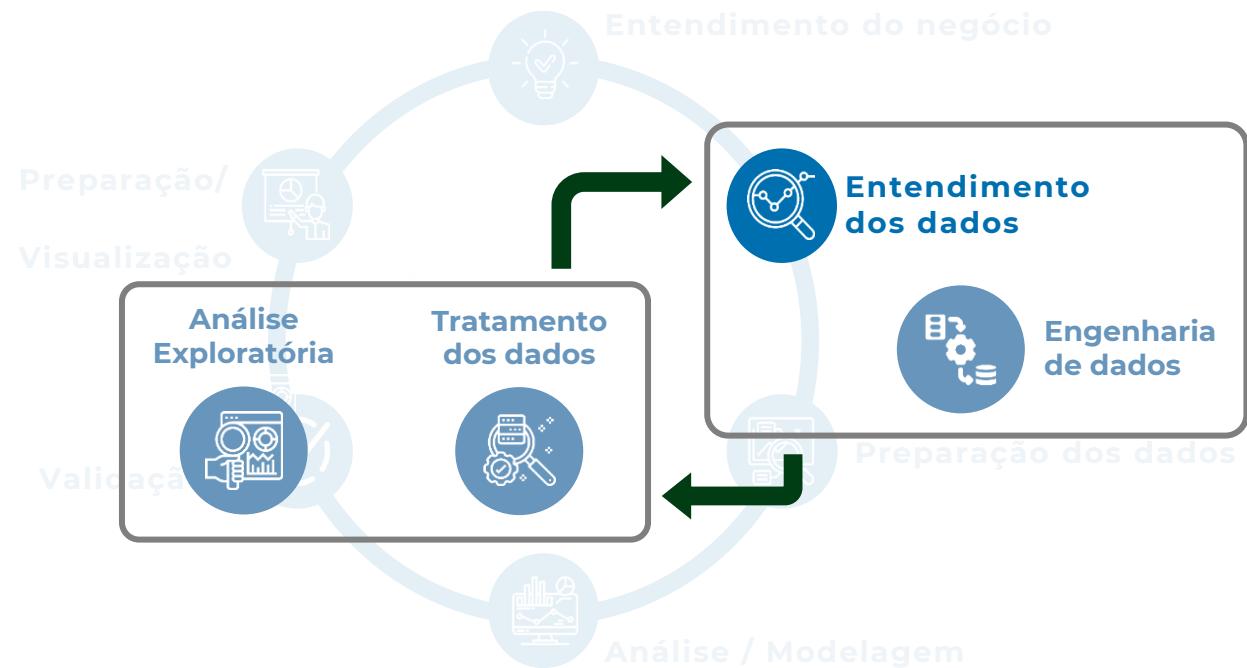
Nessa etapa vamos remover valores nulos que às vezes não foram preenchidos, mas que não podem ser zero. Por exemplo o valor da altura ser zero: nós sabemos que é impossível isso acontecer.



## Módulo 2 – Um framework para Ciência de Dados

Essas etapas não são lineares, na maioria das vezes quando fazemos o tratamento de dados percebemos um problema na base ou que precisamos de um dado novo.

Nesses casos voltamos na etapa de entendimento dos dados, se necessário uma engenharia de dados e refazemos a análise exploratória.



## Módulo 2 – Um framework para Ciência de Dados

Essas etapas serão repetidas até alcançarmos a segurança no tratamento de dados para colocar no modelo.

O modelo vai executar, selecionar qual o melhor modelo, selecionar variáveis e escolher o menor erro.



## Módulo 2 – Um framework para Ciência de Dados

Depois que o modelo escolhido for testado e validado, vamos para a parte de validação para garantir que o modelo esteja funcionando para os dados de teste e com novos dados.

Vamos fazer isso até achar o modelo perfeito, que não terá uma precisão de 100% e sim a precisão maior que a acordada com o contratante.



## Módulo 2 – Um framework para Ciência de Dados

Com o modelo validado, a próxima etapa é apresentar para o contratante com a visualização de dados, gráficos e contar a história.

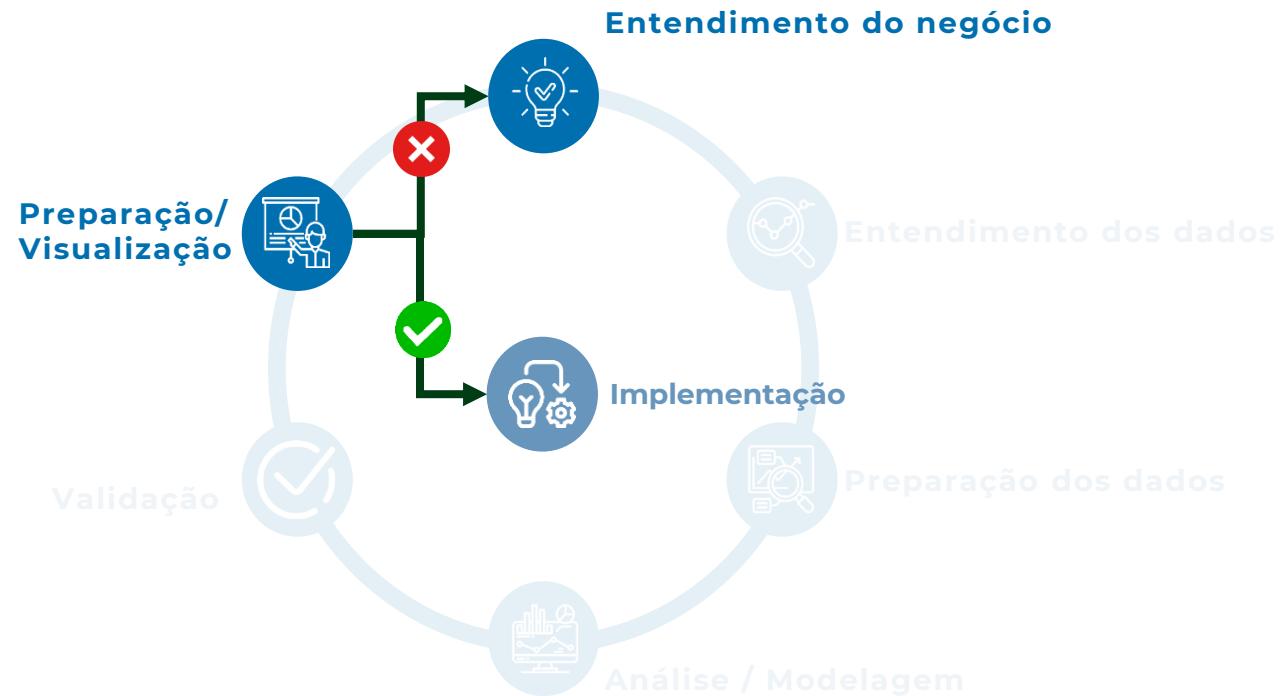
Nessa etapa não é preciso contar com todos os detalhes os modelos testados, ele quer entender o que vai resolver o problema dele.



## Módulo 2 – Um framework para Ciência de Dados

Na apresentação o projeto pode ser aceito ou o contratante pode falar que não era isso que ele queria.

Se for aceita vamos para a implementação, caso contrário precisamos refazer todo o ciclo.



## Módulo 2 – Um framework para Ciência de Dados

Com o modelo validado, vamos implementar e colocar na produção, mas por ser um método científico é necessário fazer melhorias e monitorar os dados.



## Módulo 2 – Um framework para Ciência de Dados

Todas essas etapas são muito importantes para o projeto de *Data Science*.

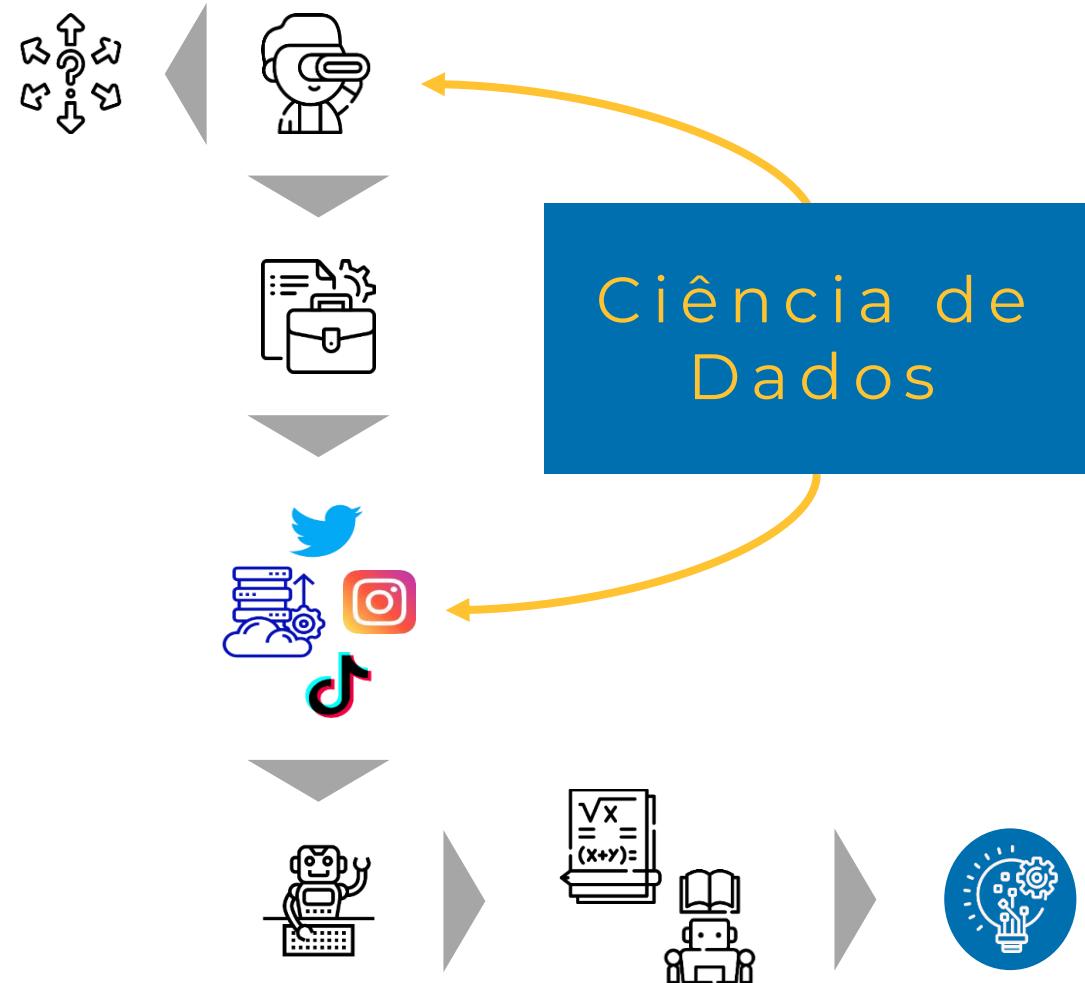


## Módulo 2 – Resumindo Ciência de Dados

A ciência de dados começa na observação e criação de hipóteses, que estão diretamente relacionadas com o negócio.

Depois começamos a trabalhar com as informações utilizando a tecnologia.

Por último, começamos a criar modelos, com a base estatística, para gerar uma conclusão com valor para o negócio.



## Módulo 2 – Um framework para Ciência de Dados

A linguagem de programação utilizada durante o curso é o Python, mas você pode usar outro programa pois a estatística é a mesma.



```
In [1]: print("Hello Python!")  
Hello Python!
```



Mais fácil entendimento e aprendizado



Comunidade forte, participativa e é mais fácil de achar dúvidas na internet



Grande crescimento no número de usuários



Cada vez mais buscada em vagas de trabalho

## Módulo 2 – Um framework para Ciência de Dados

A linguagem de programação utilizada durante o curso é o Python, mas você pode usar outro programa pois a estatística é a mesma.



```
In [1]: print("Hello Python!")  
Hello Python!
```



Usada em todo o processo de Data Science  
(não precisa de outra linguagem)



Também pode ser usada no processo de ETL  
(extração, transformação e carregamento)



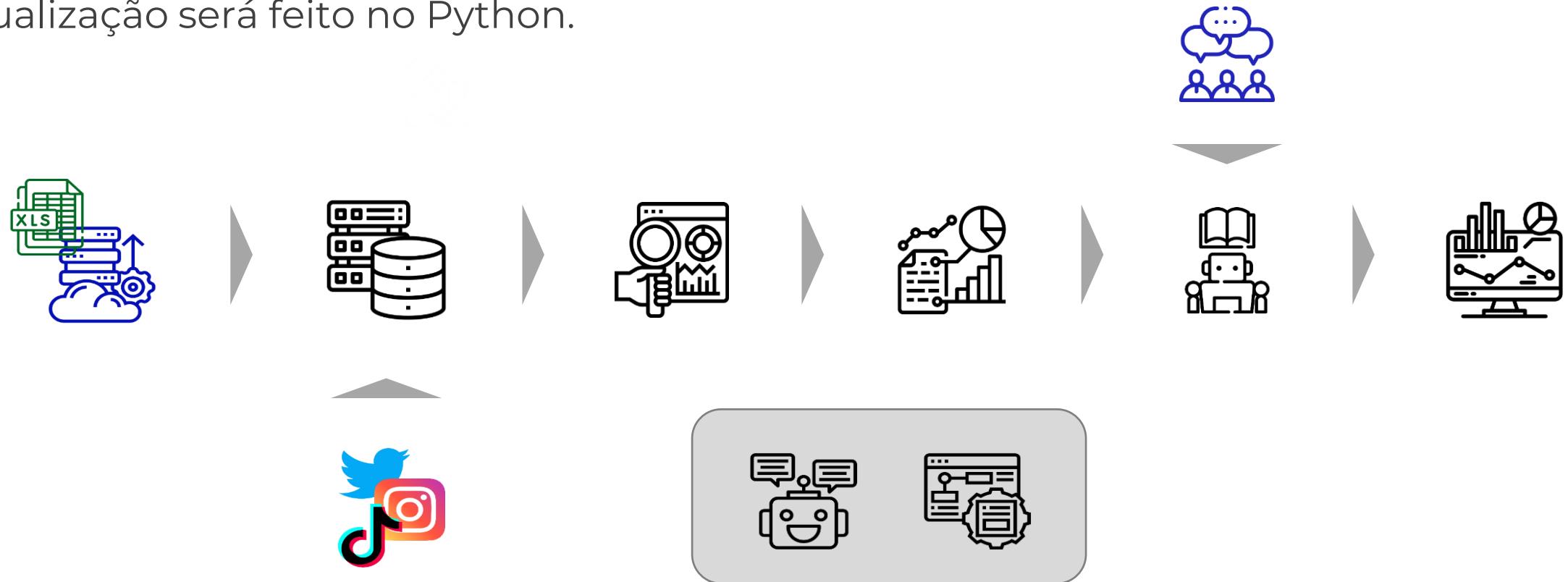
Existência de vários pacotes prontos  
(como o pandas)



Possibilidade de web scraping, chatbot, etc.

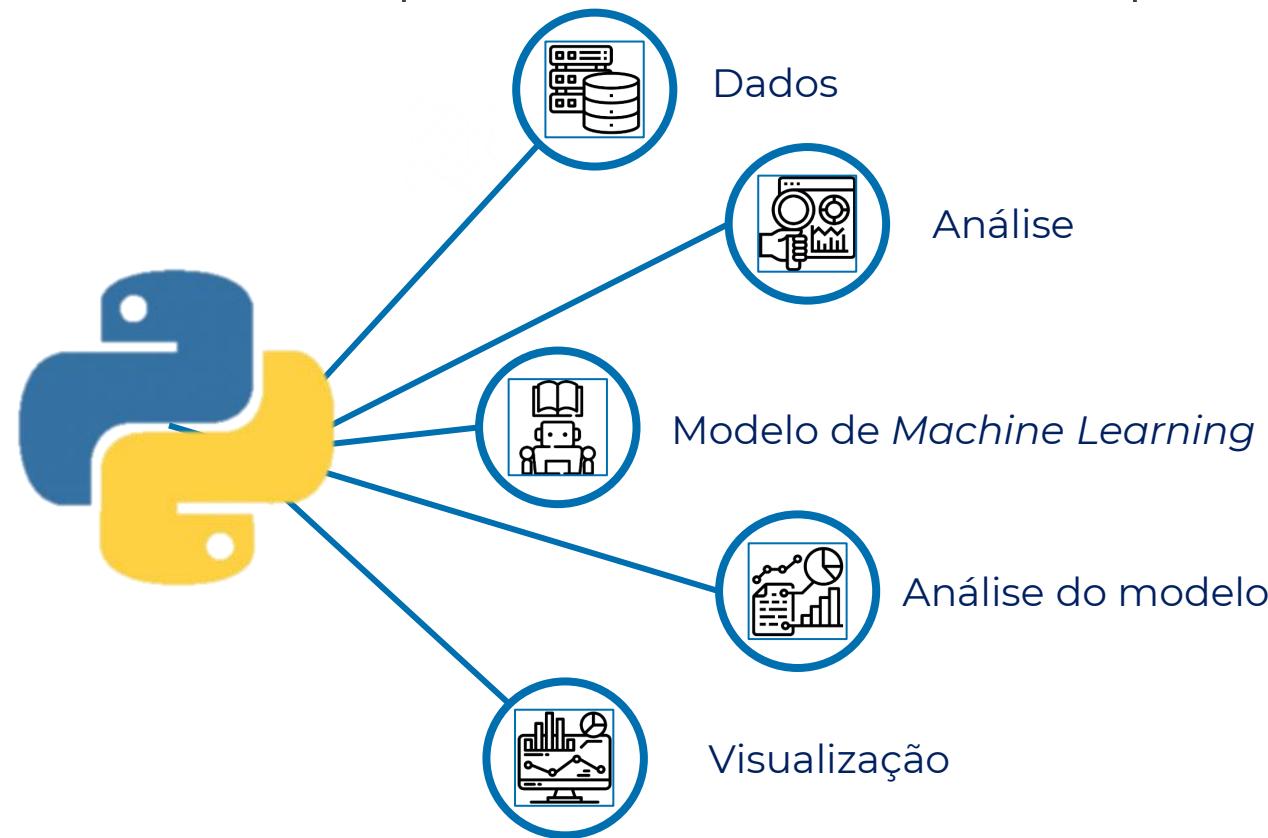
## Módulo 2 – Um framework para Ciência de Dados

Todo o processo de organização dos dados, análise exploratória, Machine Learning e visualização será feito no Python.



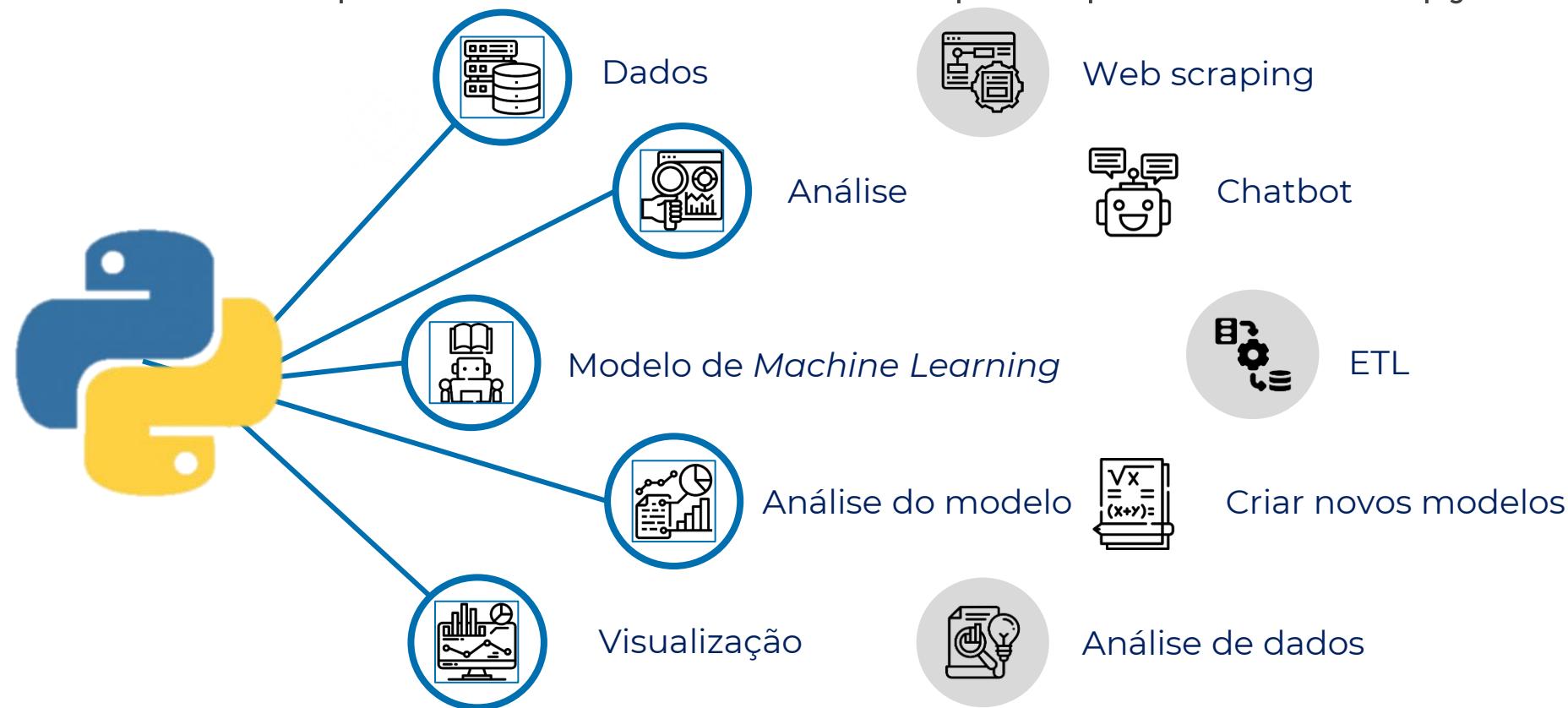
## Módulo 2 – Um framework para Ciência de Dados

O mercado de trabalho para o cientista de dados e para quem conhece python é muito amplo.



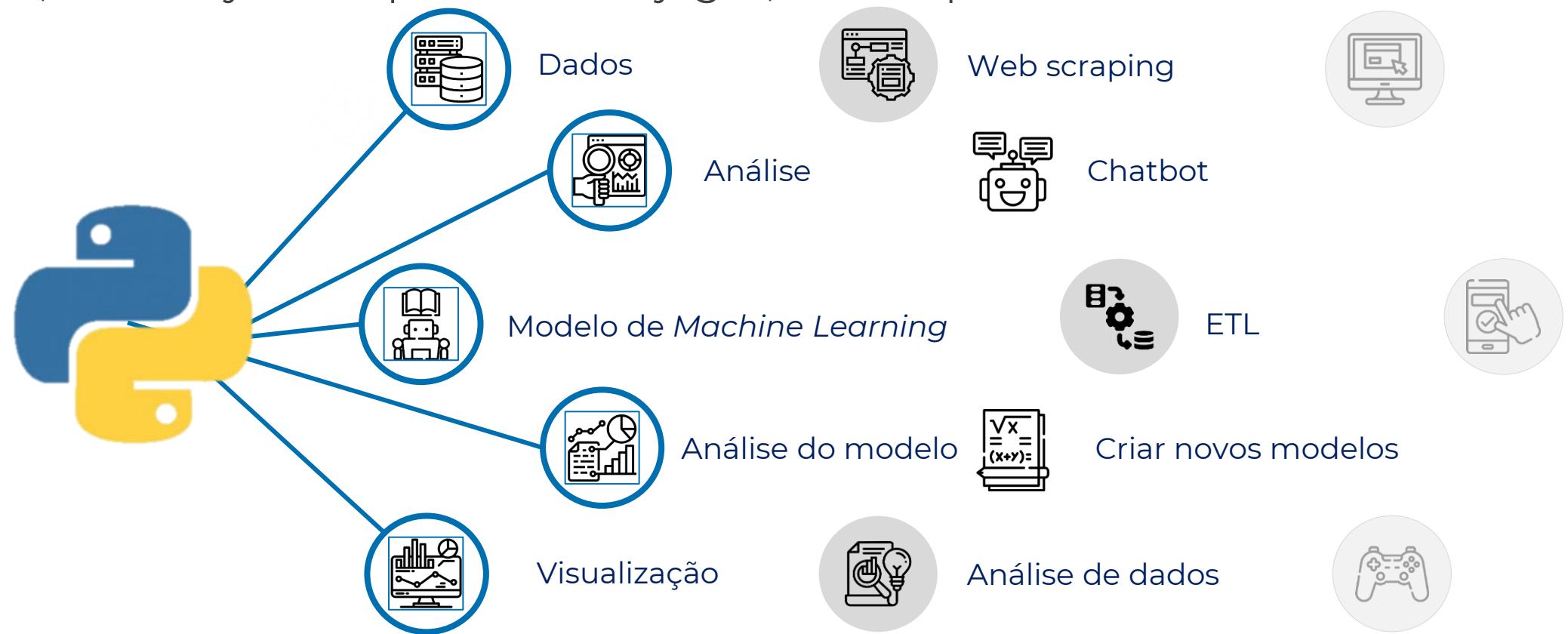
## Módulo 2 – Um framework para Ciência de Dados

O mercado de trabalho para o cientista de dados e para quem conhece python é muito amplo.



## Módulo 2 – Um framework para Ciência de Dados

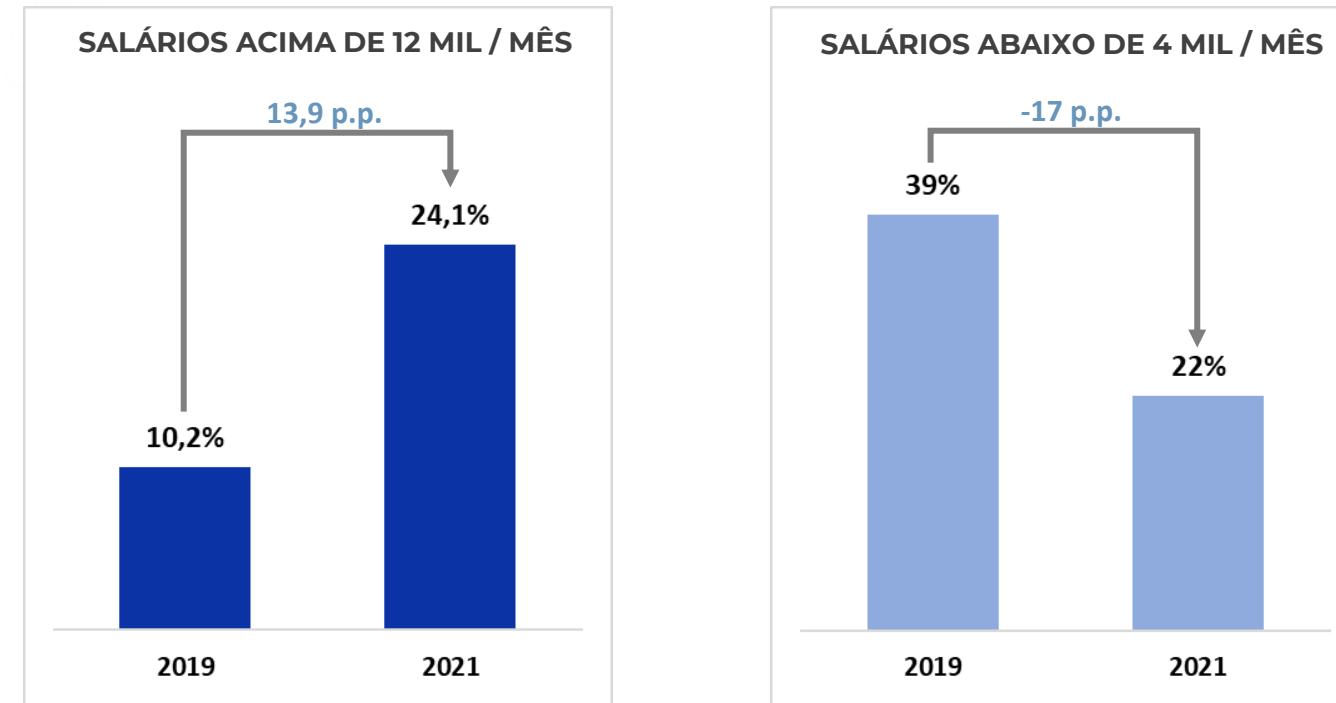
Além disso, com o Python é possível criar jogos, sites e aplicativos.



## Módulo 2 – Um framework para Ciência de Dados

O salário de um cientista de dados teve um aumento de 40%. Além disso, houve um aumento da remunerações de 12 mil e uma redução para remunerações de 4 mil.

**40%**  
de aumento na remuneração  
entre 2019 e 2021



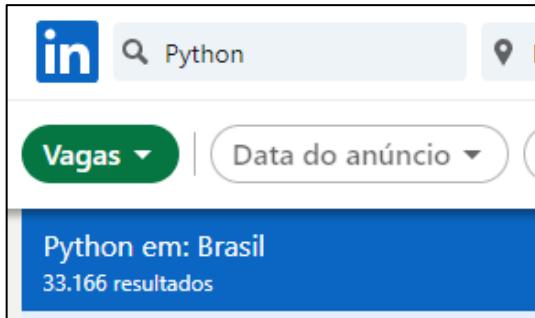
## Módulo 2 – Um framework para Ciência de Dados

A procura por cientista de dados com conhecimento em Python aumentou, pois é uma linguagem que está crescendo e as empresas estão procurando.

**Média salarial mensal de**

**7.477**

**+ de 33 mil vagas**



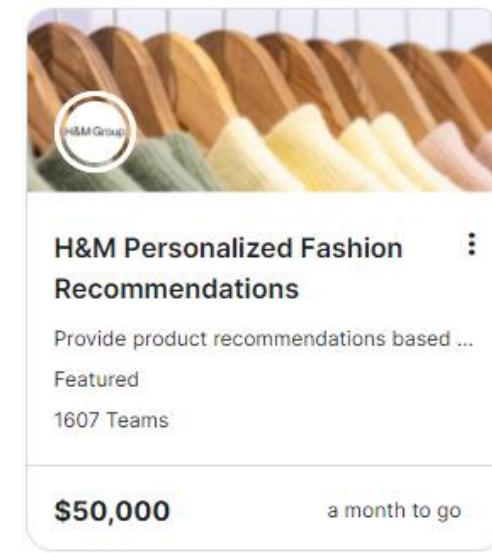
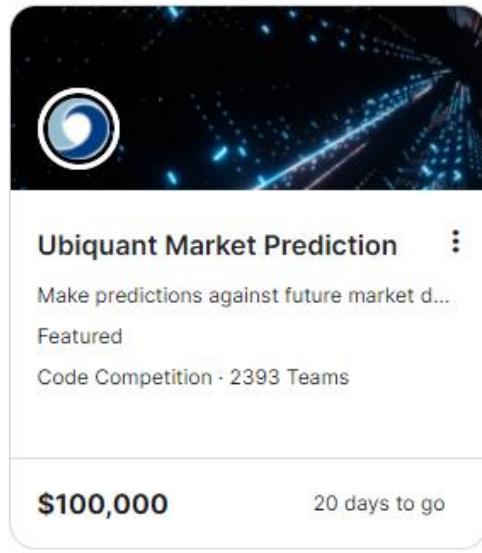
Empresas que são referência no mercado:



## Módulo 2 – Um framework para Ciência de Dados

Além de todas essas áreas, existem competições que dão prêmios em dinheiro.

### Desafios de Data Science com remuneração em dinheiro



Competições disponíveis em [www.kaggle.com/competitions](https://www.kaggle.com/competitions) na data de 29/03/2022

## Módulo 2 – Um framework para Ciência de Dados

A Netflix deu 1 milhão para quem conseguisse melhorar o seu algoritmo, ou seja, a ciência de dados é muito valorizada e as empresas reconhecem a importância disso, porque elas sabem que isso vai trazer novos clientes.



[HOME](#) > [GIZMODO](#) >

Nerds superam sistema de recomendação da Netflix e podem ganhar US\$ 1 milhão

**Netflix anuncia vencedores do concurso de US\$ 1 milhão**

22 set 2009 16h03

## MÓDULO 3

# Python Básico

## Módulo 3 – Explicando esse módulo

O módulo 3 tem como objetivo explicar conceitos de Python básico, que serão utilizados para fazer os projetos de Ciência de dados.



## Módulo 3 – Instalando o Python no Windows

O Python pode ser instalado pelo link <https://www.python.org/downloads/> mas, para esse curso, vamos usar o Anaconda que é o bloco de informações do Python.

Ao instalar o Anaconda será instalado o Jupyter junto, que é onde vamos escrever o código, executar e ele vai nos dizer o que está acontecendo.

No lugar do Jupyter poderia ser utilizado o Google Colab, porém nesse curso vamos utilizar o Jupyter porque ele nos obriga a instalar os programas no computador e não precisa de internet.

# Módulo 3 – Instalando o Python no Windows

Para fazer a instalação, entre no link abaixo e escolha o sistema operacional do seu computador:

Link: <https://www.anaconda.com/products/distribution#Downloads>

Anaconda Installers

Windows	MacOS	Linux
<a href="#">Python 3.9</a> <a href="#">64-Bit Graphical Installer (594 MB)</a> <a href="#">32-Bit Graphical Installer (488 MB)</a>	<a href="#">Python 3.9</a> <a href="#">64-Bit Graphical Installer (591 MB)</a> <a href="#">64-Bit Command Line Installer (584 MB)</a> <a href="#">64-Bit (M1) Graphical Installer (316 MB)</a> <a href="#">64-Bit (M1) Command Line Installer (305 MB)</a>	<a href="#">Python 3.9</a> <a href="#">64-Bit (x86) Installer (659 MB)</a> <a href="#">64-Bit (Power8 and Power9) Installer (367 MB)</a> <a href="#">64-Bit (AWS Graviton2 / ARM64) Installer (568 MB)</a> <a href="#">64-bit (Linux on IBM Z &amp; LinuxONE) Installer (280 MB)</a>

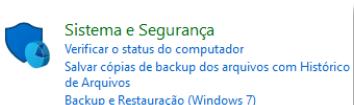
# Módulo 3 – Instalando o Python no Windows

Agora vamos conferir pra ver se está tudo funcionando direitinho:

1. Pesquise por **Painel de Controle** na barra de pesquisa



2. Selecione **Sistema e Segurança**



3. Selecione **Sistema**



4. Ao lado de tipo de sistema está escrito o sistema operacional

Tipo de sistema	Sistema operacional de 64 bits, processador baseado em x64
-----------------	--

# Módulo 3 – Instalando o Python no Windows

Para baixar, clique no **processador** para o seu dispositivo e o download começará diretamente.



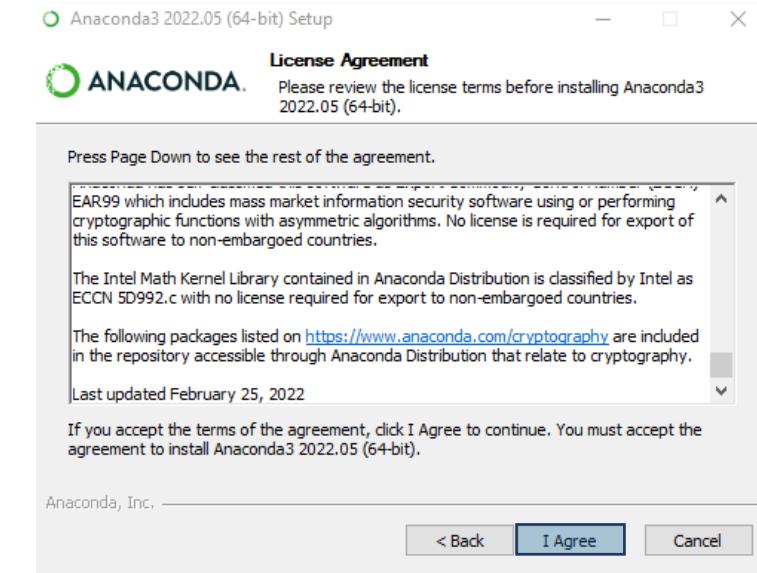
# Módulo 3 – Instalando o Python no Windows

Depois que baixar vamos executá-lo no computador.

- 1 Abra o instalador do Anaconda



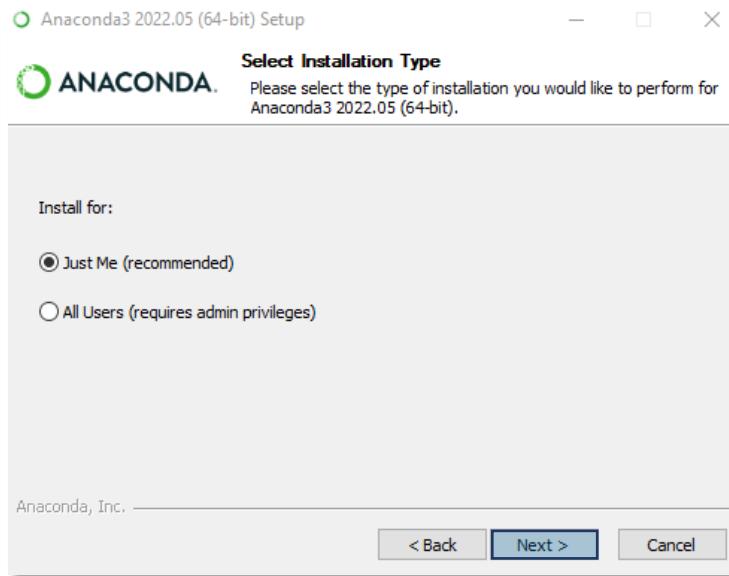
- 2 Aceite os termos de uso



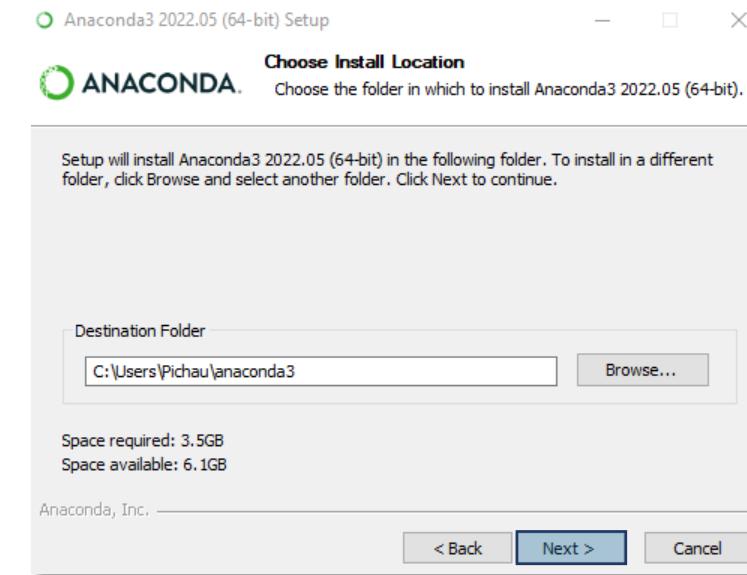
# Módulo 3 – Instalando o Python no Windows

Depois que baixar vamos executá-lo no computador.

- 3 Selecione o tipo de instalação



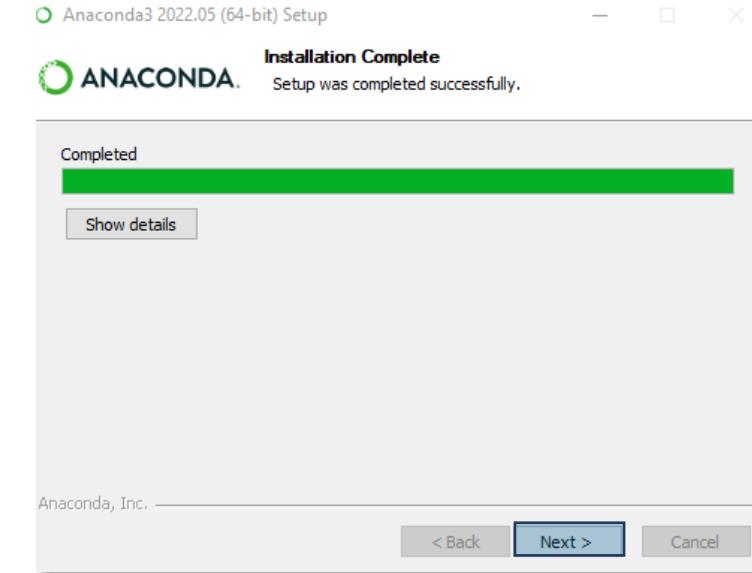
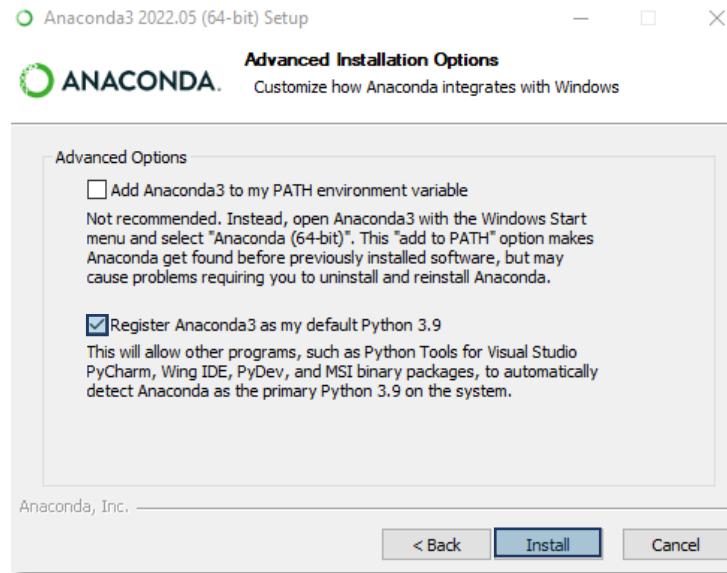
- 4 Escolha o local de destino (de preferência o que ele sugerir) e clique em **next**



# Módulo 3 – Instalando o Python no Windows

Depois que baixar vamos executá-lo no computador.

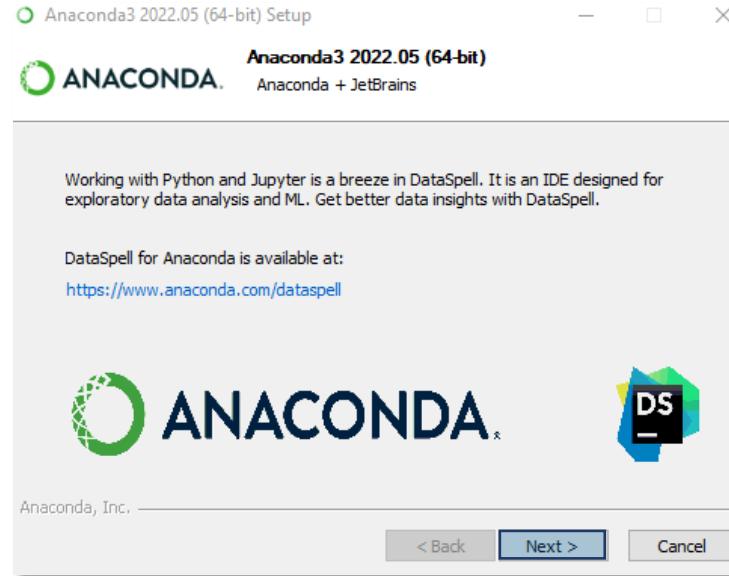
- 5 Defina o Anaconda como o seu Python padrão e siga com a instalação clicando em **install**.
- 6 Ao fim da instalação, clique em **next**.



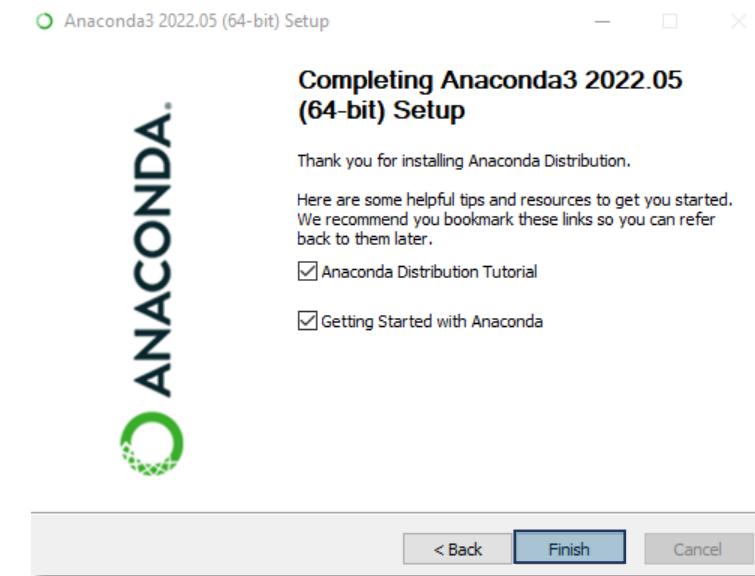
# Módulo 3 – Instalando o Python no Windows

Depois que baixar vamos executá-lo no computador.

- 7 Clique em **next**.



- 8 Ao fim da instalação, clique em **finish**.



## Módulo 3 – Instalando o Python no Windows

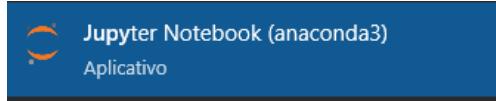
Ao baixar o Anaconda estamos instalando todas essas ferramentas juntas e se precisarmos baixar outra será pelo próprio Python.



# Módulo 3 – Instalando o Python no Windows

Agora vamos conferir pra ver se está tudo funcionando direitinho:

- 1 Na barra de pesquisa, digite Jupyter;



- 2 Clique no Jupyter Notebook;



- 3 Vai abrir uma tela preta e carregar algumas informações.

**Não feche** a tela preta;

- 4 O Jupyter irá abrir no seu navegador principal.



## Módulo 3 – Problemas na Instalação - Resolvido

Agora vamos resolver os 3 principais problemas que acontecem para abrir o Jupyter:

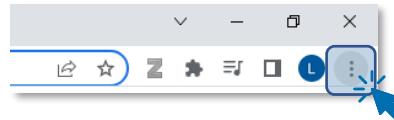
- O Chrome não abre quando eu abro o Jupyter;
- O Jupyter abre na pasta Documentos e eu queria que ele abrisse em outra Pasta;
- Eu instalo o Anaconda e não aparece o atalho do Jupyter

# Módulo 3 – Problemas na Instalação - Resolvido

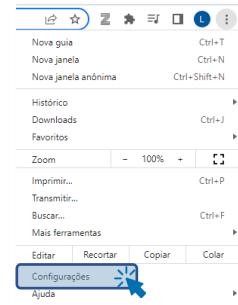
## “O meu Jupyter não abre no Google Chrome”

Normalmente o Jupyter abre no seu navegador padrão. Se o Chrome não for o seu navegador padrão, ele não irá abrir no Chrome. Para definir o Chrome como padrão, siga os passos abaixo :

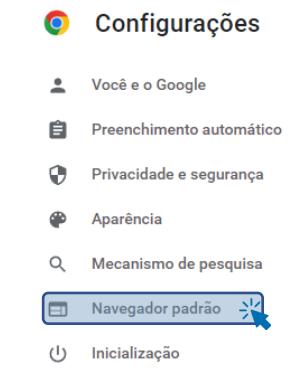
- 1 Clique nos três pontinhos



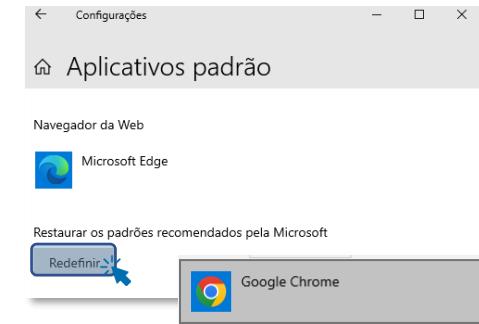
- 2 Clique em configurações



- 3 Clique em navegador padrão



- 4 Clique em redefinir e escolha o Chrome

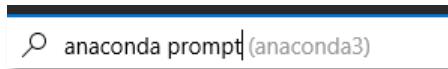


# Módulo 3 – Problemas na Instalação - Resolvido

## “O meu Jupyter não abre no Google Chrome”

Se mesmo com o procedimento anterior, o Jupyter não abrir no Chrome, então você deve seguir os passos:

- 1 Abra o Prompt Anaconda



- 2 Rode o comando:

```
jupyter notebook --generate-config
```

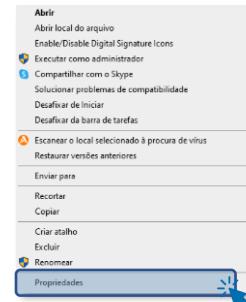
- 4 Agora procure o chrome no buscador do Windows, clique com o botão direito do mouse e selecione “Abrir o local do arquivo”.



- 3 No bloco de notas abra o arquivo que está no caminho do prompt anaconda

C:\Users\Usuario\jupyter e abra o arquivo jupyter\_notebook\_config.py

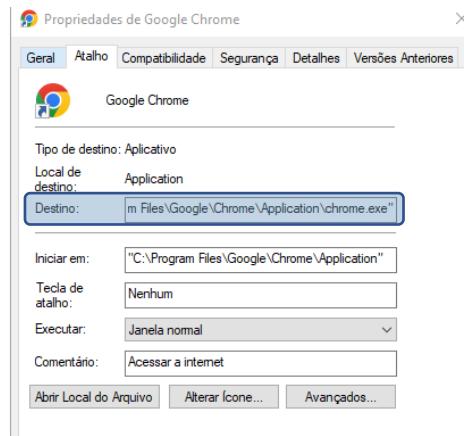
- 5 Clique com o botão direito do mouse no atalho do chrome e clique em **propriedades**.



# Módulo 3 – Problemas na Instalação - Resolvido

## “O meu Jupyter não abre no Google Chrome”

- 6 No campo Destino está exatamente o caminho do local onde está instalado o seu Google Chrome. Copie o caminho todo.



- 7 Com o arquivo aberto no bloco de notas, procure por “NotebookApp(JupyterApp) configuration” e altere a linha:

```
# c.NotebookApp.browser = " por  
c.NotebookApp.browser = 'caminho %s'
```

Coloque entre aspas simples **o local** onde está instalado o seu Google Chrome, retire o #, adicione um espaço e %s no final. Salve o arquivo e abra o Jupyter.

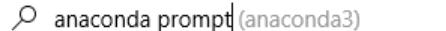
**Atenção:** Lembre-se de trocar as “\” do caminho por “/”.

Ex: c.NotebookApp.browser = 'C:/Program Files (x86)/Google/Chrome/Application/chrome.exe %s'

# Módulo 3 – Problemas na Instalação - Resolvido

“O Jupyter abre na pasta Documentos e eu queria que ele abrisse em outra Pasta”

- 1 Abra o Prompt Anaconda



- 2 Rode o comando:

```
jupyter notebook --generate-config
```

- 4 Com o arquivo aberto no bloco de notas, procure por “NotebookApp(JupyterApp) configuration” altere a linha:

```
#c.NotebookApp.notebook_dir = "  
c.NotebookApp.notebook_dir = r'Local'
```

Exemplo: c.NotebookApp.notebook\_dir = r'E:'

(Você deve colocar entre aspas simples o caminho da pasta onde você quer que o Jupyter abra. No caso do exemplo, o jupyter abrirá no disco E)

- 3 No bloco de notas abra o arquivo que está no caminho do prompt anaconda

C:\Users\Usuario\jupyter e abra o arquivo jupyter\_notebook\_config.py

- 5 Salve o arquivo

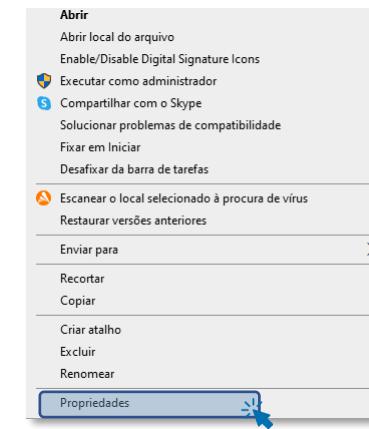
# Módulo 3 – Problemas na Instalação - Resolvido

“O Jupyter abre na pasta Documentos e eu queria que ele abrisse em outra Pasta”

- 6 Salve o arquivo e procure o Jupyter no buscador do Windows, clique com o botão direito do mouse e selecione “Abrir o local do arquivo”.



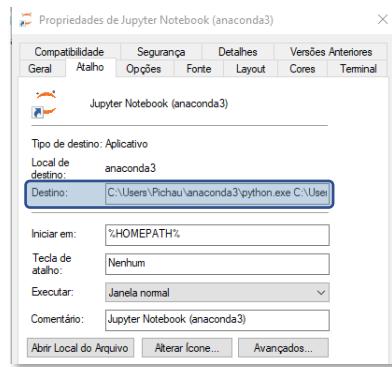
- 7 Clique com o botão direito do mouse no atalho do Jupyter e clique em **propriedades**.



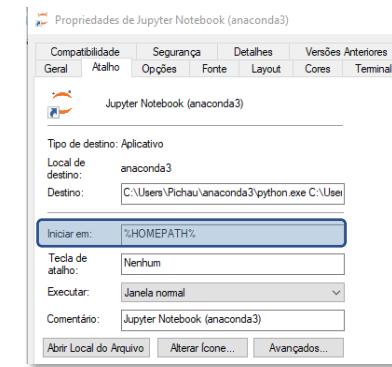
# Módulo 3 – Problemas na Instalação - Resolvido

“O Jupyter abre na pasta Documentos e eu queria que ele abrisse em outra Pasta”

- 9 No campo "Destino" delete "%USERPROFILE%".



- 10 No campo "Iniciar em" escreva o caminho do diretório onde deseja iniciar.



- 11 Feche e abra novamente o Jupyter e tudo deve funcionar normalmente.

## Módulo 3 – Problemas na Instalação - Resolvido

“Eu instalo o anaconda e não aparece o atalho do jupyter”

Geralmente o problema aqui está relacionado a algum problema na versão do anaconda/SO/arquitetura do seu computador.

- 1 Desintale o Anaconda e **REINICIE O COMPUTADOR.**

Painel de Controle > Desinstalar

(É sério, tem que reiniciar o computador)

- 2 Baixe novamente o Anaconda e instale ele, mas na opção de escolher os usuários escolha a opção "**All Users**" ao invés de "Just me"

- 3 Caso não resolva desintale o Anaconda e **REINICIE O COMPUTADOR.**

(É sério, tem que reiniciar o computador)

- 4 Depois entre nesse link que tem todas as versões que saíram até hoje:  
<https://repo.anaconda.com/archive/>

Geralmente recomendo baixar essa aqui: Anaconda3-2019.10-Windows-x86\_64.exe mas se não funcionar continue testando.

## Módulo 3 – Problemas na Instalação - Resolvido

“Eu instalo o anaconda e não aparece o atalho do jupyter”

- 5 Sempre que for testar uma nova versão é importante que você **desinstale** o Anaconda antes, **reinicie** o computador (sim, toda vez que desinstalar você precisa reiniciar) e instale uma nova versão.
- 6 Alguma versão (sobre tudo das mais recentes) deve funcionar normalmente no computador.

## Módulo 3 –Mac, Linux e Google Colab

O procedimento para fazer a instalação no Mac e no Linux é o mesmo de fazer a instalação no Windows. Para fazer a instalação entre no link abaixo e escolha o sistema operacional do seu computador:

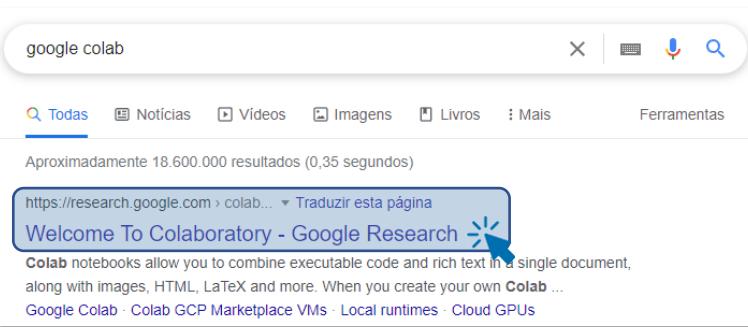
Link: <https://www.anaconda.com/products/distribution#Downloads>



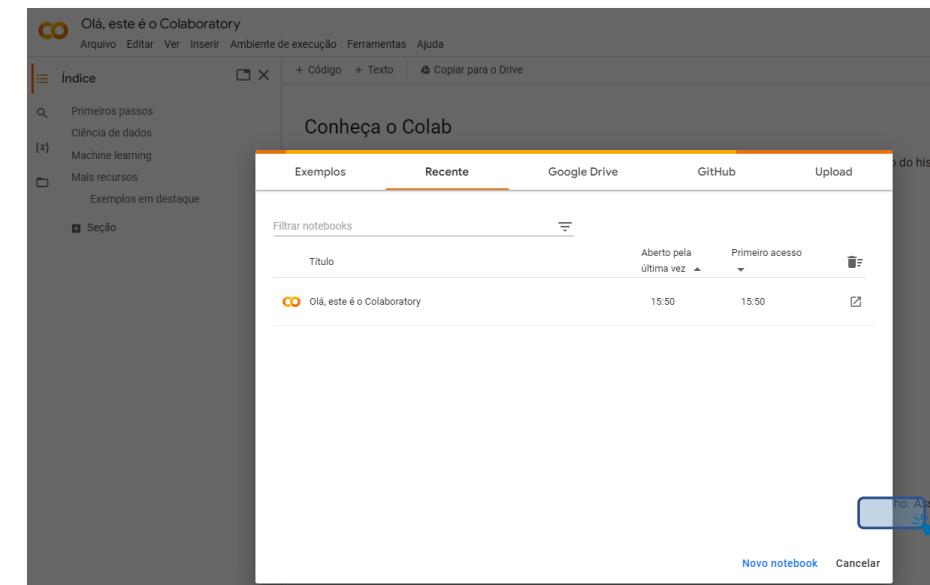
# Módulo 3 –Mac, Linux e Google Colab

Uma alternativa para quem não quer baixar nenhum programa é o **Google Colab**. Ele é bem simples de usar e só precisa de **internet** e uma conta no **gmail**.

- 1 Procure por **google colab** e clique no primeiro site



- 2 Clique em cancelar para ir para a página inicial



# Módulo 3 –Mac, Linux e Google Colab

Para criar um arquivo basta seguir o passo a passo abaixo:

1 Clique em **arquivo**



2 Clique em **novo notebook**.

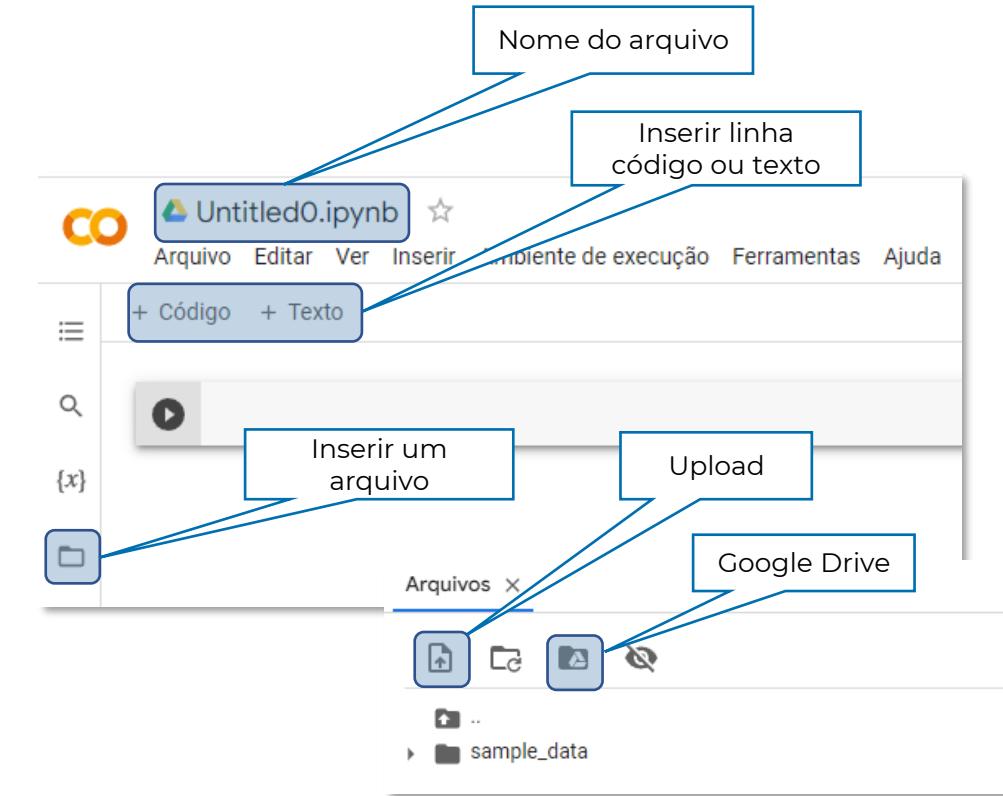


## Módulo 3 –Mac, Linux e Google Colab

Após a criação, você terá uma tela como a imagem a direita.

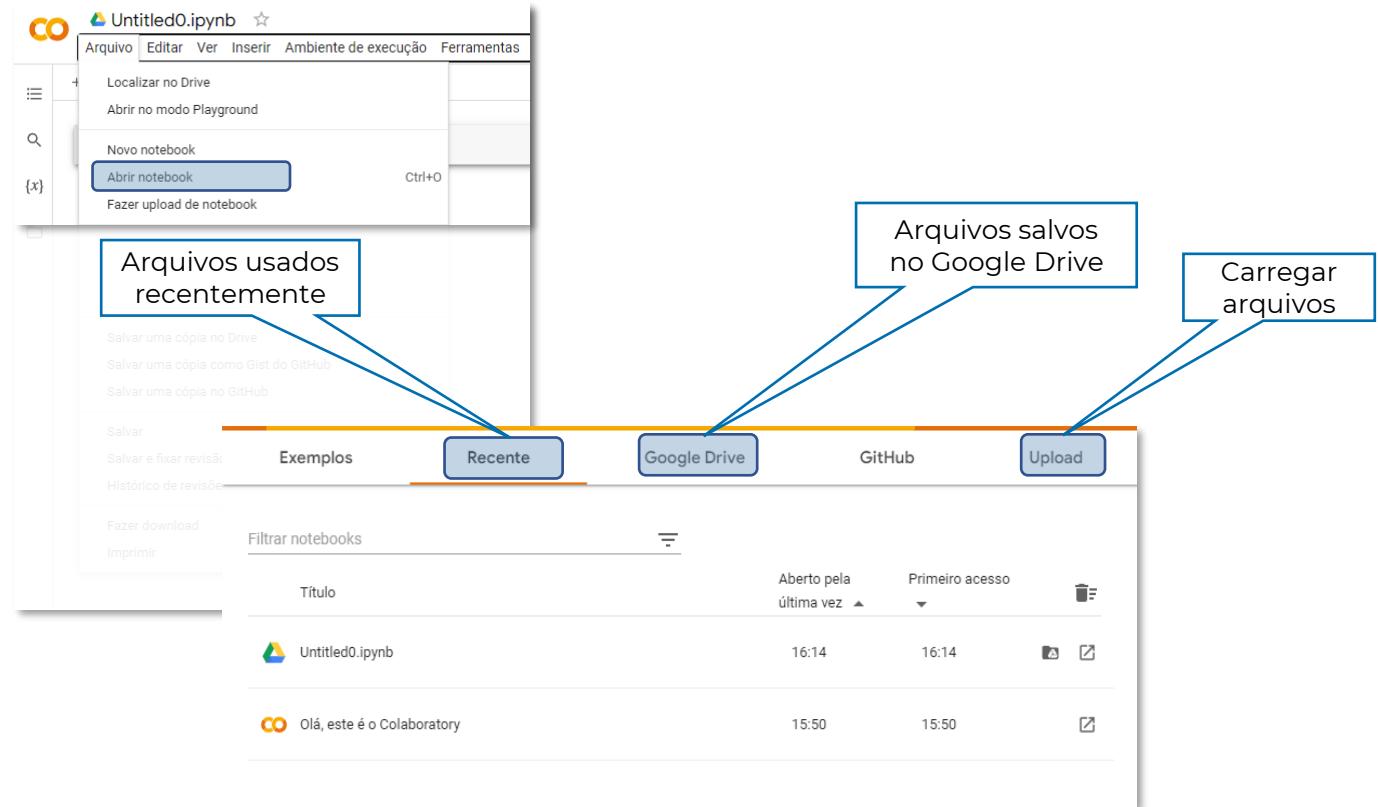
Para alterar o **nome do arquivo**, basta clicar no campo indicado e alterar.

O formato **.ipynb** indica que se trata de um formato de notebook. Ele poderá ser lido tanto no Google Colab quanto no Jupyter Notebook.



# Módulo 3 –Mac, Linux e Google Colab

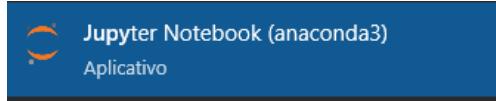
Uma outra opção para **abrir um arquivo** é clicar em **arquivo** e **abrir notebook**.



# Módulo 3 – Criando seu Primeiro Programa

Agora vamos conferir pra ver se está tudo funcionando direitinho:

- 1 Na barra de pesquisa, digite Jupyter;



- 2 Clique no Jupyter Notebook;



- 3 Vai abrir uma tela preta e carregar algumas informações.

**Não feche** a tela preta;

- 4 O Jupyter irá abrir no seu navegador principal.



# Módulo 3 – Criando seu Primeiro Programa



Quando o Jupyter rodar no seu computador, você verá uma tela como a tela ao lado.

Essa tela é basicamente um espelho das pastas do **seu computador**.

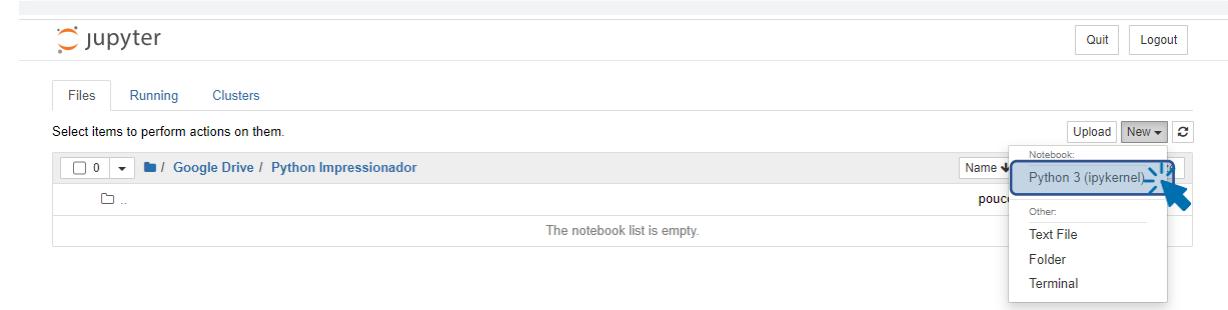
Antes de começar, vamos escolher uma pasta para salvar os arquivos.

No nosso caso, vamos salvar na Pasta Google Drive>Python Impressionador

## Módulo 3 – Criando seu Primeiro Programa

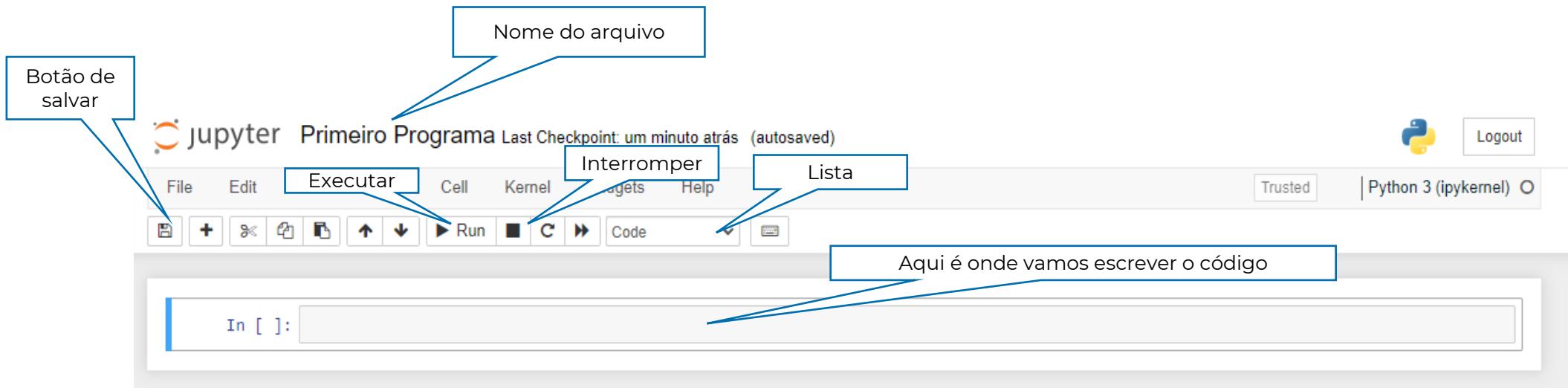
Para criar um novo arquivo do Jupyter do zero, basta clicar em NEW> Python 3.

Uma outra opção é carregar o arquivo em upload.



# Módulo 3 – Criando seu Primeiro Programa

Vamos conhecer agora os **principais ícones** da barra de tarefas.



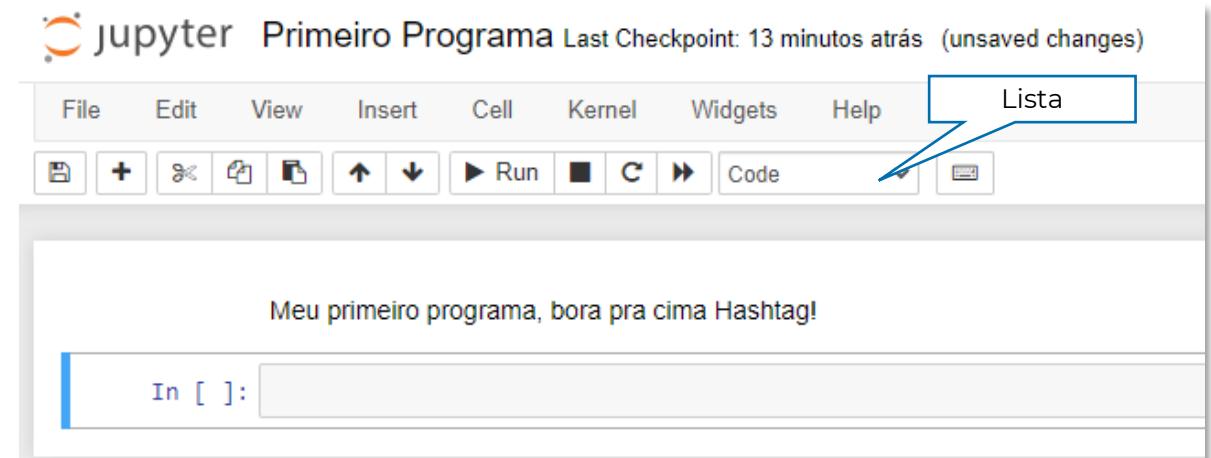
## Módulo 3 – Criando seu Primeiro Programa

O Jupyter diferencia um código de um texto pelo item escolhido na lista.

Por exemplo, o **Markdown** é utilizado para escrever **texto**.

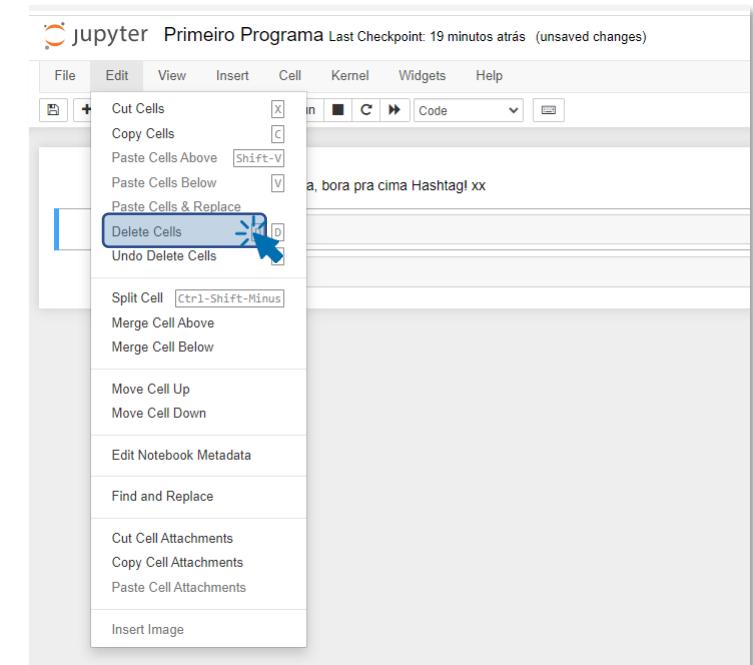
Para executar a célula selecionada, basta clicar no ícone **RUN** ou usar o atalho **CTRL + ENTER**.

Outro atalho é usar **SHIFT + ENTER** quando você quiser executar uma célula e inserir uma nova célula automaticamente.



# Módulo 3 – Criando seu Primeiro Programa

Para deletar células, basta selecionar a célula, clicar em **Edit** e **deletar**.

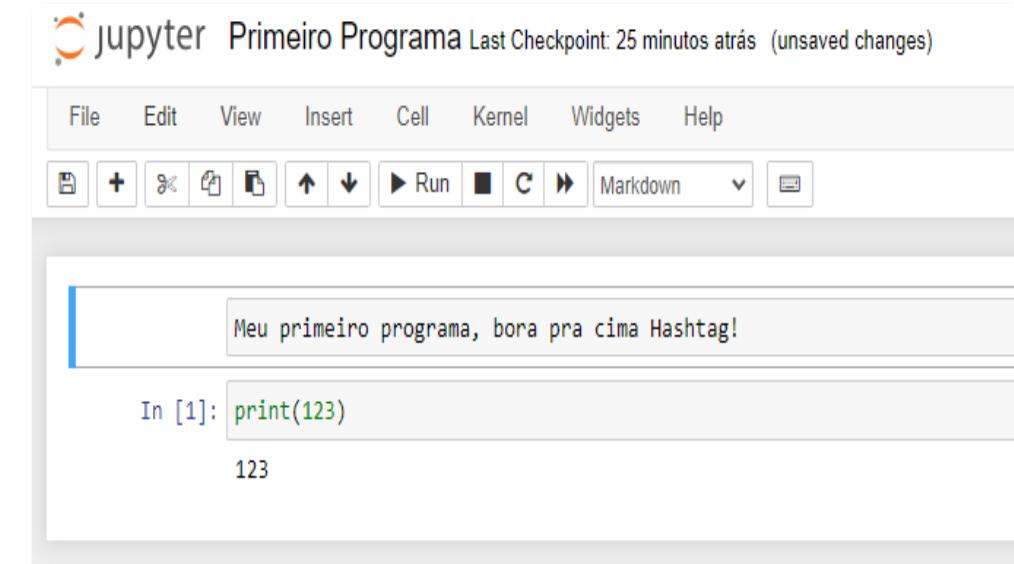


## Módulo 3 – Criando seu Primeiro Programa

Agora vamos fazer o nosso primeiro código com a função **print**, que vai nos retornar “ o valor que está dentro do parênteses”.

Estrutura: **print(número)**

Então, a gente escreve o texto e aperta **CTRL + ENTER** para executar.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Primeiro Programa Last Checkpoint: 25 minutos atrás (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with various icons. A code cell contains the text "Meu primeiro programa, bora pra cima Hashtag!". In the next cell, the input "In [1]: print(123)" is shown, followed by the output "123".

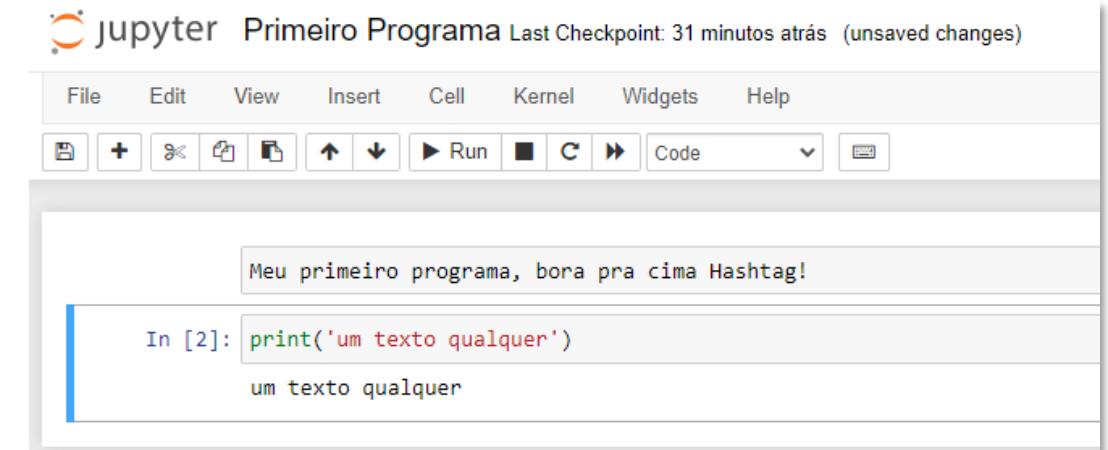
## Módulo 3 – Criando seu Primeiro Programa

A função print pode ser usada tanto para número como para textos, mas para texto é muito importante lembrar de utilizar as **aspas!**

O Python vai entender tanto as aspas duplas ("") quanto as simples (""). No curso, vamos padronizar escrever com as simples.

Estrutura: **print("texto")**

Então, a gente escreve o texto e aperta **CTRL + ENTER** para executar.



The screenshot shows a Jupyter Notebook window titled "jupyter Primeiro Programa". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with various icons. A code cell in the notebook contains the Python command `print('um texto qualquer')`. The output of this command, "um texto qualquer", is displayed directly below the cell. The status bar at the bottom indicates "Last Checkpoint: 31 minutos atrás (unsaved changes)".

## Módulo 3 – Variáveis

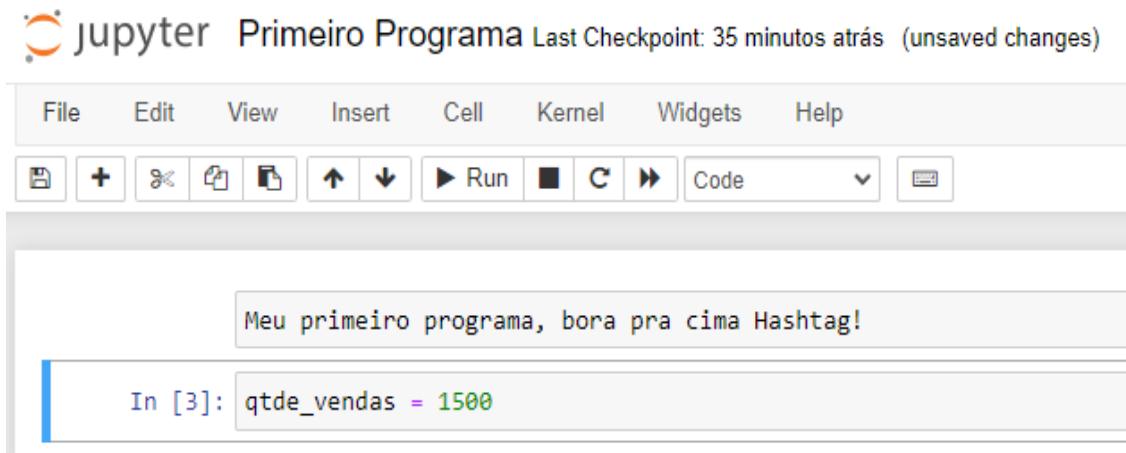
Variável é o nome atribuído para algum valor específico.

Estrutura: **nome\_variável = valor**

Por exemplo: as vendas de uma empresa no mês de janeiro foram de 1500.

Então, nesse caso vamos criar uma variável para a quantidade de vendas.

**qtde\_vendas = 1500**



jupyter Primeiro Programa Last Checkpoint: 35 minutos atrás (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

In + × ↻ ↺ ↺ Run C ► Code

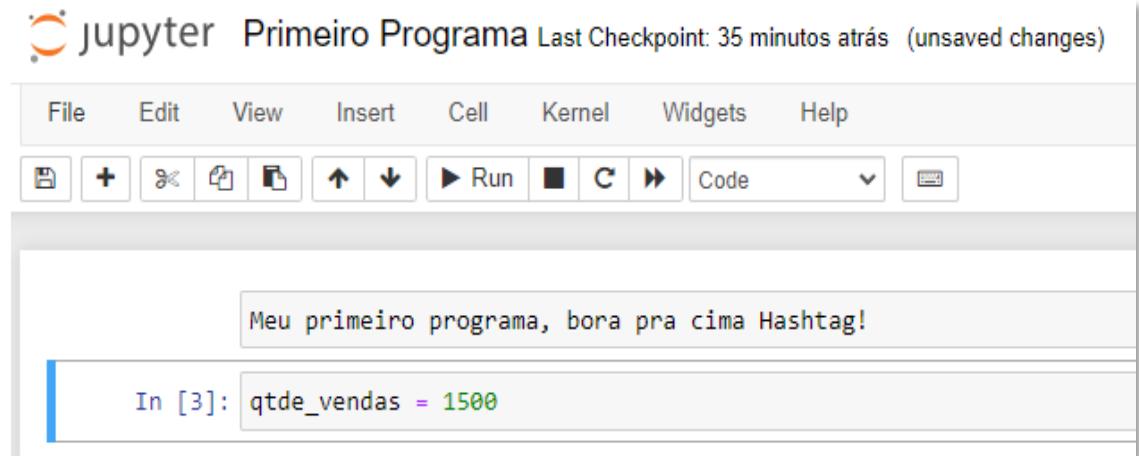
Meu primeiro programa, bora pra cima Hashtag!

In [3]: qtde\_vendas = 1500

## Módulo 3 – Variáveis

Um ponto importante é que **não pode ter espaço** no nome da variável. Você pode usar o “\_” ou pode escrever tudo junto.

No curso vamos padronizar com o uso do “\_” para variáveis com mais de uma palavra.

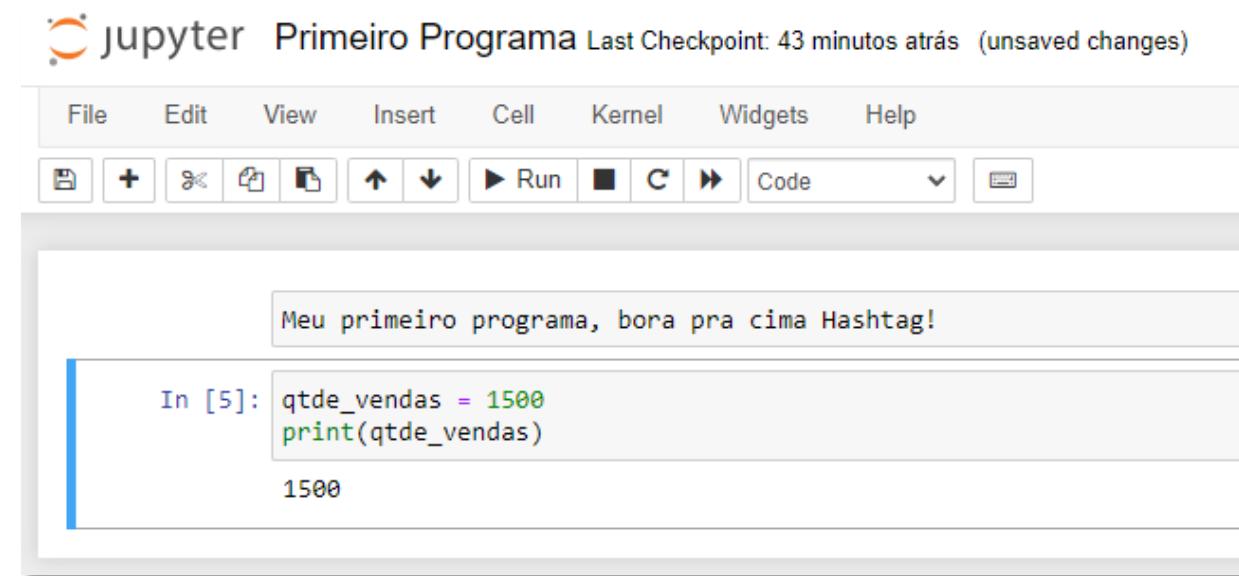


The screenshot shows a Jupyter Notebook interface. The title bar says "jupyter Primeiro Programa Last Checkpoint: 35 minutos atrás (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. A code cell contains the following text:  
Meu primeiro programa, bora pra cima Hashtag!  
In [3]: qtde\_vendas = 1500

## Módulo 3 – Variáveis

Quando estamos tratando de variáveis, o termo igual (=) não significa igual e sim, **recebe**.

Na programação, sempre o que está à esquerda do “=” **recebe** o valor do que está escrito à direita do “=”

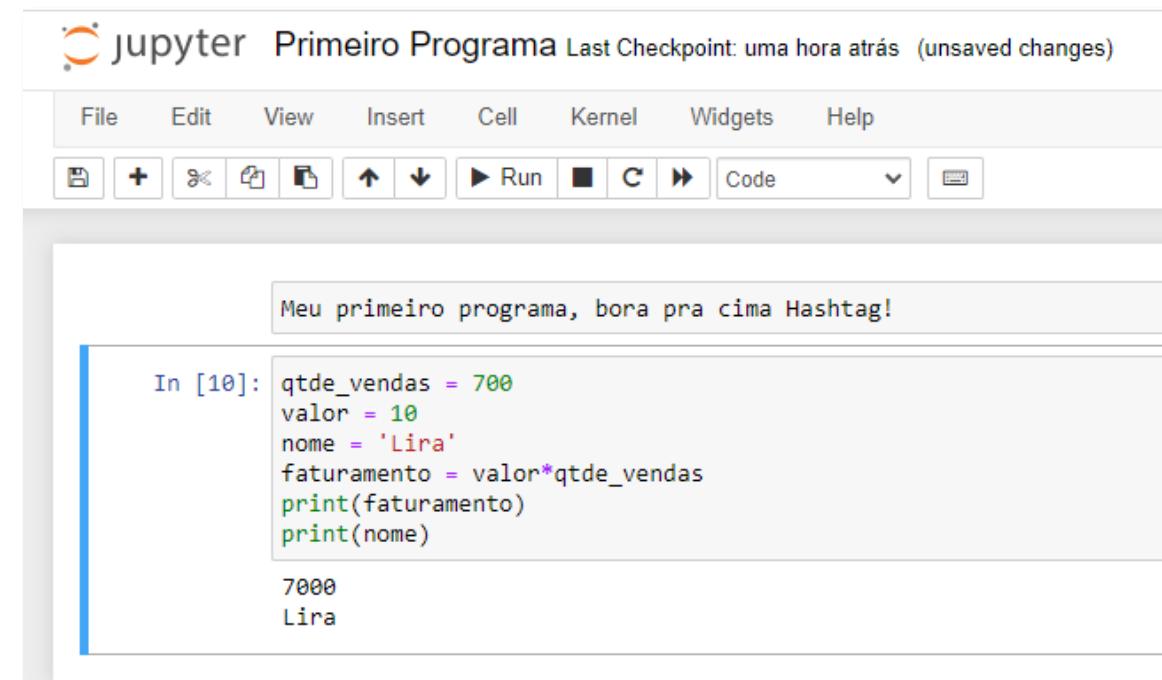


The screenshot shows a Jupyter Notebook interface. The title bar says "jupyter Primeiro Programa Last Checkpoint: 43 minutos atrás (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with various icons. The main area contains a code cell with the text "Meu primeiro programa, bora pra cima Hashtag!". Below it is another code cell labeled "In [5]:" containing the Python code "qtde\_vendas = 1500" and "print(qtde\_vendas)". The output of this cell is "1500".

## Módulo 3 – Variáveis

Uma vantagem do uso da variável é que se fosse necessário trocar o valor dela, não precisaria alterar o código todo.

No Python a gente não precisa dizer o tipo da variável, pois ele já consegue entender de forma automática.



The screenshot shows a Jupyter Notebook interface titled "jupyter Primeiro Programa". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help, along with various icons for file operations and cell execution. The main area has a header "Meu primeiro programa, bora pra cima Hashtag!". A code cell labeled "In [10]" contains the following Python code:

```
qtde_vendas = 700
valor = 10
nome = 'Lira'
faturamento = valor*qtde_vendas
print(faturamento)
print(nome)
```

The output of the code cell is:

```
7000
Lira
```

## Módulo 3 – Tipos de Variáveis

Até agora, atribuímos valores para as variáveis sem dizer qual é o tipo delas, isso porque o Python já entende o tipo da variável. Os tipos mais básicos de variáveis são:

- Int → Números inteiros.
- String → Texto. Lembrando que sempre vão estar entre aspas.
- Float → Números com casas decimais (ponto flutuante). Lembrando que no Python a casa decimal é representada pelo ponto.
- Bool ou boolean → Variáveis que só possuem 2 valores possíveis true (verdadeiro) ou false (falso).

## Módulo 3 – Tipos de Variáveis

Para saber qual o tipo de uma variável, usamos a função **Type** como nos exemplos ao lado.

Estrutura: **type(variável)**

No primeiro exemplo, faturamento é uma variável do tipo inteiro. Já no segundo, como possui casa decimal, temos o tipo float.

No terceiro a variável está entre aspas, então é um texto. Por último, a variável recebe um valor de True, então é do tipo bool.

```
In [1]: faturamento = 1000  
type(faturamento)
```

Out[1]: int

```
In [2]: faturamento = 1000.00  
type(faturamento)
```

Out[2]: float

```
In [3]: faturamento = '1.000'  
type(faturamento)
```

Out[3]: str

```
In [4]: ganha_bonus = True  
type(ganha_bonus)
```

Out[4]: bool

## Módulo 3 – Tipos de Variáveis

Cuidados que você precisa ter com os nomes das variáveis:

- **Não** escrever variáveis com caracteres especiais;
- Dica: use nomes em português;
- **Não** usar nomes reservados, por exemplo nome de uma função nativa do Python.

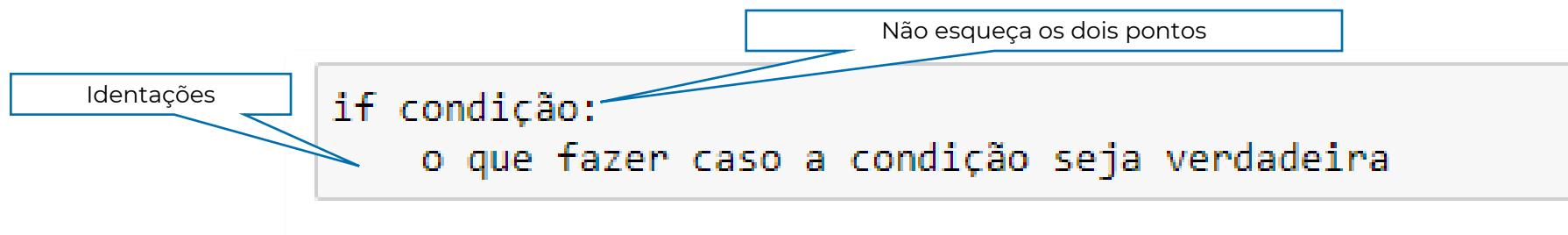
Lista com os principais nomes para você **evitar** quando for colocar nome nas variáveis:

<https://drive.google.com/file/d/15qWxRRoC5qiGnmORacbQemTsKLcAp5fa/view?usp=sharing>

## Módulo 3 – Tipos de Variáveis

Quase todo programa no Python utiliza a função **if**, uma vez que provavelmente você vai ter que fazer alguma verificação ou tratar alguma condição em algum momento.

E como funciona o **if**?



As identações indicam para o Python quais ações estão dentro daquele **if**.

## Módulo 3 – Estrutura do if – Condições no Python

Agora vamos olhar um exemplo prático: você quer exibir uma mensagem caso a quantidade vendida no mês tenha ultrapassado a meta.

Então vamos atribuir valores às variáveis:

**meta** = 50000

**qtde\_vendida** = 65300

### Exemplo Real (informações 100% hipotéticas e inventadas):

Digamos que você trabalha na Amazon (que tem centenas de milhares, se não milhões de produtos) e está analisando o resultado de vendas dos produtos.

Você precisa criar um programa que vai analisar o resultado de vendas dos produtos da Amazon em um mês. Para simplificar vamos pensar em um único produto: um Iphone.

Meta de Vendas do Iphone = 50.000 unidades

Quantidade vendida no Mês = 65.300 unidades

O seu programa deve avisar (usaremos o print por enquanto) caso o produto tenha batido a meta do mês. Então devemos fazer:

- Caso o produto tenha batido a meta, devemos exibir a mensagem: "Batemos a meta de vendas de Iphone, vendemos {} unidades"
- Se ele não bateu a meta do mês, o seu programa não deve fazer nada

```
meta = 50000
qtde_vendida = 65300

if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {}'.format(qtde_vendida))

Batemos a meta de vendas de Iphone, vendemos 65300 unidades
```

## Módulo 3 – Estrutura do if – Condições no Python

O segundo passo é construir a estrutura do **if**:

**meta** = 50000

**qtde\_vendida** = 65300

**If** **qtde\_vendida**>**meta**:

**print**('Batemos a meta de Iphone, vendemos {} unidades'.**format**(**qtde\_vendida**))

```
meta = 50000
qtde_vendida = 653000

if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))

Batemos a meta de vendas de Iphone, vendemos 653000 unidades
```

## Módulo 3 – Estrutura do if – Condições no Python

Então basicamente estamos avaliando se a quantidade vendida foi maior que a meta. Se essa condição é verdadeira o Python vai imprimir a frase ‘Batemos a meta de vendas de Iphone, vendemos {} unidades’.

Nota: a função **format** é utilizada para substituir uma variável nas chaves {}.

```
meta = 50000
qtde_vendida = 653000

if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))

Batemos a meta de vendas de Iphone, vendemos 653000 unidades
```

## Módulo 3 – Estrutura do if – Condições no Python

**Atenção:** Tudo o que estiver dentro do *tab* ele só vai executar se a condição for verdadeira.

Ambos com *tab*

```
In [2]: meta = 50000
qtde_vendida = 653000

if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))
    print('Fim do programa')

Batemos a meta de vendas de Iphone, vendemos 653000 unidades
Fim do programa
```

Porém, se não estiver dentro do *tab* e a condição for falsa ele vai executar.

Sem *tab*

Quantidade vendida menor que a meta

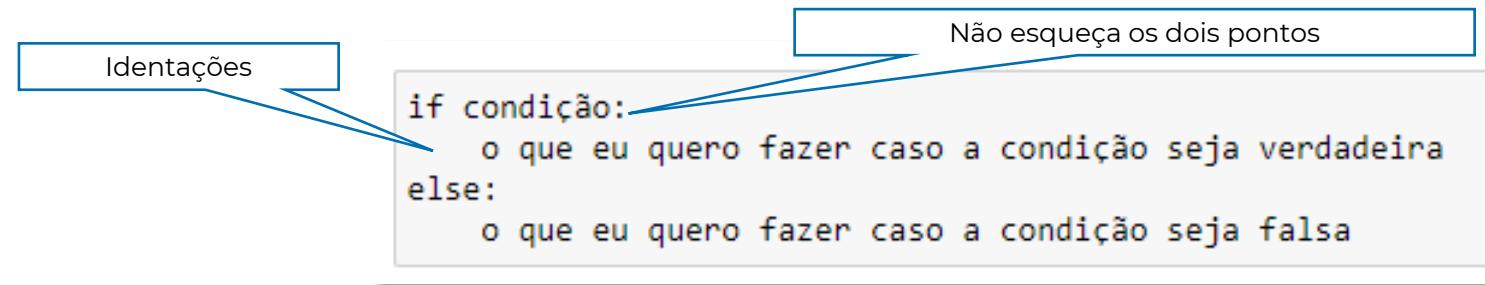
```
In [4]: meta = 50000
qtde_vendida = 3000

if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))
print('Fim do programa')

Fim do programa
```

## Módulo 3 – Estrutura do if – Condições no Python

Quando usamos o **if**, nem sempre queremos apenas analisar o caso verdadeiro, em boa parte das vezes queremos fazer alguma coisa caso a condição seja verdadeira e fazer outra coisa caso a condição seja falsa.



## Módulo 3 – Estrutura do if – Condições no Python

Utilizando o mesmo exemplo prático, agora nosso programa deve avisar nos 2 casos:

- Caso o produto tenha batido a meta, devemos exibir a mensagem: “Batemos a meta de vendas de Iphone, vendemos {} unidades”
- Se ele não bateu a meta do mês, devemos exibir a mensagem: “Infelizmente não batemos a meta, vendemos {} unidades. A meta era de {} unidades”

O primeiro passo é atribuir valores às variáveis:

**meta** = 50000

**qtde\_vendida** = 65300

## Módulo 3 – Estrutura do if – Condições no Python

O segundo passo é construir a estrutura do **if**:

```
meta = 50000
```

```
qtde_vendida = 65300
```

Se essa condição for verdadeira

```
If qtde_vendida > meta:
```

```
    print('Batemos a meta de Iphone, vendemos {} unidades'.format(qtde_vendida))
```

```
Else:
```

```
    print('Infelizmente não batemos a meta, vendemos {} unidades. A meta era de {} unidades) format(meta)
```

Caso contrário

## Módulo 3 – Estrutura do if – Condições no Python

**Observação:** a função **format** pode ser utilizada para substituir mais de uma variável nas chaves {}).

```
meta = 50000
qtde_vendida = 65300

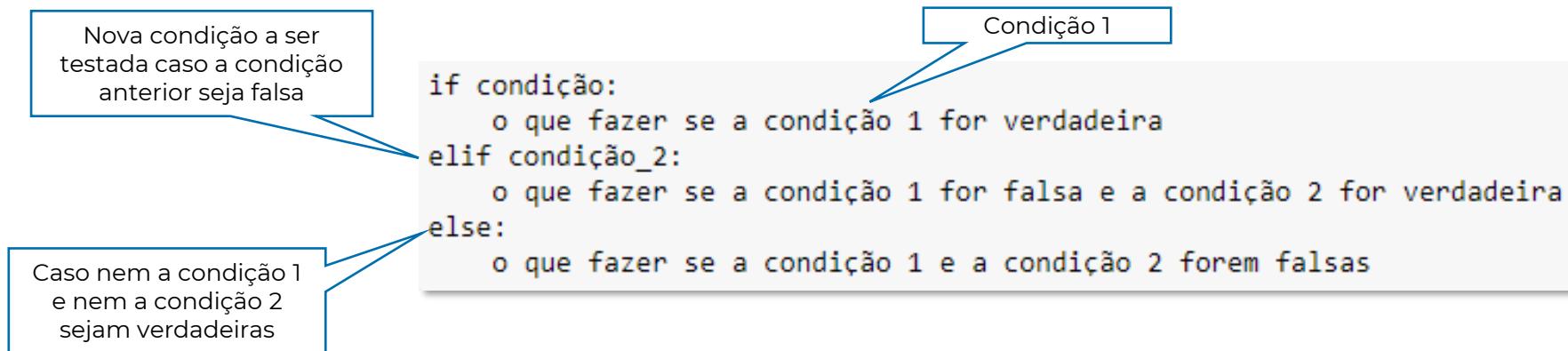
if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))
else:
    print('Infelizmente não batemos a meta, vendemos {} unidades. A meta era de {} unidades'.format(qtde_vendida,meta))

Batemos a meta de vendas de Iphone, vendemos 65300 unidades
```

The code demonstrates the use of the `format` method to insert variables into strings. It defines two variables, `meta` and `qtde_vendida`. The `if` block prints a message where the quantity sold is inserted using `format(qtde_vendida)`. The `else` block prints a more detailed message where both the quantity sold and the target value are inserted using `format(qtde_vendida,meta)`. The output shows that the quantity sold is 65300 and the target value was 50000.

## Módulo 3 – Elif

Agora vamos trabalhar com o acréscimo do if (um if composto), que é um caso mais complexo onde não temos apenas 1 condição, e sim 2 ou mais.



## Módulo 3 – Elif

Vamos para o exemplo prático ao lado.

Nesta situação, queremos criar um programa que seja capaz de analisar o bônus dos funcionários de uma empresa.

Para isso, temos que considerar alguns pontos importantes.

### Exemplo:

Vamos criar um programa para analisar o bônus dos funcionários de uma empresa (pode parecer "simples", mas uma empresa como a Amazon tem 900.000 funcionários)

Para os cargos de vendedores, a regra do bônus é de acordo com a meta de vendas da pessoa:

Se ela vendeu abaixo da meta dela, ela não ganha bônus.

Se ela vendeu acima da meta dela, ela ganha como bônus 3% do valor que ela vendeu.

Se ela vendeu mais do que o dobro da meta dela, ela ganha como bônus 7% do valor que ela vendeu.

Vamos criar um programa para avaliar uma pessoa que tinha como meta de vendas 20.000 reais e calcular o bônus dela de acordo com o valor de vendas que ela tiver.

As condições são:

- Caso a meta não seja atingida, não o vendedor não ganha bônus.
- Caso as vendas superem a meta, ele ganha 3% do valor que ele vendeu.
- Se ele vender o dobro da meta, ele ganha 3% do valor que ele vendeu.

### Exemplo:

Vamos criar um programa para analisar o bônus dos funcionários de uma empresa (pode parecer "simples", mas uma empresa como a Amazon tem 900.000 funcionários)

Para os cargos de vendedores, a regra do bônus é de acordo com a meta de vendas da pessoa:

Se ela vendeu abaixo da meta dela, ela não ganha bônus.

Se ela vendeu acima da meta dela, ela ganha como bônus 3% do valor que ela vendeu.

Se ela vendeu mais do que o dobro da meta dela, ela ganha como bônus 7% do valor que ela vendeu.

Vamos criar um programa para avaliar uma pessoa que tinha como meta de vendas 20.000 reais e calcular o bônus dela de acordo com o valor de vendas que ela tiver.

O primeiro passo é atribuir valores às variáveis:

**meta** = 20000

**vendas** = 1500

### Exemplo:

Vamos criar um programa para analisar o bônus dos funcionários de uma empresa (pode parecer "simples", mas uma empresa como a Amazon tem 900.000 funcionários)

Para os cargos de vendedores, a regra do bônus é de acordo com a meta de vendas da pessoa:

Se ela vendeu abaixo da meta dela, ela não ganha bônus.

Se ela vendeu acima da meta dela, ela ganha como bônus 3% do valor que ela vendeu.

Se ela vendeu mais do que o dobro da meta dela, ela ganha como bônus 7% do valor que ela vendeu.

Vamos criar um programa para avaliar uma pessoa que tinha como meta de vendas 20.000 reais e calcular o bônus dela de acordo com o valor de vendas que ela tiver.

## Módulo 3 – Elif

Agora vamos analisar o caso único, se a pessoa ganhou ou não bônus:

```
meta = 20000
```

```
vendas = 1500
```

```
if vendas < meta:  
    print('Não ganhou bônus')  
else:  
    bônus = 0.03 * vendas  
    print('Ganhou {} de bônus'.format(bônus))
```

```
meta = 20000  
vendas = 1500  
  
if vendas < meta:  
    print('Não ganhou bônus')  
else:  
    bônus = 0.03 * vendas  
    print('Ganhou {} de bônus'.format(bônus))
```

```
Não ganhou bônus
```

Mas agora a gente não pode falar que o vendedor vai ganhar 3% de bônus porque existem 2 condições.

## Módulo 3 – Elif

Agora vamos analisar as 2 condições juntas:

**meta** = 20000

**vendas** = 1500

**if vendas < meta:**

**print('Não ganhou bônus')**

**elif vendas > (meta \* 2):**

**bônus = 0.07 \* vendas**

**print('Ganhou {} de bônus'.format(bônus))**

**else:**

**bônus = 0.03 \* vendas**

**print('Ganhou {} de bônus'.format(bônus))**

Se ele vendeu acima de  
2 vezes a meta

Se as duas condições  
acima forem falsas

```
meta = 20000
vendas = 1500

if vendas < meta:
    print('Não ganhou bônus')
elif vendas > (meta * 2):
    bonus = 0.07 * vendas
    print('Ganhou {} de bônus'.format(bonus))
else:
    bonus = 0.03 * vendas
    print('Ganhou {} de bônus'.format(bonus))
```

Se ele vendeu menos que a meta

Não ganhou bônus

## Módulo 3 – Comparadores

Principais comparadores que você pode usar dentro de um if ou elif:

- `==` igual
- `!=` diferente
- `>` maior que (`>=` maior ou igual)
- `<` menor que (`<=` menor ou igual)
- `in` texto existe dentro de outro texto
- `not` verifica o contrário da comparação

**Observação:** Lembre-se que o sinal de igual (`=`) não significa igual e sim, **recebe**.

## Módulo 3 – Comparadores

Vamos ver agora alguns exemplos de comparadores:

No programa ao lado, se o faturamento da loja 1 for igual ao da loja 2, eu quero printar ‘Os faturamentos são iguais’, caso contrário quero printar ‘Os faturamentos são diferentes.’

Nesse exemplo, os faturamentos são diferentes (2500 e 2200).

```
faturamento_loja_1 = 2500
faturamento_loja_2 = 2200
email = "liragmail.com"

print('Programa 1')
if faturamento_loja_1 == faturamento_loja_2:
    print('Os faturamentos são iguais')
else:
    print('Os faturamentos são diferentes')
```

Símbolo de igual

Programa 1  
Os faturamentos são diferentes

## Módulo 3 – Comparadores

Agora vamos ver o que aconteceria se ao invés do “`==`” a gente colocasse o símbolo de recebe “`=`”.

Note que apareceu uma mensagem de **SyntaxError**, isso porque o Python não entende se você quer colocar um novo valor para o faturamento ou fazer uma comparação.

Então, sempre se atente ao sinal de **igual** como `==`.

```
faturamento_loja_1 = 2500
faturamento_loja_2 = 2200
email = "liragmail.com"

print('Programa 1')
if faturamento_loja_1 = faturamento_loja_2:
    print('Os faturamentos são iguais')
else:
    print('Os faturamentos são diferentes')
```

Símbolo de igual

Input In [2]  
if faturamento\_loja\_1 = faturamento\_loja\_2:  
^  
SyntaxError: invalid syntax

## Módulo 3 – Comparadores

Outro exemplo, no programa ao lado eu quero verificar se o email é [lira@gmail.com](mailto:lira@gmail.com), se for igual eu quero printar “Email correto”, caso contrário ‘Email errado’.

Como o email não possui o @, então o email está errado.

```
faturamento_loja_1 = 2500  
faturamento_loja_2 = 2200  
email = "liragmail.com"
```

Não tem o @

```
print('Programa 2')  
if email == 'lira@gmail.com':  
    print('Email correto')  
else:  
    print('Email errado')
```

```
Programa 2  
Email errado
```

## Módulo 3 – Comparadores

Por último, no programa ao lado eu quero verificar se o email é válido.

Como podemos fazer isso?

Verificando se o email possui **@** utilizando o comparador **in**.

O comparador **in** permite identificar se algo existe ao menos uma vez em um texto, ou em uma variável, etc.

Nesse caso, como não colocamos o **@**, por isso aparece que o email é inválido.

```
In [*]: print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    print('Email válido')

Insira seu e-mail: liragmail.com
```

```
In [8]: print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    print('Email válido')

Programa 3
Insira seu e-mail:liragmail.com
Email Inválido
```

## Módulo 3 – Comparadores

Normalmente quando o email é válido não precisa aparecer uma mensagem informando que ele é válido.

Nesse caso podemos usar o comparador **pass** porque não queremos fazer nada nessa condição.

```
print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    pass
```

Insira seu e-mail: lira@gmail.com

```
print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    pass
```

Programa 3  
Insira seu e-mail:lira@gmail.com

## Módulo 3 – Comparadores

Nesse exemplo também usamos o comparador NOT, que inverte o sentido da condição.

Ou seja, no exemplo ao lado, temos como condição: se o **@ não** está contido na variável email\_usuario.

Se essa condição for verdadeira, o programa imprime “Email Inválido”. Caso contrário, ele não faz nada.

```
print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    pass
```

Insira seu e-mail: lira@gmail.com

```
print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    pass
```

Programa 3  
Insira seu e-mail:lira@gmail.com

## Módulo 3 – And e Or

Até agora utilizamos o **if** para analisar uma condição de cada vez, mas em alguns casos vamos precisar usar conectores **and** e **or** para acrescentar mais condições a um mesmo teste.

Vamos utilizar o **and** para analisar se duas condições são verdadeiras **ao mesmo tempo** e **or** se **pelo menos uma** das duas condições forem verdadeiras.

```
if condicao_1 and condicao_2:  
    vai ser executado se as 2 condições forem verdadeiras ao mesmo tempo  
  
outro caso:  
  
if condicao_1 or condicao_2:  
    vai ser executado se pelo menos uma das condições forem verdadeiras
```

## Módulo 3 – And e Or

Vamos ver um exemplo prático sobre cálculo de meta de vendas dos funcionários:

Aqui o cálculo do bônus depende de 2 condições para ocorrer:

- Se o funcionário vendeu mais do que a meta de vendas **e** a loja bateu a meta de vendas da loja, o funcionário ganha 3% do que ele vendeu em forma de bônus.
- Caso o funcionário tenha batido a meta de vendas individual dele, **mas** a loja não tenha batido a meta de vendas da loja como um todo, o funcionário não ganha bônus.

## Módulo 3 – And e Or

No exemplo ao lado, como usamos o **and**, necessariamente as duas condições precisam ser atendidas para o funcionário ganhar bônus.

Nesse caso, temos que **nenhuma das condições é verdadeira**, logo o funcionário **não vai ganhar bônus**.

```
meta_funcionarios = 10000
meta_loja = 250000
vendas_funcionario = 5000
vendas_loja = 200000
if vendas_funcionario > meta_funcionarios and vendas_loja > meta_loja:
    bonus = 0.03 * vendas_funcionario
    print('Bônus do funcionário foi de {}'.format(bonus))
else:
    print('Funcionário não ganhou bônus')
```

Condição falsa      Condição falsa

Funcionário não ganhou bônus

## Módulo 3 – And e Or

Vamos considerar agora que o valor de vendas da loja, ao invés de 200000 (exemplo anterior), passe a ser 280000.

Nesse caso, temos que apenas uma das condições é verdadeira, logo o funcionário **não vai ganhar bônus**.

```
meta_funcionarios = 10000
meta_loja = 250000
vendas_funcionario = 5000
vendas_loja = 280000
if vendas_funcionario > meta_funcionarios and vendas_loja > meta_loja:
    bonus = 0.03 * vendas_funcionario
    print('Bônus do funcionário foi de {}'.format(bonus))
else:
    print('Funcionário não ganhou bônus')
```

Condição falsa

Condição verdadeira

Funcionário não ganhou bônus

## Módulo 3 – And e Or

Vamos considerar agora que o valor de vendas de um funcionário, ao invés de 5000 (exemplo anterior), passe a ser 15000.

Nesse caso, temos que ambas as condições são verdadeiras, logo o funcionário **vai ganhar bônus**.

```
meta_funcionarios = 10000
meta_loja = 250000
vendas_funcionario = 15000
vendas_loja = 280000
if vendas_funcionario > meta_funcionarios and vendas_loja > meta_loja:
    bonus = 0.03 * vendas_funcionario
    print('Bônus do funcionário foi de {}'.format(bonus))
else:
    print('Funcionário não ganhou bônus')
```

Bônus do funcionário foi de 450.0

Condição verdadeira

Condição verdadeira

## Módulo 3 – And e Or

Agora vamos para um outro exemplo e analisar mais a fundo.

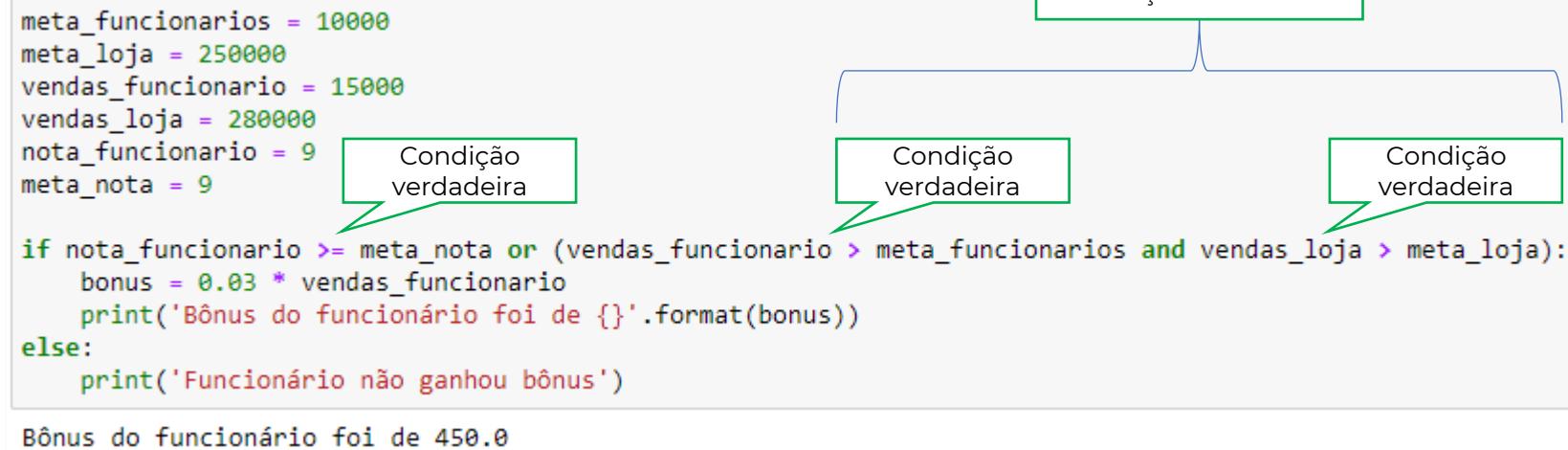
Nessa empresa, existe um outro caso também que garante que o funcionário ganhe um bônus, **independente das vendas que ele fez naquele mês**.

Se a nota do funcionário for 9 ou 10, ele também ganha o bônus de 3% do valor de vendas (os bônus não são cumulativos).

## Módulo 3 – And e Or

No exemplo abaixo, usamos o **and** e **or** ao mesmo tempo. Então, para o funcionário ganhar o bônus uma das condições do **or** precisa ser verdadeira.

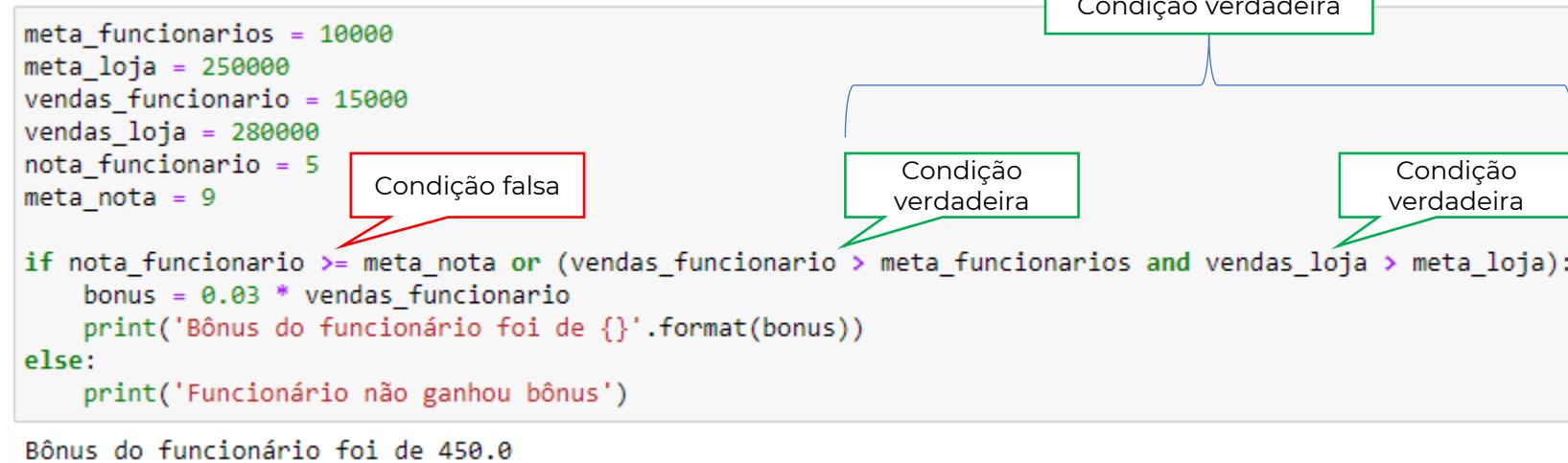
Nesse caso, temos que **todas as condições são verdadeiras**, logo o funcionário **vai ganhar bônus**.



## Módulo 3 – And e Or

Agora vamos considerar que o valor da nota, ao invés de 9 (exemplo anterior), passe a ser 5.

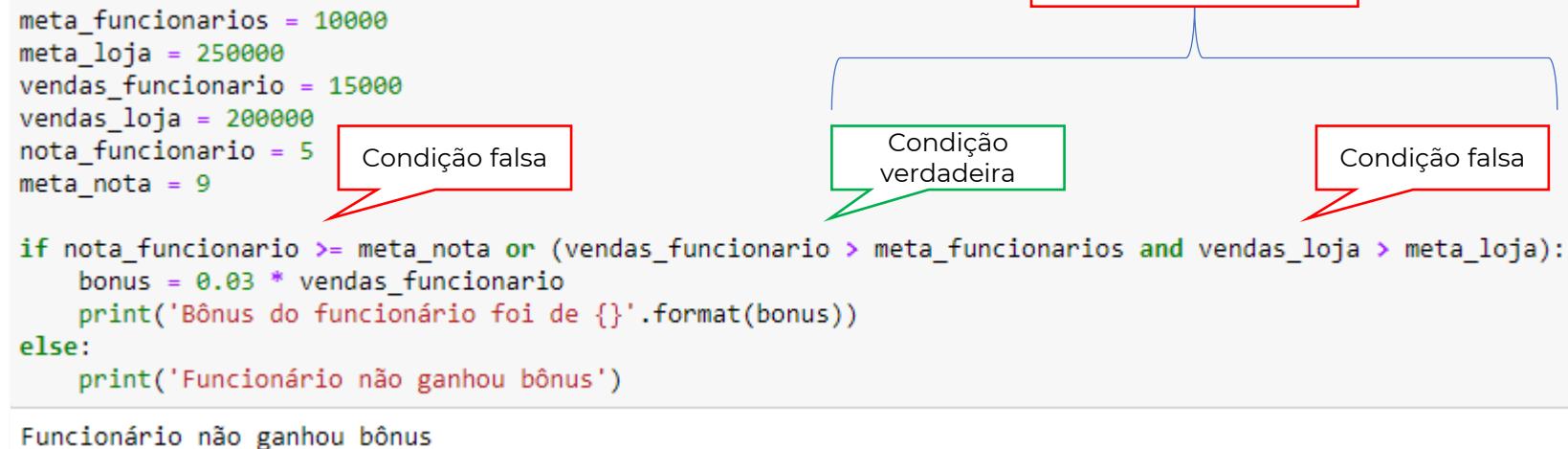
Nesse caso, temos que **apenas a condição da nota é falsa**, porém como no **or apenas uma** das condições precisa ser verdadeira, então o funcionário **vai ganhar bônus**.



## Módulo 3 – And e Or

Agora vamos considerar que o valor das vendas, ao invés de 280000 (exemplo anterior), passe a ser 200000.

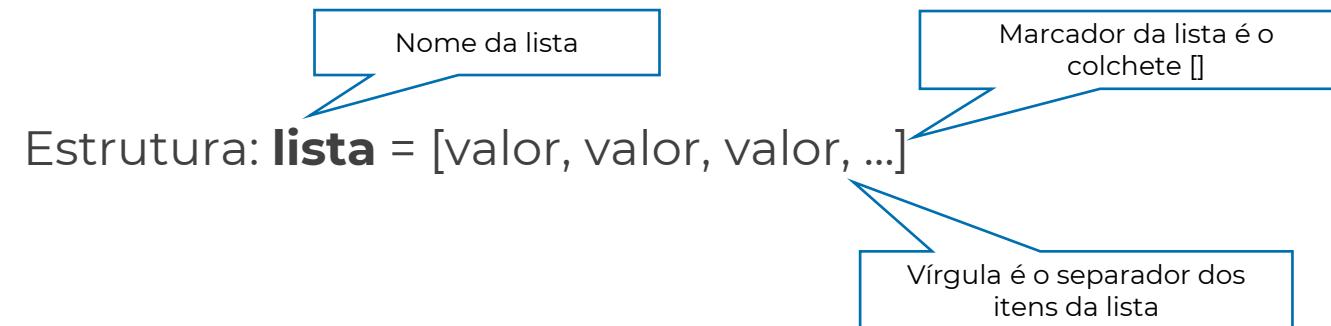
No caso do **and** ambas as condições precisam ser verdadeiras, então o resultado do **and** é uma condição **falsa**. Já que a outra condição do **or** também é **falsa**, o funcionário **não vai ganhar bônus**.



## Módulo 3 – Listas em Python

Agora vamos tratar das Listas, que são objetos muito importantes e serão muito utilizadas no curso. Quando importamos uma base de dados para o Python, normalmente ela é lida como uma "lista" ou como alguma "variação de lista".

Listas em Python foram feitas para serem homogêneas (apenas número ou texto), apesar de aceitarem valores heterogêneos.



## Módulo 3 – Listas em Python

As listas podem ser vazias. Em breve vamos explicar como adicionar itens a elas.

```
: nome = 'celular'  
produtos = ['tv', nome, 'mouse', 'teclado', 'tablet']  
print(produtos)  
  
['tv', 'celular', 'mouse', 'teclado', 'tablet']
```

As listas também podem ser representadas separando uma linha para cada item.

```
lista = []
```

Podem ser representadas com todos os itens na mesma linha. Além disso, os valores das listas podem ser textos ou variáveis.

Listas de Produtos de uma Loja:

```
produtos = ['tv',  
           'celular',  
           'mouse',  
           'teclado',  
           'tablet']
```

## Módulo 3 – Índices em Lista, Consultando e Modificando Valores

Agora vamos aprender os índices dentro da lista e como acessar ou modificar o valor de um item dentro da lista.

Se você quiser acessar um valor de um item da lista:

lista = [valor, valor, valor, valor, ...]

lista[i] → é o valor de índice i da lista que você quer como resposta.

**Observação:** Lembre que no Python o índice começa em 0, então o primeiro item de uma lista é o item lista[0].

0	1	2	3	4
produtos = ['tv', 'celular', 'mouse', 'teclado', 'tablet']				
	print(produtos[1])			
		celular		

## Módulo 3 – Índices em Lista, Consultando e Modificando Valores

No exemplo ao lado temos duas listas que são independentes, mas possuem dados complementares.

Então podemos usar a função **format** para acessarmos os dados de produto e venda de forma simples utilizando o índice da lista.

```
produtos = ['tv', 'celular', 'mouse', 'teclado', 'tablet']
           0      ,   1     ,   2    ,   3   ,   4
vendas = [ 1000,    1500 ,    350 ,    270 ,    900 ]
print('Vendas do produto {} foram de {}'.format(produtos[1], vendas[1]))
Vendas do produto celular foram de 1500
```

## Módulo 3 – Índices em Lista, Consultando e Modificando Valores

Agora vamos aprender como substituir um valor de uma lista.

Por exemplo, ao invés de 270 deveria ser 300.

Então, antes do print, vamos colocar o índice que queremos substituir recebendo o valor correto.

0 1 2 3 4

```
vendas = [1000, 1500, 350, 270, 900]
```

```
vendas[3] = 300  
print('Vendas do produto {} foram de {}'.format(produtos[3], vendas[3]))
```

## Módulo 3 – Índices em Lista, Consultando e Modificando Valores

Uma *string* no Python é como se fosse uma lista de caracteres, então o texto de índice 1 do exemplo ao lado é a letra i.

Uma diferença significativa entre a *string* e a lista é que a *string* no Python é **imutável**.

Na lista é possível substituir o valor, na *string* não. No exemplo ao lado aparece o erro de que a *string* não suporta o *item assignment*.

```
texto = 'lira@gmail.com'
print(texto[1])
i
```

---

```
texto = 'lira@gmail.com'
texto [1] = 'a'
print(texto)

-----
TypeError
Input In [7], in <cell line: 2>()
    1 texto = 'lira@gmail.com'
----> 2 texto [1] = 'a'
    3 print(texto)

TypeError: 'str' object does not support item assignment
```

Traceback (most recent call last)

## Módulo 3 – Índices em Lista, Consultando e Modificando Valores

Para substituir o valor da *string* podemos utilizar a função *replace*, mas para isso temos que criar uma nova variável.

```
texto = 'lira@gmail.com'
```

```
texto = texto.replace('i', 'a')
```

```
texto = 'lira@gmail.com'  
texto = texto.replace('i', 'a')  
print(texto)
```

```
lara@gmaal.com
```

Letra que eu quero colocar

Letra a ser substituída

## Módulo 3 – Estrutura de Repetição For

Agora vamos utilizar uma estrutura de repetição para conseguir executar códigos de forma repetitiva, ou seja, ao invés de repetir nosso código podemos pedir pra ele olhar uma lista inteira e percorrer toda aquela lista executando alguma ação. Uma das estruturas mais úteis e usadas é a **for**.

Forma mais simples de utilizar o FOR:

**for variável in range(n):**  
    repetir código n vezes

Vamos fazer um exemplo bem simples ao lado.

The diagram shows a code snippet in a light gray box. A blue callout points to the first line 'for i in range(5):' with the text 'Indentação'. Another blue callout points to the number '5' in 'range(5)' with the text 'Número de vezes'. Below the code, five instances of the word 'lira' are printed, demonstrating the loop's execution.

```
for i in range(5):
    print('lira')
lira
lira
lira
lira
lira
```

## Módulo 3 – Estrutura de Repetição For

No exemplo ao lado, o código está executando o print e voltando para a primeira linha, até terminar a contagem.

The diagram illustrates a for loop and its execution. On the left, a code block contains the Python code: `for i in range(5):  
 print('lira')`. This code is highlighted with a light gray background and a thin black border. Two blue curved arrows point from the code block to the output below. The output consists of five lines of text: `lira  
lira  
lira  
lira  
lira`, each starting with a blue arrow pointing to it.

## Módulo 3 – Estrutura de Repetição For

O **i** é um índice que funciona como um contador.

Então no código ao lado, pedimos para ele imprimir o valor de **i** um total de **5 vezes**.

Lembrando que o Python começa a contagem com o número 0.

```
for i in range(5):
    print(i)
```

0  
1  
2  
3  
4

Número de vezes

Indentação

## Módulo 3 – Estrutura de Repetição For

Imagine que você está construindo uma automação para enviar todo dia por e-mail um resumo da produção de uma fábrica. Ao invés de repetir o print 5 vezes, nós vamos colocar isso dentro de um for.

```
produtos = ['coca', 'pepsi', 'guarana', 'sprite', 'fanta']
producao = [15000, 12000, 13000, 5000, 250]

for i in range(5):
    print('{} unidades produzidas de {}'.format(producao[i],produtos[i]))
```

1<sup>a</sup> 15000 unidades produzidas de coca  
2<sup>a</sup> 12000 unidades produzidas de pepsi  
3<sup>a</sup> 13000 unidades produzidas de guarana  
4<sup>a</sup> 5000 unidades produzidas de sprite  
5<sup>a</sup> 250 unidades produzidas de fanta

Retornar o valor da tabela produção

Retornar a marca

## Módulo 3 – Estrutura de Repetição For

Agora imagine se a lista aumentar ou diminuir, nesse caso nossa estrutura for não contemplará todos os valores da lista. Para resolver esse problema vamos usar a função **len** que calculará o tamanho da lista e permitirá que mesmo com alteração do número de itens da lista o nosso for sempre vai contemplar todos.

Retorna o tamanho da lista produtos

```
produtos = ['coca', 'pepsi', 'guarana', 'sprite', 'fanta']
producao = [15000, 12000, 13000, 5000, 250]

tamanho = len(produtos)

for i in range(tamanho):

    print('{} unidades produzidas de {}'.format(producao[i],produtos[i]))
```

```
15000 unidades produzidas de coca
12000 unidades produzidas de pepsi
13000 unidades produzidas de guarana
5000 unidades produzidas de sprite
250 unidades produzidas de fanta
```

## Módulo 3 – For each - Percorrer cada item de uma lista

Agora a gente vai trabalhar com uma pequena variação do **for**.

No exemplo anterior utilizamos o **range** para informar o número de repetições executadas no for.

Porém, no Python é muito comum usarmos a própria lista para termos essa informação.

Como podemos ver no exemplo, criamos localmente uma variável **item** que irá percorrer todos os itens da lista produtos.

```
produtos = ['coca', 'pepsi', 'guarana', 'sprite', 'fanta']
texto = 'lira@gmail.com'

for item in produtos:
    print(item)

coca
pepsi
guarana
sprite
fanta
```

## Módulo 3 – For each - Percorrer cada item de uma lista

É muito comum no Python utilizarmos a palavra referente à lista no singular (**produto**) e na lista a palavra no plural (**produtos**).

Essa estrutura pode ser lida como:  
“Para cada **produto** na lista **produtos**,  
“printar” o texto”.

```
produtos = ['coca', 'pepsi', 'guarana', 'sprite', 'fanta']
texto = 'lira@gmail.com'

for produto in produtos:
    print('O produto é {}'.format(produto))

for ch in texto:
    print(ch)
```

O produto é coca  
O produto é pepsi  
O produto é guarana  
O produto é sprite  
O produto é fanta  
l  
i  
r  
a  
@  
g  
m  
a  
i  
l  
.c  
o  
m

## Módulo 3 – For e if

Vamos agora combinar o uso do for com o IF.

Note que ao lado apresentamos uma estrutura genérica. Perceba que o If possui uma identação em relação ao FOR e as linhas 3 e 5 possuem uma identação “dupla”, pois pertencem ao IF e ao FOR.

```
for item in lista:  
    if condicao:  
        faça alguma coisa  
    else:  
        outra coisa
```

## Módulo 3 – For e if

Imagine que a gente esteja analisando a meta de vendas de vários funcionários de uma empresa.

A meta de vendas é de 1000 reais em 1 dia e queremos calcular qual o % de pessoas que bateram a meta.

O primeiro passo que vamos fazer é percorrer essa lista e printar todos os itens da lista.

```
vendas = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100, 999, 900, 880, 870, 50, 1111, 120, 300, 450, 800]
meta = 1000

for venda in vendas:
    print(venda)

1200
300
800
1500
1900
2750
400
20
23
70
90
80
1100
999
900
880
870
50
1111
120
300
450
800
```

## Módulo 3 – For e if

Agora vamos analisar se as vendas foram maiores que a meta. Porém, o ideal seria termos um “contador” que acumulasse o número de funcionários que bateram a meta.

Com esse contador será possível, por exemplo, dizer automaticamente a % de atingimento de meta.

```
vendas = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100, 999, 900, 880, 870, 50, 1111, 120, 300, 450, 800]
meta = 1000

for venda in vendas:
    if venda >= meta:
        print(venda)

1200
1500
1900
2750
1100
1111
```

## Módulo 3 – For e if

Abaixo indicamos uma variável **qtde\_bateu\_meta** com valor inicial igual a zero e cada vez que a meta fosse atingida seria somado 1 a esse valor.

```
vendas = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100, 999, 900, 880, 870, 50, 1111, 120, 300, 450, 800]
meta = 1000

qtde_bateu_meta = 0

for venda in vendas:
    if venda >= meta:
        qtde_bateu_meta +=1
print(qtde_bateu_meta)
```

6

## Módulo 3 – For e if

Agora precisamos calcular a quantidade de pessoas totais, ou seja, o tamanho da minha lista. Depois vamos calcular o percentual dividindo a quantidade de pessoas que bateram a meta pela quantidade total de pessoas.

```
vendas = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100, 999, 900, 880, 870, 50, 1111, 120, 300, 450, 800]
meta = 1000

qtde_bateu_meta = 0

for venda in vendas:
    if venda >= meta:
        qtde_bateu_meta += 1
print(qtde_bateu_meta)

qtde_funcionario = len(vendas)
print('O percentual de pessoas que bateram a meta foi de {:.1%}'.format(qtde_bateu_meta / qtde_funcionario))

6
O percentual de pessoas que bateram a meta foi de 26.1%
```

O : indica que será o formato de uma forma específica

.1 significa o número de casas decimais do resultado

% indica que o resultado se trata de uma porcentagem

## Módulo 3 – Estrutura While

Agora a gente vai estudar uma outra estrutura de repetição: o **while**.

A grande diferença em relação ao **for** é que no **for** a gente executava a repetição um **número de vezes previamente definido**; já no **while**, a gente vai executar a repetição enquanto **uma determinada condição for verdadeira**.

**while condicao:**  
**repete esse código**

## Módulo 3 – Estrutura While

Vamos ver na prática como isso funciona para criar um programa que registre as vendas de uma empresa.

Nele, a pessoa deve inserir o nome do produto e o produto deve ser adicionado na lista de venda. Enquanto o usuário não encerrar o programa, significa que ele está registrando novos produtos, então o programa deve permitir e entrada de quantos produtos o usuário quiser.

```
venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')
vendas = []
    Diferente de vazio
while venda != '':
    vendas.append(venda)
    venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')

print('Registro Finalizado. As vendas cadastradas foram: {}'.format(vendas))

Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vaziaarroz
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vaziafeijão
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia
Registro Finalizado. As vendas cadastradas foram: ['arroz', 'feijão']
```

## Módulo 3 – Estrutura While

**Dica de ouro:** sempre que for usar o comando **while**, lembre-se de ter certeza que o programa vai terminar em algum momento.

Se o while não terminar em algum momento ele entra no que chamamos de loop infinito e aí duas opções podem ocorrer:

- 1) Seu computador vai travar
- 2) Seu código ficará rodando infinitamente

## Módulo 3 – Loop infinito no While

Digamos que temos uma lista de vendedores e as quantidades vendidas e queremos identificar todos os vendedores que bateram a meta de 50 vendas.

Olhando a condição do **while** temos que nosso loop será pausado quando a venda for menor que a meta.

Quando olhamos o código dentro da identação não temos nada que aumente o valor de i, que diferente do **for** aumenta automaticamente.

```
vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 269, 259, 218, 208, 163, 125, 102, 87, 47, 7]
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca',
meta = 50

i = 0

while vendas[i] > meta:
    print('{} bater a meta. Vendas: {}'.format(vendedores[i],vendas[i]))
```

# Módulo 3 – Loop infinito no While

Isso fará com que **vendas** sempre seja 941 e consequentemente sempre maior que a meta que é 50.

Logos, estamos realizando um loop infinito.

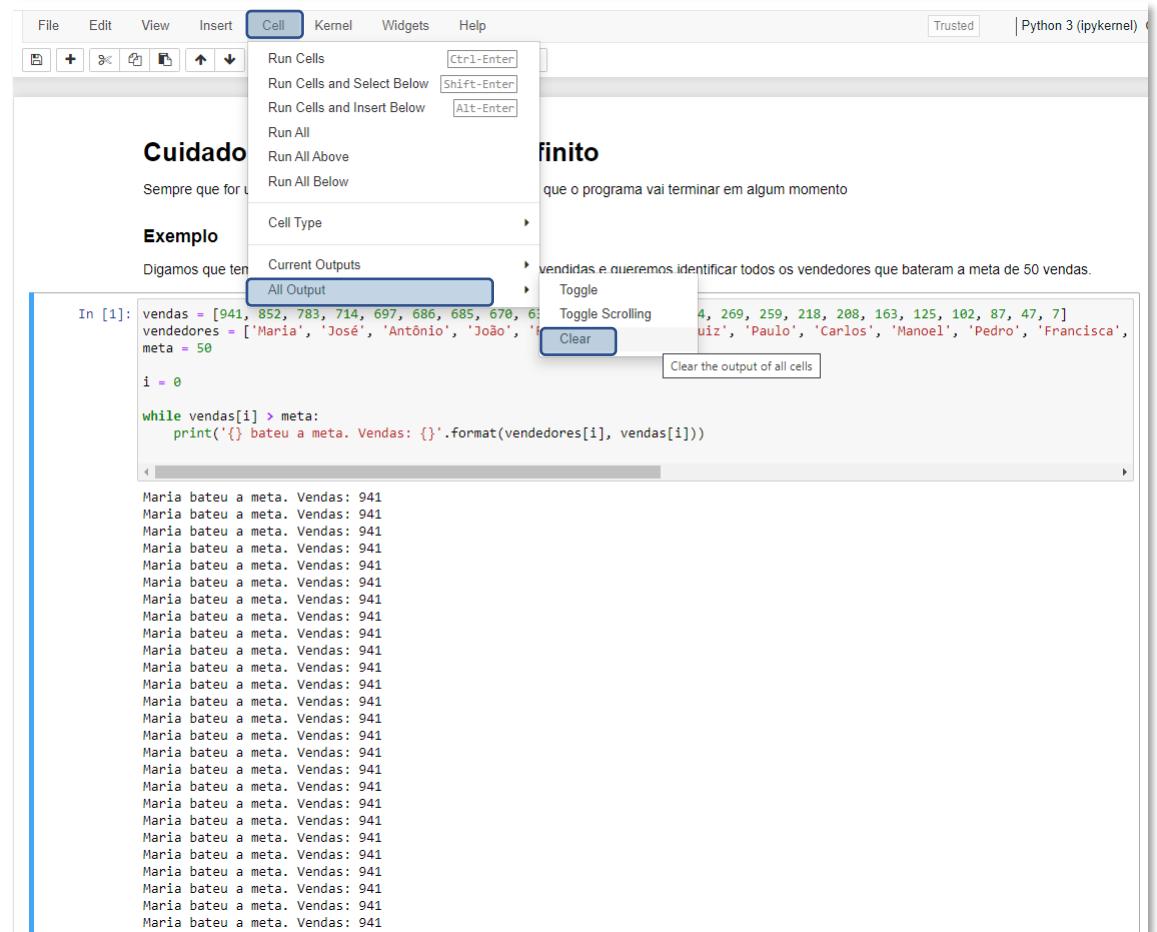
## Módulo 3 – Loop infinito no While

Existem 2 formas de parar esse loop infinito:

- Clicar em stop e corrigir o erro
  - Pausar o Jupyter pelo prompt de comando apertando as teclar CTRL + C

## Módulo 3 – Loop infinito no While

Para limpar o resultado clique em  
cell > all Outputs > Clear



## Módulo 3 – Loop infinito no While

Agora para a gente corrigir o código vamos colocar para o valor de **i receber i + 1**.

```
vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 269, 259, 218, 208, 163, 125, 102, 87, 47, 7]
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca',
meta = 50

i = 0

while vendas[i] > meta:
    print('{} bateu a meta. Vendas: {}'.format(vendedores[i], vendas[i]))
    i += 1
```

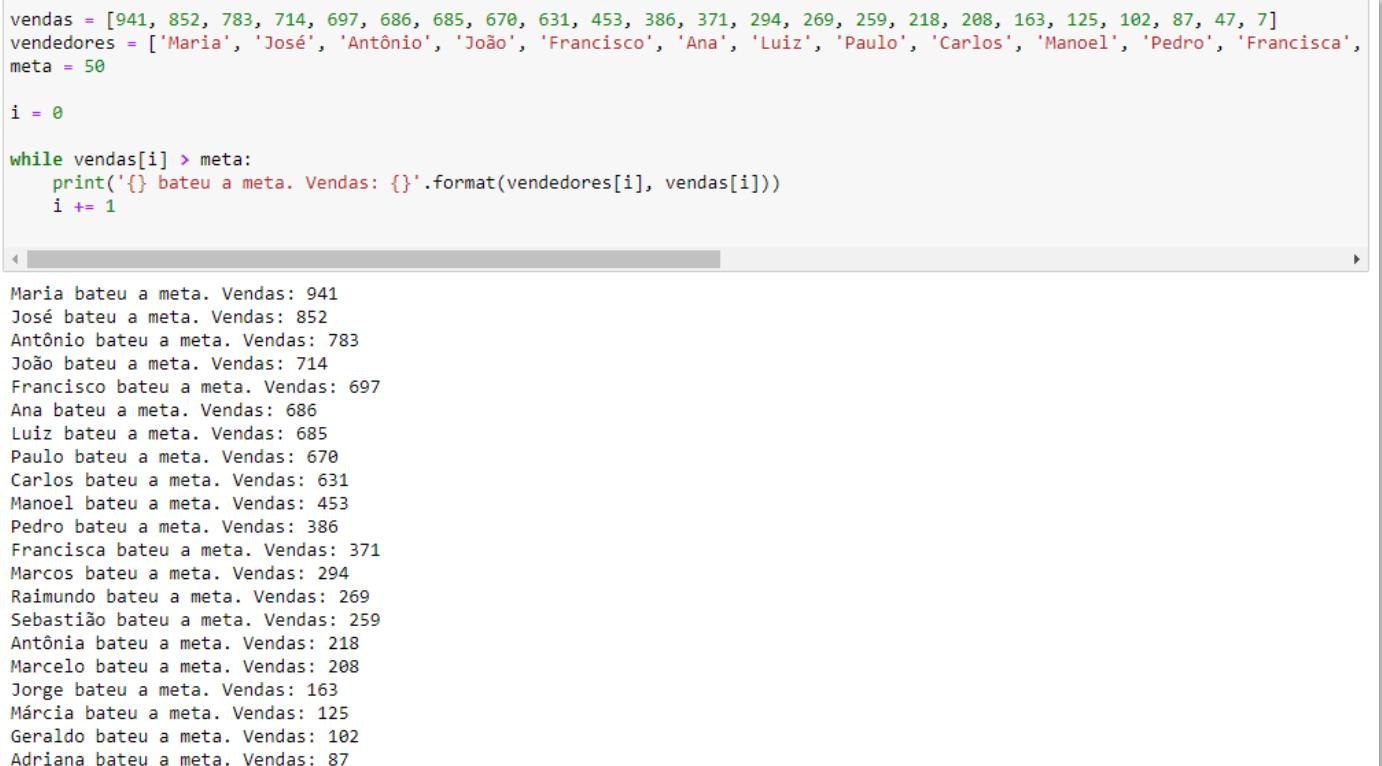
## Módulo 3 – Loop infinito no While

Agora sim o código será executado sem problemas.

```
vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 269, 259, 218, 208, 163, 125, 102, 87, 47, 7]
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca',
meta = 50

i = 0

while vendas[i] > meta:
    print('{} bateu a meta. Vendas: {}'.format(vendedores[i], vendas[i]))
    i += 1
```



```
Maria bateu a meta. Vendas: 941
José bateu a meta. Vendas: 852
Antônio bateu a meta. Vendas: 783
João bateu a meta. Vendas: 714
Francisco bateu a meta. Vendas: 697
Ana bateu a meta. Vendas: 686
Luiz bateu a meta. Vendas: 685
Paulo bateu a meta. Vendas: 670
Carlos bateu a meta. Vendas: 631
Manoel bateu a meta. Vendas: 453
Pedro bateu a meta. Vendas: 386
Francisca bateu a meta. Vendas: 371
Marcos bateu a meta. Vendas: 294
Raimundo bateu a meta. Vendas: 269
Sebastião bateu a meta. Vendas: 259
Antônia bateu a meta. Vendas: 218
Marcelo bateu a meta. Vendas: 208
Jorge bateu a meta. Vendas: 163
Márcia bateu a meta. Vendas: 125
Geraldo bateu a meta. Vendas: 102
Adriana bateu a meta. Vendas: 87
```

## Módulo 3 – Tuplas

**Criando tuplas**

```
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
```

Número  
Uso de ()  
Texto

A estrutura das Tuplas são parecidas com listas, mas elas têm algumas diferenças.

Estrutura da Tupla:

**Tupla = (valor, valor)**

As tuplas são imutáveis e podem ter dados heterogêneos.

Enquanto nas **listas** utilizávamos **[]**, nas **Tuplas** utilizamos **( )**.

## Módulo 3 – Tuplas

Algumas vantagens e desvantagens das Tuplas:

### VANTAGENS

Proteger os dados e garantir que eles nunca serão modificados dentro do programa

Mais rápida que a lista para executar

Aplicação para dados heterogêneos

### DESVANTAGENS

Menos flexível que a lista, ou seja, não é possível alterar valores

Imutável

## Módulo 3 – Tuplas

Para acessar um valor da Tupla fazemos de forma semelhante às listas:

**Nome** = vendas[**0**]

Apesar de criar a tupla com (), sempre que você for acessar o índice de algum dado você vai usar [ ].

```
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
```

### Acessando o valor de uma tupla

```
nome = vendas[0]
data_contratacao = vendas[1]
data_nascimento = vendas[2]
salario = vendas[3]
cargo = vendas[4]
```

```
print(cargo)
```

```
Estagiário
```

## Módulo 3 – Tuplas

Agora, se a gente tentar modificar o salário, por exemplo, de 2000 para 3000, repare que ele indica um erro e informa que a Tupla **não pode ser modificada**.

```
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
```

### Acessando o valor de uma tupla

```
vendas[3] = 3000
```

Atribuir um novo valor

```
nome = vendas[0]
data_contratacao = vendas[1]
data_nascimento = vendas[2]
salario = vendas[3]
cargo = vendas[4]

print(cargo)
```

```
-----  
TypeError: 'tuple' object does not support item assignment
```

TypeError  
Input In [3], in <cell line: 1>()  
----> 1 vendas[3] = 3000  
 3 nome = vendas[0]  
 4 data\_contratacao = vendas[1] Erro: Tupla não pode ser modificada

## Módulo 3 – Tuplas

Abaixo listamos as diferenças das Tuplas e Listas:

	Tuplas	Listas
Dados heterogêneos	✓	✗
Mutável	✗	✓
Utiliza [] para escrever o código	✗	✓
Utiliza () para escrever o código	✓	✗

## Módulo 3 – Unpacking em Tuplas

Agora, ao invés de identificar o índice da informação, vamos desmembrar a tupla em várias variáveis e em uma única linha de código.

Perceba que a ordem da atribuição corresponde à sequência da tupla.

```
1     2     3     4     5
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')

1     2     3     4     5
nome, data_contratacao, data_nascimento, salario, cargo = vendas
                                         Recebe a tupla

print(nome)                         Variáveis separadas por vírgula
                                         Lira
```

## Módulo 3 – Unpacking em Tuplas

**Atenção:** se você tem 5 itens na tupla, você tem que ter 5 variáveis.

Por exemplo, abaixo estamos atribuindo uma tupla com 5 itens a 4 variáveis, dessa forma o Python gera um erro pois o número de itens é maior que o de variáveis.

```
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')

nome, data_contratacao, data_nascimento, salario = vendas

print(nome)
```

5 itens

4 variáveis

```
-----  
ValueError                                     Traceback (most recent call last)
Input In [5], in <cell line: 3>()
      1 vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
----> 3 nome, data_contratacao, data_nascimento, salario = vendas
      5 print(nome)

ValueError: too many values to unpack (expected 4)
```

## Módulo 3 – Unpacking em Tuplas

O enumerate fornece 2 informações: **valor** e o **índice** dentro da lista.

No exemplo ao lado, ele vai utilizar a variável **venda** para acessar cada um dos valores da lista **vendas** e o **i** vai puxar o **índice**.

Repare que o **i, venda** é como se a gente estivesse fazendo o **unpacking** na Tupla.

```
vendas = [1000, 2000, 300, 300, 150]
funcionarios = ['João', 'Lira', 'Ana', 'Maria', 'Paula']

for i, venda in enumerate(vendas):
    print('{} vendeu {} unidades'.format(funcionarios[i], venda))

João vendeu 1000 unidades
Lira vendeu 2000 unidades
Ana vendeu 300 unidades
Maria vendeu 300 unidades
Paula vendeu 150 unidades
```

## Módulo 3 – Unpacking em Tuplas

Então o **enumerate** puxa esse valor e transforma em uma tupla, que tem 2 informações: **índice** e o **valor**.

```
vendas = [1000, 2000, 300, 300, 150]
funcionarios = ['João', 'Lira', 'Ana', 'Maria', 'Paula']

for item in enumerate(vendas):
    print(item)
        #print('{} vendeu {} unidades'.format(funcionarios[i], venda))

(0, 1000)
(1, 2000)
(2, 300)
(3, 300)
(4, 150)
```

## Módulo 3 – Unpacking em Tuplas

O enumerate transforma esses valores em uma tupla que tem 2 informações.

Então, as **tuplas** vão ser muito úteis principalmente quando você estiver trabalhando com uma lista de informações que não representam a mesma coisa, por exemplo: **nomes** e **números**.

```
vendas = [1000, 2000, 300, 300, 150]
funcionarios = ['João', 'Lira', 'Ana', 'Maria', 'Paula']

for i, venda in enumerate(vendas):
    print('{} vendeu {} unidades'.format(funcionarios[i], venda))

João vendeu 1000 unidades
Lira vendeu 2000 unidades
Ana vendeu 300 unidades
Maria vendeu 300 unidades
Paula vendeu 150 unidades
```

## Módulo 3 – Dicionários em Python

O dicionário é uma outra estrutura de dados que armazena informações. Até o momento, já trabalhamos com lista e tupla. O dicionário, assim como a tupla, pode ser heterogêneo.

No dicionário, ao invés de você usar o índice para acessar a informação, você vai usar uma **chave**, que é basicamente um nome. Cada item do dicionário é um conjunto de uma chave e um valor.

**Dicionario = {chave: valor, chave: valor, chave: valor...}**



```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'geladeira', 'vendas_tecnologia': {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000, 'celular': 25000}, 'vendas_refrigeracao': {'consul': 10000, 'brastemp': 8000, 'electrolux': 5000, 'philco': 3000, 'sanyo': 2000}, 'vendas_livros': {'o alquimista': 5000, 'o senhor dos anéis': 4000, 'o hobbit': 3000, 'o nome da rosa': 2000, 'o diário de um zumbi': 1000}, 'vendas_eletrodomestico': {'geladeira': 10000, 'fogão': 8000, 'microondas': 5000, 'lava e seca': 3000, 'secadora': 2000}, 'vendas_celular': {'apple': 25000, 'samsung': 18000, 'huawei': 12000, 'xiaomi': 8000, 'oppo': 5000}, 'vendas_tablet': {'apple': 1720, 'amazon': 1500, 'samsung': 1200, 'huawei': 800, 'xiaomi': 500}, 'vendas_ipad': {'apple': 1000, 'amazon': 800, 'samsung': 600, 'huawei': 400, 'xiaomi': 200}, 'vendas_tv': {'samsung': 10000, 'lg': 8000, 'philco': 5000, 'panasonic': 3000, 'sony': 2000}, 'vendas_ps5': {'sony': 14300, 'playstation': 12000, 'xbox': 8000, 'nintendo': 5000, 'microsoft': 3000}, 'vendas_celular_barato': {'apple': 25000, 'samsung': 18000, 'huawei': 12000, 'xiaomi': 8000, 'oppo': 5000}, 'vendas_tablet_barato': {'apple': 1720, 'amazon': 1500, 'samsung': 1200, 'huawei': 800, 'xiaomi': 500}, 'vendas_ipad_barato': {'apple': 1000, 'amazon': 800, 'samsung': 600, 'huawei': 400, 'xiaomi': 200}, 'vendas_tv_barato': {'samsung': 10000, 'lg': 8000, 'philco': 5000, 'panasonic': 3000, 'sony': 2000}, 'vendas_ps5_barato': {'sony': 14300, 'playstation': 12000, 'xbox': 8000, 'nintendo': 5000, 'microsoft': 3000}}
```

# Módulo 3 – Dicionários em Python

Imagine que a gente queira saber quanto foi vendido de iphone.

Vamos criar uma variável chamada **qtde\_iphone** para receber o valor do dicionário **vendas\_tecnologia** com a chave '**iphone**'.

```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'geladeira', 'roupas': 'camisa', 'eletronica': 'celular', 'lazer': 'xbox', 'alimentacao': 'bolacha', 'limpeza': 'spray', 'cuidados_pessoais': 'creme'}, vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000, 'celular': 8000}, qtde_iphone = vendas_tecnologia['iphone'], print(qtde_iphone)
```

**Atenção:** Você não pode ter 2 produtos com a mesma chave.

## Módulo 3 – Dicionários em Python

Imagine que a gente queira o item mais vendido nas categorias 'livros' e 'lazer'.

Vamos criar uma variável chamada **livro** para receber o valor do dicionário **mais\_vendidos** com a chave '**'livros'**'. O mesmo para o **lazer**.

## Módulo 3 – Dicionários em Python

A estrutura do código é mostrada abaixo:

```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'ge  
vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000,  
livro = mais_vendidos['livros']  
lazer = mais_vendidos['lazer']  
print(livro) Dicionário  
print(lazer) Chave  
o alquimista  
prancha surf
```

## Módulo 3 – Dicionários em Python

Imagine que a gente queira saber quanto foi vendido de 'notebook asus' e de 'ipad'.

Vamos fazer da mesma forma, igual ao exemplo ao lado.

```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'geladeira', 'lazer': 'notebook asus'}
```

```
vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000, 'notebook asus': 2450}
```

```
livro = mais_vendidos['livros']
lazer = mais_vendidos['lazer']
```

```
print(livro)
print(lazer)
```

```
qtde_notebook_asus = vendas_tecnologia['notebook asus']
qtde_ipad = vendas_tecnologia['ipad']
```

```
print(qtde_notebook_asus)
print(qtde_ipad)
```

```
o alquimista
prancha surf
2450
1000
```

## Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

Existem duas formas de buscar um valor:

- dicionario['CHAVE']
- .get['CHAVE']

Então, fazendo da forma da aula passada, vamos criar um código com **print(dicionário['chave'])**.

```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'geladeira', 'lazer': 'ps5', 'notebook': 'asus', 'tablet': 'ipad', 'outros': 'prancha surf'}
```

- Qual foi o item mais vendido nas categorias 'livros' e 'lazer'?
- Quanto foi vendido de 'notebook asus' e de 'ipad'?

```
# respondendo com a chave, igual a aula anterior
print('O livro mais vendido foi {}'.format(mais_vendidos['livros']))
print('O produto mais vendido em lazer foi {}'.format(mais_vendidos['lazer']))
# respondendo com o método get
```

```
O livro mais vendido foi o alquimista
O produto mais vendido em lazer foi prancha surf
```

## Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

Pelo método chamado get(), vamos usar a mesma premissa anterior, só mudando o formato do código.

Estrutura:

`print(dicionário.get['chave']).`

```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'geladeira electrolux', 'lazer': 'prancha surf'}
```

- Qual foi o item mais vendido nas categorias 'livros' e 'lazer'?
- Quanto foi vendido de 'notebook asus' e de 'ipad'?

```
# respondendo com a chave, igual aula anterior
print('O livro mais vendido foi {}'.format(mais_vendidos['livros']))
print('O produto mais vendido em lazer foi {}'.format(mais_vendidos['lazer']))
# respondendo com o método get
print(vendas_tecnologia.get('notebook asus'))
print(vendas_tecnologia.get('ipad'))
```

```
O livro mais vendido foi o alquimista
O produto mais vendido em lazer foi prancha surf
2450
1000
```

## Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

O que aconteceria se procurássemos por uma chave que não existe?

O método **.get** retornará **None** como resultado. Ou seja, o Python retorna dizendo que não existe nenhuma chave 'ipads' no dicionário.

Já pelo outro método, ao procurar uma chave inexistente, o Python retorna um **erro**.

```
GET  
print(vendas_tecnologia.get('ipads'))  
None
```

```
[CHAVE]  
print(vendas_tecnologia['ipads'])  
  
-----  
KeyError  
Input In [8], in <cell line: 1>()  
----> 1 print(vendas_tecnologia['ipads'])  
  
KeyError: 'ipads'
```

Traceback (most recent call last)

## Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

```
if 'copo' in vendas_tecnologia:  
    print(vendas_tecnologia['copo'])  
else:  
    print('Copo não está dentro da lista de produtos de tecnologia')  
  
Copo não está dentro da lista de produtos de tecnologia
```

Se tentarmos procurar as vendas de "copo" na lista de vendas tecnologia, o que acontece?

A primeira forma de testar é com o **if**, utilizando o método das **chaves**.

Então, se o **copo** existe dentro do **dicionário**, retorna o valor de **copo**, caso contrário retorna o **print** “Copo não está dentro da lista de produtos de tecnologia”

## Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

```
if vendas_tecnologia.get('copo') == None:  
    print('Copo não está dentro da lista de produtos de tecnologia')  
else:  
    print(vendas_tecnologia.get('copo'))  
  
Copo não está dentro da lista de produtos de tecnologia
```

Uma outra forma é utilizando o **if** com o método **get**.

Então, se valor de **dicionário.get('copo')** é **None**, retorna o print “Copo não está dentro da lista de produtos de tecnologia”. Caso contrário retorna o valor de copo.

## Módulo 3 – Range

Agora a gente vai trabalhar com as diferentes opções dentro da estrutura do range:

- range(tamanho)
- range(inicio, fim)
- range(inicio, fim, passo)

Vamos ver na prática cada um desses exemplos.

## Módulo 3 – Range

O uso mais comum do **range** é o tamanho, que é muito utilizado com o **for** para percorrer uma lista.

```
#uso mais comum no for:  
produtos = ['arroz', 'feijao', 'macarrao', 'atum', 'azeite']  
estoque = [50, 100, 20, 5, 80]  
  
for i in range(5):  
    print('{}: {} unidades em estoque'.format(produtos[i], estoque[i]))
```

arroz: 50 unidades em estoque  
feijao: 100 unidades em estoque  
macarrao: 20 unidades em estoque  
atum: 5 unidades em estoque  
azeite: 80 unidades em estoque

## Módulo 3 – Range

A segunda forma é passar o início e o fim.

Por exemplo:

**Print(range(1,10))**

Ele vai percorrer 9 itens, porque o 10 é excludente.

```
#range com inicio e fim  
print(range(1, 10))  
  
#vamos olhar no for para entender  
  
for i in range(1, 10):  
    print(i)
```

```
range(1, 10)  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

# Módulo 3 – Range

Agora um exemplo prático:

# Modelo Jack Welch da G&E

Classe A: 10% melhor

Classe B: 80% mantém/busca melhorar

## Classe C: 10% pior

# Quem são os funcionários classe B?

Nesse caso, vamos querer as pessoas que estão no meio, entre 3 e 17.

```
funcionarios = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca', 'vendas = [2750, 1900, 1500, 1200, 1111, 1100, 999, 900, 880, 870, 800, 800, 450, 400, 300, 300, 120, 90, 80, 70]
```

O último é excludente

## Módulo 3 – Range

Resolvemos o exercício anterior colocando o print para indicar o funcionário e a quantidade de venda.

```
funcionarios = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca'
vendas = [2750, 1900, 1500, 1200, 1111, 1100, 999, 900, 880, 870, 800, 800, 450, 400, 300, 300, 120, 90, 80, 70]

print('Funcionários classe B:')

for i in range(3,18):
    print('{0}: fez {1} vendas'.format(funcionarios[i],vendas[i]))
```

Funcionários classe B:  
João: fez 1200 vendas  
Francisco: fez 1111 vendas  
Ana: fez 1100 vendas  
Luiz: fez 999 vendas  
Paulo: fez 900 vendas  
Carlos: fez 880 vendas  
Manoel: fez 870 vendas  
Pedro: fez 800 vendas  
Francisca: fez 800 vendas  
Marcos: fez 450 vendas  
Raimundo: fez 400 vendas  
Sebastião: fez 300 vendas  
Antônia: fez 300 vendas  
Marcelo: fez 120 vendas  
Jorge: fez 90 vendas

## Módulo 3 – Range

A última opção que a gente tem dentro do range é utilizar ele com o **passo**.

Como assim passo?

Vamos supor que eu quero ir de 0 a 10, mas com intervalo de 2 em 2. Nesse caso, a gente utiliza o range com passo.

**Range**(início, fim, passo)

Então, o código vai retornar os valores de 0, 2, 4, 6 e 8 (lembrando que o 10 é excludente).

```
#range com passo
print(range(0, 10, 2))  
for i in range(0, 10, 2):
    print(i)  
  
range(0, 10, 2)
0
2
4
6
8
```

Passo

## Módulo 3 – Functions no Python

As **functions** são blocos de código que servem para um único propósito: fazem uma ação específica. Por exemplo: o comando print, que vimos algumas vezes anteriormente, é um exemplo de function. Só que ela é uma function nativa do Python, ou seja, ela já estava pronta pra gente usar.

Porém, podemos criar a nossa própria function e personalizar para o que a gente quiser. A estrutura para criar uma function é a seguinte:

Estrutura básica:

```
def nome_funcao():
    faça alguma coisa
    faça outra coisa
    return valor_final
```

## Módulo 3 – Functions no Python

Vamos criar uma função de cadastro de um Produto. Essa função deve garantir que o produto cadastrado está em letra minúscula.

The diagram shows a code editor with three annotations:

- A callout pointing to the function name `cadastrar_produto` is labeled "Nome da função".
- A callout pointing to the entire block of code within the function definition is labeled "Tudo está dentro da função".
- A callout pointing to the line `cadastrar_produto()` outside the function definition is labeled "Usar a função".

```
def cadastrar_produto():
    produto = input('Digite o nome do produto que deseja cadastrar')
    produto = produto.casefold()
    print('Pruduto {} cadastrado com sucesso'.format(produto))

cadastrar_produto()

Digite o nome do produto que deseja cadastrarArr0Z
Pruduto arroz cadastrado com sucesso
```

**Observação:** a função `.casefold()` é utilizada para transformar a palavra em letra minúscula.

## Módulo 3 – Retornar um valor na Function

Agora a gente vai retornar o valor para uma function. Mas, primeiro de tudo, o que é retornar um valor?

Na function, nós temos 2 opções: (1) executar as ações e (2) retornar um valor como resposta.

Retornar alguma coisa como resposta significa basicamente que a minha function vai dar um valor como resultado, que poderemos utilizar para alguma finalidade dentro do nosso código.

## Módulo 3 – Retornar um valor na Function

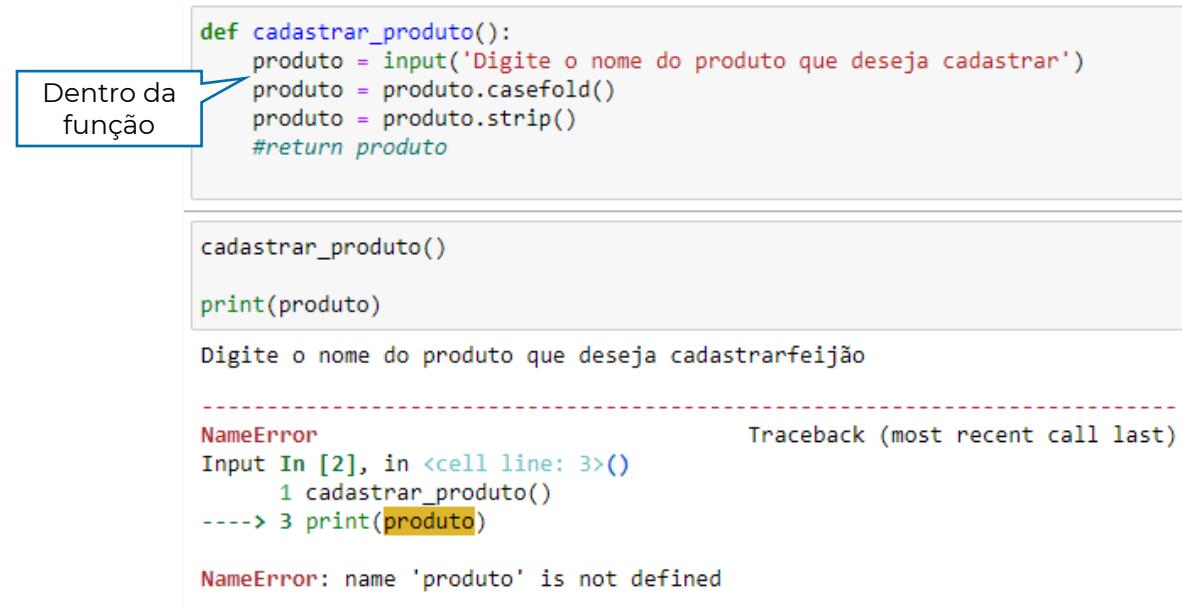
Uma variável utilizada dentro de uma function só existe dentro daquela function.

Vejamos o exemplo ao lado, onde criamos uma function para o cadastro de produtos.

Quando utilizamos o código print logo após chamar a function, ele está tentando acessar a variável produto que está dentro da function, por isso aparece um erro.

E como fazemos para acessar o valor de alguma coisa que estava dentro da function?

Utilizamos o **return**.



```
def cadastrar_produto():
    produto = input('Digite o nome do produto que deseja cadastrar')
    produto = produto.casefold()
    produto = produto.strip()
    #return produto

cadastrar_produto()
print(produto)

Digite o nome do produto que deseja cadastrarfeijão
-----
NameError: name 'produto' is not defined
Traceback (most recent call last)
Input In [2], in <cell line: 3>()
      1 cadastrar_produto()
----> 3 print(produto)

NameError: name 'produto' is not defined
```

## Módulo 3 – Retornar um valor na Function

O **return** vai basicamente te dar a resposta da function.

O retorno geralmente é composto por listas, variáveis, dicionários, etc

**Obs: Não podemos** escrever nada abaixo do return dentro da function. O comando return deve ser o último comando de uma function.

```
def cadastrar_produto():
    produto = input('Digite o nome do produto que deseja cadastrar')
    produto = produto.casefold()
    produto = produto.strip()
    return produto
```

```
variavel_produto = cadastrar_produto()

print(variavel_produto)
```

```
Digite o nome do produto que deseja cadastrar ArRoz
arroz
```

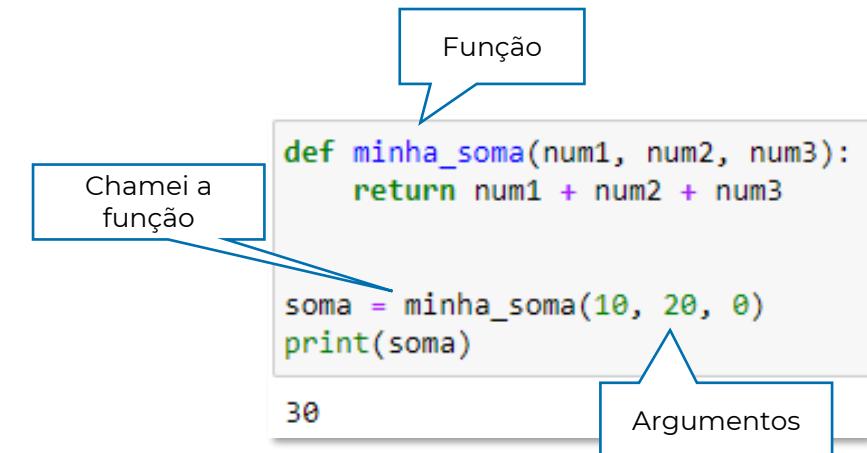
## Módulo 3 – Argumentos e Parâmetros numa Function

Agora a gente vai aprender como passar um argumento ou parâmetro para uma função.

Argumento é a informação que você passa dentro dos parênteses. Por exemplo: `print('texto')`. Nesse caso, **texto** é o argumento/parâmetro.

O **print** é um exemplo de uma função que pode receber vários parâmetros para funcionar.

Só para mostrar o funcionamento, vamos criar uma função de soma, conforme o exemplo ao lado.



## Módulo 3 – (Opcional) Aplicação em um Exemplo de argumento

Agora um exemplo prático:

Digamos que estamos criando um programa para categorizar os produtos de uma revendedora de bebidas.

Cada produto tem um código. O tipo de produto é dado pelas 3 primeiras letras do código. Exemplos:

- Vinho -> BEB12302
- Guaraná -> BSA11104

Repare que bebidas não alcóolicas começam com BSA e bebidas alcoólicas começam com BEB.

# Módulo 3 – (Opcional) Aplicação em um Exemplo de argumento

Na prática temos uma lista de produtos e eu quero olhar todos os itens.

Antes de escrever o código vamos pensar no que precisamos fazer:

1. Percorrer toda a minha lista de produtos
2. Para cada produto, verificar se ele é bebida alcoólica
3. Se for bebida alcoólica, exibir a mensagem “Enviar...”

The screenshot shows a Python script with annotations. A blue box labeled 'Função' points to the function definition. Another blue box labeled 'Coloca tudo em letra maiúscula' points to the line `bebida = bebida.upper()`. A third blue box labeled 'Retorna letra minúscula pois puxa da lista produto' points to the variable `produto` in the loop. The script itself is as follows:

```
def ehalcoolico(bebida):
    bebida = bebida.upper()
    if "BEB" in bebida:
        return True
    else:
        return False

produtos = ['beb46275', 'TFA23962', 'TFA64715', 'TFA69555', 'TFA56743', 'BSA45510', 'TFA44968', 'CAR75448', 'CAR23596', 'CAR13490', 'BEB21365', 'BEB31623', 'BEB73344', 'BEB80694', 'BEB19495', 'BEB97471', 'BEB62362', 'BEB85146', 'BEB48898', 'BEB79496', 'BEB15385', 'BEB24213', 'BEB56262', 'BEB75073']

for produto in produtos:
    if ehalcoolico(produto):
        print('Enviar {} para setor de bebidas alcóolicas'.format(produto))
```

The output of the script is shown in the terminal window below the code, listing all products that contain 'BEB' in uppercase.

## Módulo 3 – O que são módulos e qual a importância

Agora a gente vai aprender a importar um módulo no Python. Mas o que é um **módulo**?

**Módulo** é um arquivo no Python **cheio de código** e que dentro desse código você tem alguns objetos que já foram criados para você usar. Ele não vem por padrão no Python, você precisa importar para o seu código. Por isso, vamos precisar, sempre que necessário, importar esses módulos para dentro de nossos projetos.

Importância do módulo:

- Já tem muita coisa pronta, então você não precisa criar do zero.
- Se você souber usar Módulos e como usar um módulo novo, você vai conseguir fazer praticamente tudo no Python

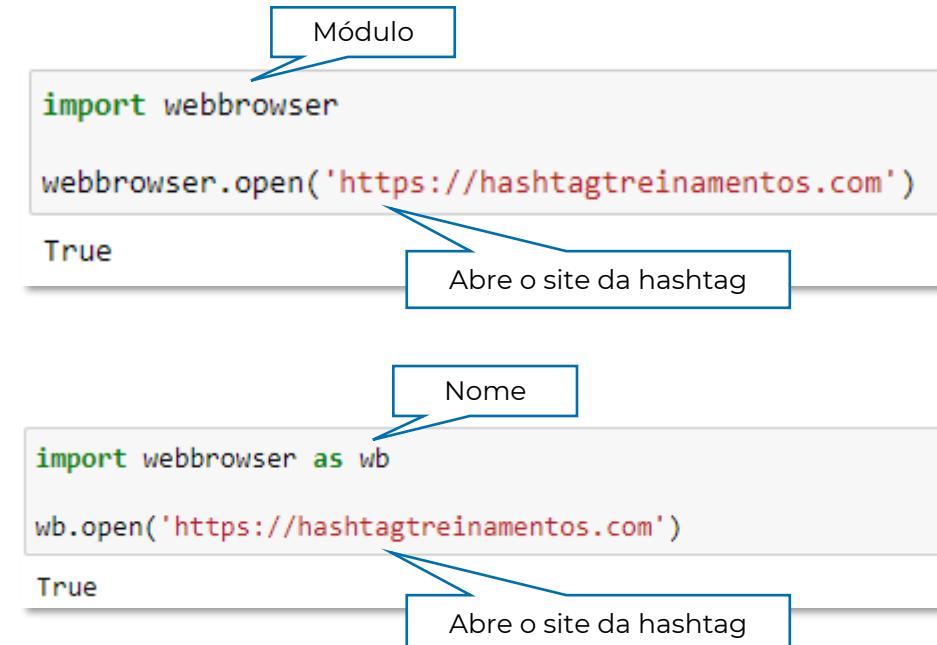
# Módulo 3 – O que são módulos e qual a importância

Para você importar um módulo a estrutura é:

## Import módulo

Ou

## Import módulo as nome (caso você queira mudar o módulo de nome)



## MÓDULO 4

# Pandas e Numpy: As bibliotecas básicas para Ciência de Dados

## Módulo 4 – As bibliotecas básicas para Ciência de Dados / Comparando Pandas e Excel

Não tem como a gente falar de Data Science sem falar do Pandas. O Pandas substitui um monte de funções que a gente iria ter que criar na mão.

Para eu poder importar qualquer biblioteca do Python a gente segue a estrutura:

**Import** pandas **as** pd

In [1]:

```
import pandas as pd
```

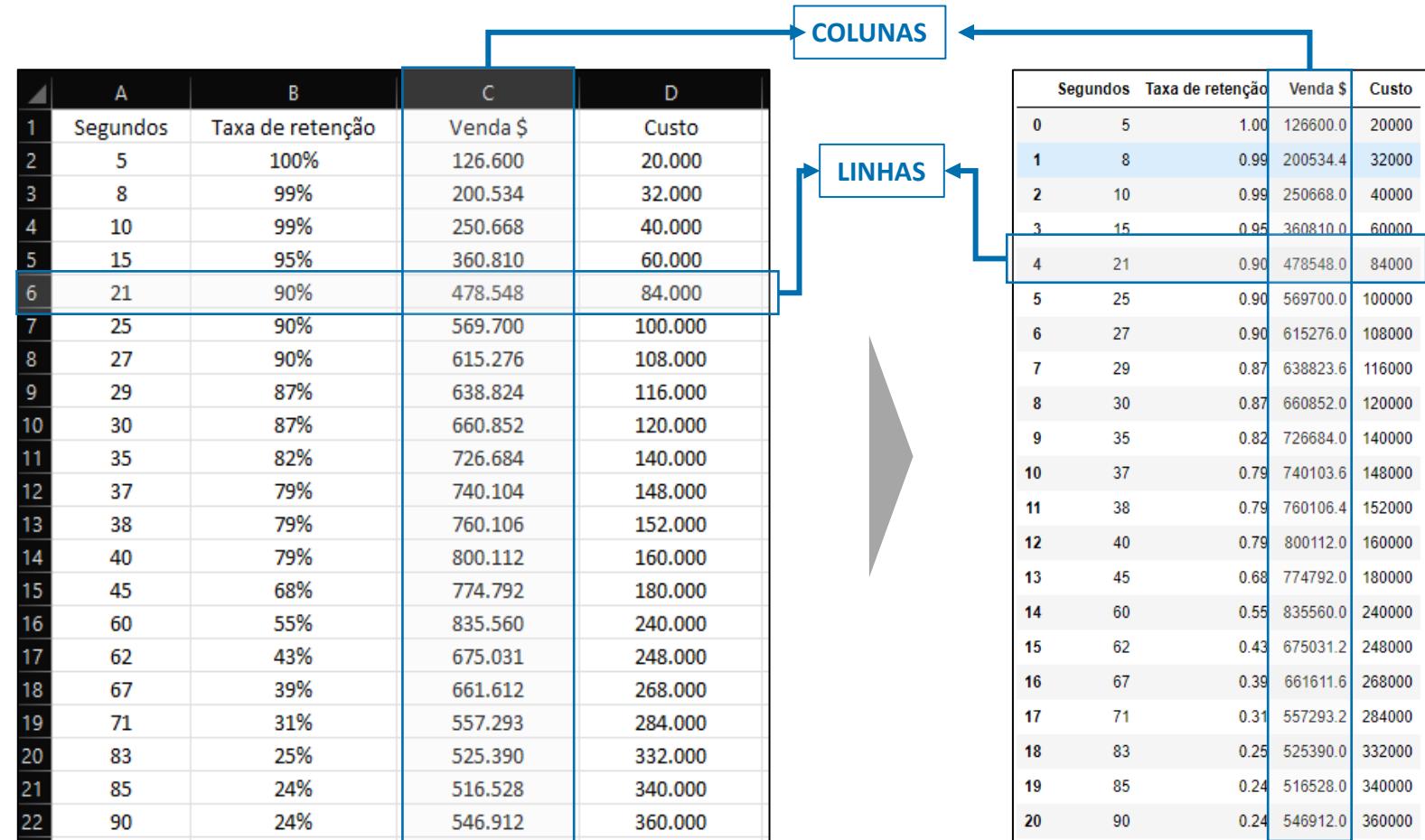
Novo nome

## Módulo 4 – As bibliotecas básicas para Ciência de Dados / Comparando Pandas e Excel

As estruturas ao lado são muito conhecidas por nós: uma composição de linhas e colunas, que chamamos de tabelas.

Essas estruturas estão presentes em todo tipo de ferramenta, como o Excel e o próprio Python.

A diferença é que no Excel chamamos de **tabela**, enquanto no Pandas (Python) a chamamos de **Dataframe**.



# Módulo 4 – As bibliotecas básicas para Ciência de Dados / Comparando Pandas e Excel

Cada coluna desse dataframe a gente vai chamar de series.

	A	B	C	D
1	Segundos	Taxa de retenção	Venda \$	Custo
2	5	100%	126.600	20.000
3	8	99%	200.534	32.000
4	10	99%	250.668	40.000
5	15	95%	360.810	60.000
6	21	90%	478.548	84.000
7	25	90%	569.700	100.000
8	27	90%	615.276	108.000
9	29	87%	638.824	116.000
10	30	87%	660.852	120.000
11	35	82%	726.684	140.000
12	37	79%	740.104	148.000
13	38	79%	760.106	152.000
14	40	79%	800.112	160.000
15	45	68%	774.792	180.000
16	60	55%	835.560	240.000
17	62	43%	675.031	248.000
18	67	39%	661.612	268.000
19	71	31%	557.293	284.000
20	83	25%	525.390	332.000
21	85	24%	516.528	340.000
22	90	24%	546.912	360.000



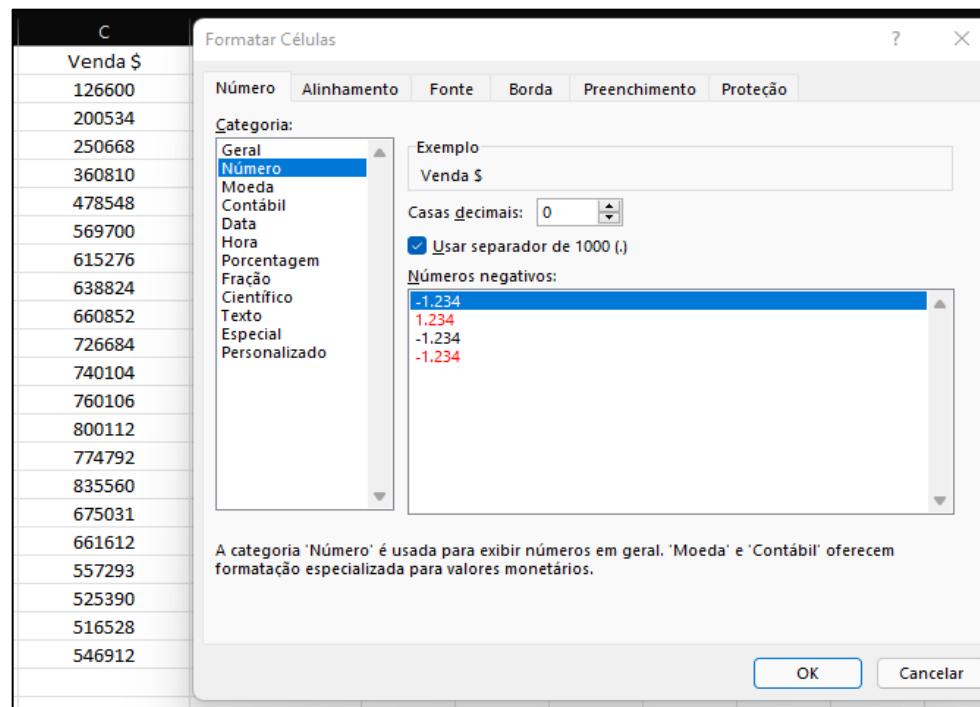
DataFrame

Series

	Segundos	Taxa de retenção	Venda \$	Custo
0	5	1.00	126600.0	20000
1	8	0.99	200534.4	32000
2	10	0.99	250668.0	40000
3	15	0.95	360810.0	60000
4	21	0.90	478548.0	84000
5	25	0.90	569700.0	100000
6	27	0.90	615276.0	108000
7	29	0.87	638823.6	116000
8	30	0.87	660852.0	120000
9	35	0.82	726684.0	140000
10	37	0.79	740103.6	148000
11	38	0.79	760106.4	152000
12	40	0.79	800112.0	160000
13	45	0.68	774792.0	180000
14	60	0.55	835560.0	240000
15	62	0.43	675031.2	248000
16	67	0.39	661611.6	268000
17	71	0.31	557293.2	284000
18	83	0.25	525390.0	332000
19	85	0.24	516528.0	340000
20	90	0.24	546912.0	360000

## Módulo 4 – As bibliotecas básicas para Ciência de Dados / Comparando Pandas e Excel

No Excel colocamos dados para cada célula. Já no pandas, os valores de uma mesma coluna devem respeitar o mesmo tipo de dados.



```
In [10]: base.dtypes
Out[10]: Segundos           int64
          Taxa de retenção   float64
          Venda $            float64
          Custo              int64
          dtype: object
```

Todos os elementos da coluna são do mesmo tipo!

## Módulo 4 – As bibliotecas básicas para Ciência de Dados / Comparando Pandas e Excel

A primeira etapa é pensar em como a gente faria no Excel e ver como se faz no Pandas. Por exemplo, no Excel, se quisermos usar um filtro, utilizamos uma ferramenta para isso; enquanto no Pandas, esse mesmo filtro será feito por meio de uma linha de código.

The screenshot shows a portion of an Excel spreadsheet with columns labeled A through H. Rows 1, 14, and 16 contain data: Row 1 has 'Segundos' in A, 'Taxa de retenção' in B, 'Venda \$' in C, and 'Custo' in D. Row 14 has 40 in A and 79% in B. Row 16 has 60 in A and 55% in B. Below the spreadsheet is a 'Personalizar AutoFiltro' dialog box. In the 'Mostrar linhas onde:' section, 'Venda \$' is selected. Under the 'é maior do que' dropdown, '800000' is entered. The 'E' radio button is selected. At the bottom right of the dialog box, the 'OK' button is highlighted with a blue border.



```
In [19]: base[base["Venda $"] > 800000]
Out[19]:
 Segundos  Taxa de retenção  Venda $  Custo
 12        40            0.79  800112.0  160000
 14        60            0.55  835560.0  240000
```

1 linha de comando

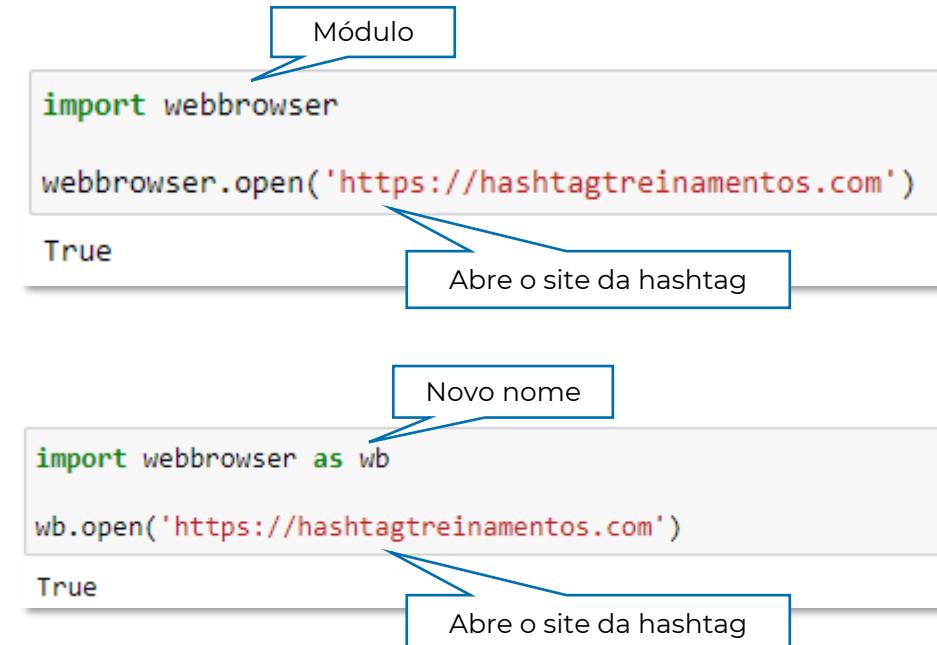
## Módulo 4 – Comparando Excel e Pandas

Para você importar um módulo a estrutura é:

### Import módulo

Ou

### Import módulo as nome (caso você queira mudar o módulo de nome)



## Módulo 4 – Comparando Excel e Pandas na prática

Agora vamos ver na prática como funciona o Pandas.

Vamos pegar essa tabela que a gente fez no Excel e trazer para o Pandas, mas antes vamos importar a biblioteca utilizando o comando **import**.

**Atenção:** Toda vez que a gente for trabalhar com qualquer biblioteca no Python, primeiro temos que importá-la.

In [1]:

```
import pandas as pd
```

Comando

Biblioteca

Novo nome

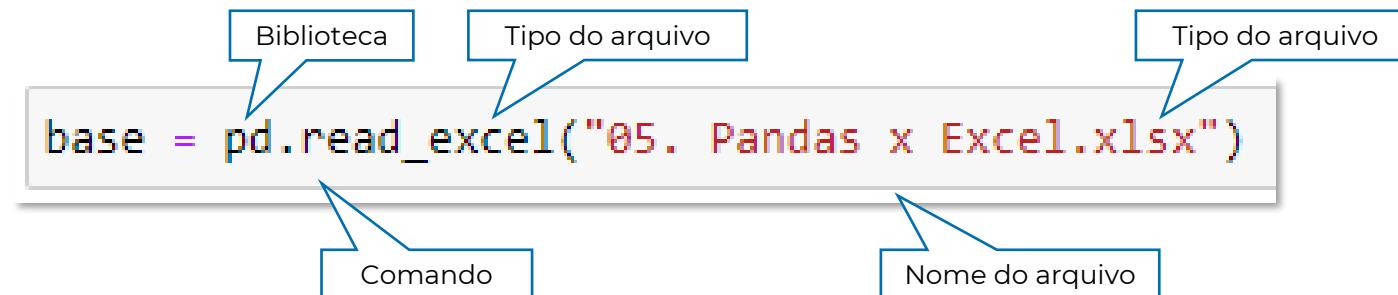
## Módulo 4 – Comparando Excel e Pandas na prática

Com o Pandas a gente consegue ler qualquer arquivo com vários formatos diferentes (CSV, html, excel, etc) utilizando a função **read**.

A estrutura é simples:

**biblioteca.read\_tipodearquivo("Nomedoarquivo.tipodoarquivo")**

Nesse caso, estamos trabalhando com um arquivo em Excel, conforme a imagem abaixo:



## Módulo 4 – Comparando Excel e Pandas na prática

No Excel a gente chama de tabela, enquanto no Pandas a gente chama de Dataframe.

Para ver a tabela no Excel basta eu abrir a tabela, enquanto no Python é necessário usar o comando **head**.

base.head(21)				
	Segundos	Taxa de retenção	Venda \$	Custo
0	5	1.00	126600.0	20000
1	8	0.99	200534.4	32000
2	10	0.99	250668.0	40000
3	15	0.95	360810.0	60000
4	21	0.90	478548.0	84000
5	25	0.90	569700.0	100000
6	27	0.90	615276.0	108000
7	29	0.87	638823.6	116000
8	30	0.87	660852.0	120000
9	35	0.82	726684.0	140000
10	37	0.79	740103.6	148000
11	38	0.79	760106.4	152000
12	40	0.79	800112.0	160000
13	45	0.68	774792.0	180000
14	60	0.55	835560.0	240000
15	62	0.43	675031.2	248000
16	67	0.39	661611.6	268000
17	71	0.31	557293.2	284000
18	83	0.25	525390.0	332000
19	85	0.24	516528.0	340000
20	90	0.24	546912.0	360000

## Módulo 4 – Comparando Excel e Pandas na prática

É possível ver o tipo de cada serie no Pandas utilizando o comando **dtypes**.

No Excel é possível mudar o tipo de uma linha, enquanto no Pandas modificamos para a coluna inteira.

Além disso, se um dos dados estiver como texto o Pandas vai entender a coluna inteira como texto, mas existem formas de tratar isso.

base.dtypes	
Segundos	int64
Taxa de retenção	float64
Venda \$	float64
Custo	int64
dtype:	object



The screenshot shows a Microsoft Excel interface. At the top, there's a ribbon with tabs like 'Arquivo', 'Home', 'Inserir', etc. Below the ribbon, the formula bar displays 'C2' and the value '126600'. To the right of the formula bar is a 'Número' (Number) button with options for 'Moeda' (Currency), 'Porcentagem' (Percentage), and 'Número' (Number). The main area shows a table with four columns and four rows of data. The columns are labeled A, B, C, D, E, F, G. The first row contains column headers: 'Segundo', 'Taxa de retenção', 'Venda \$', and 'Custo'. The second row contains values: '5', '100%', '126.600', and '20.000'. The third row contains values: '8', '99%', '200.534', and '32.000'. The fourth row contains values: '10', '99%', '250.668', and '40.000'. The 'Número' button is highlighted in red.

A	B	C	D
Segundo	Taxa de retenção	Venda \$	Custo
5	100%	126.600	20.000
8	99%	200.534	32.000
10	99%	250.668	40.000

# Módulo 4 – Comparando Excel e Pandas na prática

Agora vamos comparar o uso de filtros:

Excel:

The screenshot shows the 'Filtros de Número' (Number Filters) dialog box in Excel. The 'É Maior do que...' (Greater than...) option is selected. The underlying table has the following data:

Segundos	Taxa de retenção	Venda \$	Custo
5		20.000	
8		32.000	
10		40.000	
15		60.000	
21		84.000	
25		100.000	
27		108.000	
29			
30			
35			
37			
38			
40			
45			
60			
62			
67			
71			
83			
85			
90			

A	B	C	D
Segundos	Taxa de retenção	Venda \$	Custo
14	40	79%	800.112
16	60	55%	835.560

Pandas:

base[base["Venda \$"] > 800000]

Segundos	Taxa de retenção	Venda \$	Custo
12	40	0.79	800112.0
14	60	0.55	835560.0



# Módulo 4 – Comparando Excel e Pandas na prática

Ambos podem calcular a média utilizando uma fórmula:

Excel:

A	B	C	D	E	F	G
Segundo	Taxa de retenção	Venda \$	Custo		=MÉDIA(C2:C22)	
5	100%	126.600	20.000			
8	99%	200.534	32.000			
10	99%	250.668	40.000			
15	95%	360.810	60.000			
21	90%	478.548	84.000			
25	90%	569.700	100.000			
27	90%	615.276	108.000			
29	87%	638.824	116.000			
30	87%	660.852	120.000			
35	82%	726.684	140.000			
37	79%	740.104	148.000			
38	79%	760.106	152.000			
40	79%	800.112	160.000			
45	68%	774.792	180.000			
60	55%	835.560	240.000			
62	43%	675.031	248.000			
67	39%	661.612	268.000			
71	31%	557.293	284.000			
83	25%	525.390	332.000			
85	24%	516.528	340.000			
90	24%	546.912	360.000			

572.473,14

Pandas:



base["Venda \$"].mean()

572473.1428571427

## Módulo 4 – Comparando Excel e Pandas na prática

No Pandas é possível obter todas as informações estatísticas de uma só vez utilizando o comando **describe**.

No Pandas é possível obter algumas informações de forma mais rápida que o Excel.

```
base["Venda $"].describe()
```

count	21.000000
mean	572473.142857
std	197752.952190
min	126600.000000
25%	516528.000000
50%	615276.000000
75%	726684.000000
max	835560.000000
Name:	Venda \$, dtype: float64

## Módulo 4 – Introdução ao NumPy: A importância do NumPy

O NumPy é a base do Pandas.

É basicamente a biblioteca *Numerical Python*, ou seja, como o próprio nome diz, é uma biblioteca de **computação numérica** e é otimizada para cálculos mais pesados.

No NumPy vamos trabalhar muito com **arrays**, que são mais **rápidos** e mais **performáticos** que as listas.

Toda vez que a gente precisa importar uma biblioteca, a gente usa o comando `import` para importar a biblioteca que a gente deseja, além da possibilidade de adicionar um apelido (alias) que vai ajudar a gente a escrever o código.

## Módulo 4 – Introdução ao NumPy: A importância do NumPy

Toda vez que precisarmos importar uma biblioteca, vamos usar o comando **import**.

Estrutura:

**import** biblioteca **as** apelido

Utilizando o apelido fica mais fácil de digitar o código, pois dessa forma não precisamos escrever o nome completo da biblioteca, e sim apenas o apelido, o que otimiza bastante o nosso código.

```
# Importar o numpy  
import numpy as np
```

Comando

Biblioteca

Alias (apelido)

## Módulo 4 – Introdução ao NumPy: A importância do NumPy

Para exemplificar a importância do **numpy** vamos colocar uma lista com 2 valores: venda\_valor e comissão.

Vamos tentar calcular quanto cada funcionário vai ganhar de comissão utilizando as listas.

Repare que ocorreu um erro, pois não podemos fazer essa operação entre listas, ou seja, não conseguimos fazer de uma forma rápida.

```
# Venda e comissão
venda_valor = [150000, 230000, 82000, 143000, 184000]
comissao = [5, 8, 8, 5, 12]
```

```
# Ao tentar fazer essa operação com listas, teremos um erro
venda_valor*(comissao/100)
```

```
-----  
TypeError                                 Traceback (most recent call last)  
Input In [3], in <cell line: 2>()  
      1 # Ao tentar fazer essa operação com listas, teremos um erro  
----> 2 venda_valor*(comissao/100)  
  
TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

## Módulo 4 – Introdução ao NumPy: A importância do NumPy

Para resolver esse problema a gente pode transformar essas 2 listas em um array.

Para transformar lista em um array:

```
meu_array = np.array(minha_lista)
```

Agora, com o array conseguimos aplicar aquela função que tentamos anteriormente.

```
# Transformando em um array  
venda_valor = np.array(venda_valor)  
comissao = np.array(comissao)
```

```
# Agora vamos conseguir fazer sendo um array  
venda_valor*(comissao/100)
```

```
array([ 7500., 18400., 6560., 7150., 22080.])
```

## Módulo 4 – Introdução ao NumPy: Propriedades de um array

Agora vamos entender as propriedades do array e como ele funciona.

Uma informação importante que vamos utilizar muito é a dimensão do array, que é feita pela estrutura abaixo.

Estrutura: **array.shape**

```
print(venda_valor)  
[150000 230000 82000 143000 184000]  
  
venda_valor.shape  
(5,)
```

## Módulo 4 – Introdução ao NumPy: Propriedades de um array

Para saber o tipo dos dados, podemos fazer:

Estrutura: **meu\_array.dtype**

No exemplo ao lado, o resultado do tipo foi int32, ou seja, nesse array os nossos valores são inteiros.

```
venda_valor.dtype  
dtype('int32')
```

## Módulo 4 – Introdução ao NumPy: Propriedades de um array

Se eu tenho dados diferentes tipos (inteiro e texto), ele converte tudo para o mesmo tipo de dado.

No exemplo ao lado, temos que apenas um valor é um ponto flutuante (float).

Porém, ao converter os valores para o mesmo tipo de dado, todos ficam com casa decimal.

The diagram illustrates three examples of NumPy arrays with mixed data types and their conversion to uniform data types:

- Example 1:** `tipos_dados = np.array([1,2,3,4,5])`  
Output: [1 2 3 4 5]  
Type: `dtype('int32')` (labeled "Inteiro")
- Example 2:** `tipos_dados = np.array([1,2,3,4,5.0])`  
Output: `float64`  
[1. 2. 3. 4. 5.]  
Type: `dtype('float64')` (labeled "Ponto flutuante")
- Example 3:** `tipos_dados = np.array([1,2,3,'4',5])`  
Output: `<U11`  
['1' '2' '3' '4' '5']  
Type: `dtype('U11')` (labeled "Texto")

## Módulo 4 – Introdução ao NumPy: Propriedades de um array

O array de uma dimensão é o que chamamos de vetor e o de duas dimensões é o que chamamos de matriz.

```
dados_venda = np.array([[150000, 230000, 82000, 143000, 184000], [5, 8, 8, 5, 12]])
```

```
print(dados_venda)
```

```
[[150000 230000 82000 143000 184000]
 [ 5       8       8       5       12]]
```

2 linhas e 5 colunas

```
dados_venda.shape
```

```
(2, 5)
```

```
dados_venda.dtype
```

Inteiro

```
dtype('int32')
```

# Módulo 4 – Introdução ao NumPy: Trabalhando com um array

Agora vamos aprender como trabalhamos com um array.

Para buscar elementos em um array utilizamos as estruturas abaixo:

- Buscar por linha:

**array[N]**

- Buscar por linha e coluna:

**array[N][N]**

`dados_venda`

```
array([[ 'Lucas', 'Bia', 'Jean', 'Gabi', 'Pedro'],
       [ '150000', '230000', '82000', '143000', '184000]], dtype='<U11')
```

*# Buscando a primeira Linha (Lembrando que o Python começa no 0)*  
`dados_venda[0]`

```
array(['Lucas', 'Bia', 'Jean', 'Gabi', 'Pedro'], dtype='<U11')
```

*# Buscando na primeira Linha, a segunda coluna*  
`dados_venda[0][1]`

'Bia'  
Linha      Coluna

`dados_venda[1][2]`

'82000'

## Módulo 4 – Introdução ao NumPy: Trabalhando com um array

Além disso, podemos mostrar as informações estatísticas de um array.

Vejamos alguns exemplos:

- Média:

Definição: Soma de todos os valores dividido pela quantidade de valores.

Estrutura: **np.mean(meu\_array)**

```
# Média da primeira linha  
np.mean(dados_venda[0])
```

157800.0

Primeira  
linha

```
# Média da segunda linha  
np.mean(dados_venda[1])
```

Segunda  
linha

## Módulo 4 – Introdução ao NumPy: Trabalhando com um array

- Mediana:

Definição: Valor que está no meio da minha distribuição.

Estrutura: **np.median(meu\_array)**

```
# Mediana da primeira linha  
np.median(dados_venda[0])
```

```
150000.0
```

Primeira  
linha

## Módulo 4 – Introdução ao NumPy: Trabalhando com um array

- Desvio Padrão:

**np.std(meu\_array)**

```
# Desvio padrão da primeira linha  
np.std(dados_venda[0])
```

48836.05225650411

Primeira  
linha

## Módulo 4 – Introdução ao NumPy: Trabalhando com um array

```
# Considerando esses 2 arrays
array1 = np.array([1,2,3,4,5])
array2 = np.array([7,8,9,10,11])
```

```
# Soma
array1 + array2
```

```
array([ 8, 10, 12, 14, 16])
```

1 + 7

Podemos também realizar operações aritméticas, agregação, comparação, mínimo e máximo, etc.

- Soma:

Estrutura: **array1 + array2**

## Módulo 4 – Introdução ao NumPy: Trabalhando com um array

- Multiplicação
  - Por um valor:

Estrutura: **array1\*valor**

- De arrays:

Estrutura: **array1\* array2**

```
# Considerando esses 2 arrays  
array1 = np.array([1,2,3,4,5])  
array2 = np.array([7,8,9,10,11])
```

Multiplicação de 2 arrays

```
# Multiplicação por um valor  
array1 * 2
```

```
array([ 2,  4,  6,  8, 10])
```

1x2

```
# Multiplicação de arrays  
array1 * array2
```

```
array([ 7, 16, 27, 40, 55])
```

1x7

## Módulo 4 – Introdução ao NumPy: Trabalhando com um array

- Potência
- Soma dos valores da primeira linha

Estrutura: **np.power(array2, array1)**

Estrutura: **np.sum(array1)**

```
# Considerando esses 2 arrays  
array1 = np.array([1,2,3,4,5])  
array2 = np.array([7,8,9,10,11])
```

```
# Potência  
np.power(array2, array1)
```

```
array([ 7, 64, 729, 10000, 161051], dtype=int32)
```

7 elevado a 1

```
# Soma dos valores da primeira linha  
np.sum(array1)
```

15

$$1 + 2 + 3 + 4 + 5 = 15$$

# Módulo 4 – Introdução ao NumPy: Trabalhando com um array

- Soma acumulada

Estrutura: **np.cumsum(array2, array1)**

- Mínimo

Estrutura: **array1.min**

- Máximo

Estrutura: **array1.máx**

```
# Considerando esses 2 arrays  
array1 = np.array([1,2,3,4,5])  
array2 = np.array([7,8,9,10,11])
```

```
# Soma acumulada dos valores da primeira Linha  
np.cumsum(array1)
```

```
array([ 1,  3,  6, 10, 15], dtype=int32)
```

```
# Valor mínimo  
array1.min()
```

1  
Menor valor do array 1

```
# Valor máximo  
array1.max()
```

5  
Maior valor do array 1

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

O que é Pandas? Por que ele é importante?

O Pandas possui o Dataframe, que é uma estrutura que vamos utilizar em várias partes do nosso projeto de Data Science. Além disso, com o pandas conseguimos:

- Visualizar bases de dados;
- Analisar dados;
- Visualizar resumo estatístico das informações;
- Obter informações sobre a base de dados importada;
- Fazer gráficos e tabelas.

O pandas tem uma estrutura muito **amigável** e uma **alta performance**, muito fácil para trabalhar com dados e análises exploratórias.

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

O primeiro passo é importar o Pandas. Fazemos isso com o seguinte código:

**Import** pandas **as** pd

```
# Importando o pandas  
import pandas as pd
```

Com o Pandas a gente consegue ler qualquer arquivo com vários formatos diferentes (CSV, html, excel, etc), utilizando a função `read`.

A estrutura é simples:

**biblioteca.read\_tipodearquivo("Nomedoarquivo.tipodoarquivo")**

```
Importar uma base em csv:  
pd.read_csv("arquivo_excel.xlsx")
```

```
base = pd.read_csv("07. Pandas.csv")
```

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Fazendo um paralelo com o excel, ao abrir a base no Excel, já era possível visualizar todas a informações; no python não é tão direto assim.

Se a gente digitar apenas o código e não atribuí-lo a uma variável, o pandas só vai fazer a leitura, mas não irá salvar. Por isso, para atribuir a base em uma variável digitamos o nome da variável antes do sinal de igual (=).

**Variável = biblioteca.read\_tipodearquivo("Nomedoarquivo.tipodoarquivo")**

No exemplo abaixo a nossa base está salva na variável base.

```
base = pd.read_csv("07. Pandas.csv")
```

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

A função head mostra a visualização das **5 primeiras linhas** da base: títulos das colunas e os dados que existem nelas.

Dessa forma é possível ter uma noção de como está a base de dados importada.

Estrutura: **variável.head()**

As 5 primeiras linhas:  
base.head()

# Visualizando as 5 primeiras Linhas  
base.head()

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55

# Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

As 5 primeiras linhas:  
base.head()

# Visualizando as 5 primeiras Linhas  
base.head()

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55

Com essa função já foi possível notar que na coluna “Last Week Rank” existem linhas com traço (-).

O que isso significa esse **traço** (-)?

Em alguns casos significa **coluna vazia**, mas isso vamos ver mais para frente.

Por enquanto vamos guardar a informação que existem algumas colunas com traço.

# Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

base.head(15)										
	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55
5	2020-04-01	6	6	-	Car Masters: Rust to Riches	TV Show	Yes	Sep 14, 2018	4	14
6	2020-04-01	7	10	-	Unorthodox	TV Show	Yes	Mar 26, 2020	2	5
7	2020-04-01	8	7	5	Love is Blind	TV Show	Yes	Feb 13, 2020	9	40
8	2020-04-01	9	8	-	Badland	Movie	NaN	Mar 26, 2020	4	11
9	2020-04-01	10	9	-	Uncorked	Movie	Yes	Mar 27, 2020	4	15
10	2020-04-02	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	10	100
11	2020-04-02	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	6	54
12	2020-04-02	3	3	2	All American	TV Show	NaN	Mar 28, 2019	10	84
13	2020-04-02	4	-	-	Nailed It!	TV Show	Yes	Mar 9, 2018	1	7
14	2020-04-02	5	-	-	How to Fix a Drug Scandal	TV Show	Yes	Apr 1, 2020	1	6

O head pode ser utilizado para mostrar outras quantidades de linhas, além do padrão das 5 primeiras linhas.

Como por exemplo: visualizar 15 primeiras.

Vamos usar o código:

**base.head(nº de linhas)**

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Da mesma forma podemos visualizar as últimas 5 linhas, usando a função tail.

A estrutura é a mesma: **base.tail()**

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
7095	2022-03-11	6	5	1	Worst Roommate Ever	TV Show	Yes	Mar 1, 2022	10	81
7096	2022-03-11	7	7	2	Vikings: Valhalla	TV Show	Yes	Feb 25, 2022	14	100
7097	2022-03-11	8	8	-	Shooter	Movie	NaN	Aug 1, 2014	3	7
7098	2022-03-11	9	9	7	Shrek 2	Movie	NaN	Mar 1, 2022	10	33
7099	2022-03-11	10	10	-	Shrek	Movie	NaN	May 1, 2018	7	12

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Podemos escolher da mesma forma que o **head** o número de linhas que queremos visualizar. Por exemplo: visualizar as últimas 3 linhas.

A estrutura é a mesma: **base.tail(nº de linhas)**

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
7097	2022-03-11	8	8	-	Shooter	Movie	NaN	Aug 1, 2014	3	7
7098	2022-03-11	9	9	7	Shrek 2	Movie	NaN	Mar 1, 2022	10	33
7099	2022-03-11	10	10	-	Shrek	Movie	NaN	May 1, 2018	7	12

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Então basicamente com essas duas funções temos a visualização do topo e o final da nossa base.

Geralmente quando se tem algum problema com dado incompleto ou errado, com as primeiras e últimas linhas já é possível ter noção do erro.

In [7]: # Visualizando as 5 primeiras Linhas  
base.head()

Out[7]:

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55

As 5 últimas linhas:  
base.tail()

In [6]: # Visualizando as 5 últimas Linhas  
base.tail()

Out[6]:

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
7095	2022-03-11	6	5	1	Worst Roommate Ever	TV Show	Yes	Mar 1, 2022	10	81
7096	2022-03-11	7	7	2	Vikings: Valhalla	TV Show	Yes	Feb 25, 2022	14	100
7097	2022-03-11	8	8	-	Shooter	Movie	NaN	Aug 1, 2014	3	7
7098	2022-03-11	9	9	7	Shrek 2	Movie	NaN	Mar 1, 2022	10	33
7099	2022-03-11	10	10	-	Shrek	Movie	NaN	May 1, 2018	7	12

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Uma outra informação importante é saber quantas linhas e colunas temos nessa base e para isso utilizamos a fórmula `shape` que vai indicar exatamente o **tamanho** da nossa base.

Estrutura: **base.shape**

Nessa caso, a base de dados tem 7100 linhas e 10 colunas.

```
# Tamanho da base  
base.shape
```

(7100, 10)

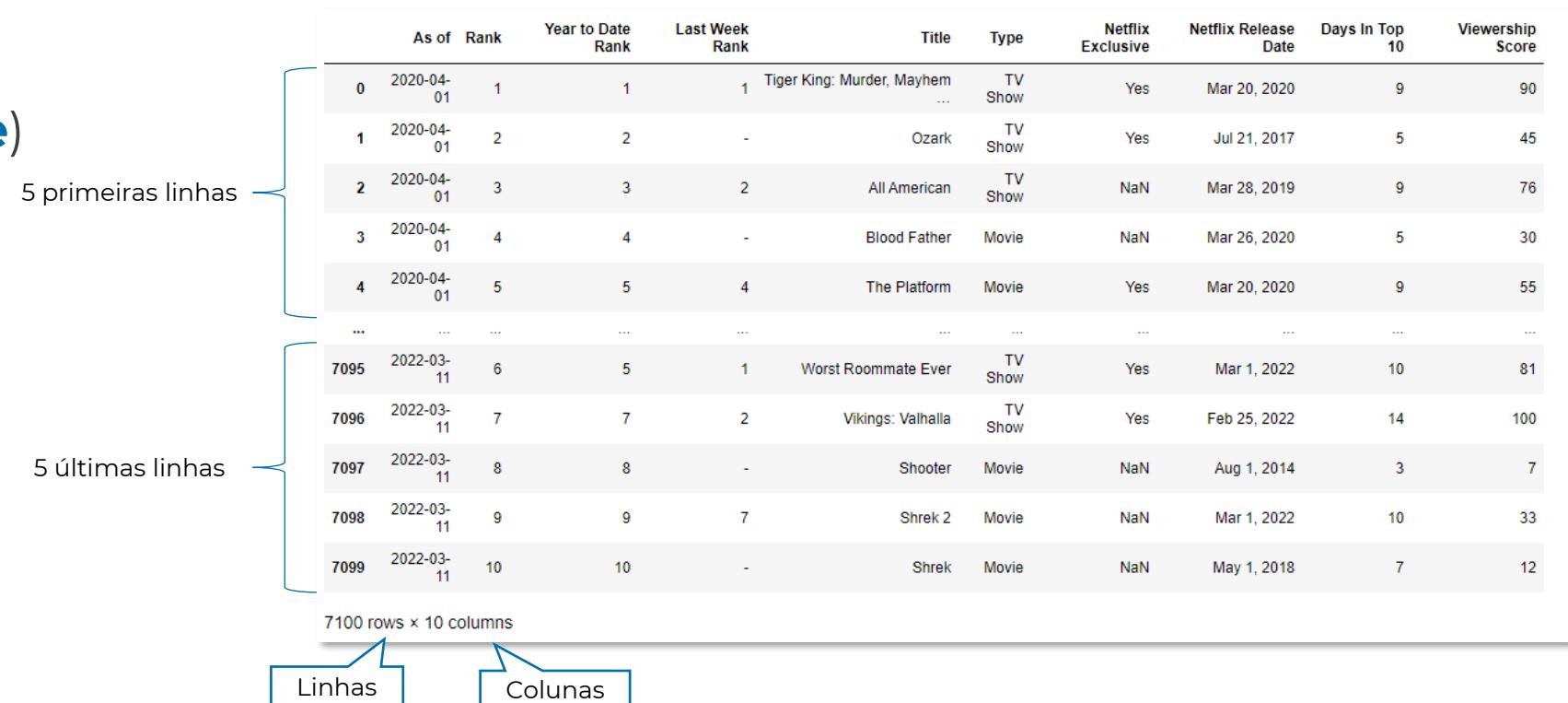
Linhas

Colunas

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Uma outra opção é a função `display`, que mostra a visualização dessas três funções juntas (`head`, `tail` e `shape`), ou seja, mostra as primeiras e últimas 5 linhas, número de linhas e colunas.

Estrutura: `display(base)`



	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55
...	...	...	...	...	...	...	...	...	...	...
7095	2022-03-11	6	5	1	Worst Roommate Ever	TV Show	Yes	Mar 1, 2022	10	81
7096	2022-03-11	7	7	2	Vikings: Valhalla	TV Show	Yes	Feb 25, 2022	14	100
7097	2022-03-11	8	8	-	Shooter	Movie	NaN	Aug 1, 2014	3	7
7098	2022-03-11	9	9	7	Shrek 2	Movie	NaN	Mar 1, 2022	10	33
7099	2022-03-11	10	10	-	Shrek	Movie	NaN	May 1, 2018	7	12

7100 rows x 10 columns

Linhos

Colunas

## Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Então fiquem a vontade para escolher qual função usar: as três funções (**head**, **tail** e **shape**) ou a função **display**. Ambas podem ser utilizadas para começar a ver seus dados.

Então, notem que em todo projeto de ciência de dados será necessário fazer os passos:



## Módulo 4 – Introdução ao Pandas: DataFrame e Series

A base que importamos está como DataFrame, uma forma fácil de confirmar é digitando a função type:

Estrutura: **type(base)**

```
type(base)
```

```
pandas.core.frame.DataFrame
```

Repare que o resultado retornado para o tipo é **DataFrame**

Isso é muito importante para usar em qualquer parte do projeto de Ciência de Dados, pois as outras bibliotecas que vamos usar esperam receber já um **DataFrame** e se não estiver nesse tipo é necessário converter.

Essa é uma vantagem do pandas porque assim que ele importa a base, ela já é um **DataFrame**.

## Módulo 4 – Introdução ao Pandas: DataFrame e Series

Supondo que temos um dicionário de funcionários, venda e comissão.

Ao utilizar o código **type(dic)** para descobrir o tipo, como mostra a imagem ao lado, o código retorna o resultado do tipo como dicionário (**dict**).

```
# Criando o dicionários
dic = {
    "funcionarios": ['Lucas', 'Bia', 'Jean', 'Gabi', 'Pedro'],
    "venda_valor": [150000, 230000, 82000, 143000, 184000],
    "comissao": [5, 8, 8, 5, 12]
}
```

The diagram illustrates the execution flow of the code. It starts with the code block containing the dictionary definition. An arrow points from the closing brace of the dictionary definition to the `type(dic)` call. Another arrow points from the `type(dic)` call to the resulting value `dict`.

## Módulo 4 – Introdução ao Pandas: DataFrame e Series

Como transformar o dicionário em DataFrame?

O pandas possui uma função:

**pd.DataFrame(varável)**

Essa função transforma automaticamente esses valores em um **DataFrame**.

Então para transformar o dicionário em DataFrame é só utilizar o código:

**pd.DataFrame(dic)**

```
# Transformando esse dicionário em um DataFrame  
base_dic = pd.DataFrame(dic)
```

```
base_dic
```

	funcionarios	venda_valor	comissao
0	Lucas	150000	5
1	Bia	230000	8
2	Jean	82000	8
3	Gabi	143000	5
4	Pedro	184000	12

## Módulo 4 – Introdução ao Pandas: DataFrame e Series

Apenas com uma linha de código transformamos um dicionário em um DataFrame, que é muito mais visual. Essa transformação nós vamos fazer muito no projeto de ciência de dados.

Dicionário:

```
dic
{'funcionarios': ['Lucas', 'Bia', 'Jean', 'Gabi', 'Pedro'],
 'venda_valor': [150000, 230000, 82000, 143000, 184000],
 'comissao': [5, 8, 8, 5, 12]}
```



DataFrame:

```
# Transformando esse dicionário em um DataFrame
base_dic = pd.DataFrame(dic)
```

```
base_dic
```

	funcionarios	venda_valor	comissao
0	Lucas	150000	5
1	Bia	230000	8
2	Jean	82000	8
3	Gabi	143000	5
4	Pedro	184000	12

## Módulo 4 – Introdução ao Pandas: DataFrame e Series

Como dito anteriormente na comparação de Excel e pandas, no **Excel** temos as **colunas** e no **pandas** temos o que chamamos de **series**. Mas o que seriam essas **series**?

Por exemplo, se o nosso objetivo fosse visualizar somente a coluna funcionários desse DataFrame, essa coluna vai ser uma **series** e vai ter suas próprias propriedades.

```
series_funcionarios = base_dic["funcionarios"]

series_funcionarios
0    Lucas
1     Bia
2    Jean
3    Gabi
4   Pedro
Name: funcionarios, dtype: object
```

Exemplo: na series funcionários temos os valores e um índice, então ela é diferente de um **array** de valores e por isso temos que saber como trabalhar com ela.

## Módulo 4 – Introdução ao Pandas: DataFrame e Series

Para passar os valores da series funcionários não podemos simplesmente digitar:  
**series\_funcionario**.

Temos que utilizar uma função mais específica: **series\_funcionarios.values**.

```
series_funcionarios.values
```

```
array(['Lucas', 'Bia', 'Jean', 'Gabi', 'Pedro'], dtype=object)
```

## Módulo 4 – Introdução ao Pandas: DataFrame e Series

O mesmo podemos fazer para visualizar somente o índice:  
**series\_funcionarios.index**

Então o índice começa em 0, termina em 5 (o 5 não é incluído) e varia de 1 em 1.

```
series_funcionarios.index
```

```
RangeIndex(start=0, stop=5,
```

## Módulo 4 – Introdução ao Pandas: DataFrame e Series

É necessário entender as definições de DataFrame e Series para saber como trabalhar.

Agora quando digitamos o tipo da **serie\_funcionários**, a função **type** retorna que é um tipo série.

```
type(series_funcionarios)  
pandas.core.series.Series
```

## Módulo 4 – Introdução ao Pandas: DataFrame e Series

Para transformar um Serie em DataFrame, vamos passar a coluna com colchetes duplos, ao invés colchetes simples.

```
Dataframe_funcionários = base_dic[["funcionários"]]
series_funcionários = base_dic ["funcionários"]
```

```
dataframe_funcionarios = base_dic[["funcionarios"]]

dataframe_funcionarios

  funcionários
  0    Lucas
  1     Bia
  2    Jean
  3    Gabi
  4   Pedro

type(dataframe_funcionarios)

pandas.core.frame.DataFrame
```



```
series_funcionarios = base_dic["funcionarios"]

series_funcionarios

  0    Lucas
  1     Bia
  2    Jean
  3    Gabi
  4   Pedro
Name: funcionarios, dtype: object

type(series_funcionarios)

pandas.core.series.Series
```

## Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Podemos aplicar vários conceitos que vimos no NumPy no Pandas, como por exemplo:

- Mostrar o tipo de dados

Estrutura: **base.dtypes**

```
# Mostrando somente o tipo dos dados  
base.dtypes
```

As of	object
Rank	int64
Year to Date Rank	object
Last Week Rank	object
Title	object
Type	object
Netflix Exclusive	object
Netflix Release Date	object
Days In Top 10	int64
Viewership Score	int64
dtype: object	

Objeto

Inteiro

## Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Uma outra função que podemos utilizar é a **info**, que vai nos mostrar o tipo de dados e os valores vazios.

Estrutura: **base.info**

Entrada de 7100  
e 4599 valores  
não nulos, então  
temos 2501  
valores nulos

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7100 entries, 0 to 7099
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   As of            7100 non-null    object  
 1   Rank             7100 non-null    int64  
 2   Year to Date Rank 7100 non-null    object  
 3   Last Week Rank   7100 non-null    object  
 4   Title            7100 non-null    object  
 5   Type             7100 non-null    object  
 6   Netflix Exclusive 4599 non-null    object  
 7   Netflix Release Date 7100 non-null    object  
 8   Days In Top 10   7100 non-null    int64  
 9   Viewership Score 7100 non-null    int64  
dtypes: int64(3), object(7)
memory usage: 554.8+ KB
```

# Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

A função `isnull` informa se uma coluna é nula.

Estrutura: `base.isnull()`

base.isnull()										
	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	False	False		False	False	False	False	False	False	False
1	False	False		False	False	False	False	False	False	False
2	False	False		False	False	False	True	False	False	False
3	False	False		False	False	False	True	Coluna nula	False	False
4	False	False		False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
7095	False	False		False	False	False	False	False	False	False
7096	False	False		False	False	False	False	False	False	False
7097	False	False		False	False	False	True	False	False	False
7098	False	False		False	False	False	True	False	False	False
7099	False	False		False	False	False	True	False	False	False
7100 rows × 10 columns										

## Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Para visualizar a coluna nula podemos utilizar a função **head**.

Estrutura: **base.head()**

base.head()										
	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55

Coluna nula

## Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Podemos utilizar a estrutura abaixo para contar os valores vazios por coluna:

Estrutura: **base.isnull().sum()**

Entrada de 7100  
e 4599 valores  
não nulos, então  
temos 2501  
valores nulos

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7100 entries, 0 to 7099
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   As of            7100 non-null    object  
 1   Rank             7100 non-null    int64  
 2   Year to Date Rank 7100 non-null    object  
 3   Last Week Rank  7100 non-null    object  
 4   Title            7100 non-null    object  
 5   Type             7100 non-null    object  
 6   Netflix Exclusive 4599 non-null    object  
 7   Netflix Release Date 7100 non-null    object  
 8   Days In Top 10   7100 non-null    int64  
 9   Viewership Score 7100 non-null    int64  
dtypes: int64(3), object(7)
memory usage: 554.8+ KB
```

## Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Antes de falarmos de estatística, vamos precisar selecionar apenas as colunas que vamos trabalhar para evitar esse erro ao fazer a média.

```
base.mean()
```

```
C:\Users\JEANLU~1\AppData\Local\Temp\ipykernel_4228\4202722464.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.  
base.mean()
```

Esse erro acontece porque estamos tentando tirar a média de colunas que não possuem esse valor.

Então o que nós vamos fazer é pegar apenas as colunas onde é possível tirar a média e separar em uma nova base.

## Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

É muito simples buscar por uma coluna no DataFrame, podemos usar o nome da coluna entre aspas ou usar o ponto.

Estrutura: **base["Title"]**

```
# Podemos usar o nome da coluna entre aspas
base["Title"]

0      Tiger King: Murder, Mayhem ...
1                      Ozark
2          All American
3          Blood Father
4          The Platform
...
7095    Worst Roommate Ever
7096      Vikings: Valhalla
7097          Shooter
7098          Shrek 2
7099          Shrek
Name: Title, Length: 7100, dtype: object
```

Estrutura: **base.Title**

```
# Ou usar o ponto
base.Title

0      Tiger King: Murder, Mayhem ...
1                      Ozark
2          All American
3          Blood Father
4          The Platform
...
7095    Worst Roommate Ever
7096      Vikings: Valhalla
7097          Shooter
7098          Shrek 2
7099          Shrek
Name: Title, Length: 7100, dtype: object
```



## Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

**Observação:** O ponto só vai funcionar quando o nome da coluna não possuir espaço.

Estrutura: **base["Title"]**

```
base['Year to Date Rank']
```

0	1
1	2
2	3
3	4
4	5
..	
7095	5
7096	7
7097	8
7098	9
7099	10

Name: Year to Date Rank, Length: 7100, dtype: object



Estrutura: **base.Title**

```
base.Year to Date Rank
```

Input In [35]

```
base.Year to Date Rank
```

SyntaxError: invalid syntax

## Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Podemos utilizar funções na coluna, como por exemplo, o **value\_counts**, que permite contar os valores de uma coluna:

Estrutura: **base["Title"].value\_counts()**

base["Title"].value_counts()	
Cocomelon	428
Ozark	85
Cobra Kai	81
Manifest	80
The Queenâ¢s Gambit	73
...	
The Office	1
Animals on the Loose: A You...	1
Dark	1
The Secret Life of Pets 2	1
Step Up Revolution	1
Name: Title, Length: 645, dtype: int64	

Série mais assistida (428 vezes)

## Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Para selecionar mais de 1 coluna, basta passar uma lista com os valores que queremos buscar.

Estrutura:

**lista\_colunas** = ["Rank","Days In Top 10","Viewership Score"]

**base[lista\_colunas]**

Dessa forma, você consegue filtrar sua base por coluna.

```
# Para selecionar mais de uma coluna  
lista_colunas = ["Rank","Days In Top 10","Viewership Score"]  
base[lista_colunas]
```

	Rank	Days In Top 10	Viewership Score
0	1	9	90
1	2	5	45
2	3	9	76
3	4	5	30
4	5	9	55
...	...	...	...
7095	6	10	81
7096	7	14	100
7097	8	3	7
7098	9	10	33
7099	10	7	12

7100 rows × 3 columns

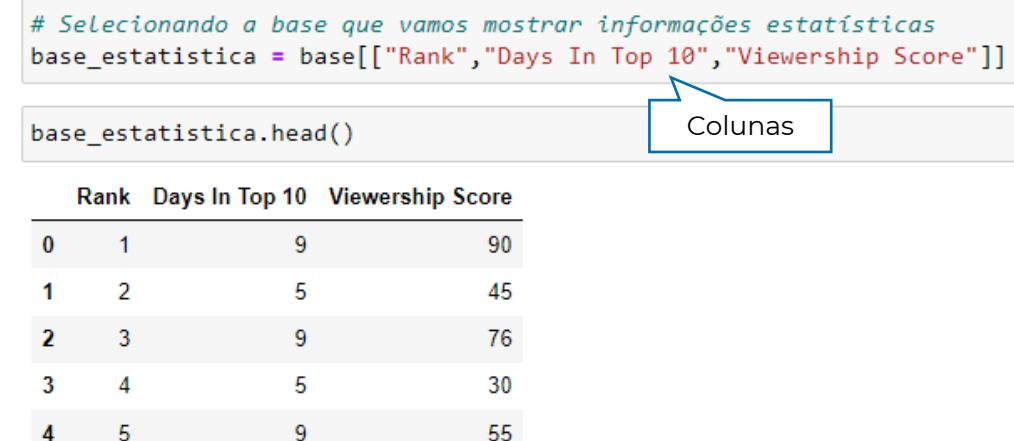
## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Da mesma forma que selecionamos as colunas que queríamos trabalhar, podemos fazer o mesmo com a base.

Estrutura:

**base\_estatistica = base[["Colunas"]]**

Para visualizar as primeiras linhas dessa nova base, podemos utilizar a função **head**.



The screenshot shows a Jupyter Notebook cell with the following content:

```
# Selecionando a base que vamos mostrar informações estatísticas
base_estatistica = base[["Rank", "Days In Top 10", "Viewership Score"]]

base_estatistica.head()
```

A blue callout box labeled "Colunas" points to the column names in the code: "Rank", "Days In Top 10", and "Viewership Score".

	Rank	Days In Top 10	Viewership Score
0	1	9	90
1	2	5	45
2	3	9	76
3	4	5	30
4	5	9	55

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Para confirmar se todos os valores são inteiros podemos utilizar a função **info**.

Estrutura:

**base\_estatistica.info()**

base\_estatistica.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7100 entries, 0 to 7099
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Rank              7100 non-null    int64  
 1   Days In Top 10   7100 non-null    int64  
 2   Viewership Score 7100 non-null    int64  
dtypes: int64(3)
memory usage: 166.5 KB
```

7100 entradas = 7100  
valores não nulos

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Como nessa base todos os valores são inteiros, é possível aplicar funções estatísticas.

- Média de cada coluna

Estrutura: **base\_estatistica.mean()**

- Contar a quantidade de registros

Estrutura: **base\_estatistica.count()**

```
# Mostrando a média  
base_estatistica.mean()
```

Rank	5.500000
Days In Top 10	24.123662
Viewership Score	122.790141
dtype:	float64

Média de cada coluna

```
# Mostrando a contagem de registros  
base_estatistica.count()
```

Rank	7100
Days In Top 10	7100
Viewership Score	7100
dtype:	int64

Quantidade de registros

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Como nessa base todos os valores são inteiros, é possível aplicar funções estatísticas.

- Mediana

Estrutura: **base\_estatistica.median()**

- Desvio Padrão

Estrutura: **base\_estatistica.std()**

# Mediana		
base_estatistica.median()		
Rank	5.5	Mediana de cada coluna
Days In Top 10	7.0	
Viewership Score	50.0	
	dtype: float64	

# Desvio Padrão		
base_estatistica.std()		
Rank	2.872484	Desvio padrão de cada coluna
Days In Top 10	58.473789	
Viewership Score	213.861642	
	dtype: float64	

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Aqui já podemos começar a fazer uma análise de dados, por exemplo, a média em Day in Top 10 é 24 e a mediana é 7, provavelmente podemos ter um **outlier**.

Além disso, o desvio padrão em Day in Top 10 é um valor alto, então esses dados estão muito distribuídos.

```
# Mostrando a média  
base_estatistica.mean()
```

Rank	5.500000
Days In Top 10	24.123662
Viewership Score	122.790141
dtype:	float64

```
# Mostrando a contagem de registros  
base_estatistica.count()
```

Rank	7100
Days In Top 10	7100
Viewership Score	7100
dtype:	int64

```
# Mediana  
base_estatistica.median()
```

Rank	5.5
Days In Top 10	7.0
Viewership Score	50.0
dtype:	float64

```
# Desvio Padrão  
base_estatistica.std()
```

Rank	2.872484
Days In Top 10	58.473789
Viewership Score	213.861642
dtype:	float64

Desvio padrão grande

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Podemos utilizar a função describe para trazer todo o resumo estatístico.

- Mediana

Estrutura: **base\_estatistica.describe()**

	Rank	Days In Top 10	Viewership Score
count	7100.000000	7100.000000	7100.000000
mean	5.500000	24.123662	122.790141
std	2.872484	58.473789	213.861642
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	19.000000
50%	5.500000	7.000000	50.000000
75%	8.000000	18.000000	128.000000
max	10.000000	428.000000	1474.000000

Annotations:

- Valor mínimo (highlighted)
- Quartil de 50% é a mediana (highlighted)
- Valor máximo (highlighted)

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Imagine que agora observamos o desvio padrão alto na coluna Days in Top 10 e queremos nos aprofundar nessas colunas.

Para isso nós podemos fazer filtros na base no pandas, da mesma forma que fazemos no Excel.

base_estatistica.describe()			
	Rank	Days In Top 10	Viewership Score
count	7100.000000	7100.000000	7100.000000
mean	5.500000	24.123662	122.790141
std	2.872484	58.473789	213.861642
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	19.000000
50%	5.500000	7.000000	50.000000
75%	8.000000	18.000000	128.000000
max	10.000000	428.000000	1474.000000

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Da mesma forma que filtramos coluna, podemos fazer para linha, só que precisamos colocar uma condição:

Estrutura:

**base[base[“Coluna”] condição]**

É possível notar que a maioria dos títulos que ficaram mais de 100 dias no top 10 é o Cocomelon.

# Filtrando apenas programas que ficaram mais de 100 dias no Top 10  
base[base[“Days In Top 10”] > 100]

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
2896	2021-01-15	7	7	10	Cocomelon	TV Show	NaN	Jun 1, 2020	101	291
2909	2021-01-16	10	7	9	Cocomelon	TV Show	NaN	Jun 1, 2020	102	292
2919	2021-01-17	10	10	9	Cocomelon	TV Show	NaN	Jun 1, 2020	103	293
3019	2021-01-27	10	-	-	Cocomelon	TV Show	NaN	Jun 1, 2020	104	294
3029	2021-01-28	10	10	-	Cocomelon	TV Show	NaN	Jun 1, 2020	105	295
...	...	...	...	...	...	...	...	...	...	...
6674	2022-01-28	5	6	-	Cocomelon	TV Show	NaN	Jun 1, 2020	424	1466
6687	2022-01-29	8	5	8	Cocomelon	TV Show	NaN	Jun 1, 2020	425	1469
6718	2022-02-01	9	-	7	Cocomelon	TV Show	NaN	Jun 1, 2020	426	1471
6959	2022-02-25	10	-	-	Cocomelon	TV Show	NaN	Jun 1, 2020	427	1472
6998	2022-03-01	9	-	-	Cocomelon	TV Show	NaN	Jun 1, 2020	428	1474

328 rows × 10 columns

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Nos filtros é possível colocar mais de uma condição, utilizando o E / OU.

Exemplo: Procurar programas que ficaram mais de 100 dias no Top 10 **e** o título é diferente de Cocomelon.

Estrutura: **base[base[“Coluna”] condição 1 & condição 2]**

Notamos que apenas o Cocomelon tem o valor maior que 100 na coluna Days in Top 10

As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
				Cocomelon	TV Series	Yes	2023-01-01	120	9.5

base[(base[“Days In Top 10”] > 100) & (base[“Title”] != “Cocomelon”)]

Condição 1

Condição utilizando o E

Condição 2

Símbolo de diferente

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Exemplo: Procurar o título Squid Game **ou** o título Ozark.

Estrutura: **base[base[“Coluna”] condição 1 | condição 2]**

Apenas o Ozark apareceu pois o Squid Game está no meio dos intervalos.

Condição 1			OU	Condição 2						
base.loc[(base["Title"] == "Squid Game")   (base["Title"] == "Ozark")]										
	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
1	2020-04-01	2			- Ozark	TV Show	Yes	Jul 21, 2017	5	45
11	2020-04-02	2			- Ozark	TV Show	Yes	Jul 21, 2017	6	54
21	2020-04-03	2			- Ozark	TV Show	Yes	Jul 21, 2017	7	63
31	2020-04-04	2			2 Ozark	TV Show	Yes	Jul 21, 2017	8	72
41	2020-04-05	2			2 Ozark	TV Show	Yes	Jul 21, 2017	9	81
...	...	...			...	...	...	...	...	...
6954	2022-02-25	5			5 Ozark	TV Show	Yes	Jul 21, 2017	85	524
6967	2022-02-26	8			8 Ozark	TV Show	Yes	Jul 21, 2017	86	527
6979	2022-02-27	10			7 Ozark	TV Show	Yes	Jul 21, 2017	87	528
6986	2022-02-28	7			7 Ozark	TV Show	Yes	Jul 21, 2017	88	532
6996	2022-03-01	7			8 Ozark	TV Show	Yes	Jul 21, 2017	89	536

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Uma outra forma de procurarmos uma informação na base é usar o **.loc**

Estrutura: **base.loc[condição]**

Condição 2                          Condição 1

```
base.loc[(base["Title"] == "Squid Game") | (base["Title"] == "Ozark")]
```

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
11	2020-04-02	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	6	54
21	2020-04-03	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	7	63
31	2020-04-04	2	2	2	Ozark	TV Show	Yes	Jul 21, 2017	8	72
41	2020-04-05	2	2	2	Ozark	TV Show	Yes	Jul 21, 2017	9	81
...	...	...	...	...	...	...	...	...	...	...
6954	2022-02-25	5	5	5	Ozark	TV Show	Yes	Jul 21, 2017	85	524
6967	2022-02-26	8	5	8	Ozark	TV Show	Yes	Jul 21, 2017	86	527
6979	2022-02-27	10	8	7	Ozark	TV Show	Yes	Jul 21, 2017	87	528
6986	2022-02-28	7	10	7	Ozark	TV Show	Yes	Jul 21, 2017	88	532
6996	2022-03-01	7	7	8	Ozark	TV Show	Yes	Jul 21, 2017	89	536

151 rows × 10 columns

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Uma outra forma de procurarmos informação na base é usar o **.loc**

Estrutura:

**base.loc[ condição,[coluna]]**

# Ele permite filtrar colunas de forma muito prática  
base.loc[(base["Title"] == "Squid Game") | (base["Title"] == "Ozark"), ["As of", "Title", 'Type']]

	As of	Title	Type
1	2020-04-01	Ozark	TV Show
11	2020-04-02	Ozark	TV Show
21	2020-04-03	Ozark	TV Show
31	2020-04-04	Ozark	TV Show
41	2020-04-05	Ozark	TV Show
...			
6954	2022-02-25	Ozark	TV Show
6967	2022-02-26	Ozark	TV Show
6979	2022-02-27	Ozark	TV Show
6986	2022-02-28	Ozark	TV Show
6996	2022-03-01	Ozark	TV Show

151 rows × 3 columns

Condição 1      Condição 2      Colunas

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Já o **.iloc** utiliza o índice para filtrar, como se fosse o índice location.

Estrutura: **base.iloc[linha]**

Buscando as linhas de 30 (incluído) e 40 (não incluído)

base.iloc[30:40]

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
30	2020-04-04	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	12	120
31	2020-04-04	2	2	2	Ozark	TV Show	Yes	Jul 21, 2017	8	72
32	2020-04-04	3	-	-	Money Heist	TV Show	Yes	Dec 20, 2017	1	8
33	2020-04-04	4	3	3	All American	TV Show	NaN	Mar 28, 2019	12	99
34	2020-04-04	5	-	-	Coffee & Kareem	Movie	Yes	Apr 3, 2020	1	6
35	2020-04-04	6	4	-	How to Fix a Drug Scandal	TV Show	Yes	Apr 1, 2020	3	18
36	2020-04-04	7	6	-	The Roommate	Movie	NaN	Apr 1, 2020	3	13
37	2020-04-04	8	5	-	Nailed It!	TV Show	Yes	Mar 9, 2018	3	16
38	2020-04-04	9	7	-	Unorthodox	TV Show	Yes	Mar 26, 2020	5	14
39	2020-04-04	10	8	-	The Players Club	Movie	NaN	Apr 1, 2020	2	4

## Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Já o **.iloc** utiliza o índice para filtrar, como se fosse o índice location.

Estrutura: **base.iloc[linha, coluna]**

```
# Também posso usar para buscar apenas colunas específicas  
base.iloc[30:40,[0,4,5]]
```

Linhas      Colunas

	As of	Title	Type
30	2020-04-04	Tiger King: Murder, Mayhem ...	TV Show
31	2020-04-04	Ozark	TV Show
32	2020-04-04	Money Heist	TV Show
33	2020-04-04	All American	TV Show
34	2020-04-04	Coffee & Kareem	Movie
35	2020-04-04	How to Fix a Drug Scandal	TV Show
36	2020-04-04	The Roommate	Movie
37	2020-04-04	Nailed It!	TV Show
38	2020-04-04	Unorthodox	TV Show
39	2020-04-04	The Players Club	Movie

## Módulo 4 – Introdução ao Pandas: criando gráficos

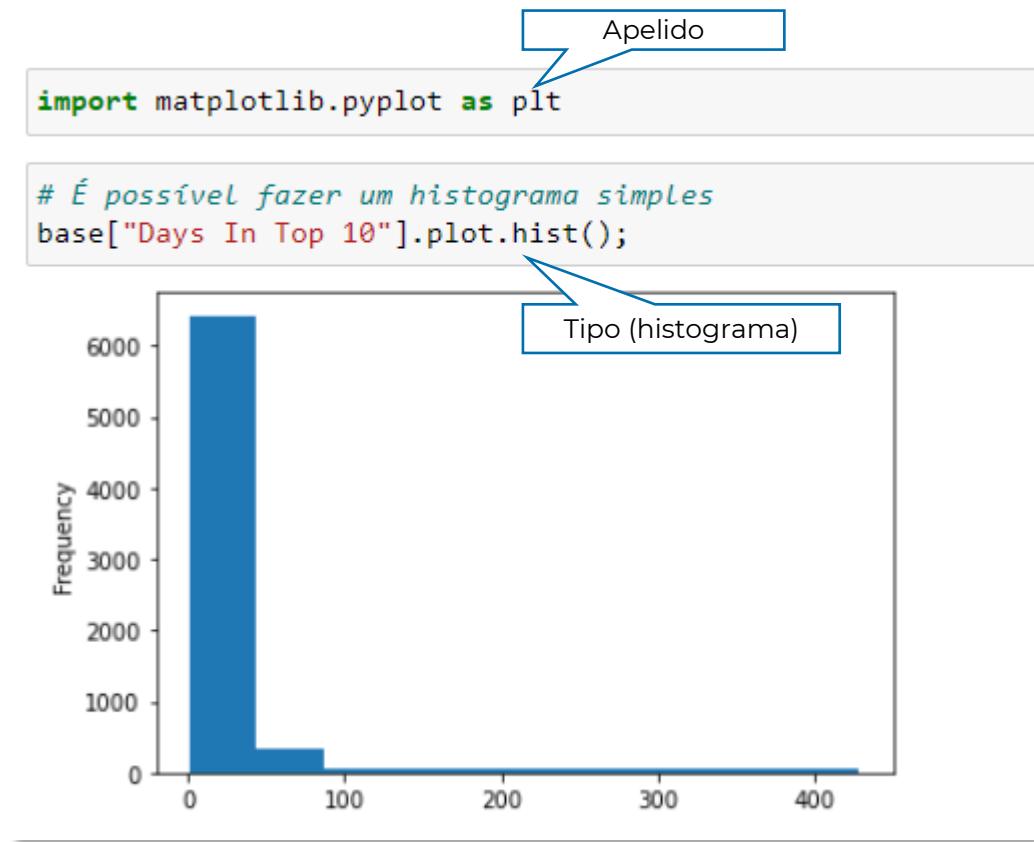
Para criar um gráfico no Pandas é extremamente simples, primeiro vamos precisar importar o **matplotlib.pyplot**.

**import matplotlib.pyplot as plt**

E depois de uma forma simples plotar um histograma com o código abaixo:

Estrutura: **base["Coluna"].plot.hist()**

Exemplo: **base["Days In Top 10"].plot.hist()**



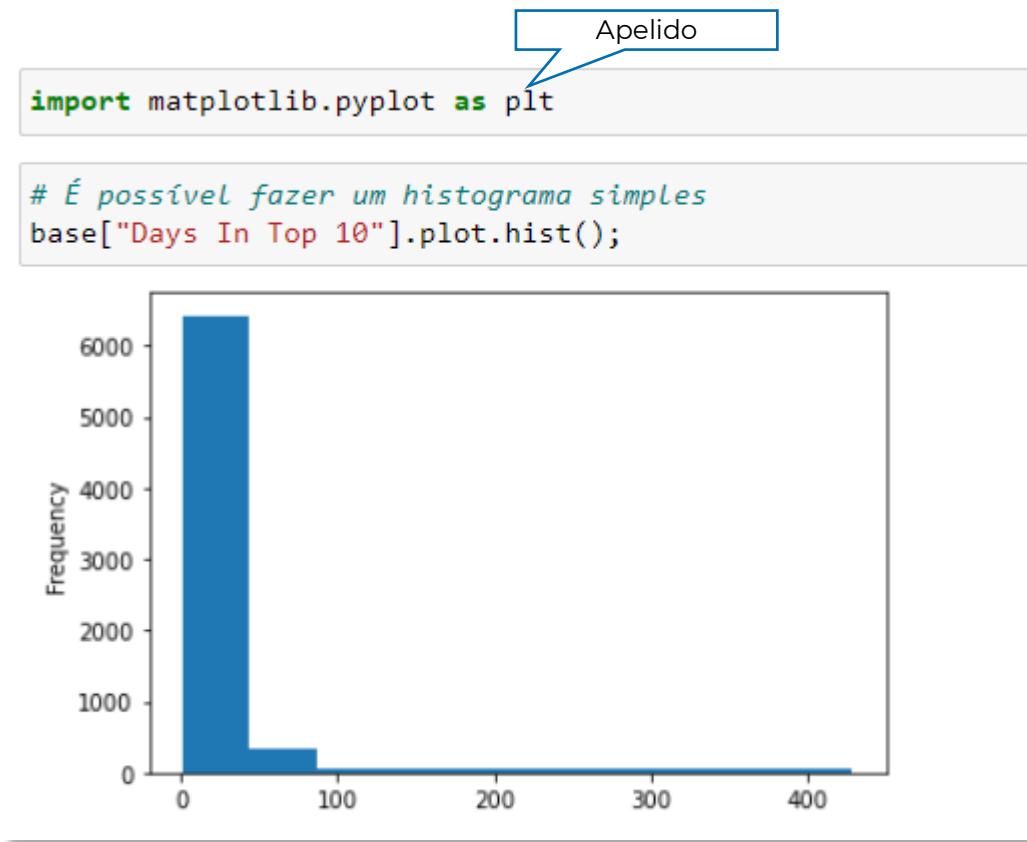
## Módulo 4 – Introdução ao Pandas: criando gráficos

Agora, por exemplo, vamos plotar um gráfico de barras de quantas vezes o título aparece.

Nesse caso, não é possível fazer direto pois o título não é um valor numérico, para isso vamos fazer o seguinte:

Estrutura: **base**["Coluna"].**plot.hist()**

Exemplo: **base**["Days In Top 10"].**plot.hist()**



## MÓDULO 5

# Projeto 1 - Analisando o engajamento do Instagram

## Módulo 5 – Explicando o Projeto

Agora, vamos aplicar o que aprendemos em um Projeto de Análise do engajamento do Instagram.

O primeiro passo do projeto foi analisar os posts que mais engajam e qual é o melhor conteúdo para sugerir dicas de conteúdos.

Nesse projeto a pergunta que queremos responder é: **Qual tipo de conteúdo mais engaja no Instagram da minha empresa?**

## Módulo 5 – Explicando o Projeto

Nesse projeto, vamos considerar que a empresa já possui uma base de dados do Instagram **desde que o usuário começou a postar na marca até o dia 27/março**.

A pessoa que nos contratou enviou alguns direcionamentos:

- Podem ignorar a coluna visualizações, queremos entender apenas **curtidas, comentários e interações**.
- **Tags vazias** é que realmente **não possuem tag** (favor tratar como vazio).

## Módulo 5 – Importando e tratando a base com Pandas

A primeira parte do nosso projeto é importar e visualizar nossa base:

- Importar o Pandas

**import pandas as pd**

- Importar a base em Excel

Estrutura: **base = pd.read\_tipo('Nome da base.formato')**

**base = pd.read\_excel('08. Analisando o engajamento no Instagram.xlsx')**

**Vamos importar e visualizar a nossa base**

```
# Importando o pandas  
import pandas as pd
```

```
# Importar a base em excel  
# - Base: 08. Analisando o engajamento no Instagram.xlsx
```

```
base = pd.read_excel('08. Analisando o engajamento no Instagram.xlsx')
```

**Atenção:** o arquivo de Excel tem que estar na mesma pasta do arquivo do jupyter.

## Módulo 5 – Importando e tratando a base com Pandas

Agora o nosso interesse é **ver** e **entender** a nossa base de dados.

Para visualizar as primeiras 5 linhas da base vamos utilizar o comando **head**.

Estrutura: **base.head()**

# Visualizando as 5 primeiras linhas  
base.head()

	Tipo	Data	Curtidas	Comentários	Visualizações	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	NaN	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	NaN	Loja/Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	NaN	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	82878.0	Produtos	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	NaN	Produtos	S	N	NaN	4437

Valor nulo

Valor nulo

## Módulo 5 – Importando e tratando a base com Pandas

Em projetos muito grandes, quanto mais colunas temos, mais o nosso modelo vai demorar para rodar.

Então, vamos adotar uma boa prática de **apagar as colunas que não vamos usar**. Para isso vamos utilizar o comando **drop**:

Estrutura: **base.drop('nome\_coluna', axis = 1)**

- **axis = 1** para apagar coluna
- **axis = 0** para apagar linha

Em caso de mais de 1 coluna, passamos a lista entre colchetes.

```
# Apagando a coluna "Visualizações"  
base = base.drop("Visualizações", axis=1)
```

Atribuir a base novamente  
para sobrepor as mudanças

# Módulo 5 – Importando e tratando a base com Pandas

```
# Visualizando novamente as 5 primeiras Linhas  
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	NaN	4437

```
# Visualizando as 5 últimas Linhas  
base.tail()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
47	IGTV	2022-03-12	5489	77	Dicas de como usar/Novos Produtos	S	N	NaN	5566
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	NaN	29563
49	Foto	2022-03-22	9087	106		NaN	S	NaN	9193
50	Foto	2022-03-26	16551	186		NaN	S	N	16737
51	IGTV	2022-03-27	4934	65	Dicas de como usar/Produtos	S	N	NaN	4999

Para visualizar novamente as primeiras 5 linhas da base vamos utilizar o comando **head**.

Estrutura: **base.head()**

E para visualizar as últimas 5 linhas vamos utilizar o comando **tail**.

Estrutura: **base.tail()**

# Módulo 5 – Importando e tratando a base com Pandas

E vamos continuar visualizando a base com o passo a passo básico.

- Tamanho

Estrutura: **base.shape()**

Outra forma de fazer isso é utilizando o comando `display`.

Estrutura: **display(base)**

Se a base for muito **pequena**, o `display` vai mostrar a **base inteira**.

# Tamanho da base  
`base.shape`

(52, 9)

Linhas      Colunas

Index	Date	Country	Continent	Index	Possessives	Commoner	Commoner	Inferior
1	Folha	2021-09-11	2050	16	Loja	N	N	2034
2	Folha	2021-09-11	2300	28	LojaProduz	N	N	2058
3	Vídeo	2021-09-11	2050	9	Loja	N	N	2016
4	Folha	2021-09-11	2111	42	Produz	N	N	2044
5	Folha	2021-09-11	5350	45	Produz	S	N	4437
6	Folha	2021-09-11	5350	82	NoneProduz	N	S	5421
7	Folha	2021-09-11	21587	82	NoneProduz	S	S	22440
8	Folha	2021-09-11	6346	33	Produz	N	S	6179
9	Folha	2021-09-11	2050	82	Produz	N	S	4444
10	Vídeo	2021-09-11	4046	81	Produz	N	N	4137
11	Folha	2021-09-11	12054	249	NoneProduz	S	N	13143
12	Folha	2021-09-11	11831	391	NaN	S	S	16222
13	Folha	2021-09-11	11831	46	NoneProduz	S	S	16222
14	Folha	2021-09-11	11840	812	NoneProduz	S	N	16442
15	Folha	2021-09-11	14121	154	DatesComunivates	S	S	14309
16	Vídeo	2021-09-11	3844	71	Produz	N	N	2717
17	Folha	2021-09-11	6306	137	Produz	N	N	8513
18	Vídeo	2021-09-11	2050	46	Produz	N	N	2230
19	Folha	2021-09-11	2111	93	Produz	N	N	2230
20	Folha	2021-09-11	14403	93	Produz	N	N	5392
21	Folha	2021-09-11	16056	285	NaN	S	S	16394
22	Folha	2021-09-11	2050	29	Produz	N	N	2910
23	Folha	2021-09-11	16056	93	Produz	N	S	4443
24	Folha	2021-09-11	11831	225	NoneProduz	S	N	16443
25	Folha	2021-09-11	11807	94	Influenciadores	S	S	16322
26	Vídeo	2021-09-11	17803	383	Influenciadores	S	S	17883
27	Folha	2021-09-11	17101	138	NoneProduz	S	N	12331
28	Folha	2021-09-11	2050	244	NoneProduz	S	S	22308
29	Folha	2021-09-11	2050	245	NoneProduz	S	N	16332
30	Folha	2021-09-11	16067	295	NoneProduz	S	N	16332
31	Folha	2021-09-11	16036	115	NoneProduz	S	S	16336
32	Folha	2021-09-11	16036	142	NaN	S	N	16336
33	Folha	2021-09-11	16036	142	None	S	S	16336
34	Folha	2021-09-11	10019	133	NoneProduz	S	N	16443
35	KTFV	2022-02-04	9270	222	NoneProduz	S	N	17122
36	Folha	2021-09-11	8191	94	Influenciadores	S	S	8289
37	Vídeo	2021-09-11	17803	383	Influenciadores	S	S	17883
38	Folha	2021-09-11	17101	138	NoneProduz	S	N	12331
39	Folha	2021-09-11	2050	244	NoneProduz	S	S	22308
40	Folha	2021-09-11	2050	245	NoneProduz	S	N	16332
41	Folha	2021-09-11	11802	102	Produz	S	N	11904
42	Vídeo	2022-02-13	11210	387	DatesComunivates	S	S	11578
43	Folha	2022-02-13	11887	213	NaN	S	N	17000
44	Folha	2022-02-13	11887	462	Influenciadores	S	S	17000
45	Folha	2022-02-13	12050	30	None	S	S	17000
46	Folha	2022-02-13	21967	72	Influenciadores	S	S	21936
47	Folha	2022-02-13	12050	30	None	S	S	17000
48	Folha	2022-02-13	20459	298	None	S	S	24680
49	Folha	2022-02-13	11831	785	DatesComunivates	S	S	11578
50	Folha	2022-02-13	11862	275	Influenciadores	S	S	11936
51	Folha	2022-02-13	4813	50	Influenciadores	S	S	4803
52	KTFV	2022-02-12	14405	479	DatesComunivatesProduz	S	N	3448
53	Folha	2022-02-13	2050	245	DatesComunivatesProduz	S	N	2050
54	Folha	2022-02-13	11807	108	None	S	S	9103
55	Folha	2022-02-13	11861	188	DatesComunivatesProduz	S	N	16737
56	KTFV	2022-02-12	4034	85	DatesComunivatesProduz	S	N	4030

## Módulo 5 – Importando e tratando a base com Pandas

Para saber as informações (tipo, valores nulos) da base vamos utilizar o comando **info**.

Estrutura: **base.info()**

```
# Verificando as informações
base.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52 entries, 0 to 51
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Tipo         52 non-null    object  
 1   Data          52 non-null    datetime64[ns]
 2   Curtidas     52 non-null    int64   
 3   Comentários  52 non-null    int64   
 4   Tags          44 non-null    object  
 5   Pessoas       52 non-null    object  
 6   Campanhas     52 non-null    object  
 7   Carrossel     8 non-null    object  
 8   Interacoes    52 non-null    int64   
dtypes: datetime64[ns](1), int64(3), object(5)
memory usage: 3.8+ KB
```

8 valores não nulos e 44  
valores nulos

## Módulo 5 – Tratando valores nulos da coluna Corrossel

Agora vamos tratar os valores nulos que aparecem na coluna Carrossel.

Porém, primeiro precisamos entender os valores de carrossel, para isso vamos contar os valores que aparecem nessa coluna.

Estrutura: **base.Coluna.value\_counts()**

Estrutura: **base.Carrossel.value\_counts()**

Só existe o valor Sim (S)

```
# Contando os valores que aparecem na coluna Carrossel
base.Carrossel.value_counts()
S    8
Name: Carrossel, dtype: int64
```

Na verdade, os valores **nulos** são de postagens que **não são carrossel**.

Sendo assim o nulo deveria ser "N".

## Módulo 5 – Tratando valores nulos da coluna Corrossel

E como que a gente poderia **atribuir um novo valor?**

Para localizar os valores que são nulos vamos utilizar o comando loc.

Estrutura: **base.loc[base.Coluna.isnull()]**

Estrutura: **base.loc[base.Carrossel.isnull()]**

# Filtrando os valores em que carrossel é nulo  
base.loc[base.Carrossel.isnull()].head()

Para poder resumir (5 primeiras linhas)

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	NaN	4437

Valor não

## Módulo 5 – Tratando valores nulos da coluna Corrossel

Para localizar os valores que não são nulos vamos utilizar o comando loc.

Estrutura: **base.loc[base.Coluna.notnull()]**

Estrutura: **base.loc[base.Carrossel.notnull()]**

```
# Buscando valores que NAO sao nulos  
base.loc[base.Carrossel.notnull()]
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
5	Foto	2021-09-17	5359	62	Novos Produtos	N	S	S	5421
8	Foto	2021-09-27	6355	89	Produtos	S	N	S	6444
12	Foto	2021-10-21	6166	55	Novos Produtos	S	S	S	6221
21	Foto	2021-12-23	8328	93	Produtos	S	N	S	8421
25	Foto	2022-01-02	12193	138	Novos Produtos	S	N	S	12331
26	Foto	2022-01-08	24585	354	Datas comemorativas	S	S	S	24939
28	Foto	2022-01-15	9936	119	Novos Produtos	S	N	S	10055
40	Foto	2022-02-21	21621	213	Influenciadores	S	S	S	21834

Valor sim

## Módulo 5 – Tratando valores nulos da coluna Corrossel

Para localizar os valores que são nulos apenas na coluna Carrossel.

Estrutura:

**base.loc[base.Coluna.isnull(), 'Coluna']**

Estrutura:

**base.loc[base.Carrossel.isnull(), 'Carrossel']**

```
# Selecionando apenas a coluna Carrossel
base.loc[base.Carrossel.isnull(),'Carrossel']

0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
6      NaN
7      NaN
9      NaN
10     NaN
11     NaN
13     NaN
14     NaN
15     NaN
16     NaN
17     NaN
18     NaN
19     NaN
20     NaN
22     NaN
23     NaN
24     NaN
27     NaN
29     NaN
30     NaN
31     NaN
32     NaN
33     NaN
34     NaN
35     NaN
36     NaN
37     NaN
38     NaN
39     NaN
41     NaN
42     NaN
43     NaN
44     NaN
45     NaN
46     NaN
47     NaN
48     NaN
49     NaN
50     NaN
51     NaN
Name: Carrossel, dtype: object
```

## Módulo 5 – Tratando valores nulos da coluna Corrossel

Agora vamos atribuir o valor N para essa coluna.

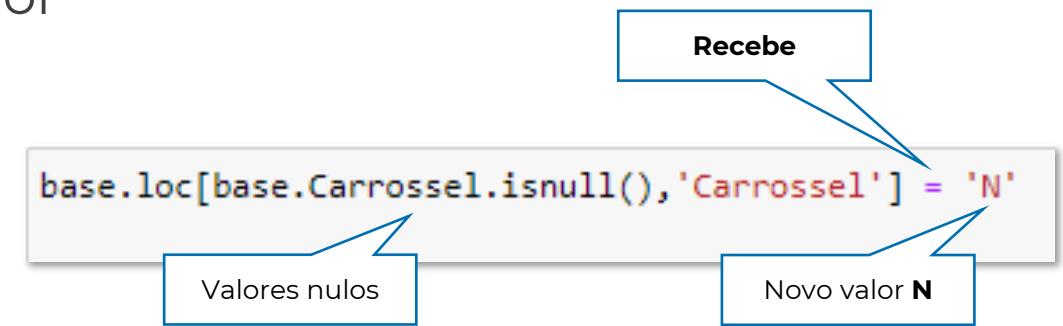
Lembrando que para atribuir um novo valor utilizamos o sinal de **igual**.

Estrutura:

**base.loc[base.Coluna.isnull(), 'Coluna'] = 'N'**

Estrutura:

**base.loc[base.Carrossel.isnull(), 'Carrossel'] = 'N'**



## Módulo 5 – Tratando valores nulos da coluna Corrossel

Verificando novamente os valores dessa coluna, utilizando o comando **head**.

base.head()									
	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437

Valor N

Verificando novamente os valores dessa coluna, podemos observar que não existe mais valor nulo.

base.Carrossel.value_counts()	
N	44
S	8
Name: Carrossel, dtype: int64	

## Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Agora, vamos nos aprofundar na nossa base utilizando o **describe**, que nos fornecerá todo o resumo estatístico.

Estrutura: **base.describe()**

É importante entender o motivo dos menores/maiores valores, para saber se o máximo é algo **replicável** e **escalável**.

# Descrição estatística da base base.describe()			
	Curtidas	Comentários	Interacoes
count	52.000000	52.000000	52.000000
mean	12262.730769	189.500000	12452.230769
std	8165.875326	170.687709	8299.390088
min	2807.000000	9.000000	2816.000000
25%	5492.000000	69.500000	5562.500000
50%	9603.000000	128.000000	9773.500000
75%	17621.750000	265.250000	17920.750000
max	37351.000000	852.000000	37853.000000

Valor  
mínimo

Valor  
máximo

## Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

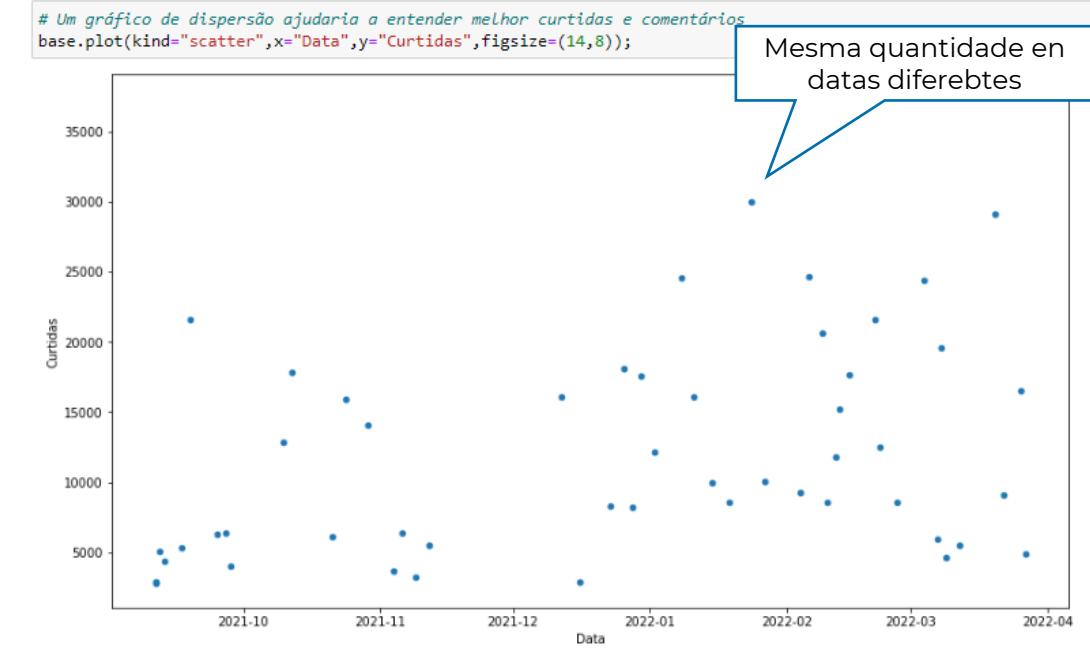
Podemos também plotar as informações pelo gráfico scatter.

Estrutura:

```
base.plot(kind="scatter",x="Data",y="Curtida  
s",figsize=(14,8));
```

```
base.plot(kind="tipo",x="Data",y="Curtidas",  
figsize=(14,8));
```

Observa-se que no tempo não tem um aumento de curtidas, então a data não é um fator tão predominante que influencia nas curtidas.

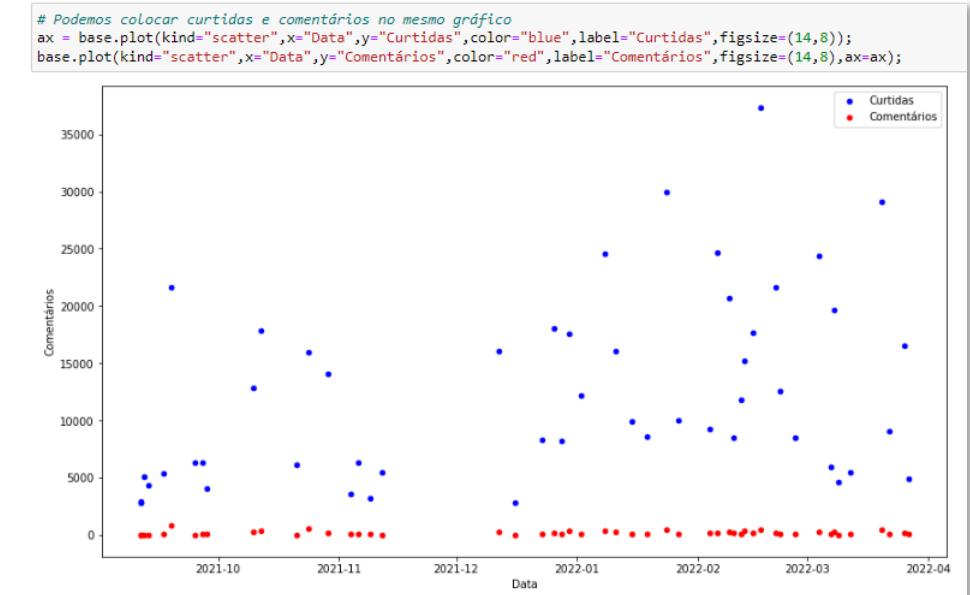


## Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Nesse gráfico não ficou bom de visualizar curtidas e comentários, porque a escala da curtida não nos permite analisar os comentários.

Estrutura:

```
Ax=base.plot(kind="scatter",x="Data",y="Curtidas",color="blue",label="Curtidas",figsize=(14,8));  
base.plot(kind="scatter",x="Data",y="Comentários",color="red",label="Comentários",figsize=(14,8),ax=ax);
```

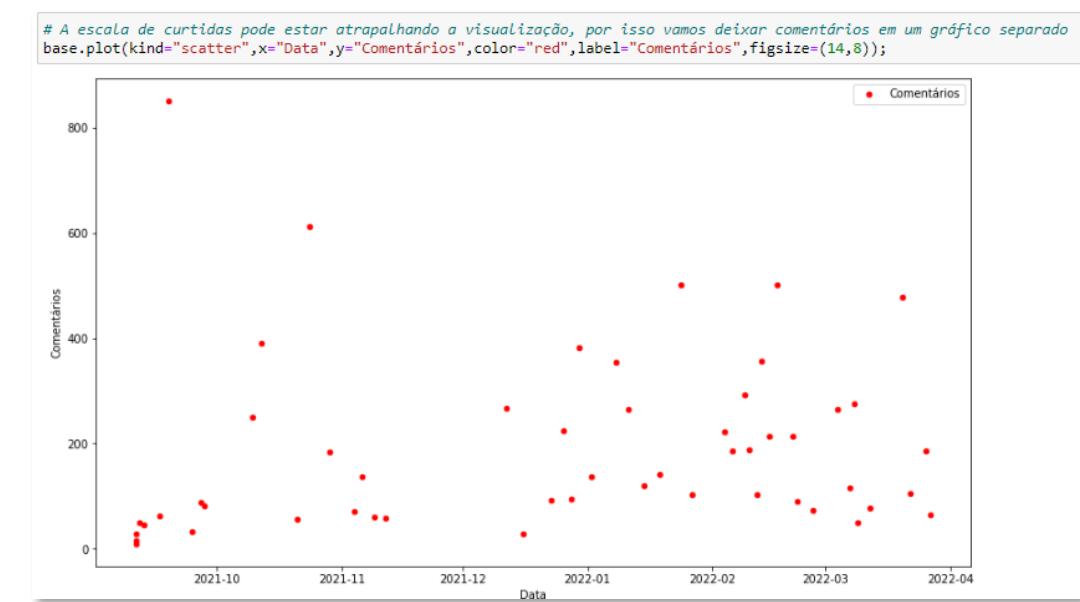


## Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Criando apenas um gráfico de comentário também observamos que não existe nenhum padrão.

O gráfico e as informações estatísticas não estão dizendo muita coisa pois existe uma grande dispersão entre curtidas e comentários.

Precisamos verificar se existe um padrão usando as outras colunas de informações.



## Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

A primeira coisa que podemos fazer é pegar os **5 primeiros** registros com **mais** e **menos curtidas**.

Para isso vamos utilizar a função **sort\_values**, que vai ordenar os meus valores por uma coluna escolhida.

Podendo escolher se vamos ordenar de forma ascendente (menor para maior) ou descendente (maior para o menor).

```
# Ordenando os valores  
base.sort_values(by="Curtidas", ascending=False).head()
```

Coluna

Maior para o menor

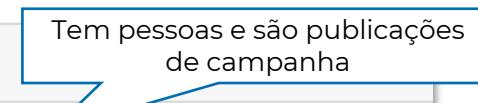
## Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

A primeira coisa que podemos fazer é pegar os **5 primeiros** registros com **mais curtidas**.

Estrutura: **base.sort\_values(by="Curtidas", ascending=False).head()**

Estrutura: **base.sort\_values(by="Curtidas", ascending=False).head()**

# Ordenando os valores  
base.sort\_values(by="Curtidas", ascending=False).head()



	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
39	Foto	2022-02-17	37351	502	Promoções	S	S	N	37853
30	Reels	2022-01-24	29981	502	Trends	S	S	N	30483
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563
33	Foto	2022-02-06	24655	186	Influenciadores	S	S	N	24841
26	Foto	2022-01-08	24585	354	Datas comemorativas	S	S	S	24939

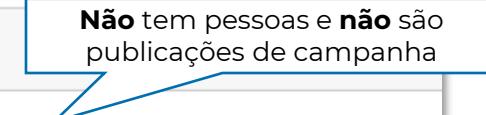
## Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Podemos utilizar o mesmo comando para encontrar os **5** registros com **menos curtidas**.

Estrutura: **base.sort\_values(by="Curtidas",ascending=True).head()**

Estrutura: **base.sort\_values(by="Curtidas",ascending= True).head()**

# Selecionando os 5 últimos valores  
base.sort\_values(by="Curtidas",ascending=True).head()



	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874
20	Foto	2021-12-16	2881	29	Produtos	N	N	N	2910
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958
17	Vídeo	2021-11-09	3213	60	Produtos	N	N	N	3273

## Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Uma outra forma de encontrar os registros com **menos curtidas** é utilizando o **tail**.

Estrutura: **base.sort\_values(by="Curtidas", ascending=False).tail()**

Estrutura: **base.sort\_values(by="Curtidas", ascending= False).tail()**

# Selecionando os 5 últimos valores base.sort_values(by="Curtidas", ascending=False).tail()										
	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	
17	Vídeo	2021-11-09	3213	60	Produtos	N	N	N	3273	
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958	
20	Foto	2021-12-16	2881	29	Produtos	N	N	N	2910	
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874	
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816	

## Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Então, podemos observar que no **top 5** todas as postagens tinham **pessoas** e eram fotos de **campanha**.

Nas 5 piores postagens, não haviam pessoas e nem eram postagens de campanhas.

Isso pode ser um indicador que **pessoas** e **campanhas** têm relação com as **curtidas**.

O próximo passo foi marcar todas as outras fotos para saber se tinha pessoa ou era foto de campanha, para saber se isso que observamos se aplicava ao restante da base.

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Agora vamos ver como nós vamos comparar as fotos que têm pessoas e as que não têm, vamos utilizar de forma muito semelhante ao que fazemos no Excel com a tabela dinâmica.

Basicamente, o que fizemos foi comprovar a hipótese de que fotos **com pessoas** possuem **mais curtidas**.

Rótulos de Linha	Média de Curtidas
N	4.257
S	14.665
Total Geral	12.263

Média até 3x maior

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Agora vamos ver como fazemos isso no Python.

O **group by** é muito análogo à tabela dinâmica que fazemos no Excel.

O groupby vai agrregar pela **Coluna1**

A função de agregação será aplicada na **Coluna2**

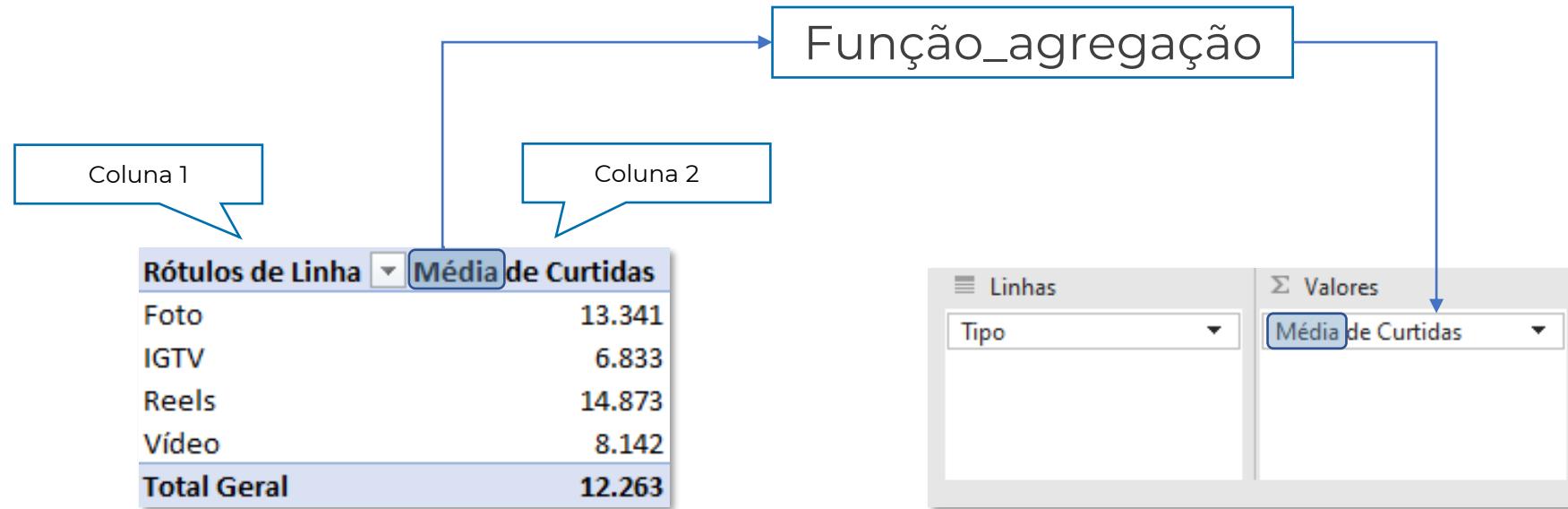
```
# Agrupando as informações por tipo  
base.groupby("Tipo")["Comentários"].count()
```

Tipo	
Foto	36
IGTV	5
Reels	5
Vídeo	6

Name: Comentários, dtype: int64

# Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Estrutura: **base.groupby("Coluna1")["Coluna2"].função\_agregação()**



# Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Aplicando no nosso exemplo temos:

**base.groupby("Tipo")["Comentários"].count()**

```
# Agrupando as informações por tipo
base.groupby("Tipo")["Comentários"].count()

Tipo
Foto      36
IGTV       5
Reels      5
Vídeo      6
Name: Comentários, dtype: int64
```

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Nós podemos adicionar mais colunas para visualizar por tipo e pessoas, por exemplo.

```
base.groupby(["Tipo", "Pessoas"])["Curtidas"].mean()
```

# Agrupando por Tipo e Pessoas		base.groupby(["Tipo", "Pessoas"])[["Curtidas", 'Comentários"]].mean()			
base.groupby(["Tipo", "Pessoas"])[["Curtidas", 'Comentários"]].mean()		Curtidas		Comentários	
Tipo	Pessoas	Tipo	Pessoas		
Foto	N	3,863.50			
	S	15,236.67			
IGTV	S	6,833.40			
	N	5,934.50			
Reels	S	20,832.00			
	N	4,007.50			
Vídeo	S	16,409.50			
	N	4,007.50			
Name: Curtidas, dtype: float64					

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Podemos também analisar Curtidas e Comentários, ao mesmo tempo.

**base.groupby(["Tipo", "Pessoas"])[["Curtidas",'Comentários']]. mean()**

Tipo	Pessoas		
		Curtidas	Comentários
Foto	N	3,863.50	29.50
	S	15,236.67	226.20
IGTV	S	6,833.40	133.60
Reels	N	5,934.50	98.00
	S	20,832.00	342.00
Vídeo	N	4,007.50	65.25
	S	16,409.50	370.00

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Agora, se a gente quiser incluir a coluna de campanhas basta:

**base.groupby(["Tipo", "Pessoas","Campanhas"])[["Curtidas",'Comentários']]. mean()**

```
# Incluindo a coluna de campanhas
base.groupby(["Tipo", "Pessoas", "Campanhas"])[["Curtidas", 'Comentários']].mean()
```

Tipo	Pessoas	Campanhas	Curtidas	Comentários
			N	S
Foto	N		2,869.00	20.50
			5,852.50	47.50
	S		10,815.29	159.93
			19,105.38	284.19
IGTV	S		6,833.40	133.60
			5,934.50	98.00
			12,894.00	249.00
Reels	N		24,801.00	388.50
			16,409.50	370.00
	S		4,007.50	65.25

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

O groupby já permite ver que publicações de campanha tem um grande engajamento e com foto de pessoas também.

Podemos então fazer os agrupamentos que achamos melhor para entender os nossos dados.

Agrupamento somente para pessoas:

**base.groupby("Pessoas")`["Curtidas","Comentários"]`.mean()**

```
# Somente para pessoas
base.groupby("Pessoas")["Curtidas","Comentários"].mean()
```

Pessoas	Curtidas	Comentários
N	4,256.67	52.83
S	14,664.55	230.50

3 vezes maior com pessoas

# Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Agrupamento somente para campanhas:

**base.groupby("Campanhas")[["Curtidas",'Comentários']]. mean()**

Campanhas	Curtidas	Comentários
	N	S
	7,928.33	123.17
	18,173.27	279.95

Postagem de campanha tem mais curtidas

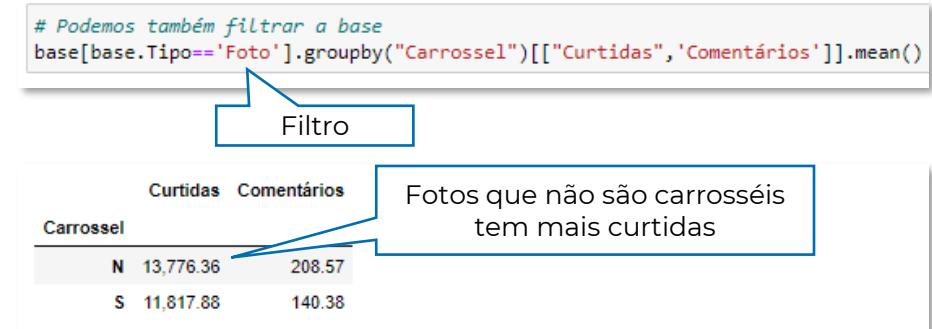
## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Nesse caso devemos **filtrar** apenas as fotos pois só temos carrossel em fotos.

Sem esse filtro estariamos comparando coisas erradas.

Então temos que **restringir a nossa base**.

```
base[base.Tipo=='Foto'].groupby("Carrossel")  
[["Curtidas",'Comentários']].mean()
```



# Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Filtrar a base é muito importante, senão podemos dar uma conclusão errada, porque comparamos dados que não podem ser comparados.

# Carrossel (sem filtrar a base)  
base.groupby("Carrossel")[["Curtidas", 'Comentários"]].mean()

	Curtidas	Comentários	
Carrossel			Diferença de 500
N	12,343.61	198.43	
S	11,817.88	140.38	



# Podemos também filtrar a base  
base[base.Tipo=='Foto'].groupby("Carrossel")[["Curtidas", 'Comentários"]].mean()

	Curtidas	Comentários	
Carrossel			Diferença de 2000
N	13,776.36	208.57	
S	11,817.88	140.38	

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Colocando pessoas e campanhas juntos podemos ver como se dá essa diferença.

A média quando tem **pessoas E campanhas** é publicação é de cerca de 19,4 mil curtidas, já quando são apenas pessoas (sem campanha) passa para quase 10 mil e se não tiverem pessoas chega no máximo a 5,9 mil mesmo em campanhas.

Nesse caso a gente já consegue mostrar para a empresa a **importância** de incluir **pessoas** usando os seus produtos, o que gera um aumento considerável no engajamento.

# Agregando por pessoas e campanhas  
base.groupby(["Pessoas", "Campanhas"])[["Curtidas", 'Comentários"]].mean()

Pessoas	Campanhas		
		Curtidas	Comentários
N	N	3,937.50	53.90
	S	5,852.50	47.50
S	N	9,923.75	157.80
	S	19,405.35	303.20

Nem Pessoas, nem campanhas: 3937 curtidas

Pessoas e campanhas: 19405 curtidas

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Podemos também analisar por pessoas, campanhas e tipo.

Analizando novamente a questão do vídeo, ele não parece mais tão ruim assim.

Quando feito em **campanha** e usando **pessoas** ele teve um resultado **bom**, inclusive próximo à foto.

# Agregando por pessoas, campanhas e tipo			
base.groupby(["Pessoas", "Campanhas", "Tipo"])[["Curtidas", 'Comentários"]].mean()			
Pessoas	Campanhas	Tipo	Curtidas Comentários
N	Foto	2.869,00	20,50
		5.934,50	98,00
		4.007,50	65,25
	Reels	5.852,50	47,50
		10.815,29	159,93
		6.833,40	133,60
S	Vídeo	12.894,00	249,00
		19.105,38	284,19
		24.801,00	388,50
	IGTV	16.409,50	370,00

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

O que poderia ter levado a média baixa é que só temos vídeo ou **COM** pessoa e **COM** campanha ou sem nenhum dos dois.

Não temos nenhum vídeo com apenas um dos dois (pessoa ou campanha).

Já IGTV, mesmo tendo pessoa, não teve um resultado tão bom.

# Agregando por pessoas, campanhas e tipo			
base.groupby(["Pessoas", "Campanhas", "Tipo"])[["Curtidas", 'Comentários"]].mean()			
Pessoas	Campanhas	Tipo	
		N	S
N	Foto	2,869.00	20.50
		Reels	5,934.50
		Vídeo	4,007.50
	S	Foto	5,852.50
		N	10,815.29
		IGTV	6,833.40
S	Foto	Reels	12,894.00
		Vídeo	19,105.38
		Reels	24,801.00
		Vídeo	16,409.50

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Se eu filtrar a base somente por tipo igual a vídeo, percebemos que a loja tentou postar 4 vídeos mostrando seus produtos (sem **nenhuma pessoa**) e o resultado foi **baixo**.

Quando o vídeo foi feito com **pessoas** aproveitando trends e **datas comemorativas** o resultado foi **muito bom!**

```
# Vamos filtrar a base apenas onde o tipo é Vídeo  
base[base.Tipo=='Vídeo']
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164
9	Vídeo	2021-09-28	4056	81	Produtos	N	N	N	4137
15	Vídeo	2021-11-04	3646	71	Produtos	N	N	N	3717
17	Vídeo	2021-11-09	3213	60	Produtos	N	N	N	3273
24	Vídeo	2021-12-30	17600	383	Trends	S	S	N	17983
37	Vídeo	2022-02-13	15219	357	Datas comemorativas	S	S	N	15576

Resultado bom  
com pessoas

## Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Conclusões:

- Em uma análise inicial, postagens incluindo **pessoas engajam** muito mais que aquelas que não possuem ninguém.
- Postagens em **épocas de campanha** também possuem um **melhor engajamento**.
- Nessa base, o **carrossel não foi um diferencial** para melhorar o engajamento da marca.

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

As tags foram criadas para a base não ficar muito grande, um exemplo de tag é o gênero de um filme.

Agora o que queremos responder é: **Qual a tag mais engaja nessas publicações?**

Agora queremos olhar apenas **tags**.

A pessoa que nos contratou também deu alguns direcionamentos:

- Podem ignorar a coluna visualizações, queremos entender apenas curtidas, comentários e interações.
- Tags vazias significa que realmente não possuem tag (tratar como vazio).

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Primeiro vamos importar a nossa base exatamente como fizemos anteriormente:

- Importar o Pandas

```
import pandas as pd
```

- Importar o Numpy

```
import numpy as np
```

```
# Importando o pandas
import pandas as pd
import numpy as np
# Usando o mesmo formato dos valores
pd.options.display.float_format = '{:, .2f}'.format
```

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

- Importar a base excel

```
base = pd.read_excel("08. Analisando o engajamento no Instagram.xlsx")
```

```
# Importar a base em excel
base = pd.read_excel("08. Analisando o engajamento no Instagram.xlsx")
```

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

- Excluir a coluna Visualizações

```
base = base.drop("Visualizações",axis=1)
```

```
# Apagando a coluna "Visualizações"  
base = base.drop("Visualizações",axis=1)
```

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

# Visualizando novamente as 5 primeiras Linhas  
base.head()

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	NaN	4437

Mais de uma tag

Comentários

Tags

Loja

Depois vamos visualizar as 5 primeiras linhas utilizando o comando head.

### base.head()

Se tem mais uma tag separamos por uma barra.

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Vamos agora aplicar o group by para Tags:

```
base.groupby("Tags")["Curtidas"].mean()
```

Repare que Datas comemorativas aparecem em mais de uma tag.

Ou seja, se eu tenho **mais de uma tag** não é possível agrupar.

Então precisamos tratar a nossa base para depois utilizar o group by.

```
# Agrupando por tags  
base.groupby("Tags")["Curtidas"].mean()
```

Tags	Curtidas
Datas comemorativas	17,975.00
Datas comemorativas/Promoções	29,084.00
Dicas de como usar/Novos Produtos	5,703.50
Dicas de como usar/Produtos	7,586.67
Influenciadores	15,197.29
Loja	2,832.50
Loja/Produtos	2,930.00
Novos Produtos	11,619.57
Produtos	5,666.92
Promoções	26,645.50
Trends	22,400.67
Trends/Produtos	12,894.00
Name: Curtidas, dtype: float64	

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (**split** e **explode**)

Para conseguir analisar separadamente as tags, podemos dividir linhas com 2 tags em 2 linhas.

- Para isso primeiro vamos usar o **split** para **separar** em uma lista com as tags.
- Depois vamos usar o **explode** para transformar as listas com 2 tags em 2 **linhas diferentes**.

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

A primeira coisa que o split vai fazer é separar as informações.

O split separa um texto em uma lista baseado em algum separador, se eu informamos nenhum argumento ele vai separar por espaço.

Estrutura: **texto.split()** ou **texto.split('delimitador')**

```
texto = "O Curso de Ciência de Dados da Hashtag é top!"  
  
# Se eu não passo nenhum argumento, ele vai separar por espaço  
texto.split()  
['O', 'Curso', 'de', 'Ciência', 'de', 'Dados', 'da', 'Hashtag', 'é', 'top!']
```

```
texto = "O-Curso-de-Ciência-de-Dados-da-Hashtag-é-top!"  
Separar pelo espaço  
texto.split()  
['O-Curso-de-Ciência-de-Dados-da-Hashtag-é-top!']  
  
Separar pelo -  
texto.split("-")  
['O', 'Curso', 'de', 'Ciência', 'de', 'Dados', 'da', 'Hashtag', 'é', 'top!']
```

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Agora vamos utilizar esse comando para a nossa coluna de Tags, transformando a coluna Tags em uma lista de tags e separando as tags por "/".

**base.Tags = base.Tags.str.split("/")**

Para visualizar as primeiras 5 linhas vamos utilizar o comando **head**.

**base.head()**

```
# Vamos usar isso para a nossa coluna "Tags"
# Transformando a coluna Tags em uma Lista de tags
base.Tags = base.Tags.str.split("/")
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	[Loja]	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	[Loja, Produtos]	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	[Loja]	N	N	NaN	2816
3	Video	2021-09-12	5115	49	[Produtos]	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	[Produtos]	S	N	NaN	4437

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

O **explode** vai separar uma coluna de um DataFrame em **1 linha para cada elemento da lista**.

Observe que na primeira o elemento A tem 2 elementos.

O **explode** vai transformar a minha primeira linha em **2 linhas**, uma com o primeiro elemento e outra com o segundo.

```
# Criando o dicionário
dic = {
    "A": [[1,2],3,[4,5,6],[]],
    "B": [1,2,3,4],
}

# Transformando esse dicionário em um DataFrame
base_dic = pd.DataFrame(dic)

base_dic
```

	A	B
0	[1, 2]	1
1		2
2	[4, 5, 6]	3
3	[]	4

2 elementos

3 elementos

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Estrutura: **base\_dic = base\_dic.explode('Coluna')**

Aplicando ao nosso exemplo, temos: **base\_dic = base\_dic.explode('A')**

```
# Criando o dicionário
dic = {
    "A": [[1,2],3,[4,5,6],[]],
    "B": [1,2,3,4],
}

# Transformando esse dicionário em um DataFrame
base_dic = pd.DataFrame(dic)

base_dic
```

2 elementos

	A	B
0	[1, 2]	1
1	3	2
2	[4, 5, 6]	3
3		4

3 elementos

# Usando o explode para separar a coluna A

```
base_dic = base_dic.explode('A')
base_dic
```

Cada elemento em uma linha

Elemento é mantido

	A	B
0	1	1
0	2	1
1	3	2
2	4	3
2	5	3
2	6	3
3	NaN	4

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Tudo que estiver em lista será separado em 1 linha por elemento da lista.

Se não tiver na lista, o elemento será mantido.

Listas vazias vão ter o valor de NaN.

Para as outras colunas, elas irão repetir os seus valores.

Inclusive o índice também irá repetir.

```
# Usando o explode para separar a coluna A  
base_dic = base_dic.explode('A')  
base_dic
```

	A	B
0	1	1
0	2	1
1	3	2
2	4	3
2	5	3
2	6	3
3	NaN	4

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Vamos aplicar o explode na nossa coluna de Tag agora que ele já está separado.

```
base = base.explode('Tags')  
base.head()
```

Feito isso agora podemos começar a trabalhar com as nossas Tags.

```
# Separando a coluna Tag em 1 Linha para cada elemento da Lista  
base = base.explode('Tags')  
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja	N	N	NaN	2958
1	Foto	2021-09-11	2930	28	Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

```
# Repetindo o cálculo da média para pessoas  
base.groupby("Pessoas")["Curtidas"].mean()
```

```
Pessoas  
N    4,154.62  
S   14,100.57  
Name: Curtidas, dtype: float64
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja	N	N	NaN	2958
1	Foto	2021-09-11	2930	28	Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164

Valores  
duplicados

Agora vamos fazer o mesmo groupby que fizemos na aula anterior para o cálculo de média para pessoas.

Repare que temos uma diferença do que vimos na última aula.

Como adicionamos linha, toda análise que estamos fazendo só pode ser feita para essa coluna nova de Tag.

Por exemplo, se eu tinha uma foto com 5000 curtidas e 2 tags, agora ela vai aparecer **2 vezes**.

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Fazendo a mesma análise para Tags, repare que agora com a base tratada temos apenas uma Tag por linha.

**base.groupby("Tags")["Curtidas"].mean()**

```
# Fazendo para Tag
base.groupby("Tags")["Curtidas"].mean()

Tags
Datas comemorativas    20,752.25
Dicas de como usar      6,833.40
Influenciadores          15,197.29
Loja                     2,865.00
Novos Produtos           10,304.89
Produtos                  6,269.82
Promoções                 27,458.33
Trends                    20,024.00
Name: Curtidas, dtype: float64
```

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Para melhorar a visualização podemos colocar em ordem decrescente os valores de curtidas. Para isso, vamos utilizar o comando **sort\_values**.

**sort\_values("Coluna", ascending=False)**

False – maior para o menor

True – menor para o maior

Ordenando a nossa base por curtida temos:

**base.groupby("Tags")  
[["Curtidas", "Comentários"]].mean().sort\_values("Curtidas", ascending=False)**

```
# Ordenando por curtidas
base.groupby("Tags")  
[["Curtidas", "Comentários"]].mean().sort_values("Curtidas", ascending=False)
```

Tags	Curtidas	Comentários
Promoções	27,458.33	531.00
Datas comemorativas	20,752.25	343.50
Trends	20,024.00	352.25
Influenciadores	15,197.29	161.71
Novos Produtos	10,304.89	198.56
Dicas de como usar	6,833.40	133.60
Produtos	6,269.82	94.12
Loja	2,865.00	17.67

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Dessa forma, é possível notar que postagens de **promoções** são as que mais **engajam**.

Além de promoções, datas comemorativas e trends também possuem um bom engajamento.

```
# Ordenando por curtidas
base.groupby("Tags")[["Curtidas","Comentários"]].mean().sort_values("Curtidas",ascending=False)
```

Tags	Curtidas	Comentários
Promoções	27,458.33	531.00
Data comemorativas	20,752.25	343.50
Trends	20,024.00	352.25
Influenciadores	15,197.29	161.71
Novos Produtos	10,304.89	198.56
Dicas de como usar	6,833.40	133.60
Produtos	6,269.82	94.12
Loja	2,865.00	17.67

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

E o que está sem tag? Como vamos tratar isso?

O primeiro passo é identificar os valores sem Tag utilizando o filtro.

Estrutura: **base[base.Tags.isnull()]**

# Filtrando valores sem tag  
base[base.Tags.isnull()]

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
11	Foto	2021-10-12	17831	391	NaN	S	S	NaN	18222
19	Foto	2021-12-12	16086	268	NaN	S	S	NaN	16354
29	Foto	2022-01-19	8612	142	NaN	S	N	NaN	8754
38	Foto	2022-02-15	17687	213	NaN	S	N	NaN	17900
41	Foto	2022-02-22	12530	90	NaN	S	N	NaN	12620
43	Foto	2022-03-04	24399	266	NaN	S	S	NaN	24665
49	Foto	2022-03-22	9087	106	NaN	S	S	NaN	9193
50	Foto	2022-03-26	16551	186	NaN	S	N	NaN	16737

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Da mesma forma que fizemos para Carrossel, podemos fazer para as tags escrevendo "Sem tag" e nesse caso iria aparecer no groupby.

**base.loc[base.Tags.isnull(),'Tags'] = 'Sem Tags'**

```
# Atribuindo o texto sem tag para as colunas onde a tag é NaN  
base.loc[base.Tags.isnull(),'Tags'] = 'Sem Tags'
```

Valor  
atribuído

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Agora vamos repetir a tabela de curtidas por Tag considerando valores sem tag.

Observe que a categoria Sem Tags aparece na quarta posição. Uma interpretação possível é que podem ser fotos muito específicas ou quando faço algo muito genérico e tem um engajamento bom.

```
# Mostrando novamente a tabela de curtidas por tag  
base.groupby("Tags") [["Curtidas", "Comentários"]].mean().sort_values("Curtidas", ascending=False)
```

Tags	Curtidas	Comentários
Promoções	27,458.33	531.00
Datas comemorativas	20,752.25	343.50
Trends	20,024.00	352.25
Sem Tags	15,347.88	207.75
Influenciadores	15,197.29	161.71
Novos Produtos	10,304.89	198.56
Dicas de como usar	6,833.40	133.60
Produtos	6,269.82	94.12
Loja	2,865.00	17.67

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Podemos voltar como NaN caso a gente queira somente ignorar esses valores conforme orientado.

Podemos utilizar o Numpy:

```
import numpy as np  
base.loc[base.Tags == 'Sem Tags','Tags'] = np.nan
```

E essas linhas novamente param de ser consideradas na agregação.

The screenshot shows a Jupyter Notebook cell with the following code:

```
# Podemos voltar como NaN caso a gente queira somente ignorar esses valores conforme orientado  
import numpy as np  
base.loc[base.Tags == 'Sem Tags','Tags'] = np.nan  
  
# E voltamos com as colunas com valores nulos  
base[base.Tags.isnull()]
```

A blue box highlights the line `base[base.Tags.isnull()]` with the annotation "Transforma em valor nulo".

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
11	Foto	2021-10-12	17831	391	NaN	S	S	NaN	18222
19	Foto	2021-12-12	16086	268	NaN	S	S	NaN	16354
29	Foto	2022-01-19	8612	142	NaN	S	N	NaN	8754
38	Foto	2022-02-15	17687	213	NaN	S	N	NaN	17900
41	Foto	2022-02-22	12530	90	NaN	S	N	NaN	12620
43	Foto	2022-03-04	24399	266	NaN	S	S	NaN	24665
49	Foto	2022-03-22	9087	106	NaN	S	S	NaN	9193
50	Foto	2022-03-26	16551	186	NaN	S	N	NaN	16737

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Agora que a gente já tem a nossa base de Tag, podemos relacionar pessoas com Tag:

```
base.groupby(['Pessoas','Tags'])[['Curtidas','Comentários']].mean()
```

Podemos concluir que todas as tags com pessoas têm um valor bom.

Comparando Novos Produtos com pessoas e sem pessoas, é possível observar que ele funciona muito melhor com pessoas.

```
# Fazendo para Pessoas e Tag  
base.groupby(['Pessoas','Tags'])[['Curtidas','Comentários']].mean()
```

Pessoas	Tags	Curtidas	Comentários
		N	Loja
S	Novos Produtos	5,359.00	62.00
	Produtos	4,450.67	60.78
	Datas comemorativas	20,752.25	343.50
N	Dicas de como usar	6,833.40	133.60
	Influenciadores	15,197.29	161.71
	Novos Produtos	10,923.12	215.62
T	Produtos	8,316.38	131.62
	Promoções	27,458.33	531.00
	Trends	20,024.00	352.25

Sem pessoas

Com pessoas

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Se a gente quiser ordenar por curtidas podemos utilizar o sort\_values.

```
base.groupby(['Pessoas','Tags'])[['Curtidas','Comentários']].mean().sort_values("Curtidas",ascending=False)
```

Observe que tudo o que tem pessoas tem uma boa quantidade de curtidas.

# Também podemos ordenar por curtidas

```
base.groupby(['Pessoas','Tags'])[['Curtidas','Comentários']].mean().sort_values("Curtidas",ascending=False)
```

Pessoas	Tags	Curtidas	Comentários
S	Promoções	27.458.33	531.00
	Datas comemorativas	20.752.25	343.50
	Trends	20.024.00	352.25
	Influenciadores	15.197.29	161.71
	Novos Produtos	10.923.12	215.62
	Produtos	8.316.38	131.62
	Dicas de como usar	6.833.40	133.60
N	Novos Produtos	5.359.00	62.00
	Produtos	4.450.67	60.78
	Loja	2.865.00	17.67

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Podemos aplicar o mesmo para Campanhas e Tags.

```
base.groupby(['Campanhas','Tags'])[['Curtidas','Comentários']].mean().sort_values("Curtidas", ascending=False)
```

Observe que algumas vezes, mesmo não tendo campanha, promoção funciona.

Campanhas não é tão determinante assim em algumas Tags.

# Fazendo para Campanhas e Tag		
Campanhas	Tags	Curtidas Comentários
S	Promoções	33.217,50 490,50
	Trends	22.400,67 386,67
	Datas comemorativas	20.752,25 343,50
	Influenciadores	18.715,40 197,60
N	Promoções	15.940,00 612,00
	Trends	12.894,00 249,00
S	Novos Produtos	11.040,67 323,00
N	Novos Produtos	9.937,00 136,33
S	Produtos	9.074,00 67,50
N	Dicas de como usar	6.833,40 133,60
	Influenciadores	6.402,00 72,00
	Produtos	5.895,93 97,67
	Loja	2.865,00 17,67

Valor muito mais alto em campanhas

Só faz sentido fazer Trends em campanhas

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Com tudo isso que observamos, podemos fechar a nosso análise com algumas conclusões:

- Ter o **rosto de outras pessoas é fundamental para um bom engajamento** na publicação.
  - Em todas as tags, quando havia o rosto, o resultado foi muito melhor.
- Criar **campanhas ajuda muito na divulgação** da marca.
- **Promoções** tiveram um desempenho absurdamente **maior que qualquer outra tag**.
  - Porém é uma tag que pode ter **custo para a loja**, o que deve ser analisado.

## Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

- Usar conteúdos que estão em **trend** também ajudam na divulgação da marca, **mesmo a trend sendo de outros nichos** .
- A melhor maneira de **mostrar produtos** é através de outras **pessoas utilizando-os**, e se possível em campanhas de datas especiais.
- Para **novos produtos** a inclusão de **pessoas** é mais crítica ainda, sendo quase o dobro quando há um rosto junto ao produto.
- **Não** podemos afirmar que a **tag Loja é ruim** até testarmos essa tag incluindo pessoas ou em uma campanha. Vale o teste para verificar.
- Continuaremos a monitorar as postagens para encontrar novos padrões, dado que ainda temos poucas informações da base.

## MÓDULO 6

# Introdução a Estatística



# Módulo 6 – Introdução a Estatística e Estatística Descritiva

O que é estatística?

Descrição

```
base.describe()
```

	altura
count	8.000000
mean	1.728750
std	0.058661
min	1.650000
25%	1.695000
50%	1.720000
75%	1.755000
max	1.840000



Tabela de Frequência

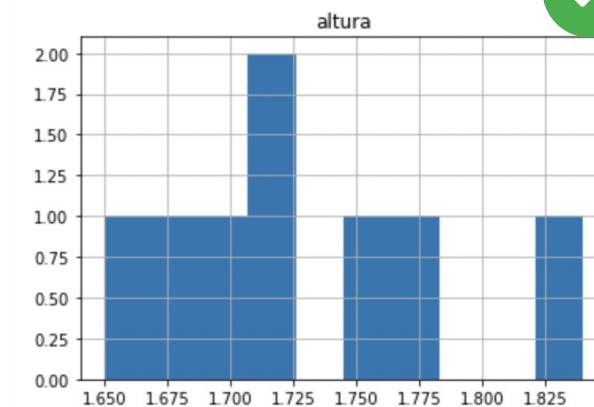
```
pd.crosstab(index=base.altura,columns='count')
```

col_0	count
altura	
1.65	1
1.68	1
1.70	1
1.72	2
1.75	1
1.77	1
1.84	1



Histograma

```
base.hist();
```



Amostragem

```
base.sample(3)
```

alunos	altura	
2	Jean	1.70
6	Vanessa	1.77
0	Lucas	1.65



## Módulo 6 – Introdução a Estatística e Estatística Descritiva

Podemos dizer que tudo isso é estatística, desde a parte de coleta até a análise de dados.

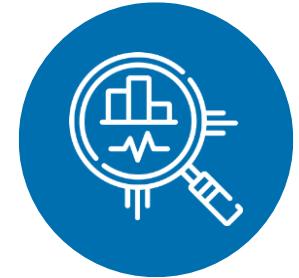
**COLETA**



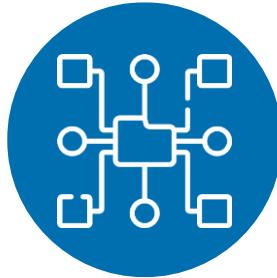
**APRESENTAÇÃO**



**ANÁLISE**



**ORGANIZAÇÃO**



**INTERPRETAÇÃO**



## Módulo 6 – Introdução a Estatística e Estatística Descritiva

Na estatística também podemos fazer:

- Inferências
- Amostragem: para obter uma confiança da amostra que estamos trabalhando.
- Machine learning: utiliza estatística para fazer previsões.

### INFERÊNCIAS



### AMOSTRAGEM

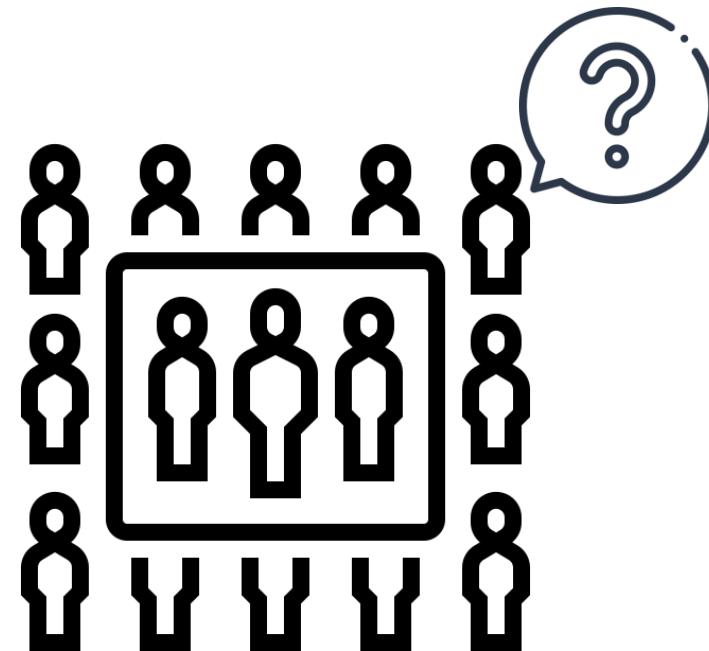


### MACHINE LEARNING



## Módulo 6 – Introdução a Estatística e Estatística Descritiva

Com a estatística nós vamos representar todas as informações que estão em 1000 linhas através de **indicadores** (média, mediana, tabela de frequência), de forma que seja **prático** trabalhar com essas informações.



## Módulo 6 – Introdução a Estatística e Estatística Descritiva

Estatística descritiva é a **Descrição** e **apresentação** dos dados de forma a facilitar o entendimento de um grande conjunto de dados

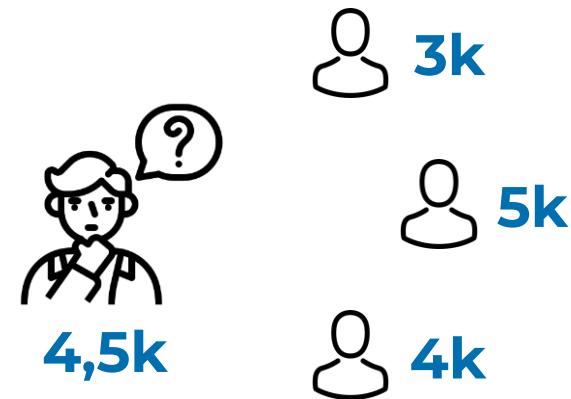
Fornece um resumo sobre os dados apresentados, podendo ser **visual** (tabelas de frequência, histogramas, gráficos) ou **quantitativo** (media, mediana, moda)

## Módulo 6 – Introdução a Estatística e Estatística Descritiva

Vamos analisar um exemplo: supondo que você recebeu uma promoção e seu salário a partir de agora é de 4,5K.

E aí surgem os questionamentos: **É um salário bom? Faz sentido esse salário?**

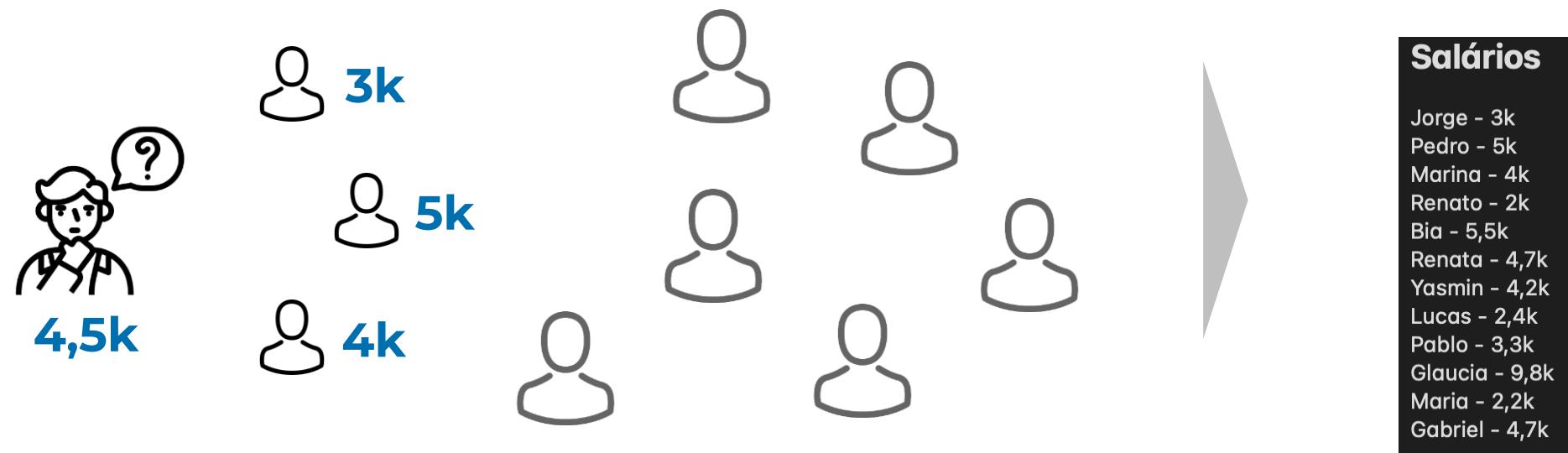
Perguntando a 3 amigos, você percebe que o seu salário é um valor bom.



## Módulo 6 – Introdução a Estatística e Estatística Descritiva

Você decide continuar perguntando e tem uma lista maior de dados. Como você vai comparar o seu salário agora?

Fica **inviável** fazer comparações listando todos os dados.



## Módulo 6 – Introdução a Estatística e Estatística Descritiva

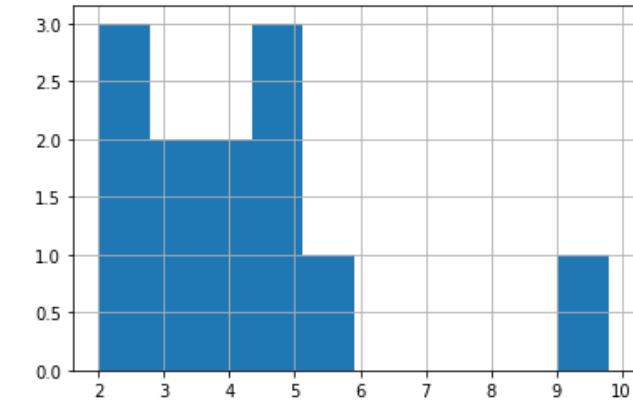
Quando transformamos a nossa tabela em um histograma e colocamos indicadores (média, mediana e moda), podemos concluir que o seu salário está acima da média.

A **estatística** fornece com poucos indicadores **informações** muito importantes sobre a base inteira.



Salários	
Jorge	- 3k
Pedro	- 5k
Marina	- 4k
Renato	- 2k
Bia	- 5,5k
Renata	- 4,7k
Yasmin	- 4,2k
Lucas	- 2,4k
Pablo	- 3,3k
Glaucia	- 9,8k
Maria	- 2,2k
Gabriel	- 4,7k

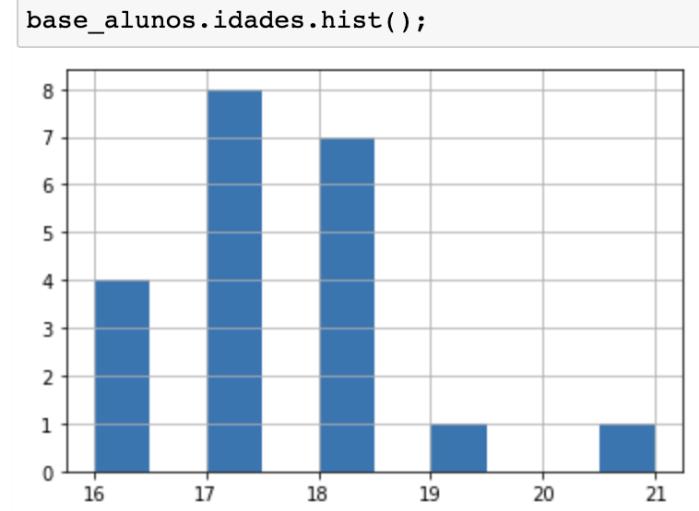
média: 4.23333333333333  
mediana: 4.1  
moda: 4.7



## Módulo 6 – Tabela de frequência e histograma

Agora que a gente já entendeu que a estatística é utilizada para explicar dados sobre uma base e mostrar de forma gráfica ou quantitativa informações sobre uma base, vamos mostrar como podemos fazer isso utilizando a **tabela de frequência** e o **histograma**.

```
base = {  
    "id_aluno": [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21],  
    "idades": [16,16,16,16,17,17,17,17,17,17,17,17,17,18,18,18,18,18,18,19,21]  
}  
  
base = pd.DataFrame(base)  
  
print(pd.crosstab(index=base.idades,columns="contagem"))  
  
col_0  contagem  
idades  
16        4  
17        8  
18        7  
19        1  
21        1
```



## Módulo 6 – Tabela de frequência e histograma

A tabela de frequência apresenta os dados e a **quantidade de ocorrência / repetições daquele dado.**

Vamos supor que você tem a idade dos alunos e deseja informar quantos alunos possuem 17 anos.

Olhando para a lista você não consegue dizer rapidamente.

Agora, criando um tabela de frequência, listando a idade e o total de alunos, já temos muita informação sobre a base.

Difícil visualizar informação

```
base = {
    "id_aluno": [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21],
    "idades": [16,16,16,16,17,17,17,17,17,17,17,17,17,17,17,17,17,18,18,18,18,18,19,21]
}

base = pd.DataFrame(base)

print(pd.crosstab(index=base.idades,columns="contagem"))
```

col_0	contagem
idades	
16	4
17	8
18	7
19	1
21	1

Informação resumida

ALUNOS	ALUNOS
16	4
17	8
18	7
19	1
21	1

## Módulo 6 – Tabela de frequência e histograma

Vejamos um outro exemplo: em 2018, cerca de 30% dos funcionários de uma determinada empresa recebiam um salário de 4000 a 5500.

Em 2022, esse número diminuiu, mas em compensação o número de pessoas com salário acima de 4000 aumentou.

Podemos concluir que a empresa está aumentando os salários.

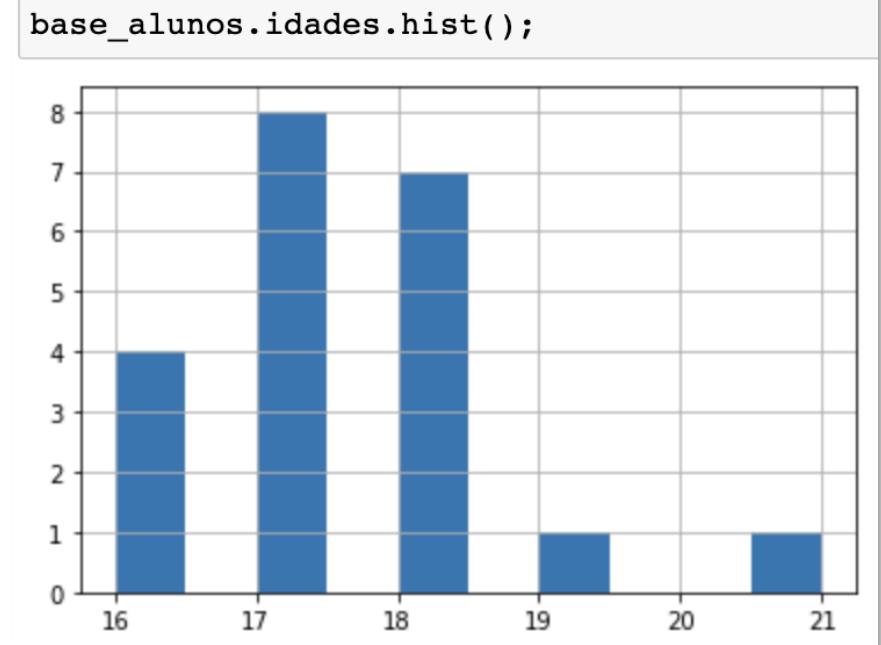
FAIXA SALARIAL	FREQUÊNCIA		FREQUÊNCIA ACUMULADA	
	2018	2022	2018	2022
1.000 a 2.500	5%	3%	5%	3%
2.500 a 4.000	21%	18%	26%	21%
4.000 a 5.500	38%	25%	64%	46%
5.500 a 7.000	14%	25%	78%	71%
7.000 a 8.500	12%	17%	90%	88%
8.500 a 10.000	7%	9%	97%	97%
acima 10.000	3%	3%	100%	100%

## Módulo 6 – Tabela de frequência e histograma

O **histograma** é basicamente a **exibição gráfica** de uma distribuição de frequências.

Pelo histograma ao lado podemos observar a distribuição de alunos.

Podemos escolher o histograma ou a tabela de frequência para representar distribuição, escolher com base na análise que estamos criando.



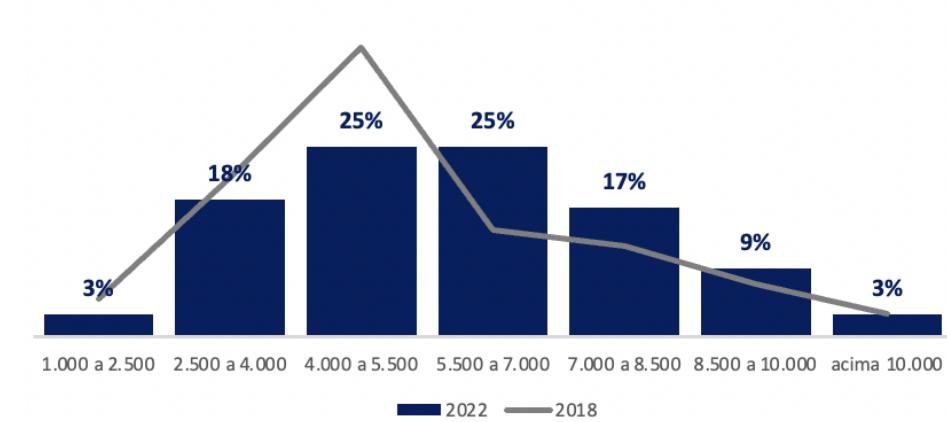
## Módulo 6 – Tabela de frequência e histograma

Podemos ver que tanto a tabela de frequência quanto o histograma passam a mensagem que os salários estão melhorando.

FAIXA SALARIAL	FREQUÊNCIA	
	2018	2022
1.000 a 2.500	5%	3%
2.500 a 4.000	21%	18%
4.000 a 5.500	38%	25%
5.500 a 7.000	14%	25%
7.000 a 8.500	12%	17%
8.500 a 10.000	7%	9%
acima 10.000	3%	3%



Crescimento de faixas salariais maiores em nossa empresa

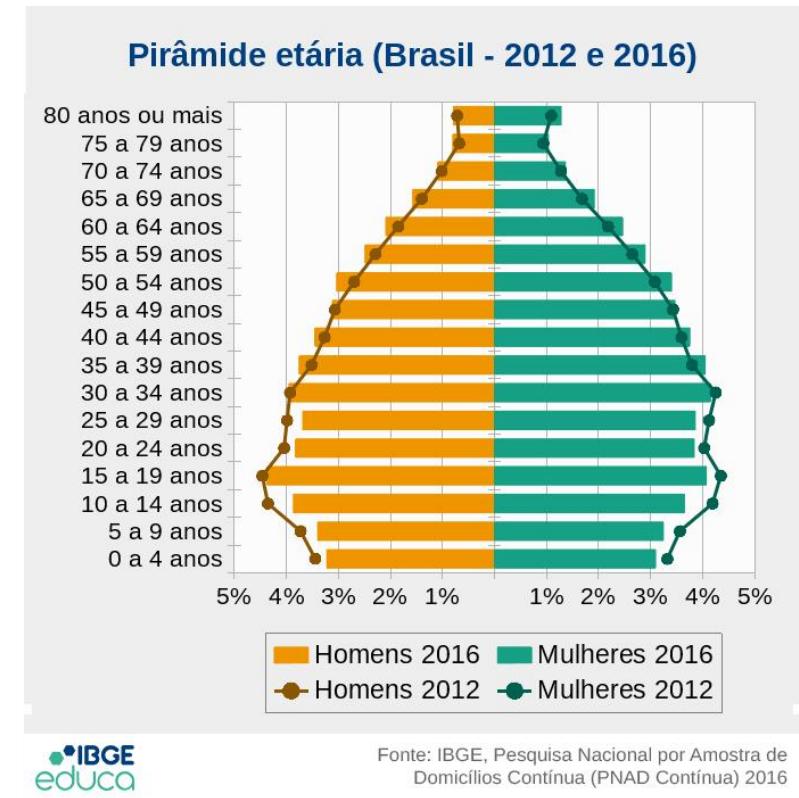


## Módulo 6 – Tabela de frequência e histograma

Utilizamos o histograma também para mostrar a população.

Por exemplo, no histograma de pirâmide etária é possível notar que em 2012 tínhamos mais pessoas mais novas, agora o numero de pessoas mais novas caiu.

Pode ser que poucas pessoas estão nascendo ou menos pessoas querendo ter filhos.



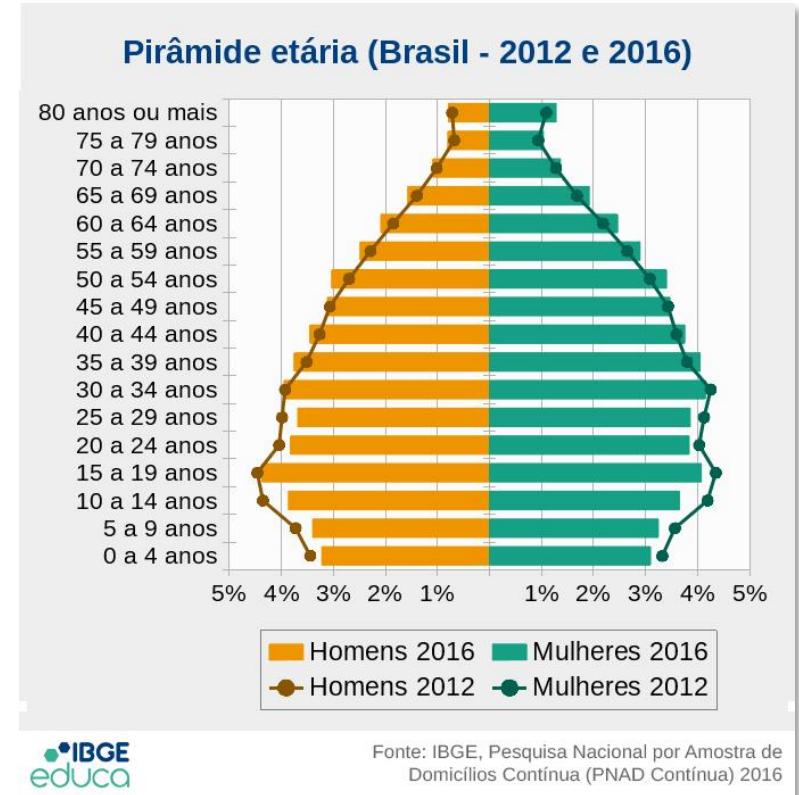
## Módulo 6 – Tabela de frequência e histograma

Em compensação as idades maiores estão aumentando ao longo dos anos.

Então a expectativa de vida está maior e as pessoas estão vivendo mais tempo.

Podemos usar essa informação para entender a questão de aposentadoria.

Se temos menos pessoas contribuindo e mais pessoas estão ficando mais velhas, precisamos entender melhor como vamos fazer um plano de aposentadoria que consiga atender a todos.

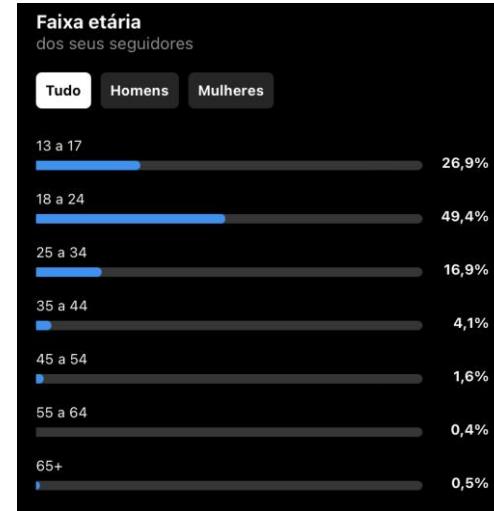


## Módulo 6 – Tabela de frequência e histograma

Podemos utilizar informações estatísticas para rede social para aumentar o engajamento.

Podemos visualizar a faixa etária dos seguidores, períodos mais ativos, melhor dia e horário para postar.

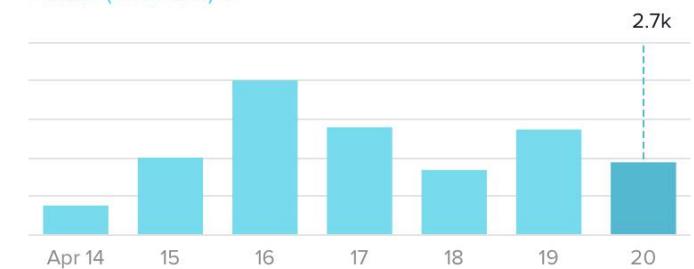
Todas essas informações ajudam a aumentar o engajamento.



### Visualizações de vídeo

22.7K

+15.8K (229.49%) ↑



## Módulo 6 – Entendendo o conceito de média

Agora vamos entender como mostrar informações estatísticas de forma quantitativa.

Resumo estatístico é uma forma de apresentar os principais conceitos sobre a distribuição dos dados de forma resumida, usando para isso:

- Média
- Mediana
- Moda
- Desvio Padrão

## Módulo 6 – Entendendo o conceito de média

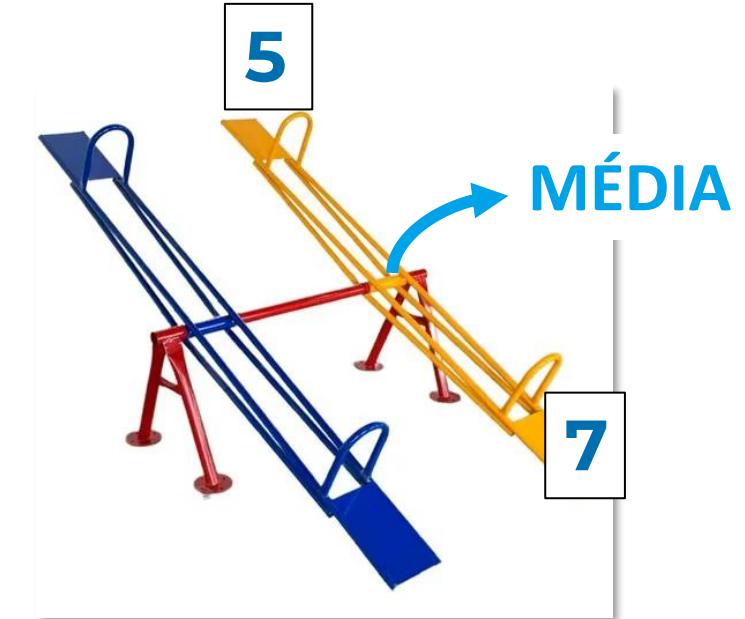
É basicamente a posição central de determinado conjunto de dados, é análogo ao “meio” (ou o centro de massa físico).



## Módulo 6 – Entendendo o conceito de média

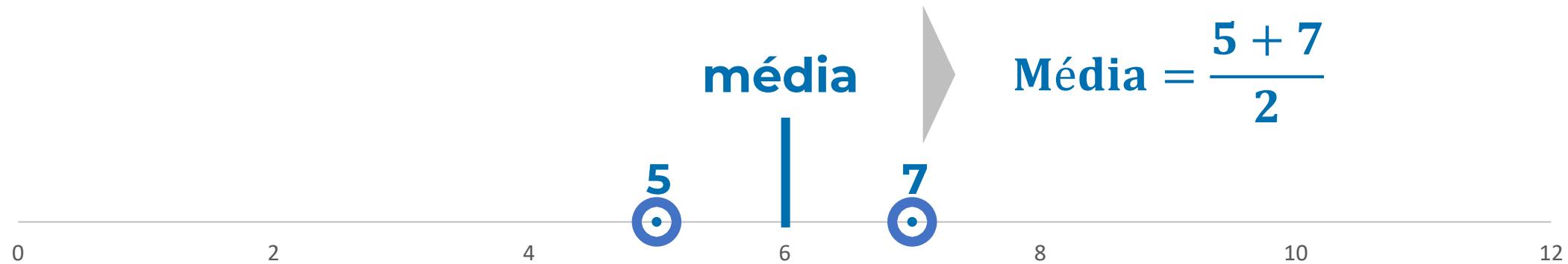
Vamos supor que estamos com dois números: 5 e 7.

Como que conseguiríamos o ponto de equilíbrio entre esses dois números?



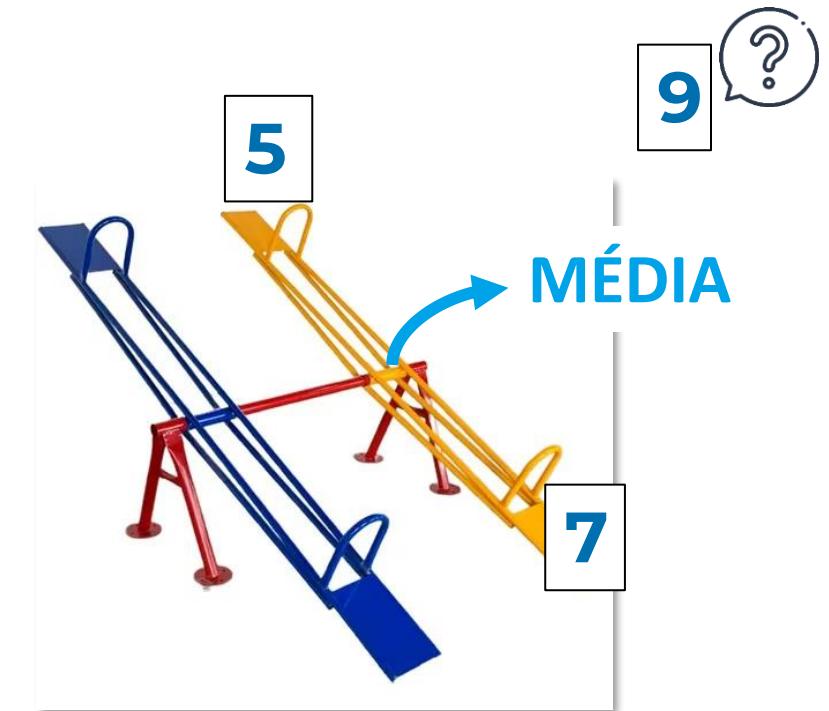
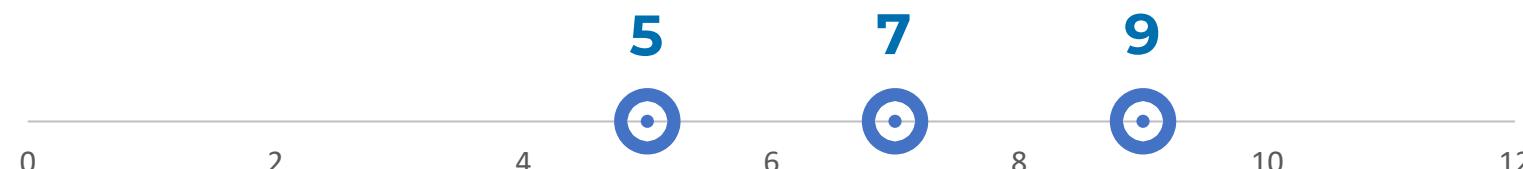
## Módulo 6 – Entendendo o conceito de média

A média que estamos falando desse sistema seria exatamente o 6, que é o meio desses 2 números.



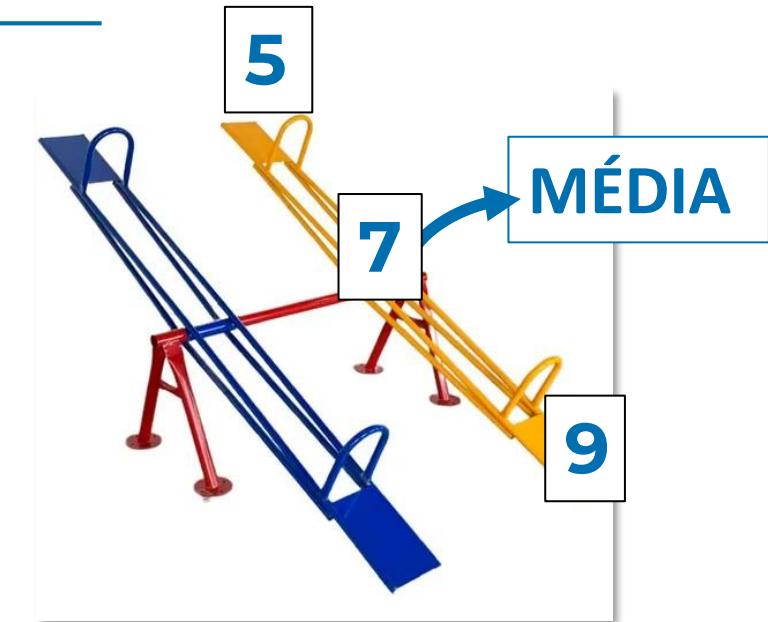
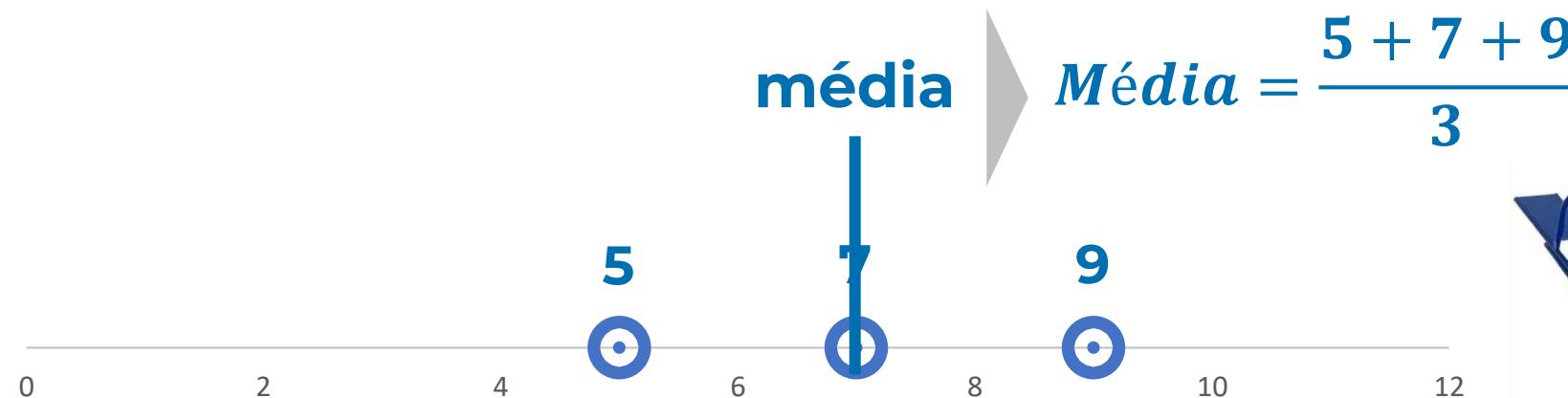
## Módulo 6 – Entendendo o conceito de média

Vamos supor que colocamos um outro número. Como calculamos a média?



## Módulo 6 – Entendendo o conceito de média

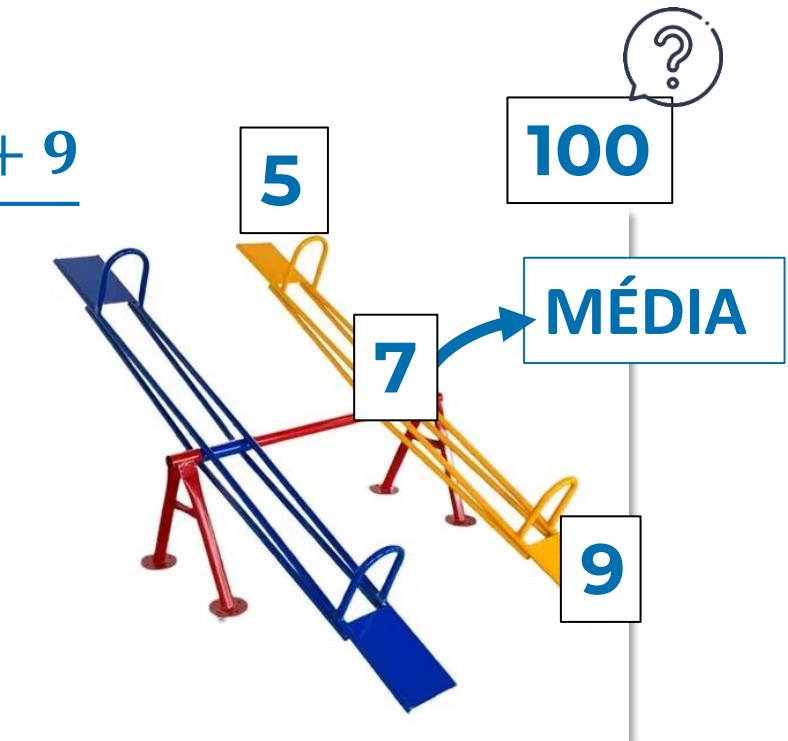
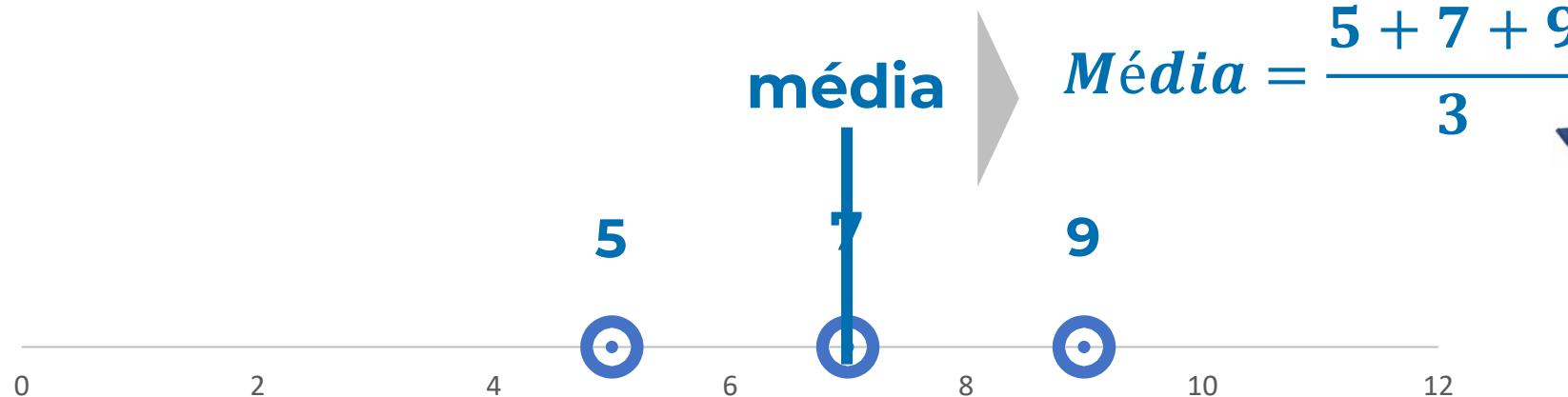
Vamos supor que colocamos um outro número. Como calculamos a média?



## Módulo 6 – Entendendo o conceito de média

E se agora tivéssemos um *outlier*, por exemplo, o número 100.

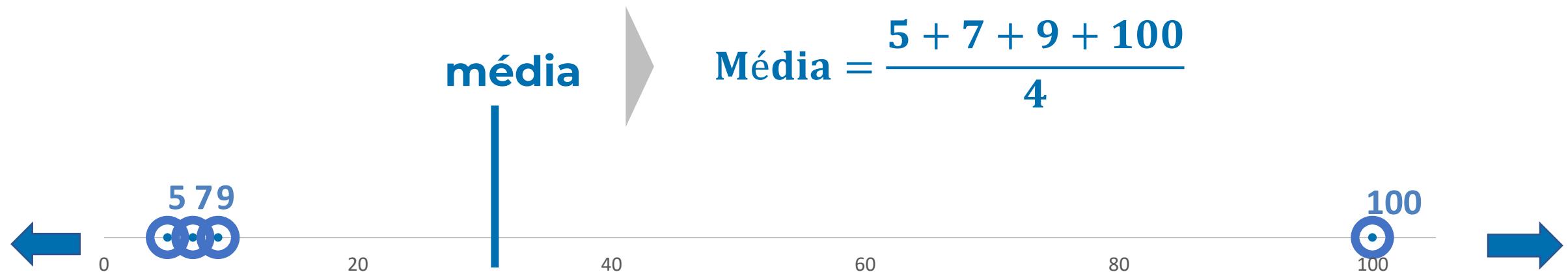
Como calculamos a média?



## Módulo 6 – Entendendo o conceito de média

Observa que o 100 é um número mais distante, então ele deslocou muito a média por ser um valor muito maior se comparado aos demais.

A **média é muito sensível a valores muito altos** e nós temos que tomar cuidado para não interpretar resultados de forma imprecisa apenas utilizando a média.



# Módulo 6 – Entendendo o conceito de média

Para fazer a média no Pandas, basta utilizar o comando **mean**.

Estrutura: **base.mean()**

A média nos ajuda a **começar a entender a distribuição dos nossos dados**

```
base = {  
    "valores": [5,7]  
}  
  
base = pd.DataFrame(base)  
  
print(base.mean())  
  
valores    6.0  
dtype: float64
```

```
base = {  
    "valores": [5,7,9]  
}  
  
base = pd.DataFrame(base)  
  
print(base.mean())  
  
valores    7.0  
dtype: float64
```

```
base = {  
    "valores": [5,7,9,100]  
}  
  
base = pd.DataFrame(base)  
  
print(base.mean())  
  
valores    30.25  
dtype: float64
```

A média é muito **influenciada por outliers!**

## Módulo 6 – Mediana e sua relação com a média

A mediana é o valor central de um conjunto de dados (ou seja, o valor que está no meio).

**Obs:** para isso, os dados precisam estar **ordenados!**

[1 , 5 , 100 , 12 , 28 , 37 , 9 ]

O que a mediana vai nos dizer?

A **mediana** vai nos dizer metade dos valores abaixo da mediana e metade acima, então se eu tenho **outlier não vai influenciar tanto.**

## Módulo 6 – Mediana e sua relação com a média

Vamos observar um exemplo com a lista de valores abaixo:

[1, 5, 100, 12, 28, 37, 9]



O primeiro passo é **ORDENAR OS DADOS!**

[1, 5, 9, 12, 28, 37, 100]



A **MEDIANA** será o valor na posição **central** do nosso conjunto de dados

Então, podemos dizer que temos 3 valores antes do 12 e 3 valores depois.

## Módulo 6 – Mediana e sua relação com a média

Agora, por exemplo, se tivéssemos uma quantidade par na nossa lista.

[1 , 5 , 9 , 12 , 15 , 28 , 37 , 100 ]



No caso de um número par de elementos, **a mediana será a media entre os 2 elementos centrais**



$$\text{Mediana} = \frac{12 + 15}{2}$$

# Módulo 6 – Mediana e sua relação com a média

Como que podemos relacionar a mediana com a média?

```
base = {  
    "valores": [5,7]  
}  
  
base = pd.DataFrame(base)  
  
print(base.median())  
  
valores    6.0  
dtype: float64
```

Média igual a mediana

```
base = {  
    "valores": [5,7,9]  
}  
  
base = pd.DataFrame(base)  
  
print(base.median())  
  
valores    7.0  
dtype: float64
```

Média igual a mediana

```
base = {  
    "valores": [5,7,9,100]  
}  
  
base = pd.DataFrame(base)  
  
print(base.median())  
  
valores    8.0  
dtype: float64
```

Média é de 30,25 e mediana 8.

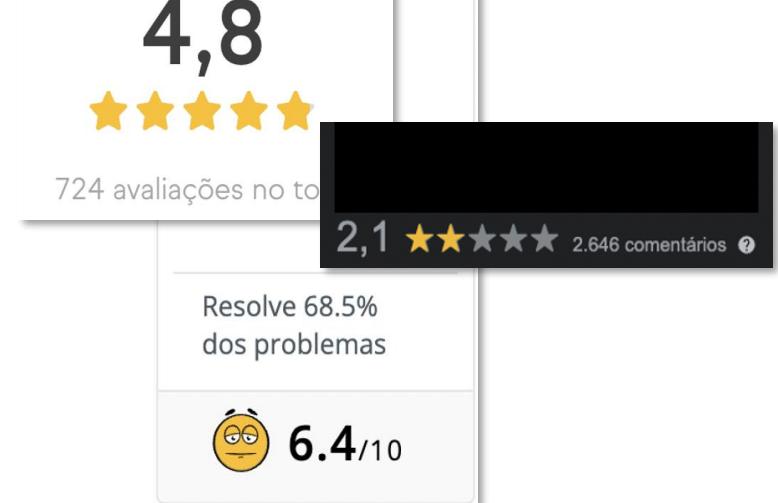
A mediana é **pouco influenciada por outliers**

## Módulo 6 – Mediana e sua relação com a média

Utilizamos média e mediana no nosso dia a dia:

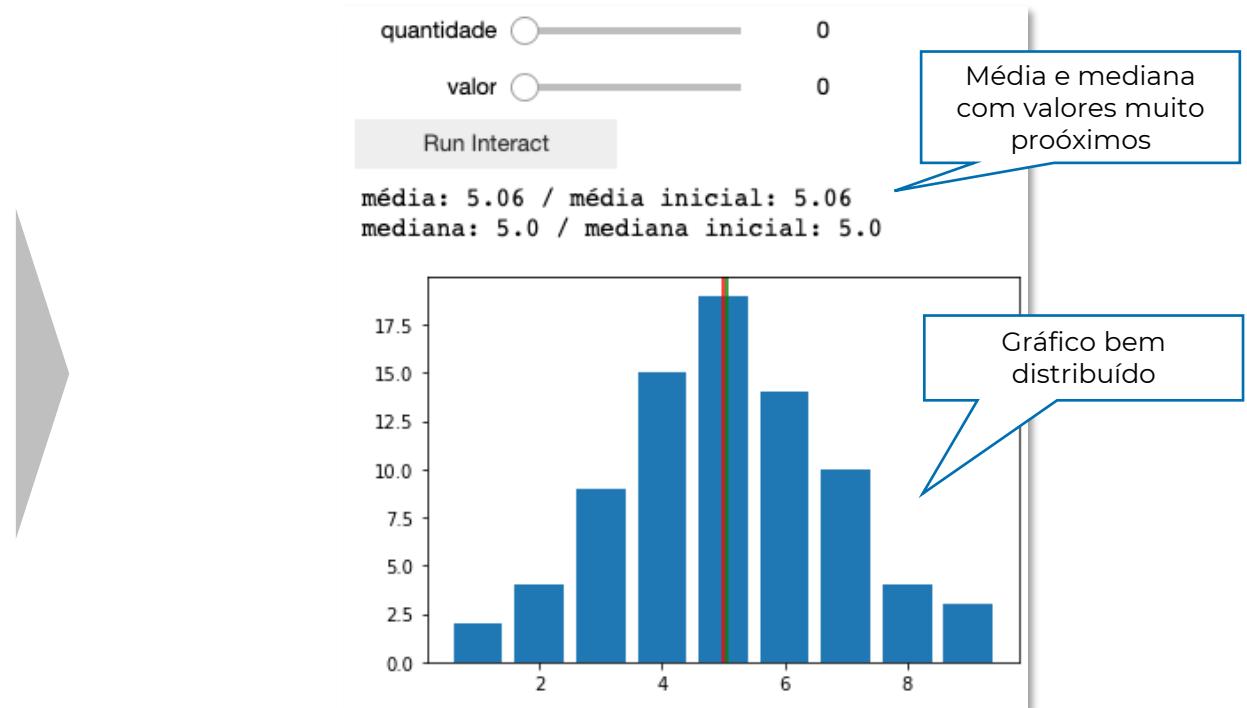
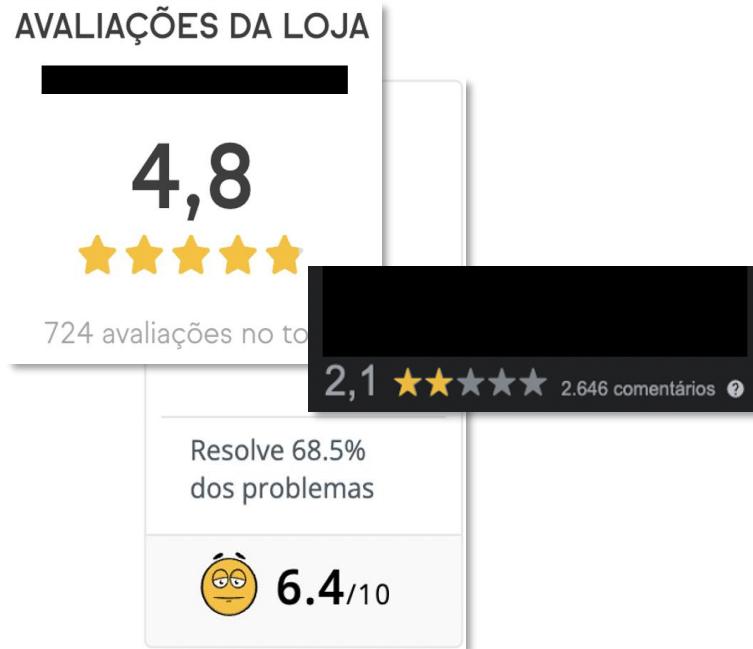
- Avaliação de loja no Reclame aqui
- Avaliação do motorista da uber

### AVALIAÇÕES DA LOJA



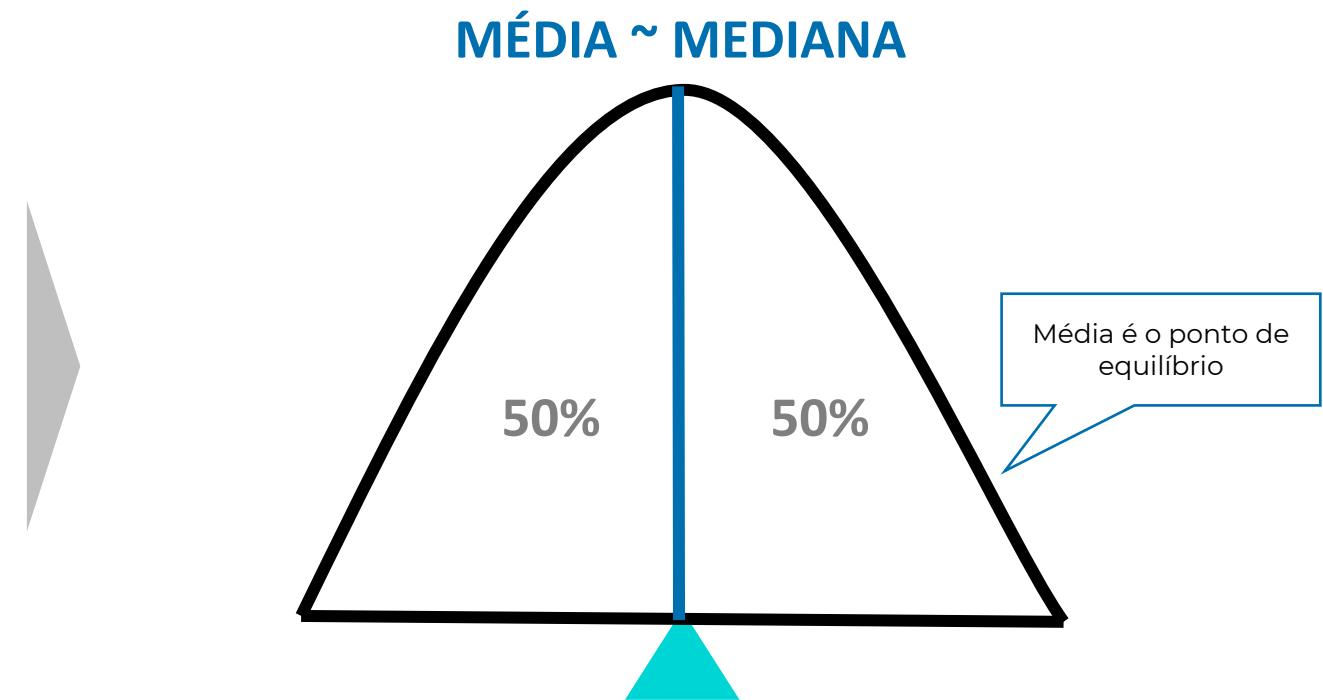
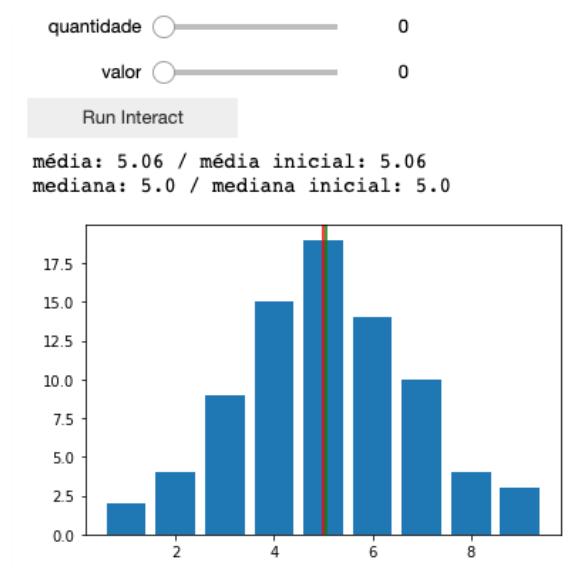
## Módulo 6 – Usando Python para entender a relação entre média e mediana

Ao começar a relacionar mediana com a média, já começamos a ter uma ideia da distribuição dos nossos dados



## Módulo 6 – Usando Python para entender a relação entre média e mediana

Ao fazer um desenho passando por todos os pontos de maior frequência, obtemos algo muito próximo de um sino, conforme imagem à direita.



## Módulo 6 – Usando Python para entender a relação entre média e mediana

Vamos ver agora como alterar a **média** e **mediana**.

Supondo que temos a base abaixo onde todos os valores são 5, nesse caso a **média** e **mediana** são **iguais**.

```
1 2 3 4 5 6 7 8 9 10 11  
notas2 = np.array([5,5,5,5,5,5,5,5,5,5,5])  
  
print(np.median(notas2))  
print(np.mean(notas2))  
print(len(notas2))  
  
5.0  
5.0  
11
```

Mediana é o valor da posição 6

## Módulo 6 – Usando Python para entender a relação entre média e mediana

Agora, se adicionarmos um valor 6 nessa base, podemos notar que apenas a **média** altera.

Como a **mediana** é o valor da posição central, por mais que o valor não tenha sido alterado, observe que a posição foi.

```
1 2 3 4 5 6 7 8 9 10 11 12  
notas2 = np.array([5,5,5,5,5,5,5,5,5,5,5,5,6])  
  
print(np.median(notas2))  
print(np.mean(notas2))  
print(len(notas2))  
  
5.0  
5.083333333333333  
12
```

Mediana é o valor da soma dos elementos da posição 6 e 7 dividido por 2

## Módulo 6 – Usando Python para entender a relação entre média e mediana

Observe que para alterar a **média** foi necessário apenas acrescentar um valor, enquanto a **mediana** alterou somente a posição do elemento.

Então, se adicionarmos um valor de 100 na nossa base, ele afeta a **média** e a **mediana** continua com o mesmo valor.

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14
notas2 = np.array([5,5,5,5,5,5,5,5,5,5,6,6,6,100])

print(np.median(notas2))
print(np.mean(notas2))
print(len(notas2))

5.0
11.928571428571429
14
```

Mediana é o valor da soma dos elementos da posição 7 e 8 dividido por 2

## Módulo 6 – Usando Python para entender a relação entre média e mediana

Só vamos conseguir alterar o valor da **mediana** se adicionarmos muitos valores a ponto de colocar o 6 como a posição central.

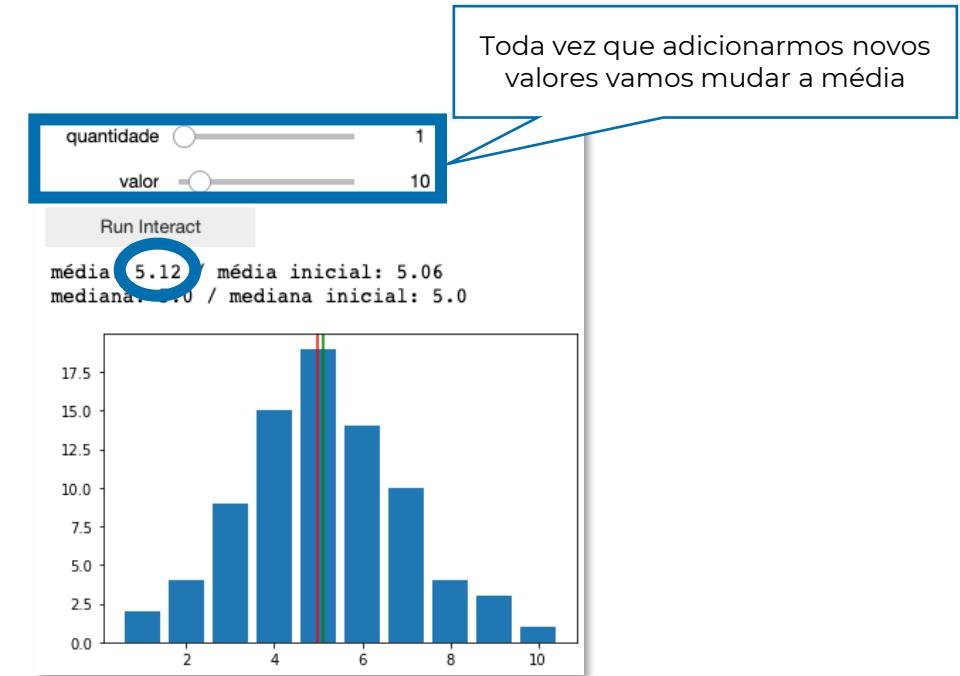
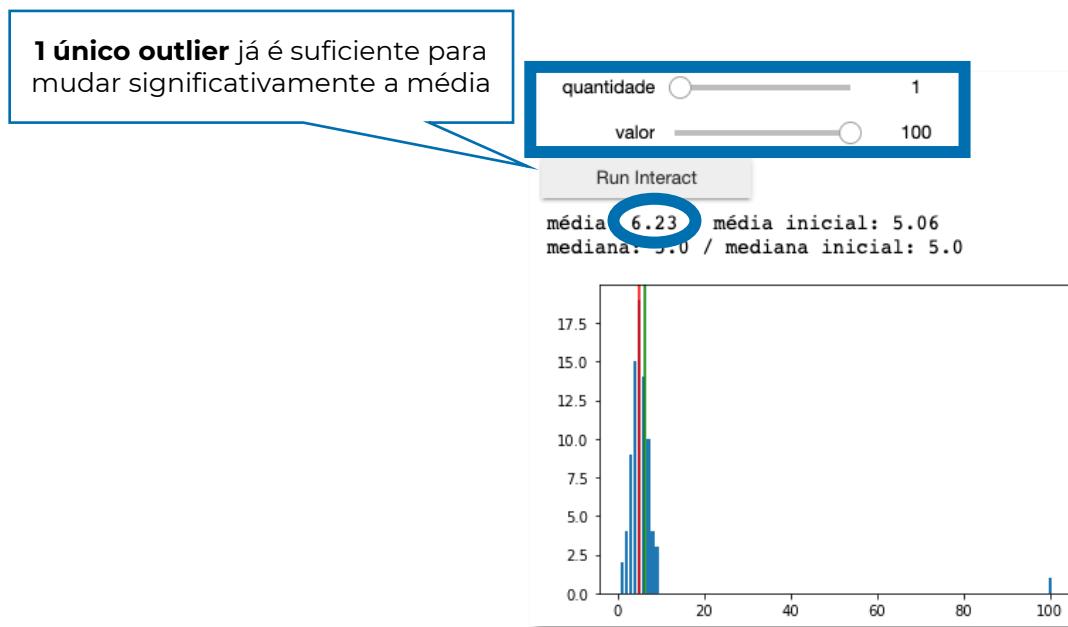
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22  
notas2 = np.array([5,5,5,5,5,5,5,5,5,5,5,5,6,6,100,100,100,100,100,100,100,100,500])  
  
print(np.median(notas2))  
print(np.mean(notas2))  
print(len(notas2))  
  
5.5  
62.13636363636363  
22
```

Mediana é o valor da soma dos elementos da posição 11 e 12 dividido por 2

# Módulo 6 – Usando Python para entender a relação entre média e mediana

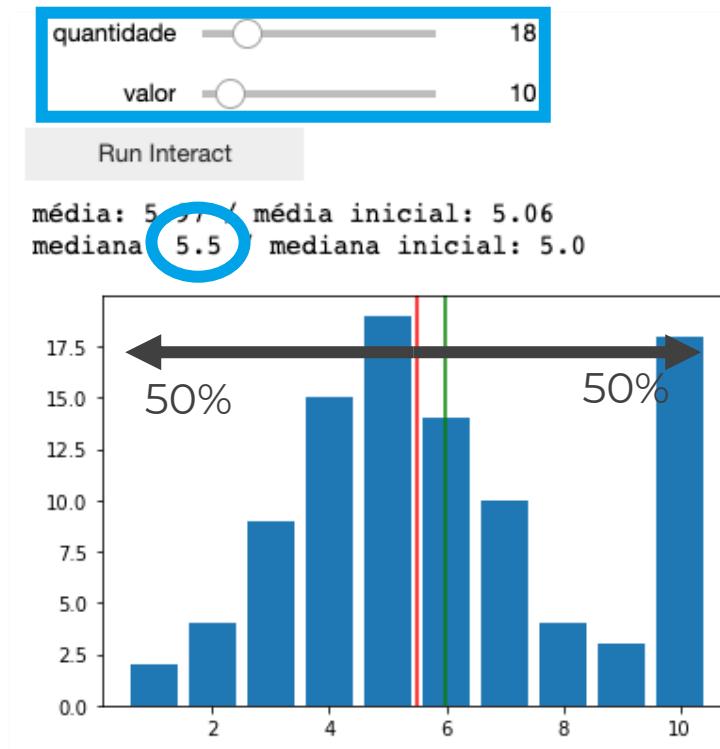
Então, a **média** alteramos com a **adição de um único valor**.

Quanto mais alto for o valor, mais alta a minha **média**.



## Módulo 6 – Usando Python para entender a relação entre média e mediana

Só vamos mudar a **mediana** quando mudarmos a **quantidade de dados** adicionada a base.



## Módulo 6 – Média, mediana e moda

Imagine que temos essa distribuição de dados abaixo e queremos adicionar o 5.

O que isso impacta na **média** e **mediana**?

[ 1 , 1 , 2 , 2 , 3 ]



Para a **média** importa esse **valor**. Se vai ser 5, 10, 1.000

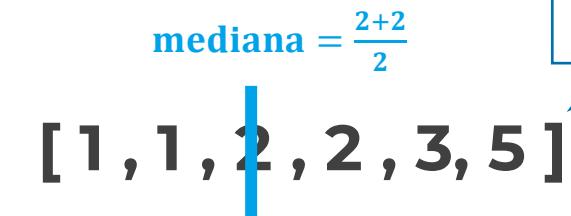
## Módulo 6 – Média, mediana e moda

Imagine que temos essa distribuição de dados abaixo e queremos adicionar o 5.

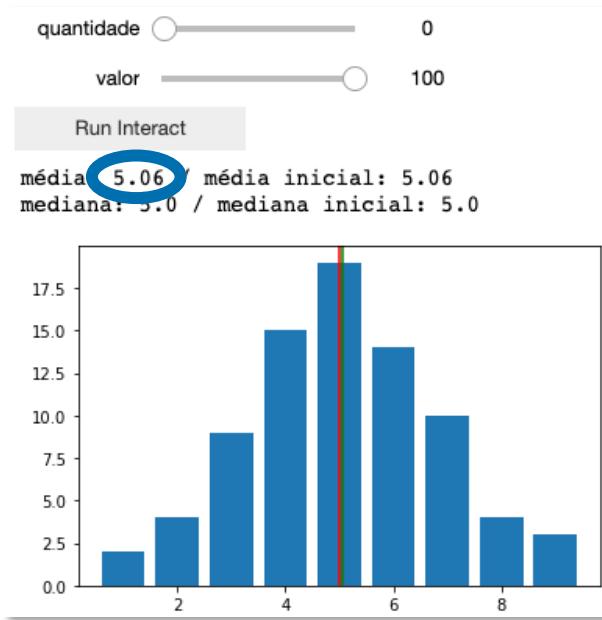
O que isso impacta na **média** e **mediana**?



Para a **mediana**, só vai importar se ele vai entrar **antes ou depois** da mediana atual (ou seja, a distribuição dos dados)



## Módulo 6 – Média, mediana e moda

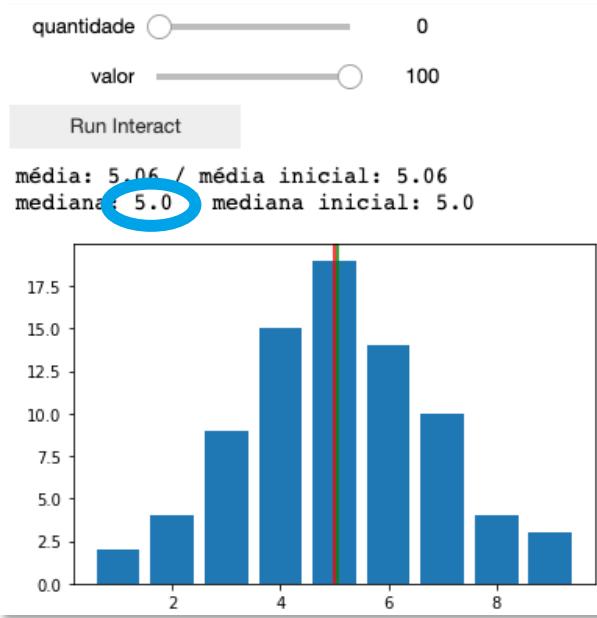


Podemos utilizar a média para **analisar a avaliação de empresas**.

Quando escolhemos um range para a nota, tornamos ele **menos sensível a outliers**.

Por exemplo: a nota varia de 0 a 10 ou de 1 a 5

## Módulo 6 – Média, mediana e moda



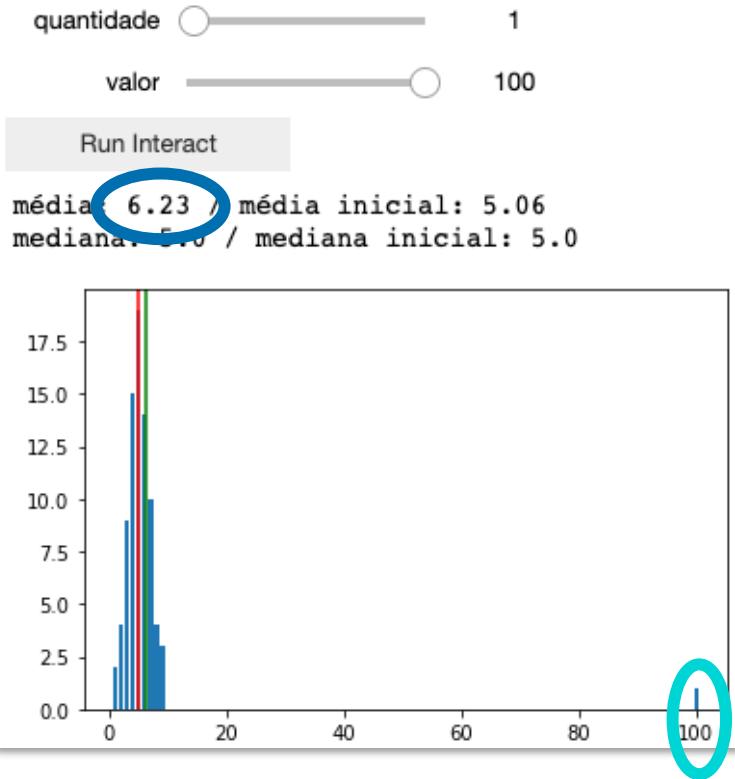
Já a **mediana** podemos usar para avaliar o **tempo de resposta de uma solicitação**.

Se ocorreu um problema no sistema que a solicitação não foi fechada, isso não vai penalizar a empresa

## Módulo 6 – Média, mediana e moda

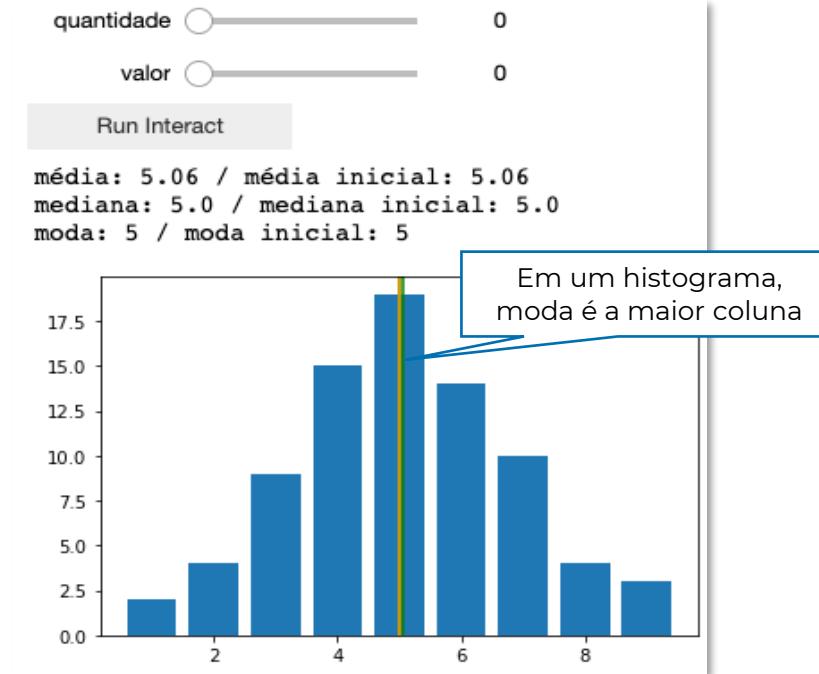
Podemos usar a **média** e os **outliers** para identificação de fraudes.

Um outro exemplo é para verificar o que está fora do padrão dos dados.



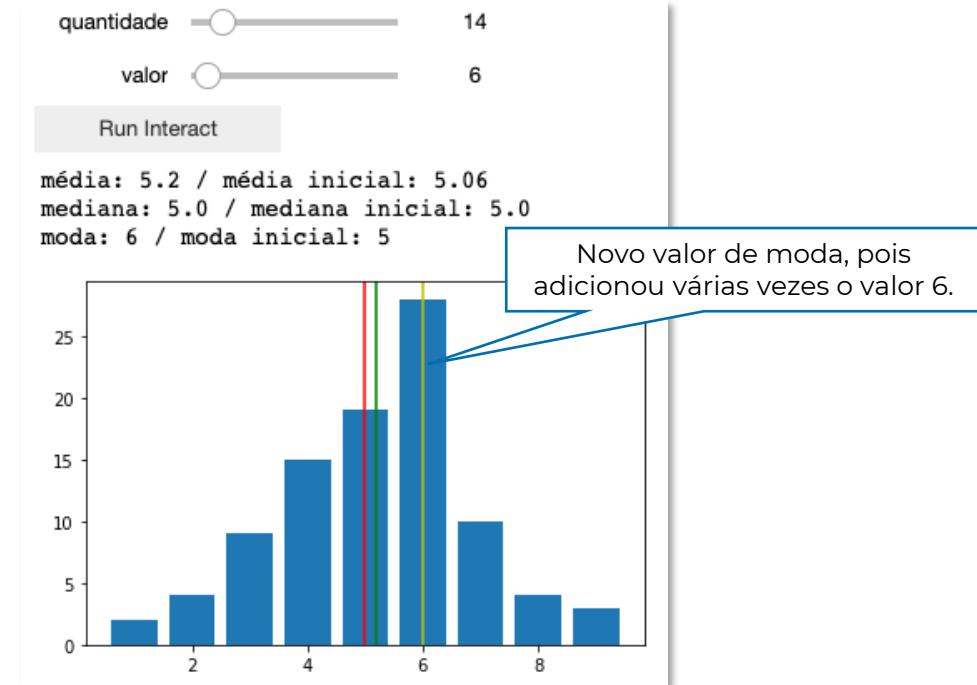
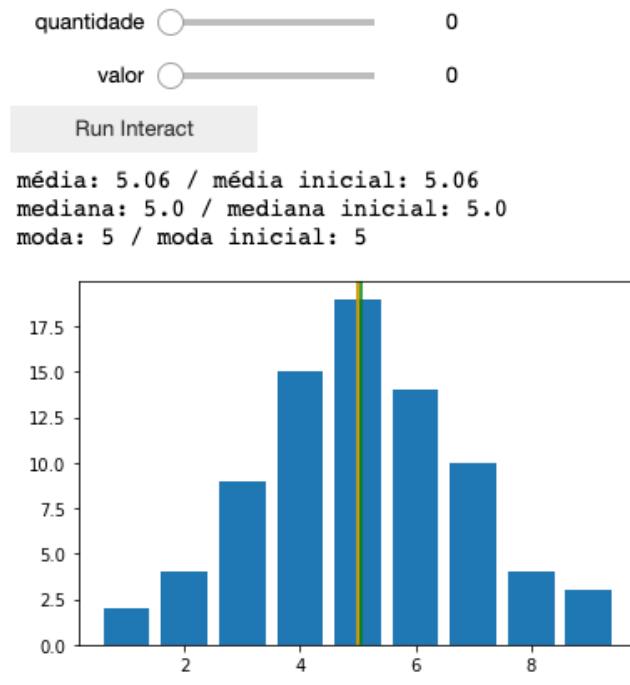
## Módulo 6 – Média, mediana e moda

Vamos agora falar de Moda. Consiste no **valor mais frequente** do conjunto de dados.



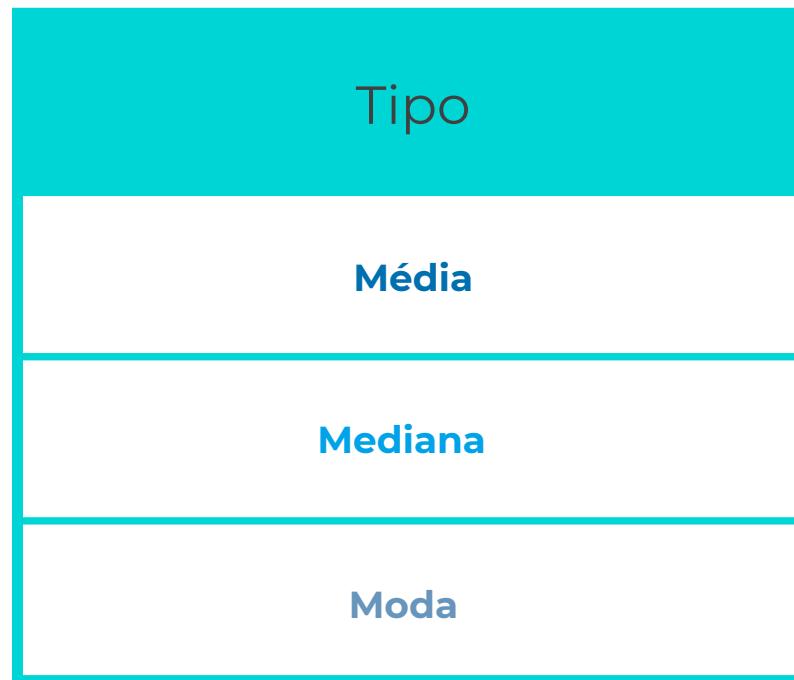
## Módulo 6 – Média, mediana e moda

Só vamos conseguir mudar a moda se a quantidade de registros adicionado for suficiente para fazer **esse valor ser o mais frequente**.



# Módulo 6 – Média, mediana e moda

Então relembrando:



## DESVANTAGENS

É influenciada pela **adição de qualquer valor.**

É influenciada pela **quantidade de valores**, independente do valor.

É influenciada pela quantidade de **valores iguais**.

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Agora vamos ver alguns exemplos práticos:

Média: 1.67

Mediana: 1.5

Moda: 1

Registros: 6

Mínimo: 1

Máximo: 3



Todos são valores inteiros

Com essas informações conseguimos entender como está a base.

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Média: 1.67

**Mediana: 1.5**

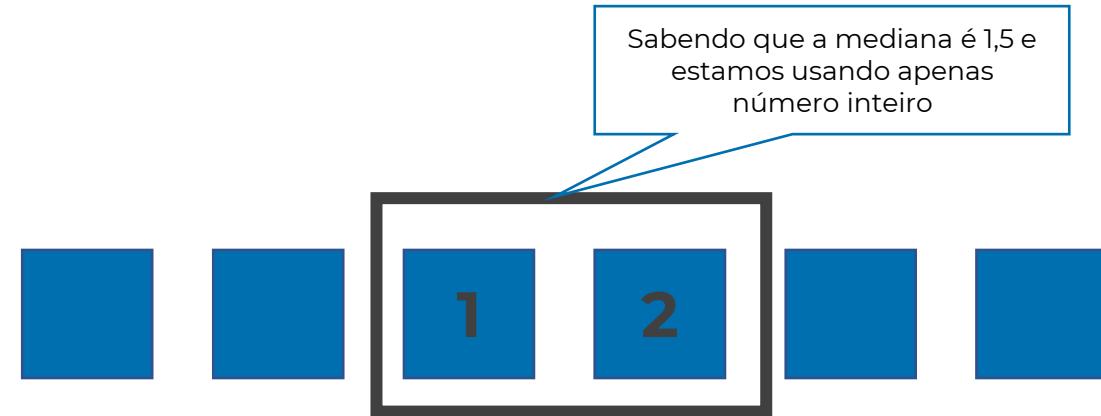
Moda: 1

Registros: 6

Mínimo: 1

Máximo: 3

**Todos são valores inteiros**



## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Média: 1.67

Mediana: 1.5

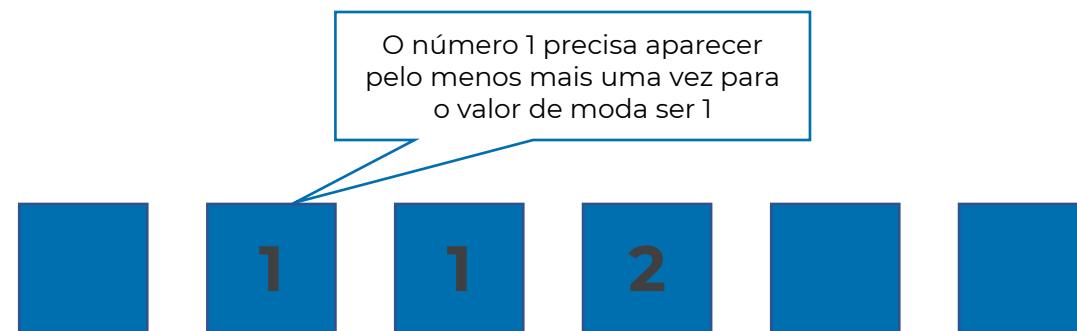
**Moda: 1**

Registros: 6

Mínimo: 1

Máximo: 3

Todos são valores inteiros



## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Média: 1.67

Mediana: 1.5

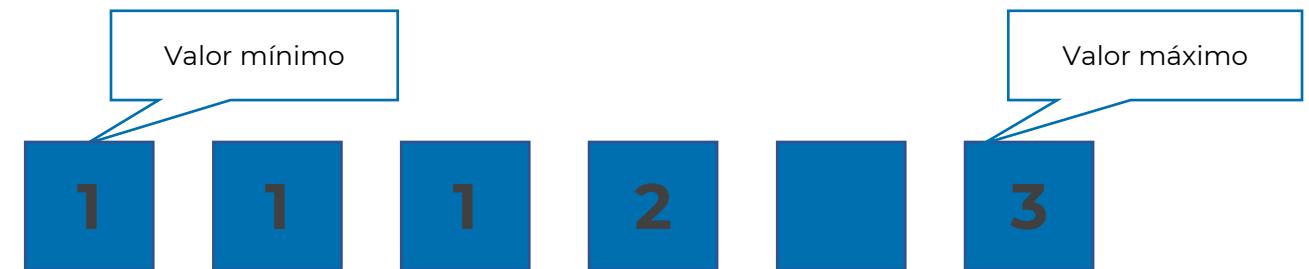
Moda: 1

Registros: 6

**Mínimo: 1**

**Máximo: 3**

Todos são valores inteiros



## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

**Média: 1.67**

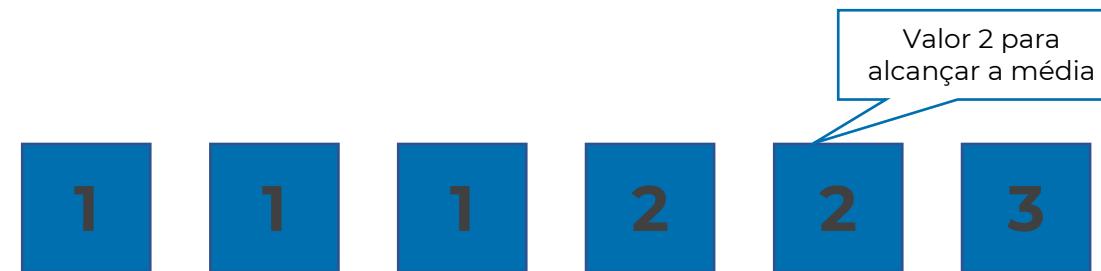
Mediana: 1.5

Moda: 1

Registros: 6

Mínimo: 1

Máximo: 3



Todos são valores inteiros

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

**Média: 1.67**

Mediana: 1.5

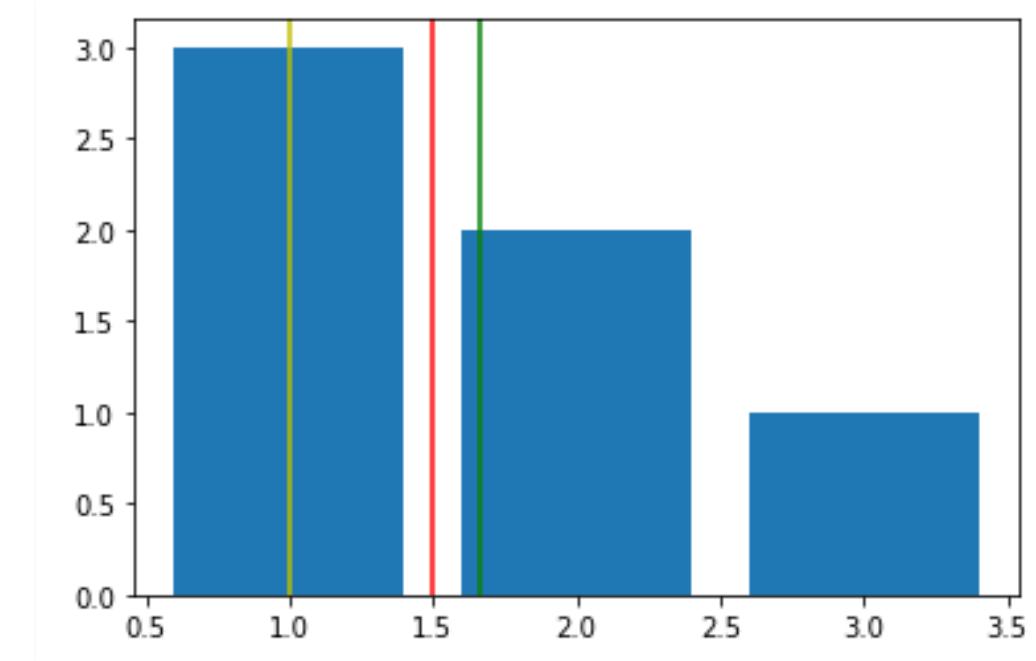
Moda: 1

Registros: 6

Mínimo: 1

Máximo: 3

Todos são valores inteiros



## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

```
base.mean()
```

```
Produto1      6.9  
Produto2      6.7  
Produto3      7.4  
Produto4      9.0  
Produto5      2.8  
Produto6      2.0  
dtype: float64
```

Agora, vamos ver uma base de avaliação de produtos, conforme mostrado ao lado.

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

```
base.mean()
```

```
Produto1      6.9  
Produto2      6.7  
Produto3      7.4  
Produto4      9.0  
Produto5      2.8  
Produto6      2.0  
dtype: float64
```

Você precisa **retirar um produto de linha!**

Com as avaliações do produto mostradas ao lado, **qual você escolheria?**



## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

```
base.mean()
```

```
Produto1      6.9  
Produto2      6.7  
Produto3      7.4  
Produto4      9.0  
Produto5      2.8  
Produto6      2.0  
dtype: float64
```

Apenas com a média, nenhum!

Precisamos de **mais informações** para responder a essa pergunta!

AVALIAÇÕES DA LOJA



4,8



724 avaliações no total

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Com as informações de média, mediana e moda, podemos ver que o produto 6 é ó pior.

Porém, repare que os valores são **exatamente iguais**

	produtos	média	mediana	moda
0	Produto1	6.9	7.0	6.0
1	Produto2	6.7	7.0	9.0
2	Produto3	7.4	7.0	7.0
3	Produto4	9.0	9.0	9.0
4	Produto5	2.8	2.0	2.0
5	Produto6	2.0	2.0	2.0



## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Agora, avaliando o número de registros, mínimo e máximo, podemos observar que o produto 6 só teve **uma avaliação**.

Como a avaliação é mensal, então ele é um **produto novo** e talvez as pessoas não tiveram tempo de conhecer.

	produtos	média	mediana	moda	registros	mínimo	máximo
0	Produto1	6.9	7.0	6.0	10	6.0	8.0
1	Produto2	6.7	7.0	9.0	10	3.0	10.0
2	Produto3	7.4	7.0	7.0	10	5.0	9.0
3	Produto4	9.0	9.0	9.0	1	9.0	9.0
4	Produto5	2.8	2.0	2.0	10	1.0	10.0
5	Produto6	2.0	2.0	2.0	1	2.0	2.0

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

O produto 5 possui 10 registros e uma **média bem abaixo** dos outros produtos.

Então, provavelmente ele seria o escolhido.

	produtos	média	mediana	moda	registros	mínimo	máximo
0	Produto1	6.9	7.0	6.0	10	6.0	8.0
1	Produto2	6.7	7.0	9.0	10	3.0	10.0
2	Produto3	7.4	7.0	7.0	10	5.0	9.0
3	Produto4	9.0	9.0	9.0	1	9.0	9.0
4	Produto5	2.8	2.0	2.0	10	1.0	10.0
5	Produto6	2.0	2.0	2.0	1	2.0	2.0

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Outro exemplo que podemos analisar é a média de salário de duas empresas.

Qual empresa você preferiria?

Se olharmos apenas para a média, escolheríamos a empresa 1. Porém vamos analisar os outros dados.

empresas	média	mediana	moda	registros	mínimo	máximo
0	1 6954.2	2155.0	5000	12	1000	57000
1	2 5983.3	5100.0	4200	12	3000	15000

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Esse valor 57.000 está fazendo **a média ser mais alta** (outlier)!

empresas	média	mediana	moda	registros	mínimo	máximo
0	6954.2	2155.0	5000	12	1000	57000
1	5983.3	5100.0	4200	12	3000	15000

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Se retirarmos esse valor de 57.000, a **média** seria de aproximadamente 2.404,6.

empresas		<b>média</b>	mediana	moda	registros	mínimo	<b>máximo</b>
0	1	6954.2	2155.0	5000	12	1000	57000
1	2	5983.3	5100.0	4200	12	3000	15000

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Na empresa 1, metade das pessoas ganha menos que 2.155 reais, já na empresa 2 a metade ganha até 5.100 reais.

empresas	média	mediana	moda	registros	mínimo	máximo
0	1 6954.2	2155.0	5000	12	1000	57000
1	2 5983.3	5100.0	4200	12	3000	15000

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Na empresa 2, você vai receber **pelo menos** 3 mil reais, já na empresa 1 você pode começar ganhando mil reais.

Então, na empresa 2 metade dessa pessoas ganha entre 3000 e 5100, enquanto na empresa 1 metade ganha entre 1000 e 2155.

empresas	média	mediana	moda	registros	mínimo	máximo
0	1	6954.2	2155.0	5000	12	1000 57000
1	2	5983.3	5100.0	4200	12	3000 15000

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

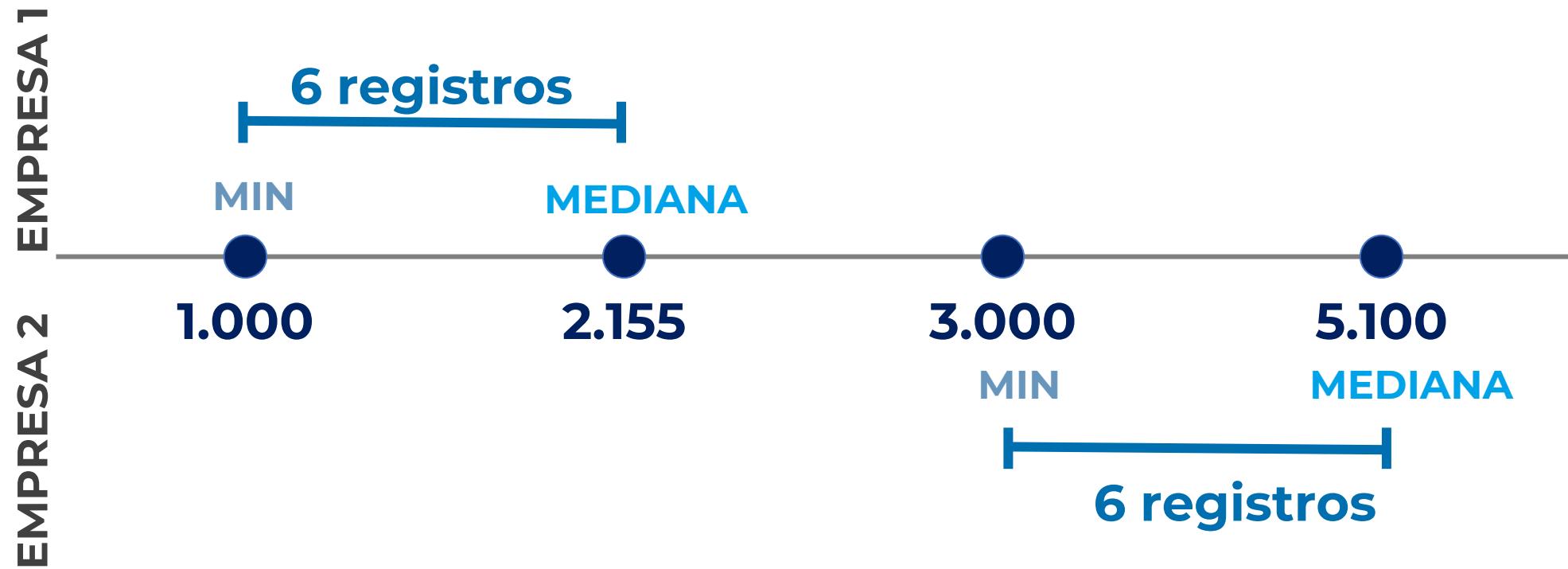
Na empresa 2, o **menor salário é MAIOR que a mediana** da empresa 1.

Ou seja, a pessoa que menos ganha na empresa 2, ganha mais que a metade da empresa 1.

empresas	média	mediana	moda	registros	mínimo	máximo
0	1 6954.2	2155.0	5000	12	1000	57000
1	2 5983.3	5100.0	4200	12	3000	15000

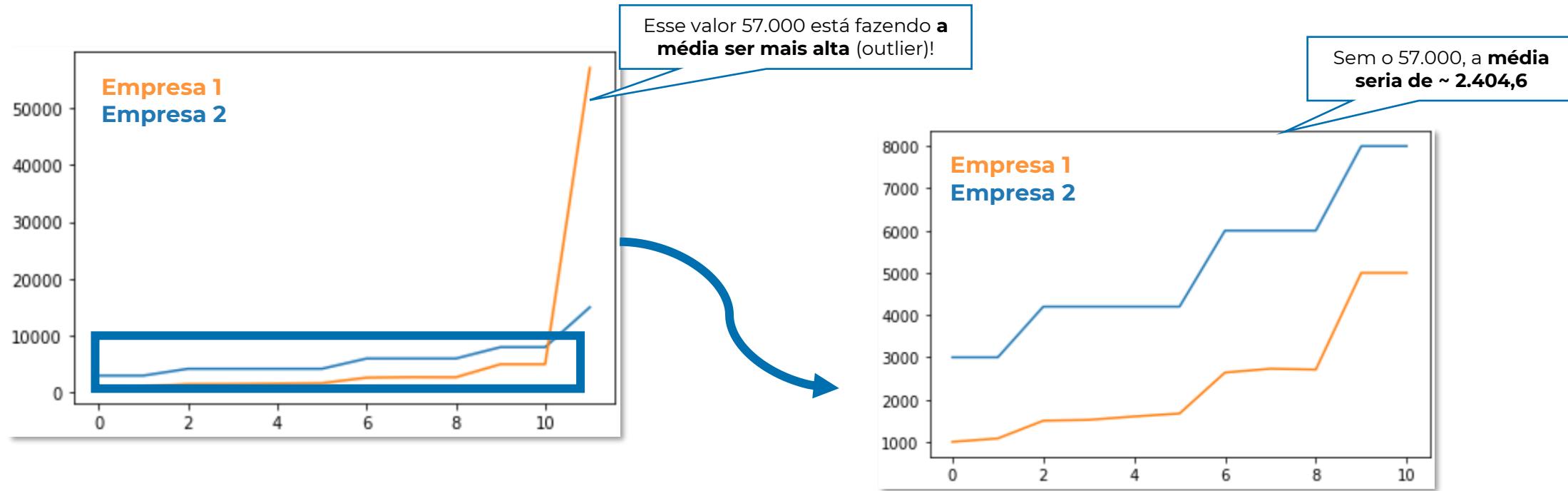
## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Podemos ver de uma forma mais visual e nesse caso já podemos perceber que a **empresa 2 é muito mais vantajosa**.



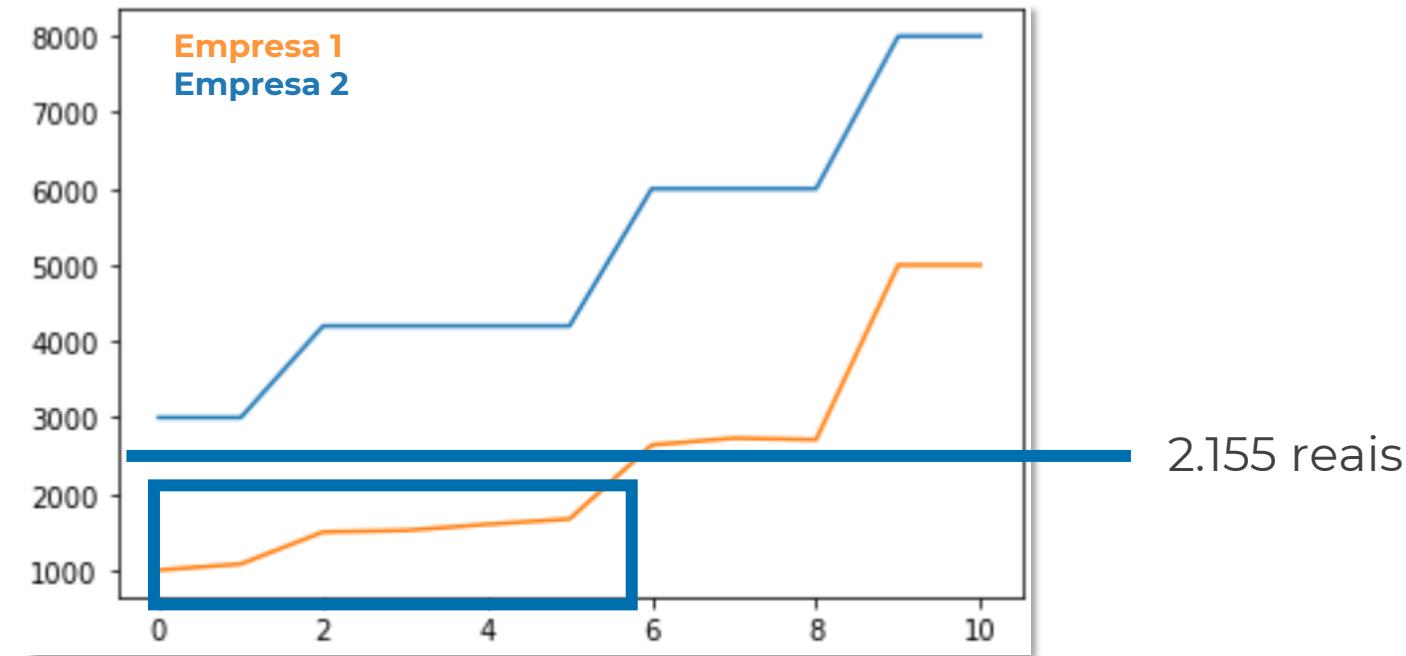
## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Se olharmos para os gráficos de salário, podemos observar que o outlier na empresa 1 está fazendo a média ser mais alta.



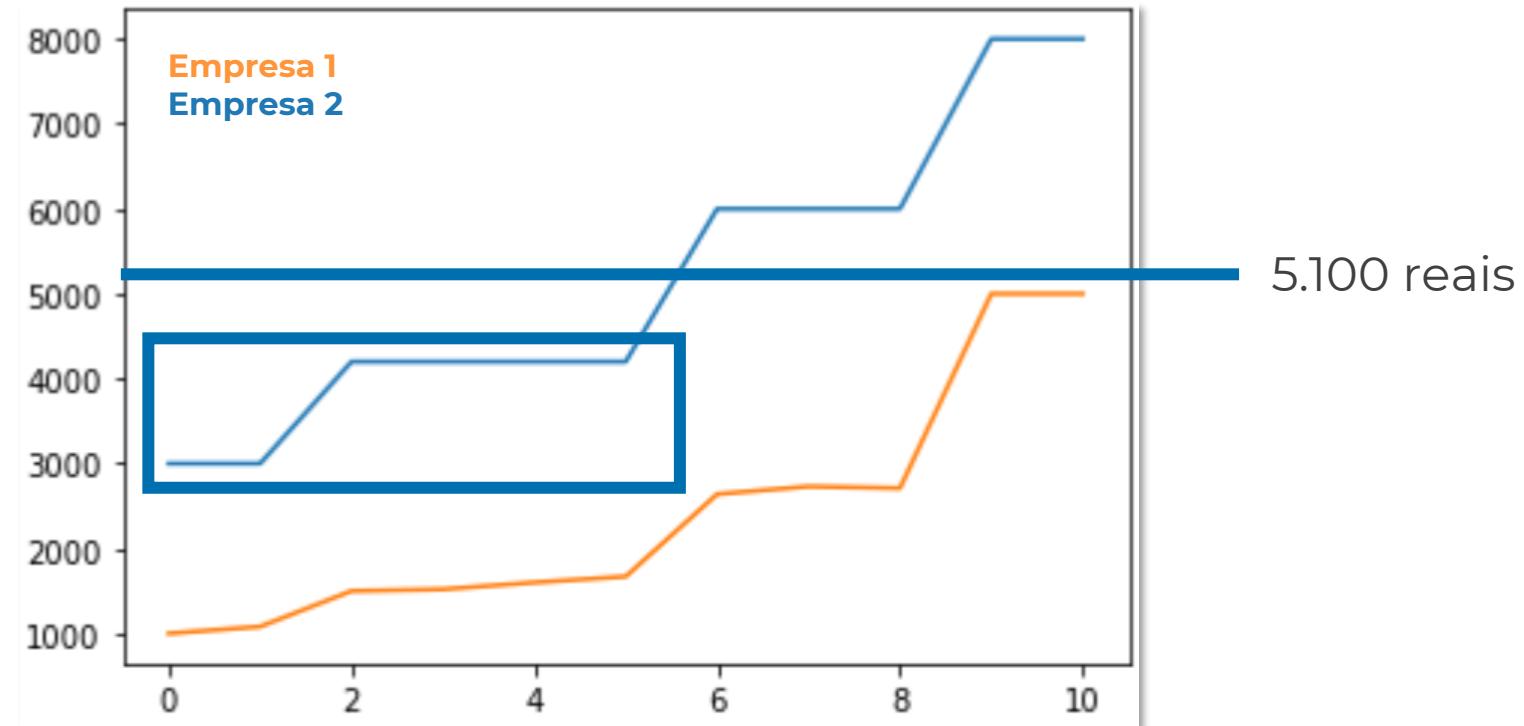
## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Na empresa 1, metade das pessoas ganha menos que 2.155 reais.



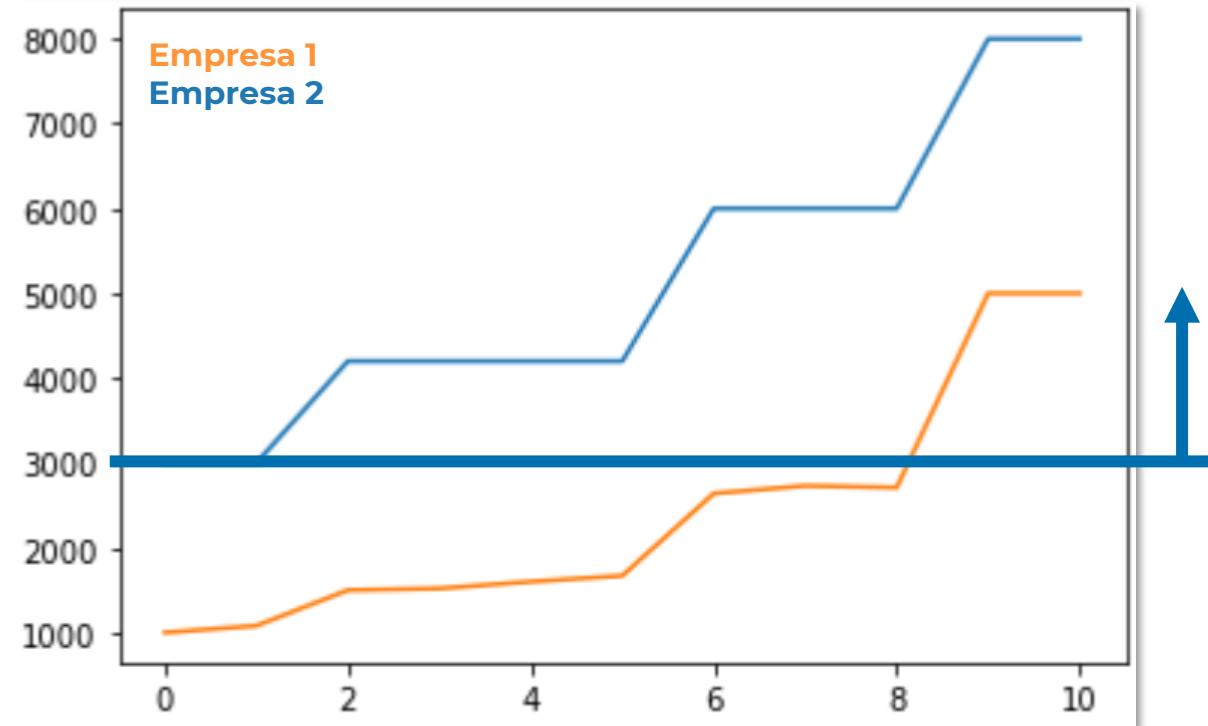
## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Já na empresa 2, a metade ganha até 5.100 reais.



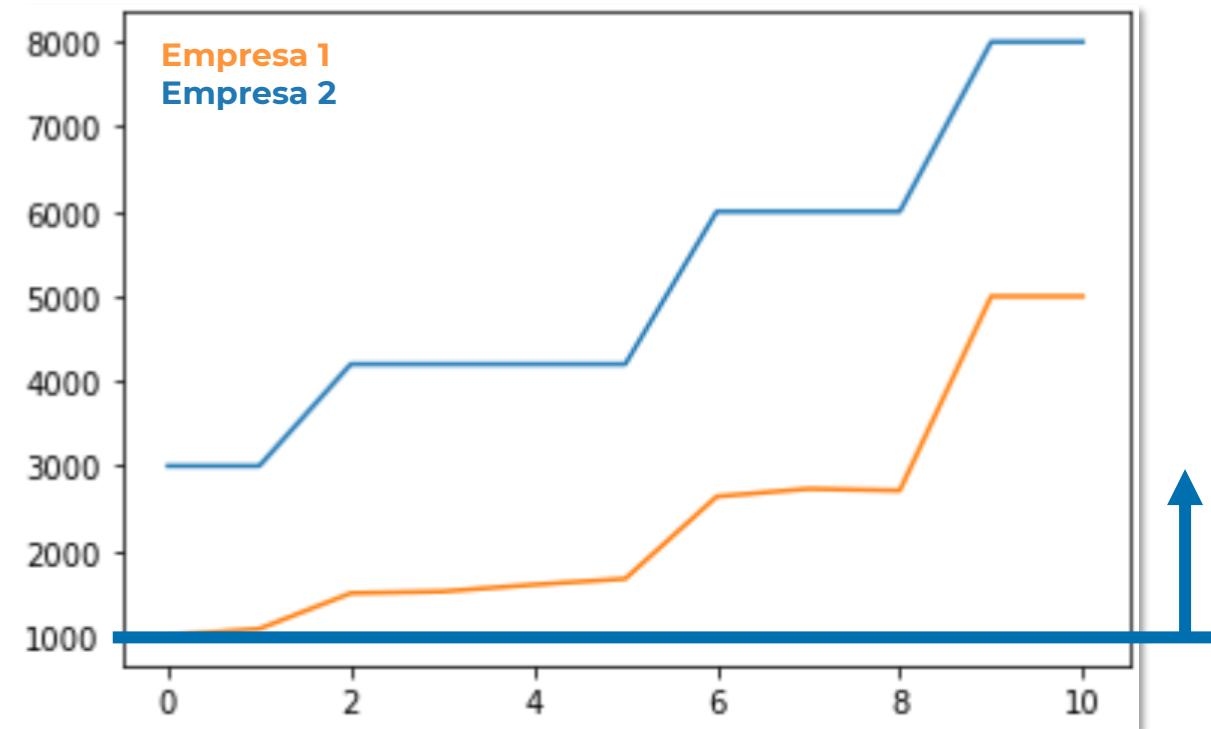
## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Na empresa 2, você vai receber **pelo menos 3 mil reais**.

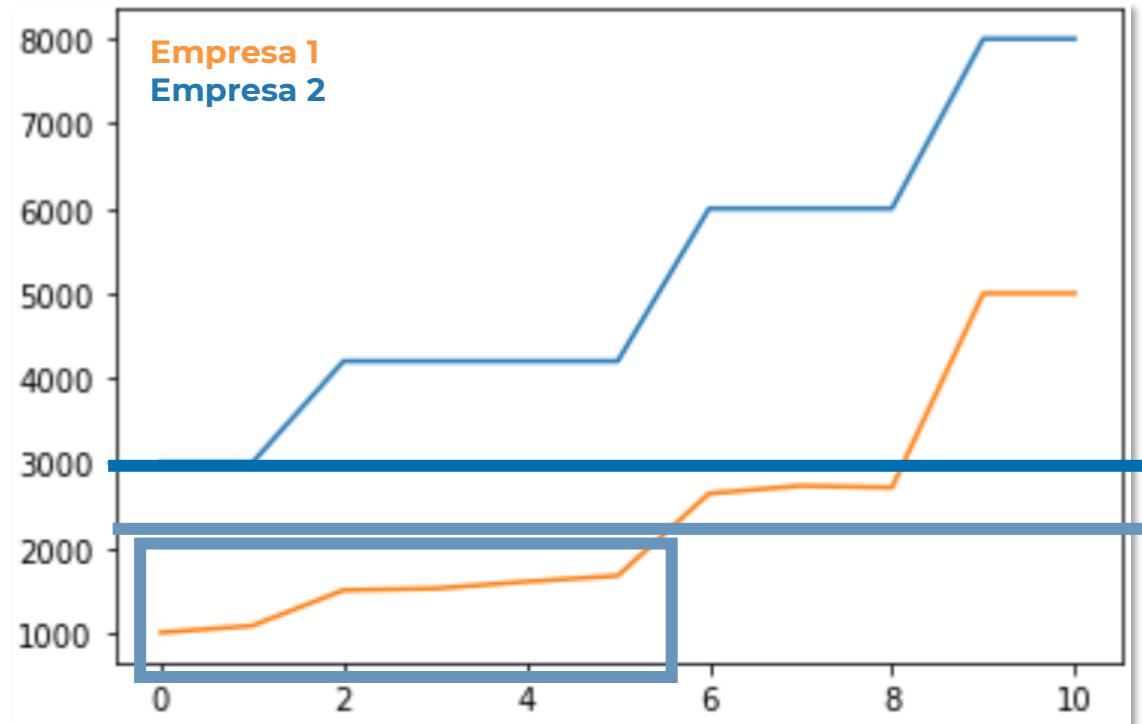


## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Já na empresa 1, você pode começar ganhando **mil reais** e chegar a 9 graduações de cargo para receber mais que 3000 reais.



## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda



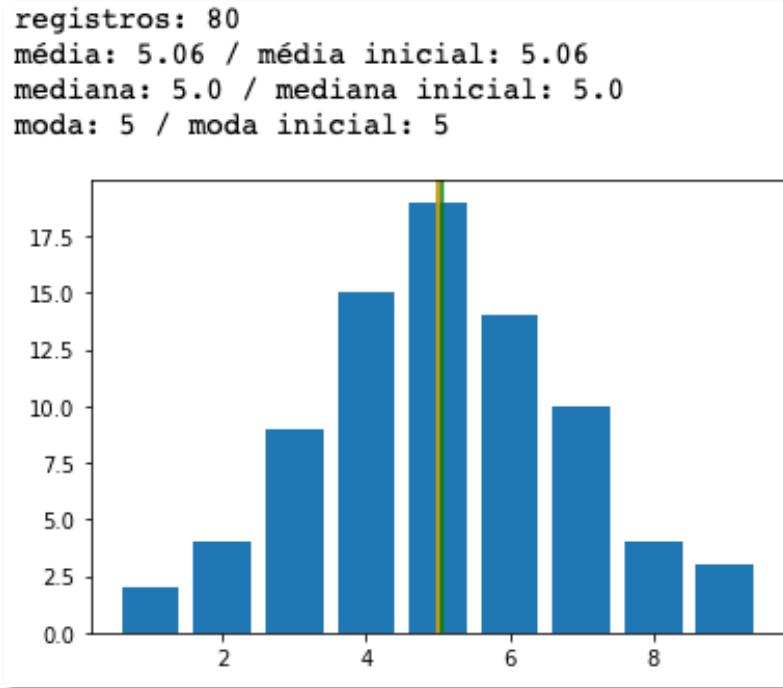
Na empresa 2, o **menor salário é MAIOR que a mediana** da empresa 1.

**MÍNIMO DA EMPRESA 2**

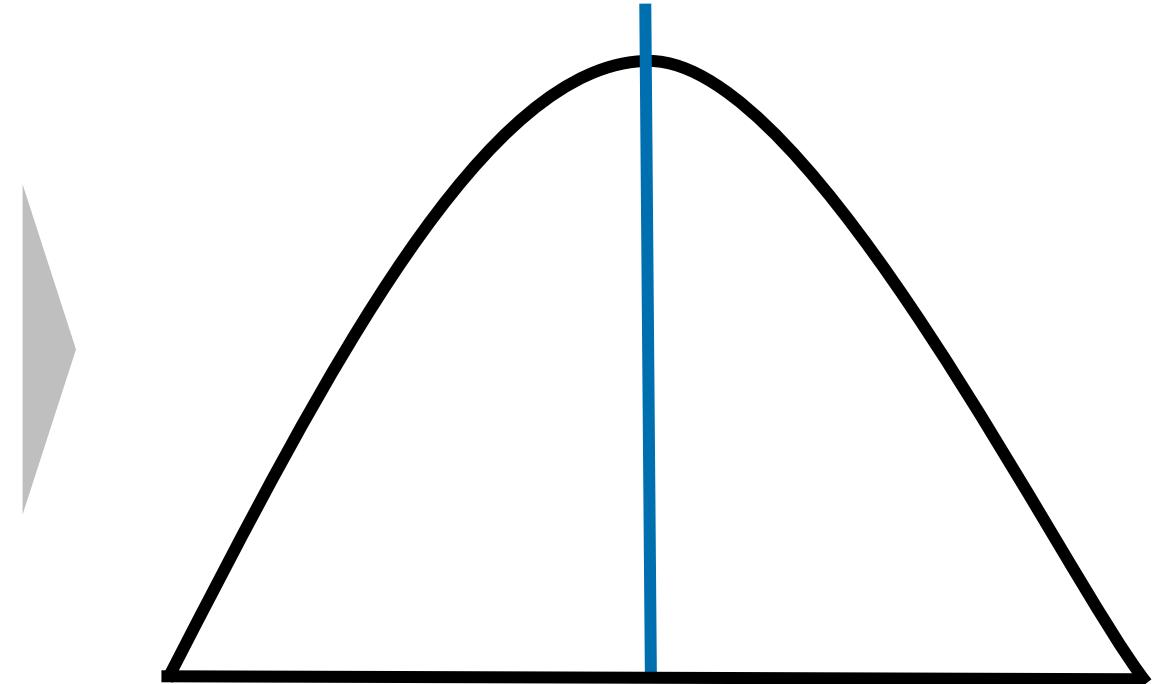
**MEDIANA DA EMPRESA 1**

## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Quando temos muitos dados, vemos que eles estão muito bem distribuídos.



**MÉDIA = MEDIANA = MODA**

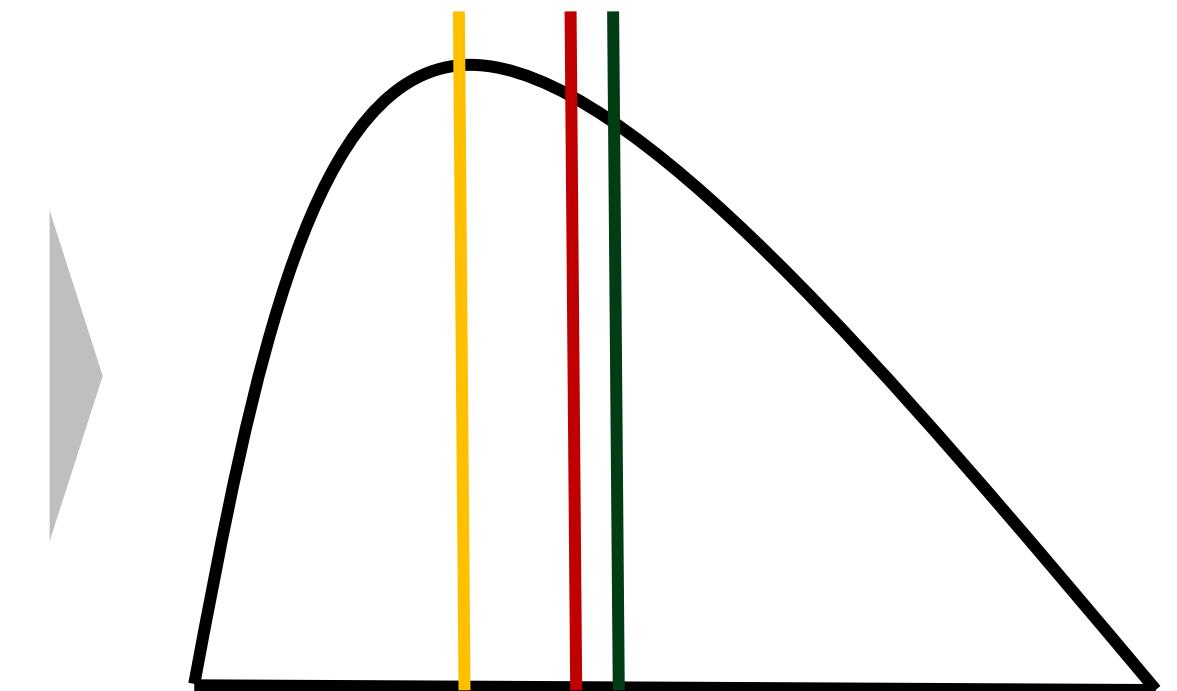
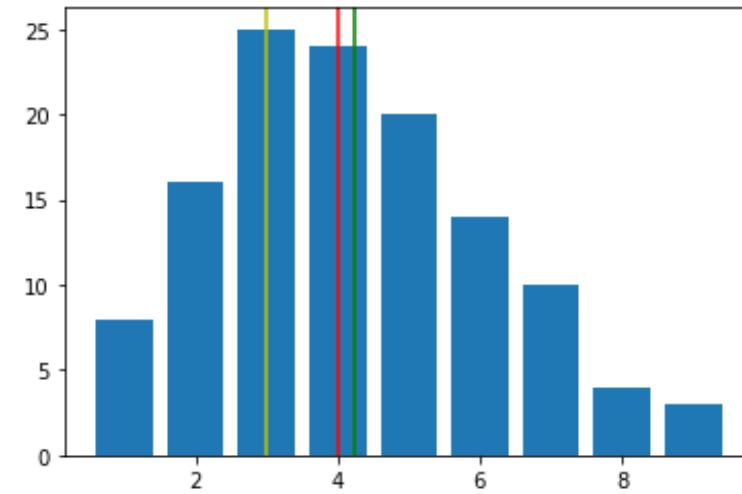


## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Já quando temos média maior que mediana e moda, os dados tendem mais para os valores menores.

**MODA < MEDIANA < MÉDIA**

registros: 124  
média: 4.23 / média inicial: 5.06  
mediana: 4.0 / mediana inicial: 5.0  
moda: 3 / moda inicial: 5

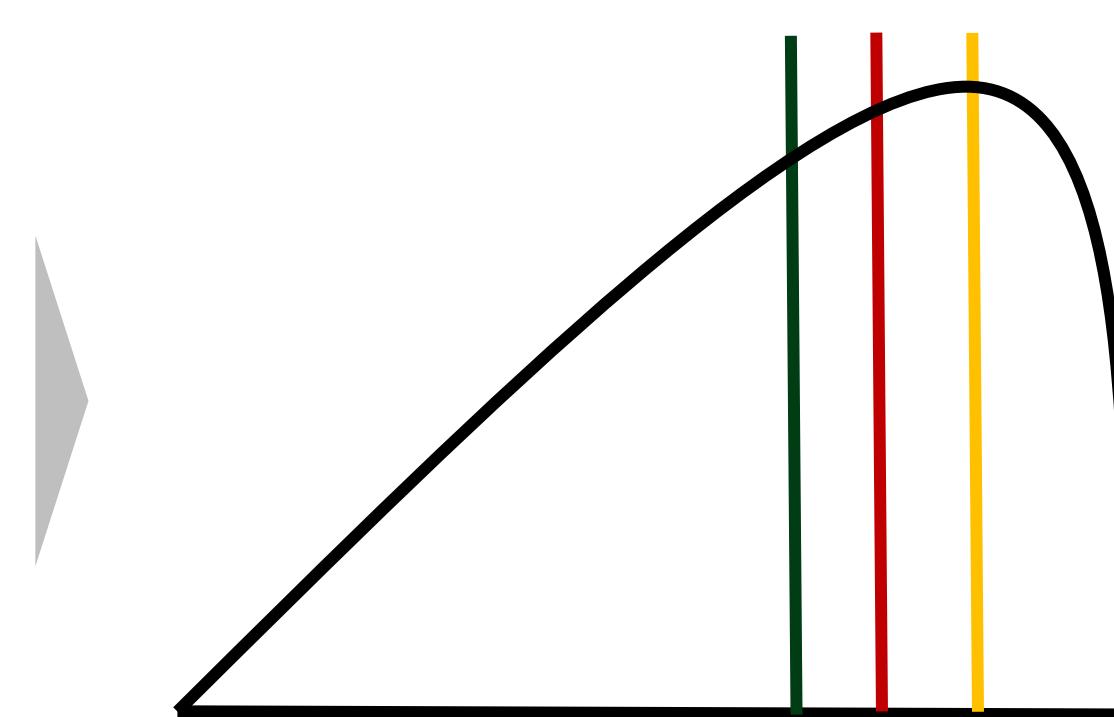
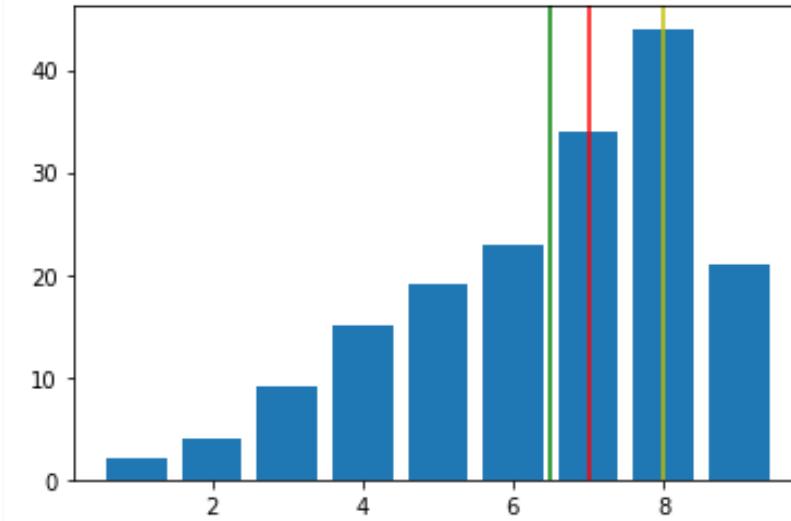


## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Quando temos média menor que mediana e moda, os dados tendem mais para os valores maiores.

**MÉDIA < MEDIANA < MODA**

registros: 171  
média: 6.49 / média inicial: 5.06  
mediana: 7.0 / mediana inicial: 5.0  
moda: 8 / moda inicial: 5

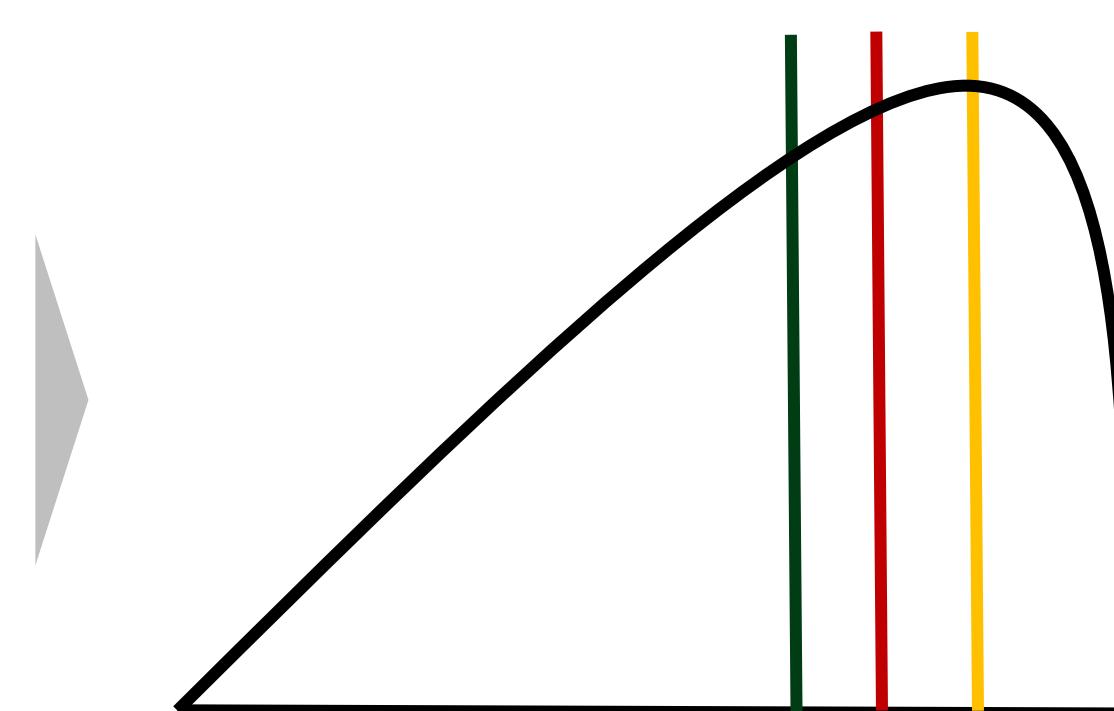
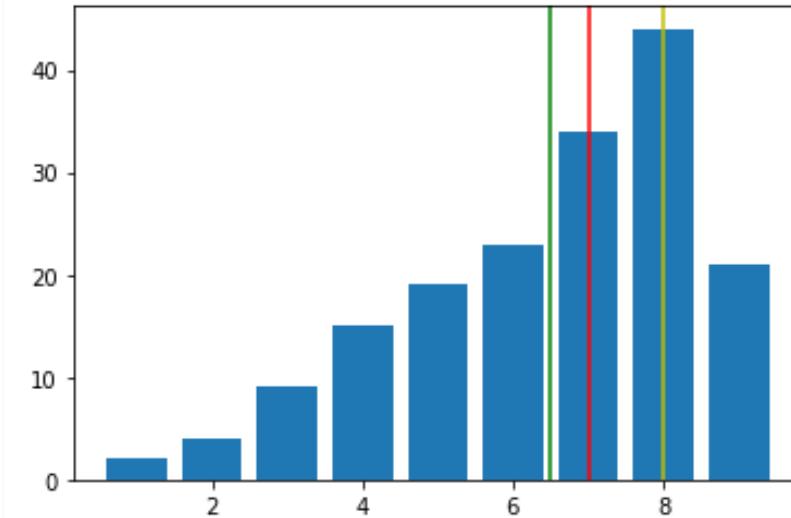


## Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Agora, apenas com a média, mediana e moda você já consegue ter uma boa noção de como está a **distribuição dos seus dados!**

**MÉDIA < MEDIANA < MODA**

```
registros: 171  
média: 6.49 / média inicial: 5.06  
mediana: 7.0 / mediana inicial: 5.0  
moda: 8 / moda inicial: 5
```



## MÓDULO 7

# Matplotlib: Criando gráficos em Python

# Módulo 7 – Apresentando o Matplotlib: Criando gráficos em Python

Nesse módulo vamos falar do Matplotlib, que é uma biblioteca que nos ajuda a criar gráficos no Python.

```
import matplotlib.pyplot as plt  
import numpy as np  
  
# make data  
x = [1,2,3,4] Dados  
y = [10,50,23,100]  
y2 = [100,500,230,1000]  
  
# plot  
fig, ax = plt.subplots()  
  
ax.plot(x, y, linewidth=2.0) Limite de X  
ax.set(xlim=(0,8), xticks=np.arange(1,8),  
       ylim=(0,100),yticks=np.arange(1,8))  
plt.show() Limite de Y Números que vão aparecer no gráfico
```

## Módulo 7 – Apresentando o Matplotlib: Criando gráficos em Python

Se removermos os limites dos eixos o próprio Matplotlib vai procurar a melhor forma de mostrar o gráfico.

Mas antes disso vamos entender o que significa os termos do exemplo ao lado.

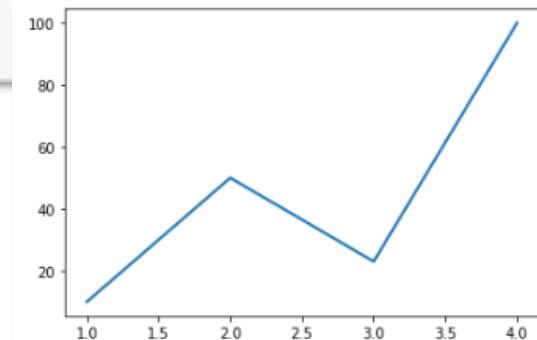
```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = [1,2,3,4]
y = [10,50,23,100]

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

plt.show()
```



## Módulo 7 – Apresentando o Matplotlib: Criando gráficos em Python

A função **plt.subplot** retorna 2 argumentos: fig (figura) e ax (eixo).

Por exemplo: **plt.subplot(nrows=2, ncols = 2)**

Então tudo isso vai aparecer dentro da figura.

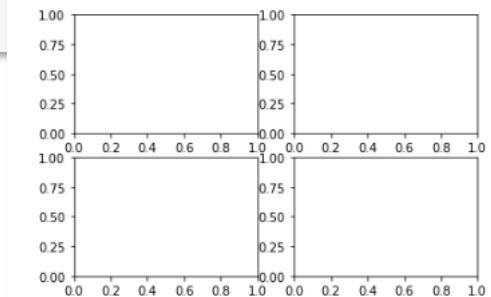
```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = [1,2,3,4]
y = [10,50,23,100]

# plot
fig, ax = plt.subplots(nrows=2, ncols = 2)

ax.plot(x, y, linewidth=2.0)

plt.show()
```



# Módulo 7 – Apresentando o Matplotlib: Criando gráficos em Python

Agora cada quadrante é uma área de plotagem, então podemos fazer:

Para falar do primeiro elemento:

- 1 - Ax[0][0].plot(x, y, linewidth=2.0)
- 2 - Ax[0][1]. plot(x, y, linewidth=2.0)
- 3 - Ax[1][0].plot(x, y, linewidth=2.0)

Dessa forma, é possível criar diferentes gráficos.

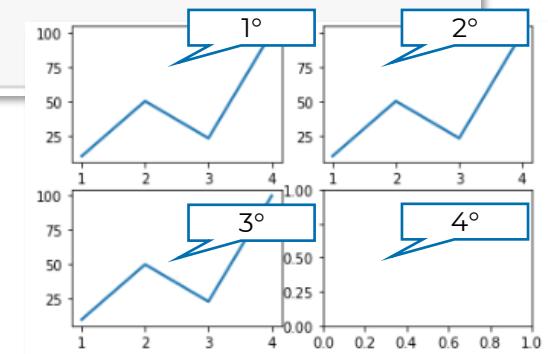
```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = [1,2,3,4]
y = [10,50,23,100]

# plot
fig, ax = plt.subplots(nrows=2, ncols = 2)

ax[0][0].plot(x, y, linewidth=2.0)
ax[0][1].plot(x, y, linewidth=2.0)
ax[1][0].plot(x, y, linewidth=2.0)

plt.show()
```



## Módulo 7 – Apresentando o Matplotlib: Criando gráficos em Python

Além disso, é possível criar diferentes gráficos e colocar 2 dados em um mesmo gráfico.

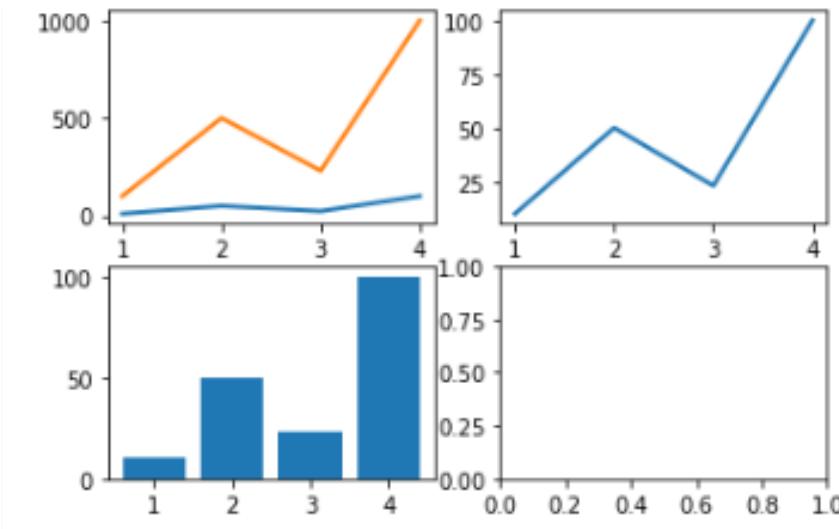
```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = [1,2,3,4]
y = [10,50,23,100]
y2 = [100,500,230,1000]

# plot
fig, ax = plt.subplots(nrows=2, ncols = 2)

ax[0][0].plot(x, y, linewidth=2.0)
ax[0][0].plot(x, y2, linewidth=2.0)
ax[0][1].plot(x, y, linewidth=2.0)
ax[1][0].bar(x, y, linewidth=2.0)

plt.show()
```



# Módulo 7 – Apresentando o Matplotlib: Criando gráficos em Python

Nós podemos mexer nos parâmetros também. Por exemplo, mudar a cor do eixo x para vermelho.

**Ax.tick\_params(axis='x', labelcolor='r')**

Se colocarmos apenas a função color estamos falando dos traços que acompanham o número.

**Ax.tick\_params(axis='x', color='r')**

E para aumentar o tamanho do rótulo utilizamos a função **label**.

**Ax.tick\_params(axis='x', labelcolor='r', labelsize=12)**

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

# make data
x = [1,2,3,4]
y = [10,50,23,100]
y2 = [100,500,230,1000]

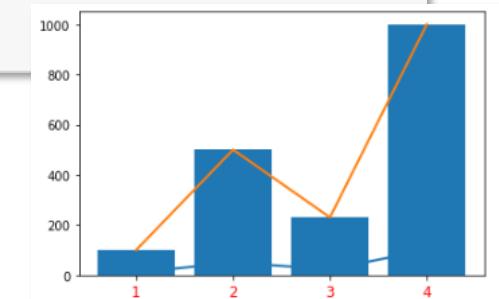
# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)
ax.plot(x, y2, linewidth=2.0)
ax.bar(x, y2, linewidth=2.0)

ax.set(xticks=np.arange(1, 5))

ax.tick_params(axis='x', labelcolor='r', labelsize=12)

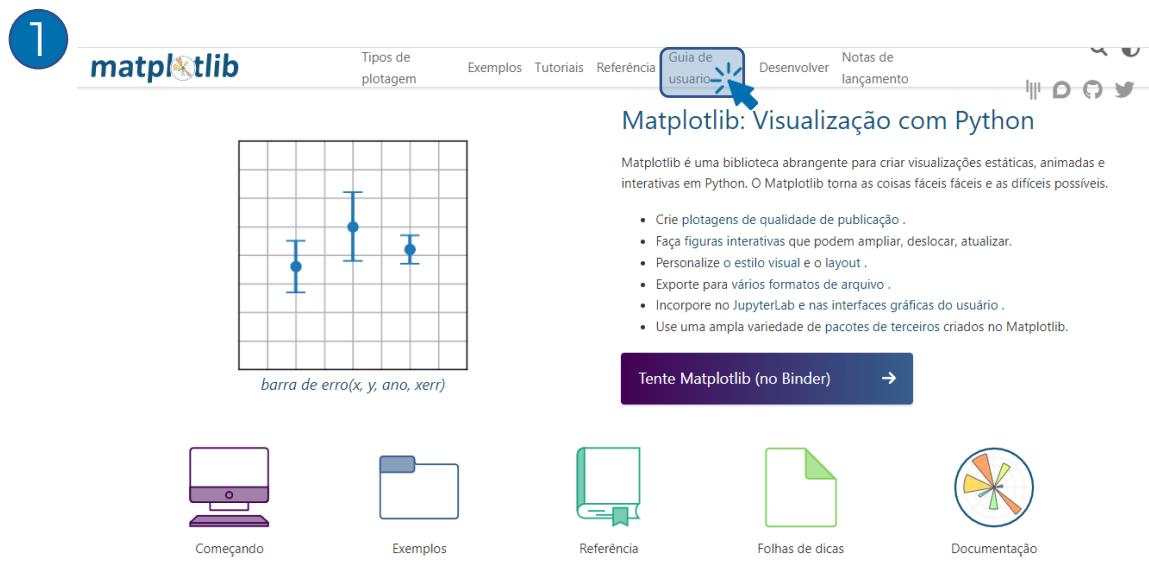
plt.show()
```



# Módulo 7 – Introdução ao Matplotlib

Existem várias bibliotecas no Python que nos ajudam a fazer gráficos, mas a Matplotlib é a mais utilizada.

A documentação do Matplotlib está disponível no [link](#).



2

The screenshot shows the 'Users guide' section of the Matplotlib documentation. The left sidebar has a 'General' category with 'Getting started' (highlighted with a blue box and a cursor arrow) and other sections like 'Installation quick-start', 'Draw a first plot', 'Where to go next', 'Installation', 'Explanations', and 'Backends'. The main content area is titled 'Getting started' and contains two code snippets: 'Install using pip:' with the command 'pip install matplotlib' and 'Install using conda:' with the command 'conda install matplotlib'.

# Módulo 7 – Introdução ao Matplotlib

Se você estiver utilizando o Jupyter Notebook ele já está instalado, caso contrário é necessário fazer o pip install matplotlib.

The screenshot shows the 'Getting started' section of the Matplotlib documentation. It includes links for 'Install using pip:' (with the command `pip install matplotlib`) and 'Install using conda:' (with the command `conda install matplotlib`). Below these, a link leads to the 'Installation Guide'. The 'Draw a first plot' section shows a minimal example plot with the following Python code:

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

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

# Módulo 7 – Introdução ao Matplotlib

Para fazer o primeiro plot utilizamos o código abaixo:

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

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

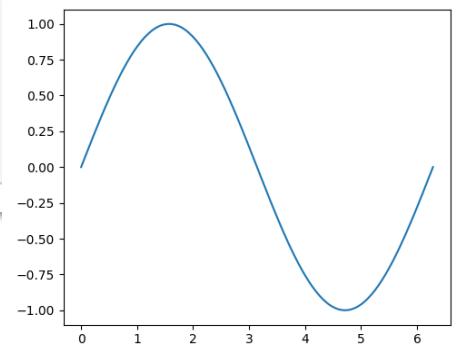
## Draw a first plot

Here is a minimal example plot:

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

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```



## Módulo 7 – Introdução ao Matplotlib

O Matplotlib **permite desde a criação de visualizações** estáticas mais simples até os gráficos mais complexos e elaborados.

Como diz a própria documentação, ele torna o que é fácil, fácil e as coisas difíceis, possíveis de serem feitas!

Ela é bem **completa** e também é útil para **analytics**.

Não possui tanta customização quanto outras bibliotecas e **pode deixar a desejar no apelo estético em alguns casos**.

## Módulo 7 – Introdução ao Matplotlib

O **Pyplot** é o módulo do Matplotlib que possui os **gráficos básicos** normalmente usados nessa biblioteca.

Vamos usar bastante o **pyplot** e, por isso, vamos importar diretamente esse módulo.

## Módulo 7 – Introdução ao Matplotlib

Então vamos copiar exatamente o código disponível na documentação:

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

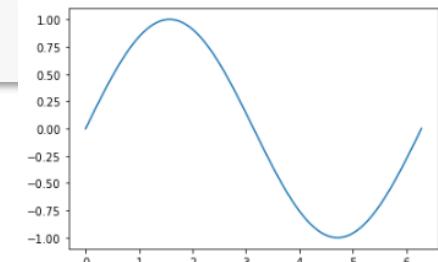
x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

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

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```



## Módulo 7 – Introdução ao Matplotlib

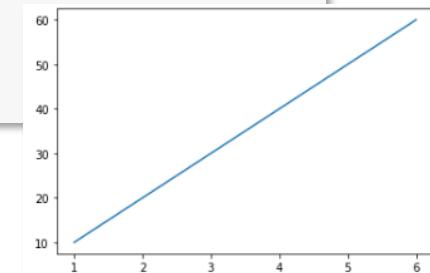
Agora, vamos supor que nós não queremos utilizar o numpy, então podemos alterar os eixos para:

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

```
x =[1,2,3,4,5,6]  
y =[10,20,30,40,50,60]
```

```
fig, ax = plt.subplots()  
ax.plot(x, y)  
plt.show()
```

```
import matplotlib.pyplot as plt  
  
x =[1,2,3,4,5,6]  
y =[10,20,30,40,50,60]  
  
fig, ax = plt.subplots()  
ax.plot(x, y)  
plt.show()
```



## Módulo 7 – Introdução ao Matplotlib

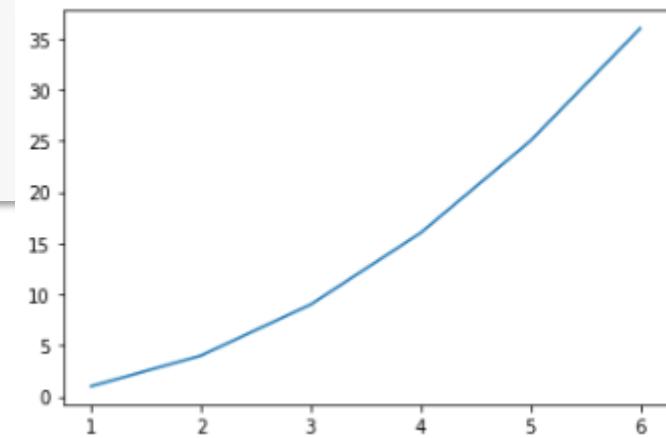
Agora, vamos supor que temos uma função quadrática:

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

```
x = [1,2,3,4,5,6]  
y = [1,4,9,16,25,36]
```

```
fig, ax = plt.subplots()  
ax.plot(x, y)  
plt.show()
```

```
# Criando nosso primeiro plot  
# Disponível em: https://matplotlib.org/stable/users/getting\_started/  
import matplotlib.pyplot as plt  
  
x = [1,2,3,4,5,6]  
y = [1,4,9,16,25,36]  
  
fig, ax = plt.subplots()  
ax.plot(x, y)  
plt.show()
```



## Módulo 7 – Introdução ao Matplotlib

Para esse módulo vamos utilizar a nossa base do Projeto do Instagram.

Para isso vamos importar as bibliotecas utilizando a estrutura abaixo:

### Import biblioteca as apelido

```
# Importando o pyplot
import matplotlib.pyplot as plt

# Importando as outras bibliotecas
import numpy as np
import pandas as pd
import datetime as dt
```

## Módulo 7 – Introdução ao Matplotlib

Vamos importar e tratar a nossa base em Excel.

Para importar a base utilizamos:

**Base = pd.read\_tipodearquivo('Nome do arquivo'.tipodoarquivo)**

```
# Importar a base em excel  
base = pd.read_excel("08. Analisando o engajamento no Instagram.xlsx")  
# Apagando a coluna "Visualizações"  
base = base.drop("Visualizações",axis=1)  
# Agora vamos atribuir o valor N para essa coluna  
base.loc[base.Carrossel.isnull(),"Carrossel"] = "N"
```

Nome da base

Tipo de arquivo

## Módulo 7 – Introdução ao Matplotlib

Para apagar uma coluna utilizamos:

```
base = base.drop('Coluna',axis=1)
```

E para atribuir um valor N para essa coluna utilizamos a estrutura abaixo:

```
base.loc[base.Coluna.isnull(),"Coluna"] = "N"
```

```
# Apagando a coluna "Visualizações"  
base = base.drop("Visualizações",axis=1)  
  
# Agora vamos atribuir o valor N para essa coluna  
base.loc[base.Carrossel.isnull(),"Carrossel"] = "N"
```

# Módulo 7 – Introdução ao Matplotlib

base.head()									
	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437

Para visualizarmos as primeiras 5 linhas utilizamos o comando **head**.

**Base.head()**

## Módulo 7 – Usando a documentação para criar nosso primeiro gráfico (gráfico de linha)

Agora que já temos a nossa base importada vamos começar a fazer os plots.

Vamos começar com um exemplo da documentação:

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

# make data
x = np.linspace(0, 10, 100)
y = 4 + 2 * np.sin(2 * x)

# plot
fig, ax = plt.subplots()
ax.plot(x, y, 'r--', linewidth=5.0)
ax.set(xlim=(0,8),xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```

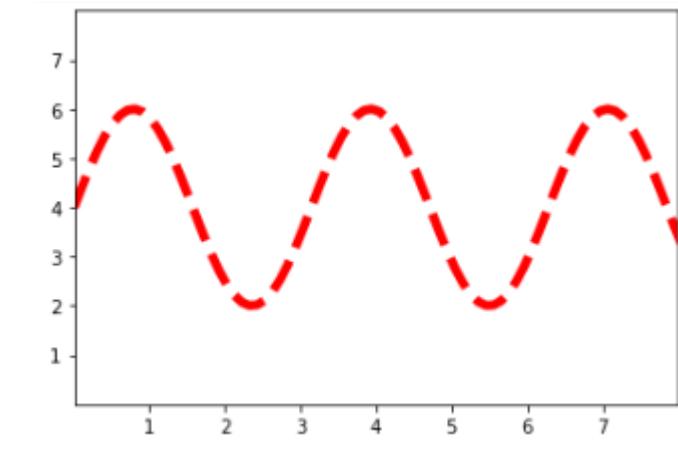
Importando as bibliotecas

Criando dados

Tipo de gráfico

Limites dos eixos

Comando para mostrar o gráfico



## Módulo 7 – Usando a documentação para criar nosso primeiro gráfico (gráfico de linha)

Agora vamos aplicar esse mesmo gráfico para a nossa base.

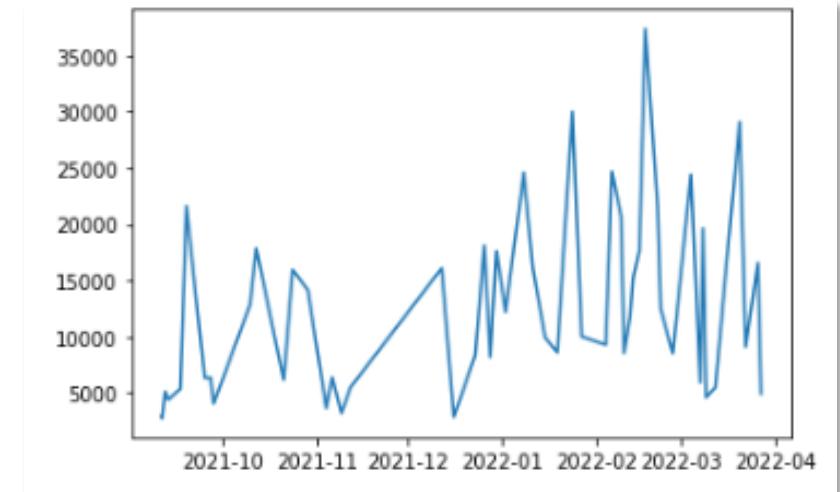
```
x = base.Data
```

```
y = base.Curtidas
```

```
fig, ax = plt.subplots()
```

```
ax.plot(x, y)
```

```
plt.show()
```



## Módulo 7 – Usando a documentação para criar nosso primeiro gráfico (gráfico de linha)

E se quiséssemos mostrar esse mesmo gráfico só que no modelo de dispersão? O que mudaria?

Trocariamos apenas o **ax.plot** por **ax.scatter**:

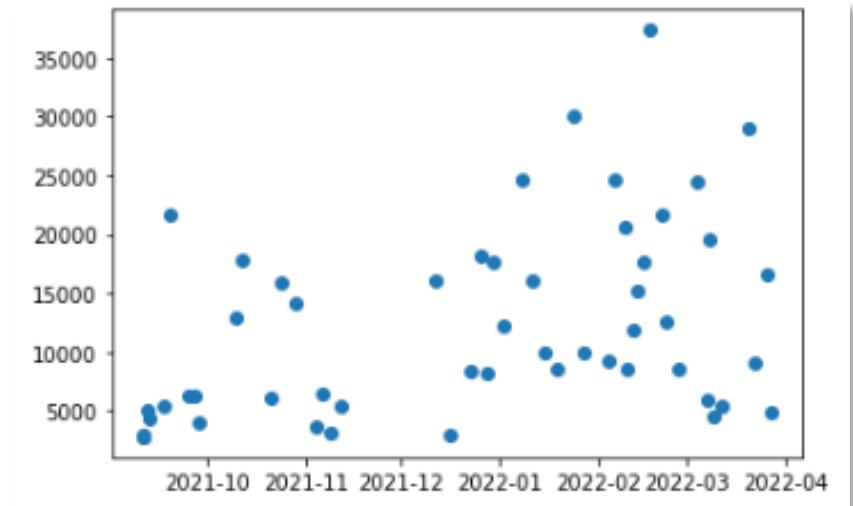
**x = base.Data**

**y = base.Curtidas**

**fig, ax = plt.subplots()**

**ax.scatter(x, y)**

**plt.show()**



### Atenção!

Confira na documentação os **argumentos** de cada gráfico!

## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Agora vamos entender mais um pouco sobre como utilizar a documentação do Matplotlib.

Caso você tenha dificuldade em inglês, existe a opção de traduzir a página.

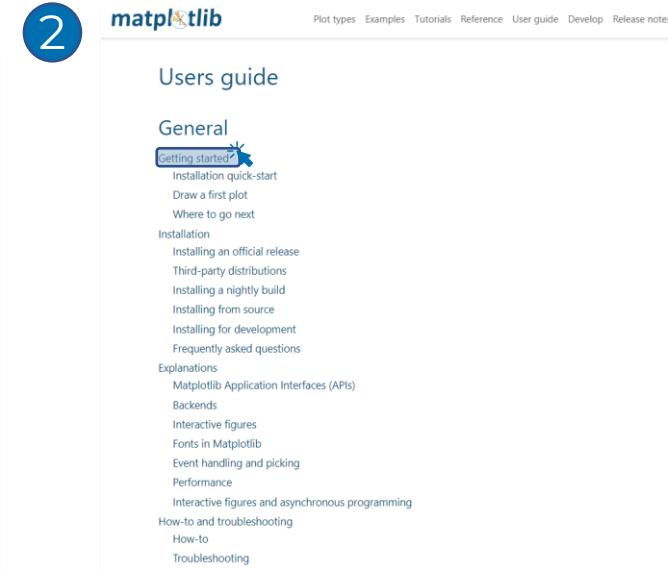
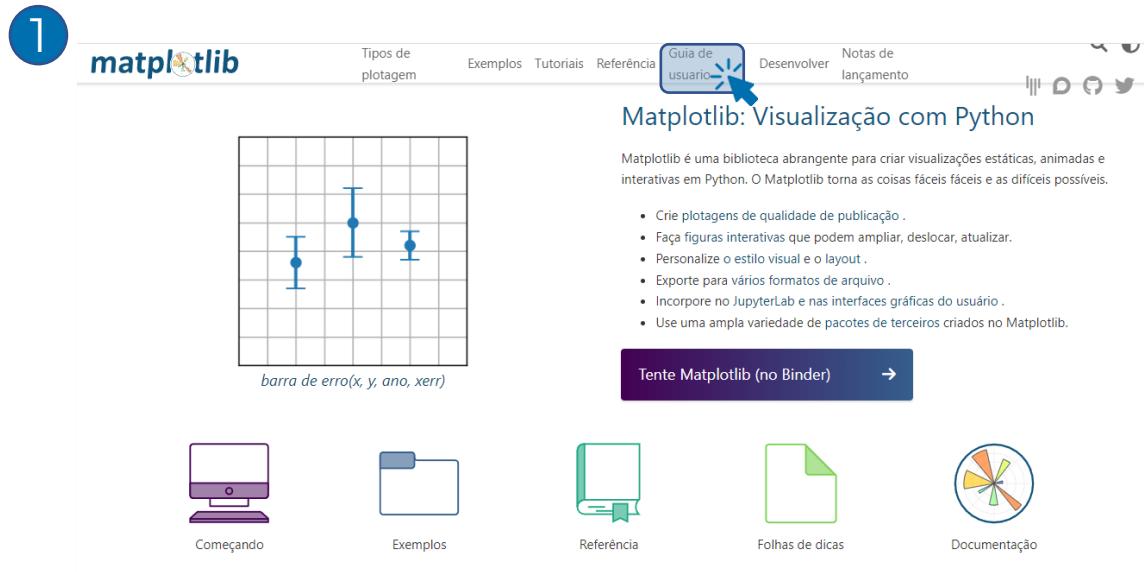
- 1) Clique com o botão direito.
- 2) Clique em traduzir para o português.



# Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Como podemos começar a utilizar a biblioteca do Matplotlib?

Clicamos em ‘Guia do usuário’ e em seguida: General > Getting started.



## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Se você estiver utilizando o Anaconda ele já está instalado, caso contrário é necessário fazer o pip install matplotlib.

### Início rápido da instalação #

Instale usando o pip :

```
pip install matplotlib
```

Instale usando conda :

```
conda install matplotlib
```



## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Então vamos entender linha a linha o primeiro gráfico que copiamos da documentação:

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

Importando as bibliotecas

```
x = np.linspace(0, 2 * np.pi, 200)  
y = np.sin(x)
```

Criando os dados

```
fig, ax = plt.subplots()  
ax.plot(x, y)  
plt.show()
```

Fazendo o plot

Desenhe um primeiro gráfico #

Aqui está um gráfico de exemplo mínimo:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.linspace(0, 2 * np.pi, 200)  
y = np.sin(x)  
  
fig, ax = plt.subplots()  
ax.plot(x, y)  
plt.show()
```

## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Vamos substituir os nossos dados por números para facilitar:

```
import matplotlib.pyplot as plt
```

```
x = [1,2,3,4,5]
```

```
y = [1,4,17,50,92]
```

Nossos dados

```
fig, ax = plt.subplots()  
ax.plot(x, y)  
plt.show()
```

Desenhe um primeiro gráfico #

Aqui está um gráfico de exemplo mínimo:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.linspace(0, 2 * np.pi, 200)  
y = np.sin(x)  
  
fig, ax = plt.subplots()  
ax.plot(x, y)  
plt.show()
```



## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Agora vamos entender a parte do gráfico.

O comando abaixo cria uma figura, então com ele nós criamos 2 imagens diferentes dentro da nossa figura.

```
import matplotlib.pyplot as plt
```

```
x = [1,2,3,4,5]
```

```
y = [1,4,17,50,92]
```

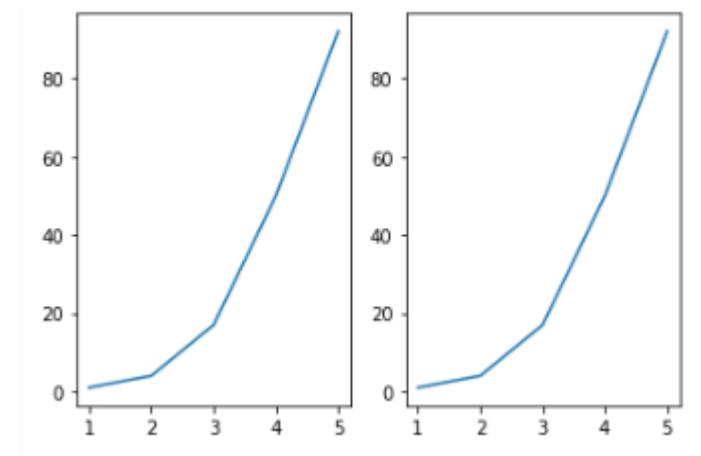
```
fig, ax = plt.subplots(nrows=1, ncols=2)
```

```
ax[0].plot(x, y)
```

```
ax[1].plot(x, y)
```

Cada um dos gráficos que  
estamos criando

```
plt.show()
```



## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Podemos colocar mais de um gráfico em uma coluna, por exemplo:

```
import matplotlib.pyplot as plt
```

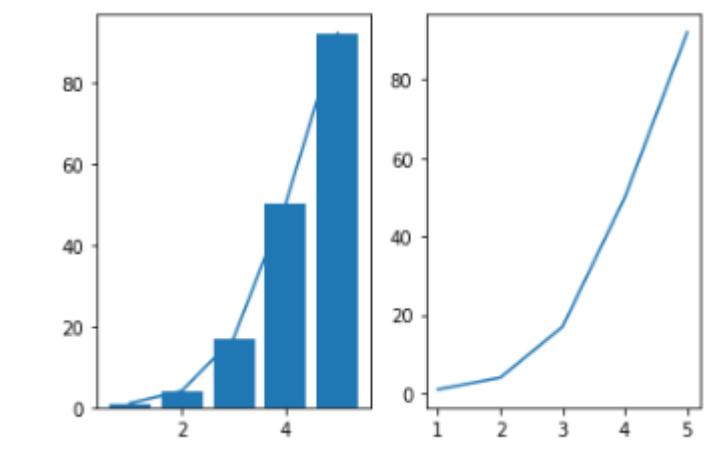
```
x = [1,2,3,4,5]  
y = [1,4,17,50,92]
```

Cada um dos gráficos que  
estamos criando

```
fig, ax = plt.subplots(nrows=1, ncols=2)
```

```
ax[0].plot(x, y)  
ax[0].bar(x, y)  
ax[1].plot(x, y)
```

```
plt.show()
```



## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Agora vamos olhar os tipos de gráficos clicando em **Tipos de plotagem**.

### Para onde ir a seguir

- Confira [Tipos de plotagem](#) para obter uma visão geral dos tipos de plotagens que você pode criar com o Matplotlib.
- Aprenda Matplotlib desde o início no guia de início rápido .



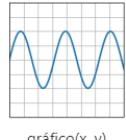
# Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Agora vamos clicar no gráfico de linha para entendermos como ele funciona.

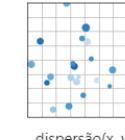
1

Básico #

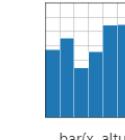
Tipos básicos de gráficos, geralmente y versus x.



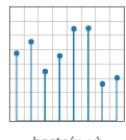
gráfico(x, y)



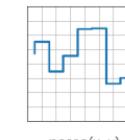
dispersão(x, y)



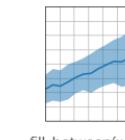
bar(x, altura)



haste(x, y)



passo(x, y)



fill\_between(x, y1, y2)

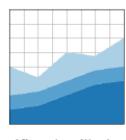
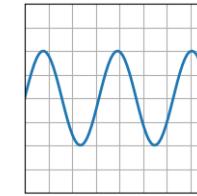


gráfico de pilha(x, y)

2

plot(x, y)

Veja [plot](#).



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

plt.style.use('_mpl-gallery')

# make data
x = np.linspace(0, 10, 100)
y = 4 + 2 * np.sin(2 * x)

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Copiando o código da documentação temos:

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

```
x = np.linspace(0, 10, 100)
```

Definindo os valores de X

```
y = 4 + 2 * np.sin(2 * x)
```

Definindo os valores de Y

```
fig, ax = plt.subplots()
```

Espessura da linha

```
ax.plot(x, y, linewidth=2.0)
```

```
ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
```

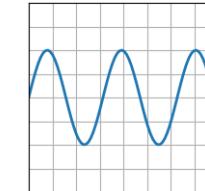
Limites dos eixos

```
    ylim=(0, 8), yticks=np.arange(1, 8))
```

```
plt.show()
```

plot(x, y)

Veja [plot](#).



```
import matplotlib.pyplot as plt  
import numpy as np  
  
plt.style.use('_mpl-gallery')  
  
# make data  
x = np.linspace(0, 10, 100)  
y = 4 + 2 * np.sin(2 * x)  
  
# plot  
fig, ax = plt.subplots()  
  
ax.plot(x, y, linewidth=2.0)  
  
ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  
       ylim=(0, 8), yticks=np.arange(1, 8))  
  
plt.show()
```

## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Para aprender mais sobre um argumento, podemos pesquisar na documentação



## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

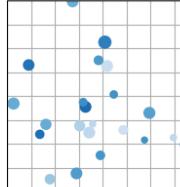
Podemos aplicar isso para outros tipos de gráficos também e vamos notar que o código para o gráfico fica parecido.

Navegação da Seção

- básico
  - gráfico(x, y)
    - dispersão(x, y)
  - bar(x, altura)
  - haste(x, y)
  - passo(x, y)
  - fill\_between(x, y1, y2)
  - gráfico de pilha(x, y)
- Gráficos de arrays e campos
- Gráficos estatísticos
- Coordenadas não estruturadas
- 3D

scatter(x, y) #

Veja [scatter](#).



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

plt.style.use('_mpl-gallery')

# make the data
np.random.seed(3)
x = 4 + np.random.normal(0, 2, 24)
y = 4 + np.random.normal(0, 2, len(x))
# size and color
sizes = np.random.uniform(15, 80, len(x))
colors = np.random.uniform(15, 80, len(x))

# plot
fig, ax = plt.subplots()

ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Agora vamos voltar para a aba *Getting started* e vamos clicar em Guia do início rápido.

### Para onde ir a seguir

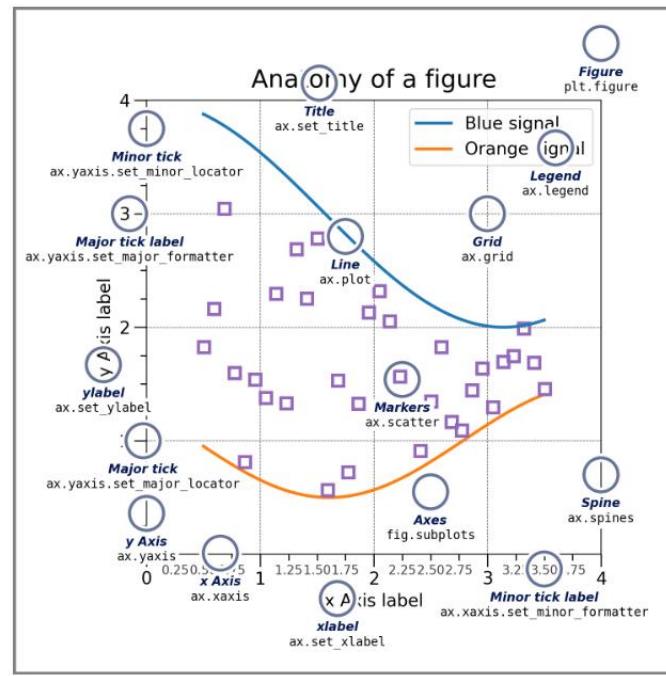
- Confira Tipos de plotagem para obter uma visão geral dos tipos de plotagens que você pode criar com o Matplotlib.
- Aprenda Matplotlib desde o início no [guia de início rápido](#).



## Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Nessa opção, além de ter a forma e um exemplo simples, temos a parte da figura.

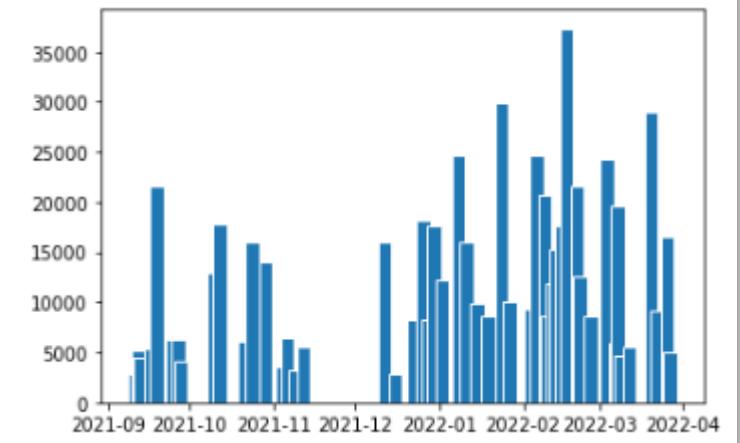
A parte da figura mostra como mexer na linha, bordas, etc.



## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Os gráficos vão nos ajudar a entender se existe uma relação entre os dados, comportamentos que podem prever o futuro.

A parte da figura mostra como mexer na linha, bordas, etc.



## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Agora repare que ao trocar o gráfico de barra para um de linha ocorre um erro no nosso código.

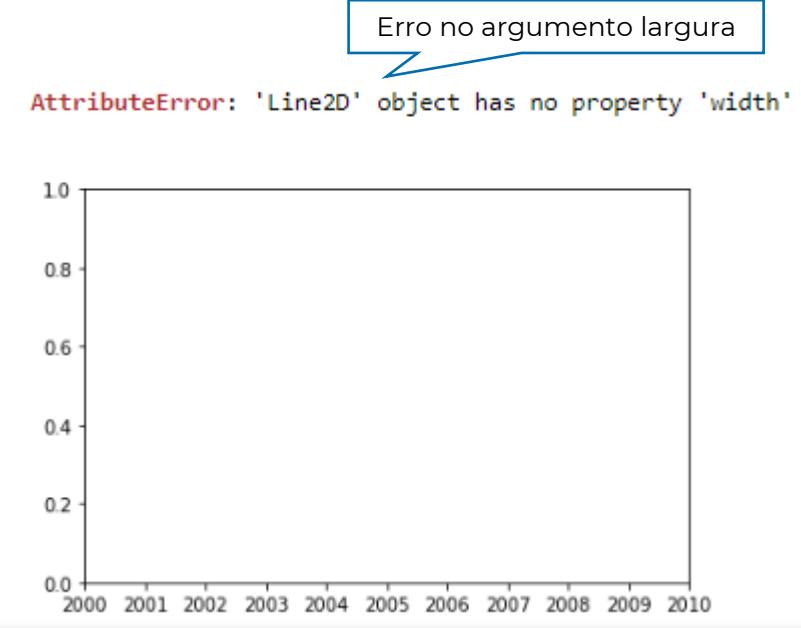
Para resolvemos esse problema, basta apagar o argumento **width**.

```
# make data
x = base.Data
y = base.Curtidas

# plot
fig, ax = plt.subplots()

ax.plot(x, y, width=5, edgecolor='white')

plt.show()
```



## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Será que existe uma tendência de queda nas curtidas?

- Vamos fazer uma análise mensal
  - Para isso precisamos criar a coluna mês/ano:

Criando a coluna do mês:

**base["coluna\_que\_queremos\_criar"] = função\_que\_vai\_criar\_essa\_coluna**

```
# Criando a coluna do mês
base["mes"] = base.Data.dt.year*100 + base.Data.dt.month
```

Nome da coluna que queremos criar

Função

## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Vamos importar uma biblioteca que vai nos ajudar a trabalhar com datas: **datetime**.

Antes de mais nada, utilizamos o comando import datetime as dt, conforme mostrado na imagem abaixo.

```
import datetime as dt
```

## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Em seguida, complementamos com o seguinte código:

```
import datetime as dt

# Criando a coluna do mês
base["mes"] = base.Data.dt.year*100 + base.Data.dt.month

# Verificando a coluna criada
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874	202109
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958	202109
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816	202109
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164	202109
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437	202109

## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Agora como poderíamos colocar essa tabela na forma de um gráfico?

Vamos utilizar o comando **groupby** para somar os valores do mês.

Relembrando o groupby:

Variável = base.groupby("Coluna")["Coluna\_que\_queremos\_agrupar"].função()

```
media_mensal = base.groupby("mes")["Curtidas"].mean()
print(media_mensal)

mes
202109    6181.500000
202110    13390.400000
202111    4682.000000
202112    11863.833333
202201    15913.285714
202202    17081.363636
202203    13299.555556
Name: Curtidas, dtype: float64
```

## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Se a gente quiser o tipo da média mensal utilizando o comando `type`, vamos observar que ele é uma série, que possui 2 elementos: índice e os valores.

O índice vai ser o mês e os valores vão ser as médias.

E podemos usar exatamente essas informações para criar o nosso gráfico.

```
type(media_mensal)
pandas.core.series.Series

# O groupby vai ter o index
media_mensal.index
Int64Index([202109, 202110, 202111, 202112, 202201, 202202, 202203], dtype='int64', name='mes')

# E os valores
media_mensal.values
array([ 6181.5        , 13390.4        , 4682.        , 11863.83333333,
       15913.28571429, 17081.36363636, 13299.55555556])
```

## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Então vamos utilizar o nosso código e substituir os valores de x e y:

**x** = media\_mensal.index

**y** = media\_mensal.values

fig, ax = **plt.subplots()**

ax.plot(**x**, **y**, linewidth=2.0)

**plt.show()**

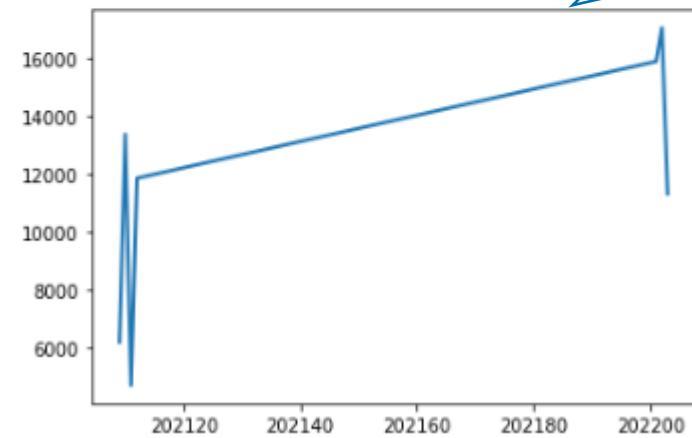
```
# make data
x = media_mensal.index
y = media_mensal.values

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

plt.show()
```

Repara que o nosso gráfico ficou estranho!



## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Vamos entender o motivo do gráfico ter ficado assim, se observarmos os valores do índice vamos notar que existe um gap entre eles.

Como podemos corrigir?

Considerar o índice como um **texto** e não um inteiro.

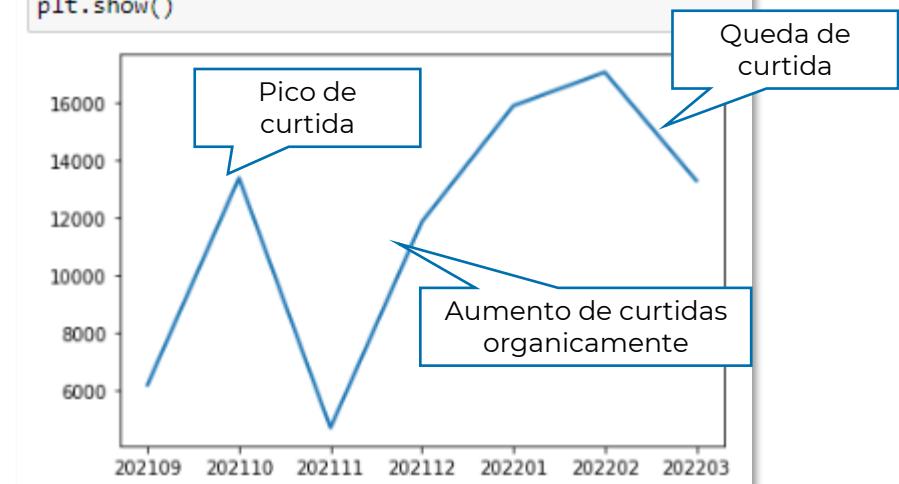
Agora conseguimos mostrar de forma visual as curtidas de cada mês.

```
# make data
x = media_mensal.index.astype(str)
y = media_mensal.values

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

plt.show()
```



## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Como já vimos a média é interferida pelos outliers, então vamos aplicar novos dados nesse gráfico:

```
minimo = base.groupby("mes")["Curtidas"].min()
maximo = base.groupby("mes")["Curtidas"].max()
```

```
# Fazendo para o mínimo e máximo de curtidas
minimo = base.groupby("mes")["Curtidas"].min()
maximo = base.groupby("mes")["Curtidas"].max()
```

## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Plotando esse gráfico:

```
x = media_mensal.index.astype(str)
```

```
y = media_mensal.values
```

```
x1 = minimo.index.astype(str)
```

```
y1 = minimo.values
```

```
x2 = maximo.index.astype(str)
```

```
y2 = maximo.values
```

```
fig, ax = plt.subplots()
```

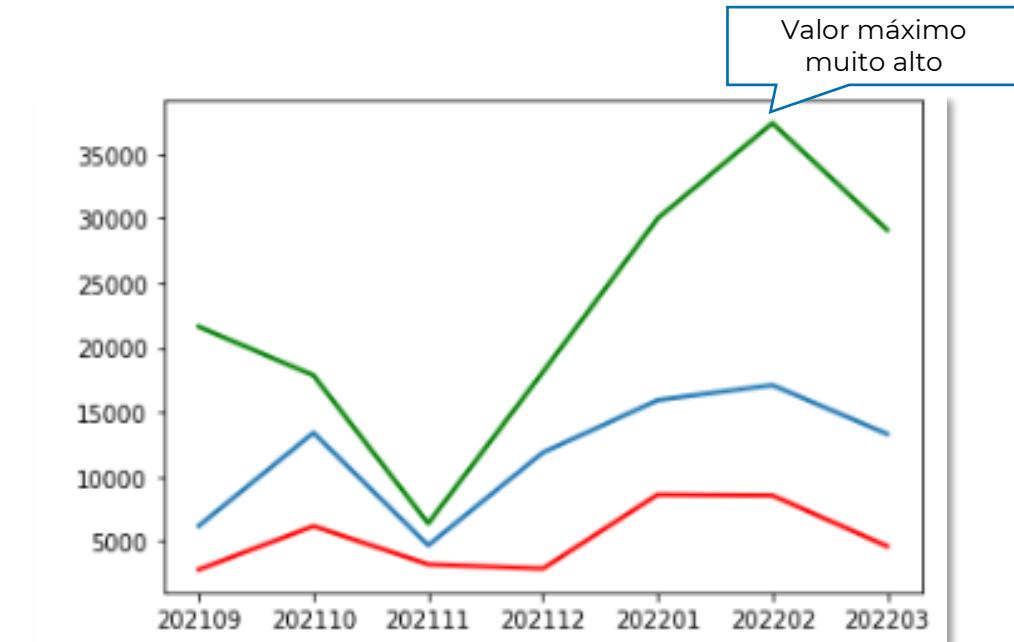
```
ax.plot(x, y, linewidth=2.0)
```

Argumento da cor

```
ax.plot(x1, y1, linewidth=2.0, color="r")
```

```
ax.plot(x2, y2, linewidth=2.0, color='g')
```

```
plt.show()
```



## Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Agora vamos fazer uma visualização ordenando por mês e curtidas para entender nosso outlier:

base[(base.mes >= 202202)].sort_values(["mes", "Curtidas"], ascending=False)											
	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes	
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203	
43	Foto	2022-03-04	24399	266		NaN	S	S	24665	202203	
45	Reels	2022-03-08	19621	275		Trends	S	S	19896	202203	
50	Foto	2022-03-26	16551	186		NaN	S	N	16737	202203	
49	Foto	2022-03-22	9087	106		NaN	S	S	9193	202203	
44	IGTV	2022-03-07	5918	116	Dicas de como usar/Novos Produtos	S	N	N	6034	202203	
47	IGTV	2022-03-12	5489	77	Dicas de como usar/Novos Produtos	S	N	N	5566	202203	
51	IGTV	2022-03-27	4934	65	Dicas de como usar/Produtos	S	N	N	4999	202203	
46	Foto	2022-03-09	4613	50	Influenciadores	S	N	N	4663	202203	
39	Foto	2022-02-17	37351	502	Promoções	S	S	N	37853	202202	
33	Foto	2022-02-06	24655	186	Influenciadores	S	S	N	24841	202202	
40	Foto	2022-02-21	21621	213	Influenciadores	S	S	S	21834	202202	
34	Foto	2022-02-09	20660	292	Influenciadores	S	S	N	20952	202202	
38	Foto	2022-02-15	17687	213		NaN	S	N	17900	202202	
37	Vídeo	2022-02-13	15219	357	Datas comemorativas	S	S	N	15576	202202	
41	Foto	2022-02-22	12530	90		NaN	S	N	12620	202202	
36	Foto	2022-02-12	11802	102	Produtos	S	S	N	11904	202202	
32	IGTV	2022-02-04	9270	222	Dicas de como usar/Produtos	S	N	N	9492	202202	
35	IGTV	2022-02-10	8556	188	Dicas de como usar/Produtos	S	N	N	8744	202202	
42	Foto	2022-02-26	8544	72	Influenciadores	S	S	N	8616	202202	

Nosso outlier foi uma promoção

## Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Notamos na aula passada que em uma publicação de promoção foi onde tivemos o maior número de curtidas.

Mas será que essa foi a única promoção que a gente fez?

Para verificar se tivemos outras promoções vamos utilizar o **contains**.

base[(base.mes >= 202202)].sort_values(["mes","Curtidas"],ascending=False)											
	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes	
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203	
43	Foto	2022-03-04	24399	266		NaN	S	S	24665	202203	
45	Reels	2022-03-08	19621	275		Trends	S	S	N	19896	202203
50	Foto	2022-03-26	16551	186		NaN	S	N	N	16737	202203
49	Foto	2022-03-22	9087	106		NaN	S	S	N	9193	202203
44	IGTV	2022-03-07	5918	116	Dicas de como usar/Novos Produtos	S	N	N	6034	202203	
47	IGTV	2022-03-12	5489	77	Dicas de como usar/Novos Produtos	S	N	N	5566	202203	
51	IGTV	2022-03-27	4934	65	Dicas de como usar/Produtos	S	N	N	4999	202203	
46	Foto	2022-03-09	4613	50		Influenciadores	S	N	N	4663	202203
39	Foto	2022-02-17	37351	502		Promoções	S	S	N	37853	202202
33	Foto	2022-02-06	24655	186		Influenciadores	S	S	N	24841	202202
40	Foto	2022-02-21	21621	213		Influenciadores	S	S	S	21834	202202
34	Foto	2022-02-09	20660	292		Influenciadores	S	S	N	20952	202202
38	Foto	2022-02-15	17687	213		NaN	S	N	N	17900	202202
37	Vídeo	2022-02-13	15219	357		Datas comemorativas	S	S	N	15576	202202
41	Foto	2022-02-22	12530	90		NaN	S	N	N	12620	202202
36	Foto	2022-02-12	11802	102		Produtos	S	S	N	11904	202202
32	IGTV	2022-02-04	9270	222		Dicas de como usar/Produtos	S	N	N	9492	202202
35	IGTV	2022-02-10	8556	188		Dicas de como usar/Produtos	S	N	N	8744	202202
42	Foto	2022-02-26	8544	72		Influenciadores	S	S	N	8616	202202

## Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Nós vamos utilizar o **contains** para verificar se existe uma palavra dentro de um texto.

Porém, o **contains não aceita se tiver valores NaN**, então precisamos tratar isso antes

```
# Tentando usar o contains sem tratar os valores NaN
base[base.Tags.str.contains("Promoções")]

-----
ValueError                                                 Traceback (most recent call last)
Input In [25], in <cell line: 2>()
      1 # Tentando usar o contains sem tratar os valores NaN
----> 2 base[base.Tags.str.contains("Promoções")]

File ~/anaconda3/lib/site-packages/pandas\core\frame.py:3495, in DataFrame.__getitem__(self, key)
 3492     return self.where(key)
 3494 # Do we have a (boolean) 1d indexer?
-> 3495 if com.is_bool_indexer(key):
 3496     return self._getitem_bool_array(key)
 3498 # We are left with two options: a single key, and a collection of keys,
 3499 # We interpret tuples as collections only for non-MultiIndex

File ~/anaconda3/lib/site-packages/pandas\core\common.py:144, in is_bool_indexer(key)
 140     na_msg = "Cannot mask with non-boolean array containing NA / NaN values"
 141     if lib.infer_dtype(key) == "boolean" and isna(key).any():
 142         # Don't raise on e.g. ["A", "B", np.nan], see
 143         # test_loc_getitem_list_of_labels_categoricalindex_with_na
--> 144         raise ValueError(na_msg)
 145     return False
 146 return True

ValueError: Cannot mask with non-boolean array containing NA / NaN values
```

Não aceita se tiver valores **nulos**

## Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Para corrigir os valores nulos vamos utilizar o `.fillna()` que vai substituir os valores NaN pelo novo valor que passarmos.

`base.fillna("novo_valor")`

Então vamos substituir os valores nulos para vazios:

`base.fillna("")`

# O `fillna()` vai substituir os valores NaN pelo novo valor que passarmos  
`base.fillna("").tail()`

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
47	IGTV	2022-03-12	5489	77	Dicas de como usar/Novos Produtos	S	N	N	5566	202203
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203
49	Foto	2022-03-22	9087	106	Vazio	S	S	N	9193	202203
50	Foto	2022-03-26	16551	186		S	N	N	16737	202203
51	IGTV	2022-03-27	4934	65	Dicas de como usar/Produtos	S	N	N	4999	202203

## Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Então podemos usar o vazio apenas para visualizar todas as tags que possuem marcação de patrocinado.

```
base[base.fillna("").Tags.str.contains("Promoções")]
```

Então, podemos observar que temos 3 resultados que contém promoções

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
13	Foto	2021-10-24	15940	612	Promoções	S	N	N	16552	202110
39	Foto	2022-02-17	37351	502	Promoções	S	S	N	37853	202202
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203

## Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Agora vamos excluir da nossa base as linhas de promoções:

**base = base.drop([13,39,48],axis=0)**



	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
13	Foto	2021-10-24	15940	612	Promoções	S	N	N	16552	202110
39	Foto	2022-02-17	37351	502	Promoções	S	S	N	37853	202202
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203

## Módulo 7 – Filtrando a base usando o contains (efillna para tratar valores vazios)

Refazendo os cálculos pra base sem as promoções e refazendo o plot:

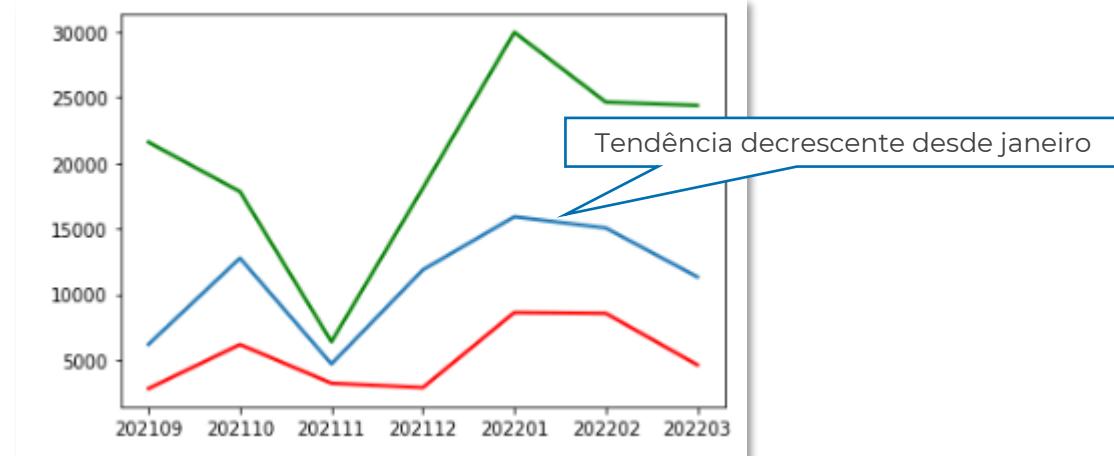
```
media_mensal = base.groupby("mes")["Curtidas"].mean()
minimo = base.groupby("mes")["Curtidas"].min()
maximo = base.groupby("mes")["Curtidas"].max()

# make data
x = media_mensal.index.astype(str)
y = media_mensal.values
x1 = minimo.index.astype(str)
y1 = minimo.values
x2 = maximo.index.astype(str)
y2 = maximo.values

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)
ax.plot(x1, y1, linewidth=2.0, color="r")
ax.plot(x2, y2, linewidth=2.0, color='g')

plt.show()
```



## Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Como podemos visualizar as tags que mais impactaram o negócio?

Para isso vamos precisar fazer o tratamento da nossa base e separar a linha de tag em duas, utilizando o **split**.

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874	202109
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958	202109
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816	202109
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164	202109
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437	202109

## Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Separando a linha em 2:

**baseTags = base**

Elemento de separação

**baseTags.Tags = baseTags.Tags.str.split("//")**

**baseTags = baseTags.explode('Tags')**

**baseTags.head()**

```
baseTags = base
baseTags.Tags = baseTags.Tags.str.split("//")
baseTags = baseTags.explode('Tags')
baseTags.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874	202109
1	Foto	2021-09-11	2930	28	Loja	N	N	N	2958	202109
1	Foto	2021-09-11	2930	28	Produtos	N	N	N	2958	202109
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816	202109
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164	202109

## Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Vamos agrupar Curtidas por Tags.

```
baseTags.groupby("Tags")["Curtidas"].mean()
```

Só que a visualização dessa tabela é difícil para o nosso cliente.

Tags	Curtidas
Datas comemorativas	17975.000000
Dicas de como usar	6833.400000
Influenciadores	15197.285714
Loja	2865.000000
Novos Produtos	10304.888889
Produtos	6269.823529
Promoções	26645.500000
Trends	20024.000000
Name: Curtidas, dtype: float64	

## Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Então vamos colocar em um gráfico para uma melhor visualização:

```
media = baseTags.groupby ("Tags")["Curtidas"].mean() .sort_values(ascending=False)
```

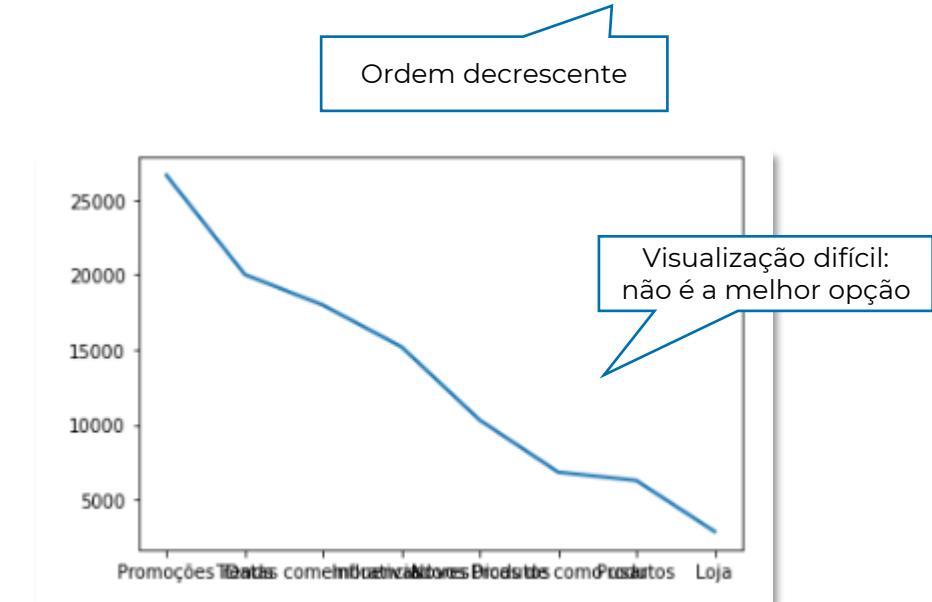
```
x = media.index
```

```
y = media.values
```

```
fig, ax = plt.subplots()
```

```
ax.plot(x, y, linewidth=2.0)
```

```
plt.show()
```



## Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Criamos um gráfico de barra com o auxílio da documentação:

```
media = baseTags.groupby ("Tags")["Curtidas"].mean() .sort_values(ascending=False)
```

```
x = media.index
```

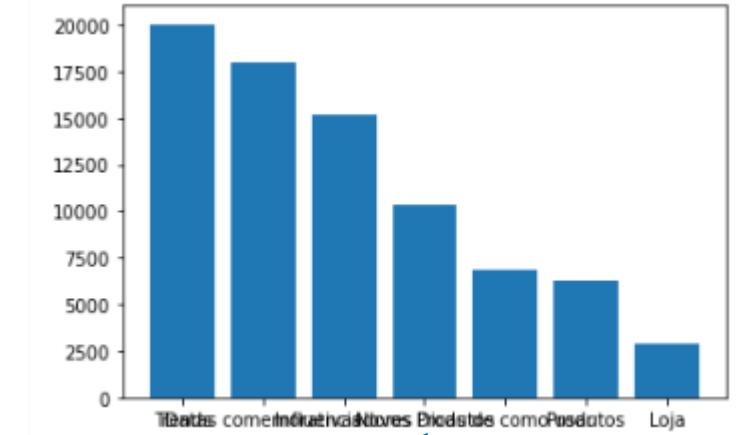
```
y = media.values
```

```
fig, ax = plt.subplots()
```

```
ax.bar(x, y, linewidth=2.0)
```

```
plt.show()
```

Alterou **plot** para **bar**

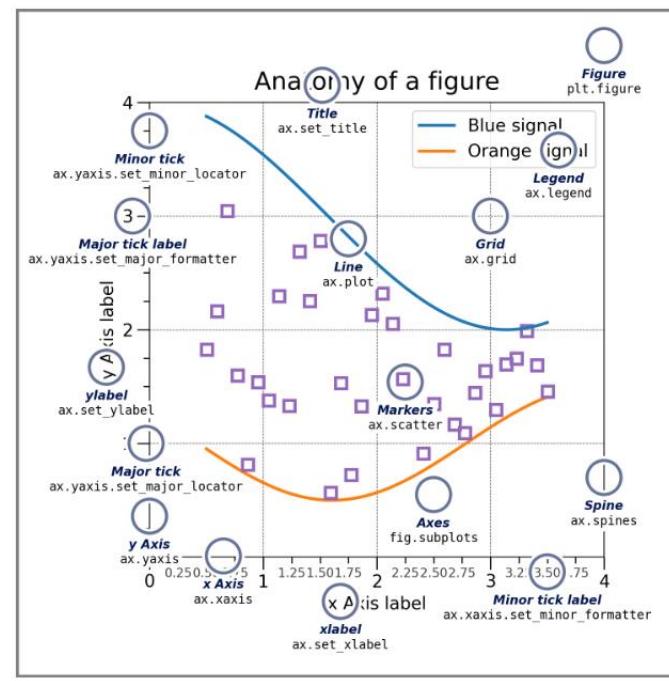


Difícil de identificar o eixo x

## Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Então agora vamos melhorar a visualização do gráfico:

O próprio matplotlib já mostra como podemos mudar a linha, a borda, o eixo, etc.



## Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Vamos rotacionar o eixo x utilizando o **tick\_params**, ou seja, os parâmetros dos rótulos:

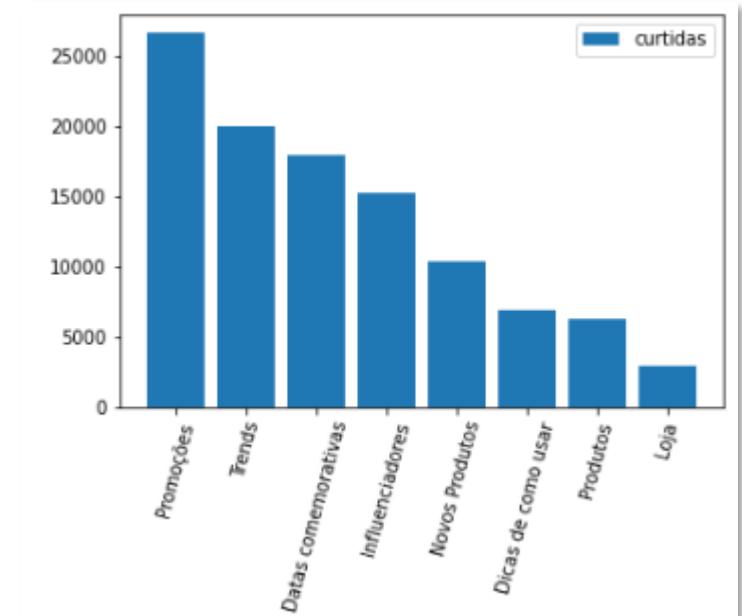
```
ax.tick_params('eixo',labelrotation=valor_da_rotação)
```

Aplicando no nosso exemplo:

```
fig, ax = plt.subplots()
```

```
ax.bar(x, y, label="curtidas")  
ax.tick_params('x',labelrotation=75)
```

```
ax.legend()  
plt.show()
```



## Módulo 7 – Usando o `annotate` para adicionar rótulos de dados no gráfico

O `annotate` vai servir para a gente colocar qualquer texto no nosso gráfico.

Estrutura:

```
plt.annotate("TEXTO", (x,y))
```

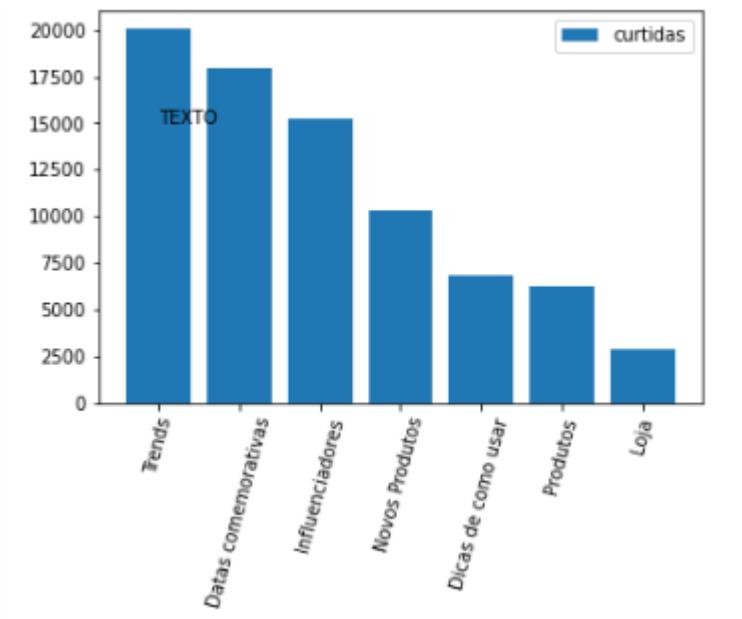
Precisamos passar os argumentos:

- Texto que vai ser escrito
- Posição (x,y) onde o texto vai estar

## Módulo 7 – Usando o `annotate` para adicionar rótulos de dados no gráfico

Vejamos um exemplo colocando a palavra texto na posição x=0 e y=15000.

```
fig, ax = plt.subplots()  
  
ax.bar(x, y, label="curtidas")  
ax.tick_params('x',labelrotation=75)  
  
ax.legend()  
  
plt.annotate('TEXTO',(0,15000))  
  
plt.show()
```



## Módulo 7 – Usando o `annotate` para adicionar rótulos de dados no gráfico

Melhorando a formatação do `annotate`:

```
for i in np.arange(0,8):
    plt.annotate(' {:.0f}'.format(y[i]),
                 (i,y[i]),
                 ha="center",
                 xytext=(0,5),
                 textcoords="offset points"
                )
```

```
ax.set(ylim=(0, 30000))
```

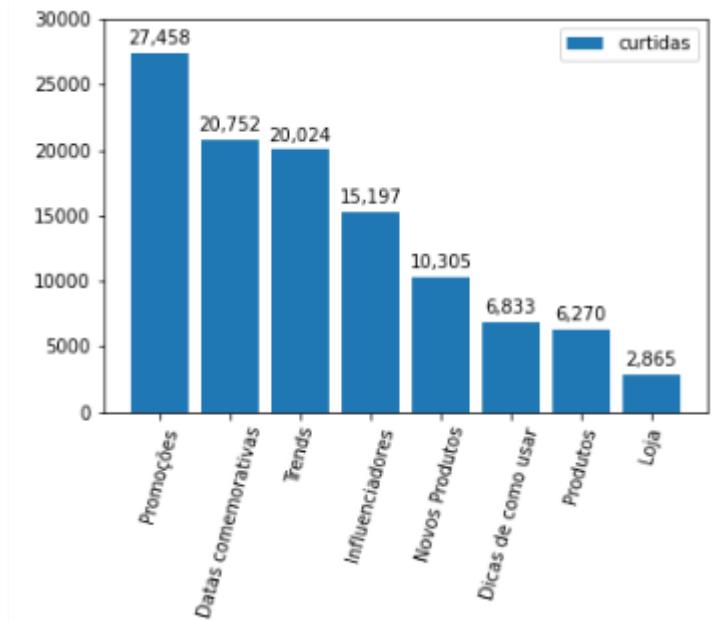
formatando o número

alinhamento horizontal central

colocando a posição do elemento

deslocamento x,y do texto

referencial que vamos fazer o deslocamento



# Módulo 7 – Criando um scatter plot usando apenas a documentação

Agora vamos fazer um exemplo de gráfico de dispersão a partir da documentação.



Tipos de  
plotagem

Exemplos Tutoriais Referência

Guia de  
usuario

Desenvolver

Notas de  
lançamento



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

plt.style.use('_mpl-gallery')

# make the data
np.random.seed(3)
x = 4 + np.random.normal(0, 2, 24)
y = 4 + np.random.normal(0, 2, len(x))
# size and color:
sizes = np.random.uniform(15, 80, len(x))
colors = np.random.uniform(15, 80, len(x))

# plot
fig, ax = plt.subplots()
ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```

## Módulo 7 – Criando um scatter plot usando apenas a documentação

Vamos copiar o código da documentação e aplicar esse gráfico a nossa base, utilizando a relação curtidas e comentários.

**x = base.Curtidas**

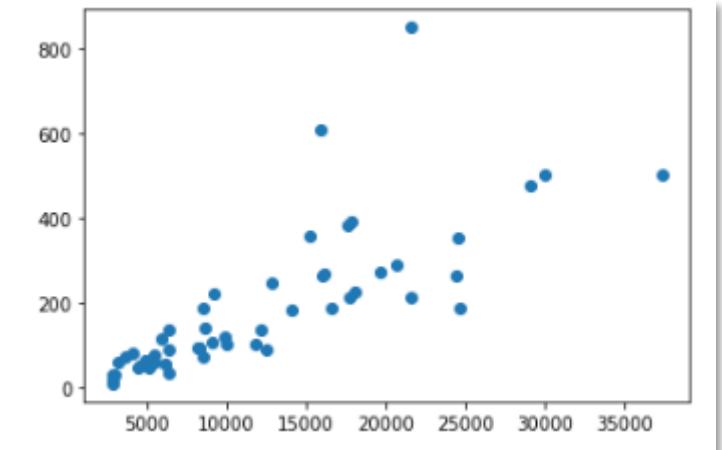
**y = base.Comentários**

Depois já podemos ir direto para o plot, usando apenas o básico:

**fig, ax = plt.subplots()**

**ax.scatter(x, y)**

**plt.show()**

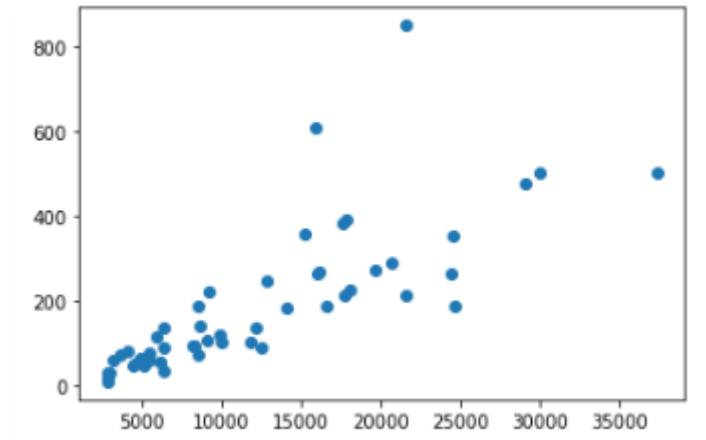


## Módulo 7 – Criando um scatter plot usando apenas a documentação

Agora vamos supor que precisamos diferenciar foto, vídeo, reels e IGTV. Vamos atribuir um número para cada tipo.

```
base.Tipo.unique()
```

```
def tipo_para_numero(tipo):  
    if(tipo == 'Foto'):  
        return 1  
    elif(tipo == 'Vídeo'):  
        return 2  
    elif(tipo == 'Reels'):  
        return 3  
    elif(tipo == 'IGTV'):  
        return 4
```



## Módulo 7 – Criando um scatter plot usando apenas a documentação

Aplicando a função dentro da nossa base utilizando o apply, teremos o seguinte.

```
base['NrTipo'] = base.Tipo.apply(tipo_para_numero)
```

base.head()												
	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes	NrTipo	
0	Foto	2021-09-11	2858	16	[Loja]	N	N	N	2874	202109	1	
1	Foto	2021-09-11	2930	28	[Loja, Produtos]	N	N	N	2958	202109	1	
2	Foto	2021-09-11	2807	9	[Loja]	N	N	N	2816	202109	1	
3	Vídeo	2021-09-12	5115	49	[Produtos]	N	N	N	5164	202109	2	
4	Foto	2021-09-13	4392	45	[Produtos]	S	N	N	4437	202109	1	

## Módulo 7 – Criando um scatter plot usando apenas a documentação

Agora podemos utilizar a coluna ‘**NrTipo**’ para a nossa variável **colors**.

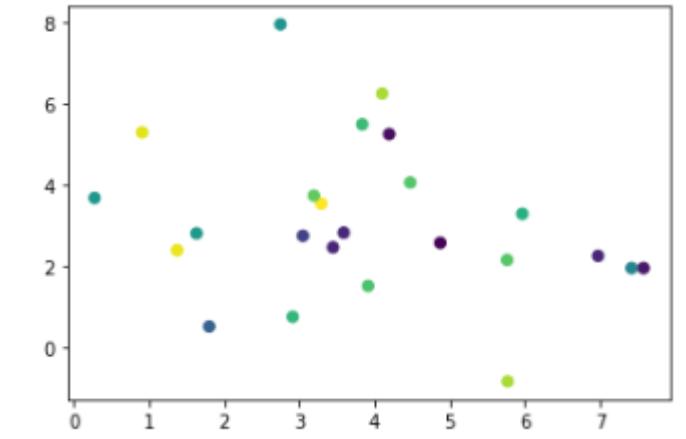
**colors = base.NrTipo**

Depois já podemos ir direto para o plot, usando apenas o básico:

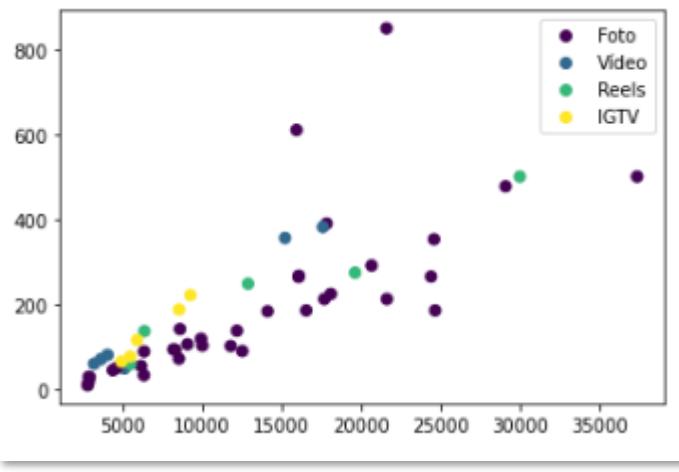
```
fig, ax = plt.subplots()
```

```
ax.scatter(x, y, c=colors)
```

```
plt.show()
```



## Módulo 7 – Criando um scatter plot usando apenas a documentação



**DICA:** Também podemos adicionar a legenda em relação a cada um dos pontos utilizando o código abaixo:

```
fig, ax = plt.subplots()
```

```
scatter_plot = ax.scatter(x, y, c=colors)
```

```
ax.legend(handles=scatter_plot.legend_elements()[0],  
         labels=['Foto','Vídeo','Reels','IGTV'])
```

```
plt.show()
```

## Módulo 7 – (Opcional) Revisando o datetime e o astype

Nessa aula vamos entender passo a passo como fazer os ajustes nos nossos cálculos de data. O primeiro passo é importar a biblioteca.

**Import** datetim`e` as dt

**Base**[‘Mes’] = **base**.Data.**dt**.month

Mas, repare que nós temos valores de 2021 e 2022. Quando utilizamos a função mês ele somaria o mesmo mês dos dois anos.

Uma forma de resolver esse problema é considerar o ano.

**Base**[‘Ano’] = **base**.Data.**dt**.year

## Módulo 7 – (Opcional) Revisando o datetime e o astype

Só que existe uma forma melhor que é colocando **Ano\_mês**, pois dessa forma ele já fica ordenado.

Por exemplo:

- 202109
- 202110
- 202201

Caso contrário não seria possível ordenar, por exemplo:

- 92021
- 102021
- 12022

## Módulo 7 – (Opcional) Revisando o datetime e o astype

Então nós queremos:

Ano mês

2021 001 → multiplicamos por 100

E como que podemos fazer essa fórmula?

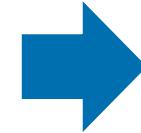
**base[“DataAjustada”] = base[‘Ano’]\*100 + base[‘Mes’]**

base.head()													
	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	Mes	Ano	DataAjustada	
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874	9	2021	202109	
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958	9	2021	202109	
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816	9	2021	202109	
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164	9	2021	202109	
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437	9	2021	202109	

## Módulo 7 – (Opcional) Revisando o datetime e o astype

Utilizamos o **astype** para transformar qualquer tipo de dado.

```
base.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 52 entries, 0 to 51  
Data columns (total 12 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          -----          ----  
 0   Tipo         52 non-null    object    
 1   Data          52 non-null    datetime64[ns]  
 2   Curtidas     52 non-null    int64     
 3   Comentários  52 non-null    int64     
 4   Tags          44 non-null    object    
 5   Pessoas       52 non-null    object    
 6   Campanhas     52 non-null    object    
 7   Carrossel     52 non-null    object    
 8   Interacoes    52 non-null    int64     
 9   Mes           52 non-null    int64     
 10  Ano           52 non-null    int64     
 11  DataAjustada 52 non-null    int64     
dtypes: datetime64[ns](1), int64(6), object(5)  
memory usage: 5.0+ KB
```



Inteiro

```
base['DataAjustada'] = base['DataAjustada'].astype(str)  
  
base.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 52 entries, 0 to 51  
Data columns (total 12 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          -----          ----  
 0   Tipo         52 non-null    object    
 1   Data          52 non-null    datetime64[ns]  
 2   Curtidas     52 non-null    int64     
 3   Comentários  52 non-null    int64     
 4   Tags          44 non-null    object    
 5   Pessoas       52 non-null    object    
 6   Campanhas     52 non-null    object    
 7   Carrossel     52 non-null    object    
 8   Interacoes    52 non-null    int64     
 9   Mes           52 non-null    int64     
 10  Ano           52 non-null    int64     
 11  DataAjustada 52 non-null    object    
dtypes: datetime64[ns](1), int64(5), object(6)  
memory usage: 5.0+ KB
```

Objeto

## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos aprender agora como inserir rótulos em um gráfico de dispersão.

O primeiro passo é importar a base do Excel:

```
Import pandas as pd
```

```
base = pd.read_excel('Analisando o engajamento no Instagram.xlsx')
```

## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos criar a coluna do tipo de publicação, como feito anteriormente:

```
def tipo_para_numero(tipo):
    if(tipo == 'Foto'):
        return 1
    elif(tipo == 'Vídeo'):
        return 2
    elif(tipo == 'Reels'):
        return 3
    elif(tipo == 'IGTV'):
        return 4
```

```
base['NrTipo'] = base.Tipo.apply(tipo_para_numero)
```

## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos visualizar a nossa base:

**base.head**

base.head()											
	Tipo	Data	Curtidas	Comentários	Visualizações	Tags	Pessoas	Campanhas	Carrossel	Interacoes	NrTipo
0	Foto	2021-09-11	2858	16	NaN	Loja	N	N	NaN	2874	1
1	Foto	2021-09-11	2930	28	NaN	Loja/Produtos	N	N	NaN	2958	1
2	Foto	2021-09-11	2807	9	NaN	Loja	N	N	NaN	2816	1
3	Vídeo	2021-09-12	5115	49	82878.0	Produtos	N	N	NaN	5164	2
4	Foto	2021-09-13	4392	45	NaN	Produtos	S	N	NaN	4437	1

## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos criar um gráfico de dispersão a partir do código da documentação:

```
fig, ax = plt.subplots()
```

Vamos deixar só  
os eixos

```
ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)
```

```
ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  
       ylim=(0, 8), yticks=np.arange(1, 8))
```

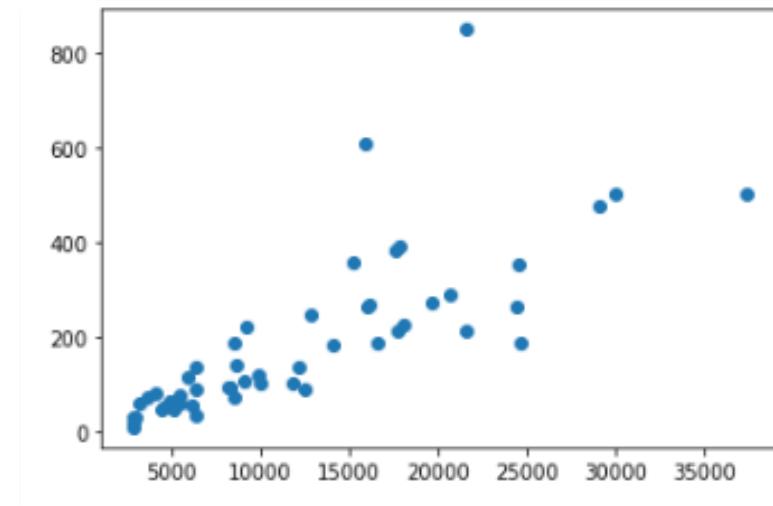
Podemos excluir

```
plt.show()
```

## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Agora, basta substituirmos o x para curtidas e o y para comentários.

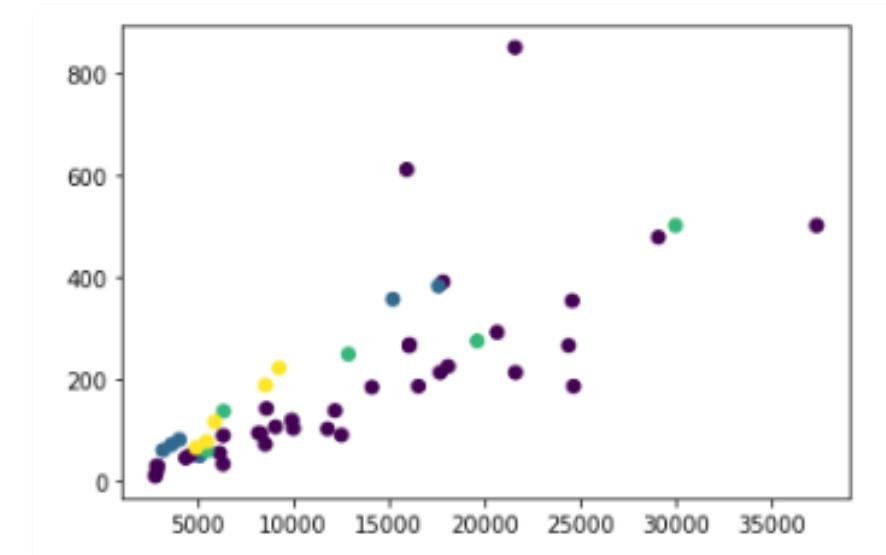
```
import matplotlib.pyplot as plt  
  
fig, ax = plt.subplots()  
  
x = base.Curtidas  
y = base.Comentários  
  
ax.scatter(x, y)  
  
plt.show()
```



## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Separando os tipos por cores:

```
import matplotlib.pyplot as plt  
  
fig, ax = plt.subplots()  
  
x = base.Curtidas  
y = base.Comentários  
  
ax.scatter(x, y, c= base.NrTipo)  
  
plt.show()
```



## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos salvar esse nosso gráfico em uma variável **scatter\_plot** que será um **pathcollection**.

```
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots()
```

```
x = base.Curtidas
```

```
y = base.Comentários
```

```
scatter_plot = ax.scatter(x, y, c= base.NrTipo)
```

```
plt.show()
```

```
scatter_plot
```

```
<matplotlib.collections.PathCollection at 0x1ed79031e80>
```

Atribuir a uma  
variável

## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

**Paths:** um módulo para lidar com polilinhas usadas no matplotlib

- Quase todos os desenhos vetoriais usam Paths em algum lugar no pipeline de desenho

### **legend\_elements()**

- Crie identificadores e rótulos de legenda para um PathCollection
- Retornos: handles (elementos 2D) e labels (texto dos rótulos)

## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Aplicando o **legend\_elements** para a nossa variável do gráfico temos como retorno os pontos 2D e o texto.

Então vamos definir isso como uma nova variável:

```
handles, labels = scatter_plot.legend_elements()
```

```
scatter_plot.legend_elements()
([<matplotlib.lines.Line2D at 0x1ed79038160>,
 <matplotlib.lines.Line2D at 0x1ed79038430>,
 <matplotlib.lines.Line2D at 0x1ed79038580>,
 <matplotlib.lines.Line2D at 0x1ed78f43ac0>],
 ['$\\mathdefault{1}$',
 '$\\mathdefault{2}$',
 '$\\mathdefault{3}$',
 '$\\mathdefault{4}$'])
```

The code block shows the output of the `scatter_plot.legend_elements()` method. It returns a tuple containing two lists. The first list contains four `Line2D` objects, each represented by a small blue line segment. The second list contains four text labels: '\$\\mathdefault{1}\$', '\$\\mathdefault{2}\$', '\$\\mathdefault{3}\$', and '\$\\mathdefault{4}\$'. A callout box labeled 'Pontos 2D' points to the first list, and another callout box labeled 'Textos' points to the second list.

## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos colocar o **ax.legend** no nosso código do gráfico utilizando como argumento a nossa variável **handles, labels**:

```
import matplotlib.pyplot as plt
```

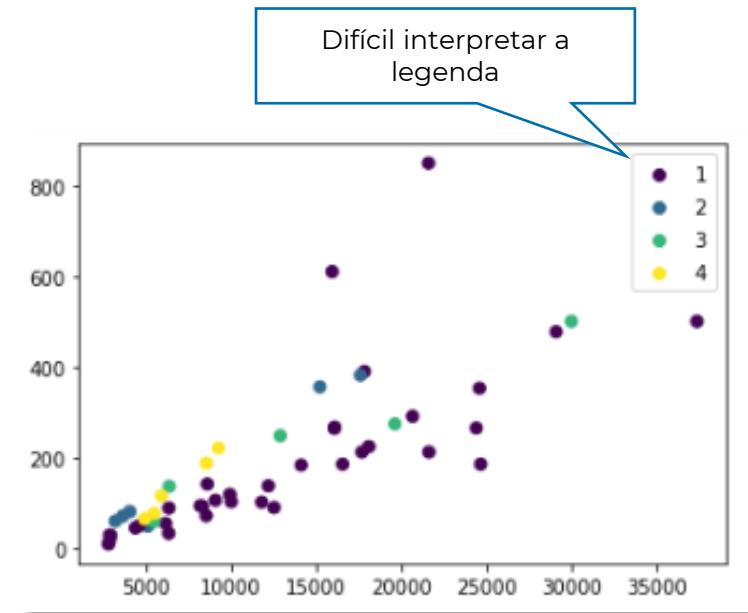
```
fig, ax = plt.subplots()
```

```
x = base.Curtidas
```

```
y = base.Comentários
```

```
scatter_plot = ax.scatter(x, y, c= base.NrTipo)  
ax.legend(handles, labels)
```

```
plt.show()
```



## Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Alterando a legenda para os nossos valores:

```
legendas = ['Foto', 'Vídeo', 'Reels', 'IGTV']
```

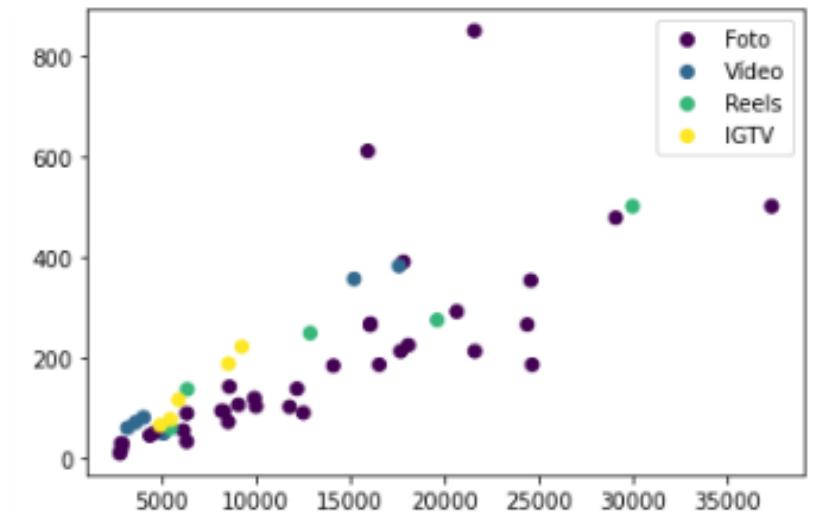
```
fig, ax = plt.subplots()
```

```
x = base.Curtidas
```

```
y = base.Comentários
```

```
scatter_plot = ax.scatter(x, y, c= base.NrTipo)  
ax.legend(handles=handles, labels=legendas)
```

```
plt.show()
```

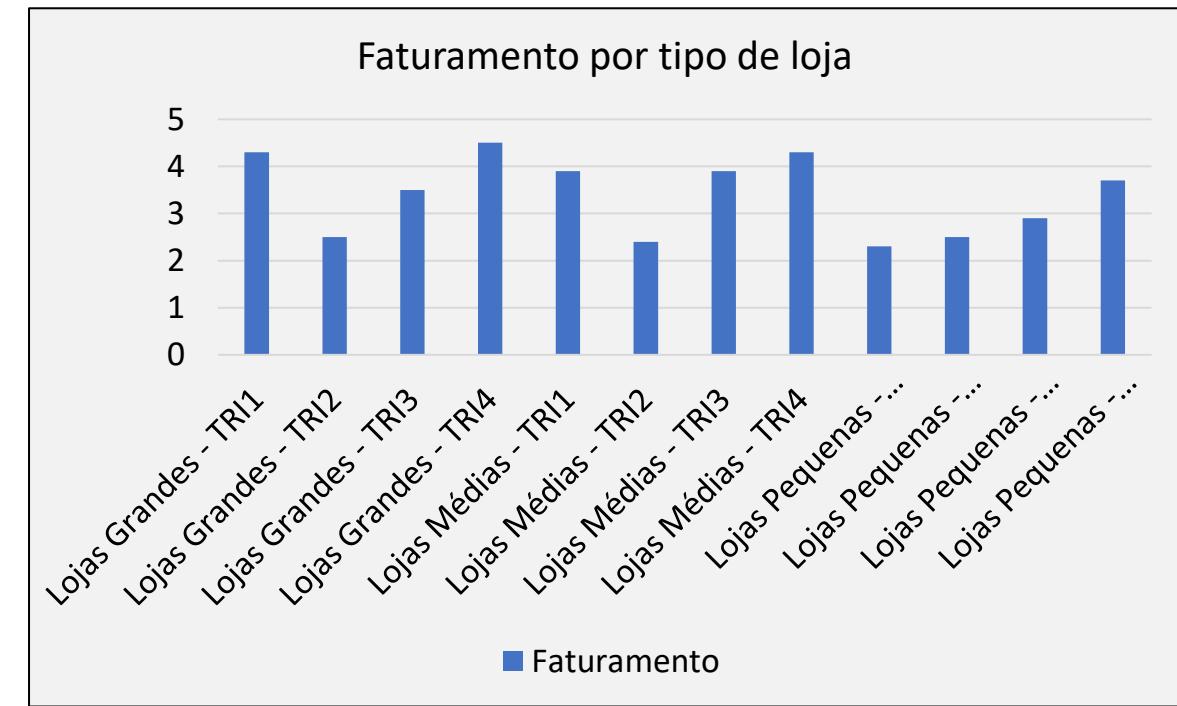
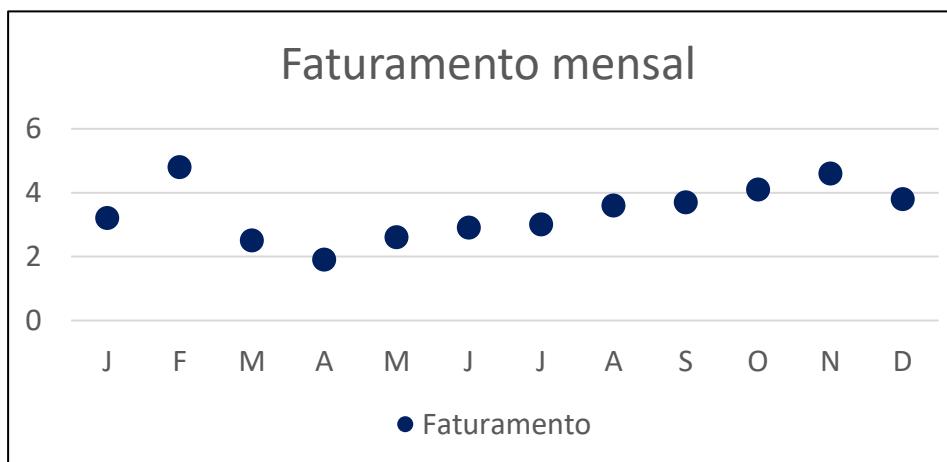


## MÓDULO 8

# Boas práticas para apresentação de dados

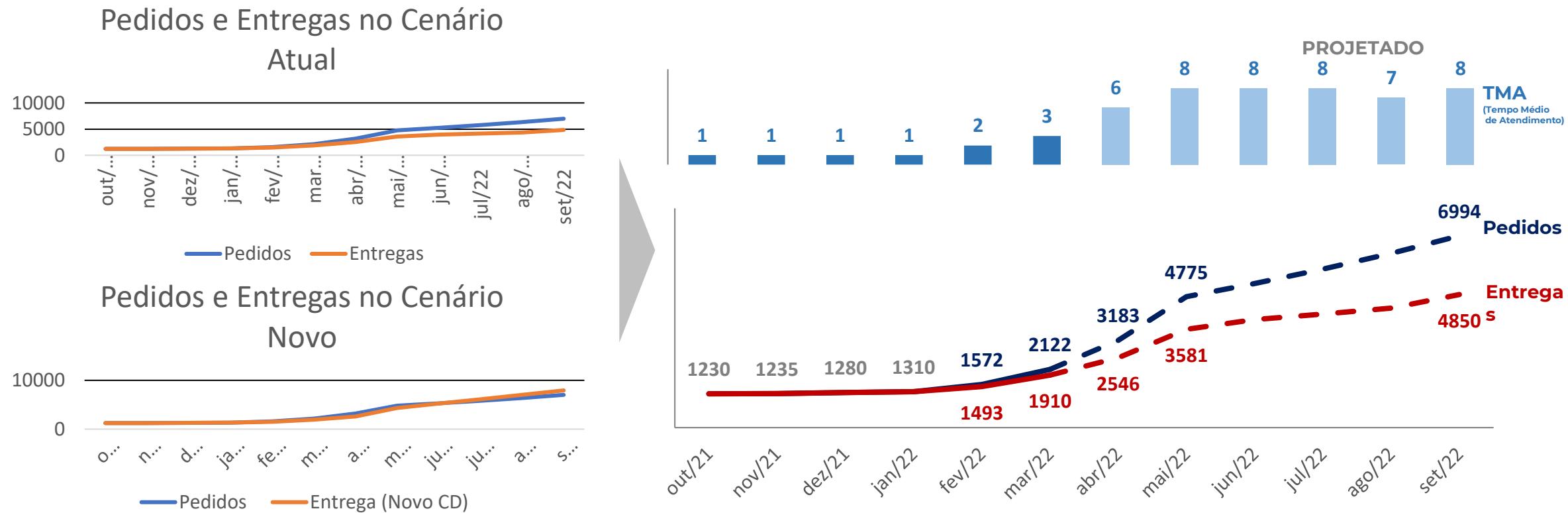
## Módulo 8 – Introdução aos conceitos básicos de apresentação de dados

Todas as pessoas conseguem apresentar dados. Porém, a forma como você **apresenta**, a forma como esses **gráficos estão mostrados**, isso sim vai ser o nosso diferencial como um **bom cientista de dados**.



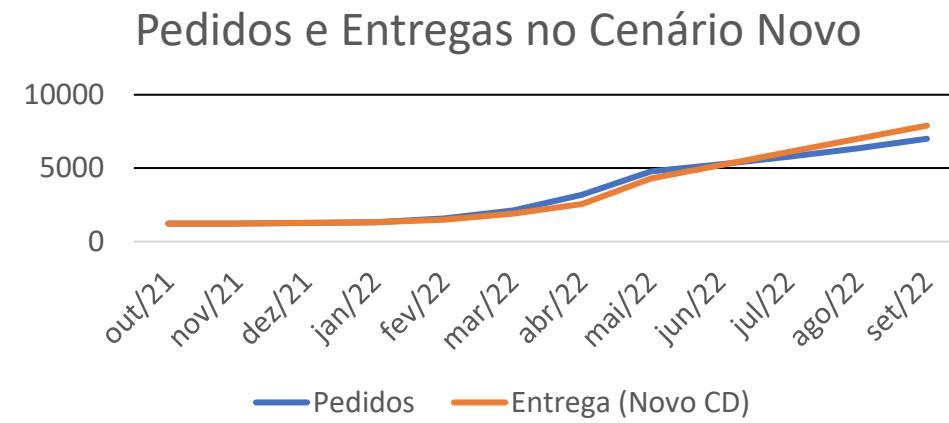
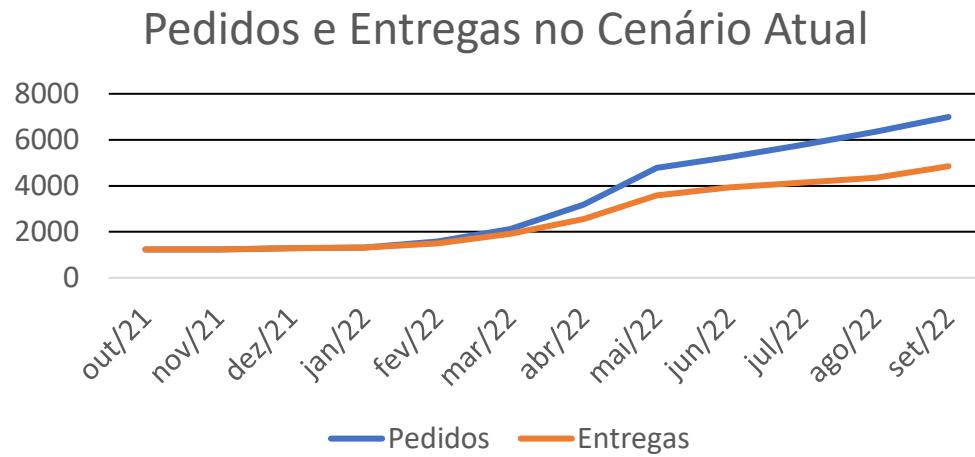
# Módulo 8 – Introdução aos conceitos básicos de apresentação de dados

Repare as diferenças dos gráficos abaixo:



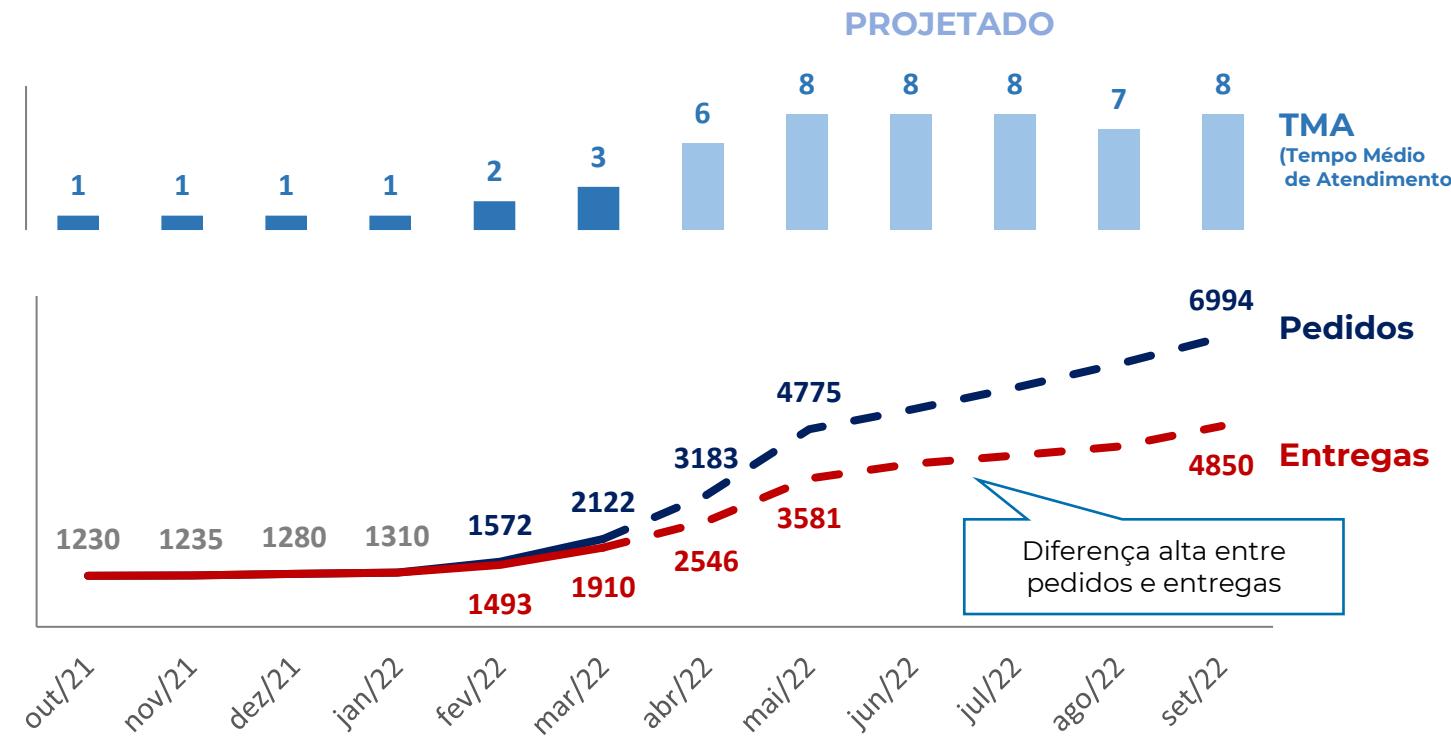
## Módulo 8 – Introdução aos conceitos básicos de apresentação de dados

Repare que nos gráficos abaixo é **mais difícil de visualizar os valores**.



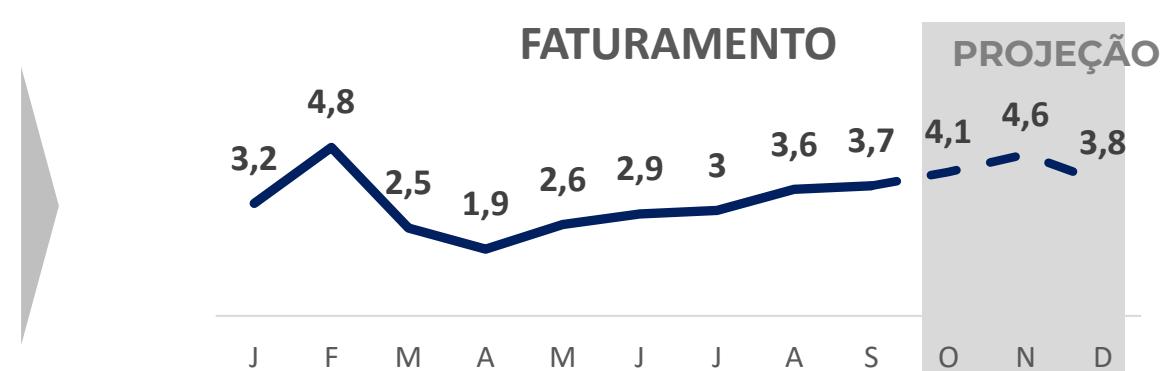
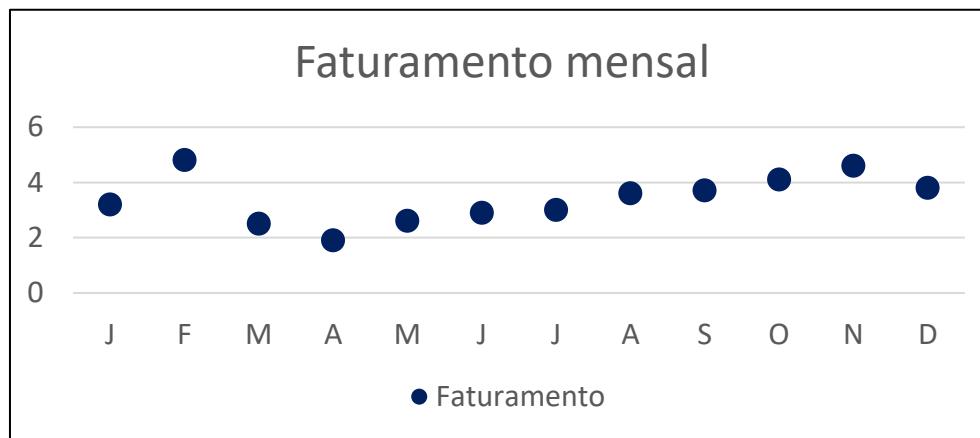
## Módulo 8 – Introdução aos conceitos básicos de apresentação de dados

Note que agora é **mais fácil de observar os valores**, a diferença do real e do projetado.



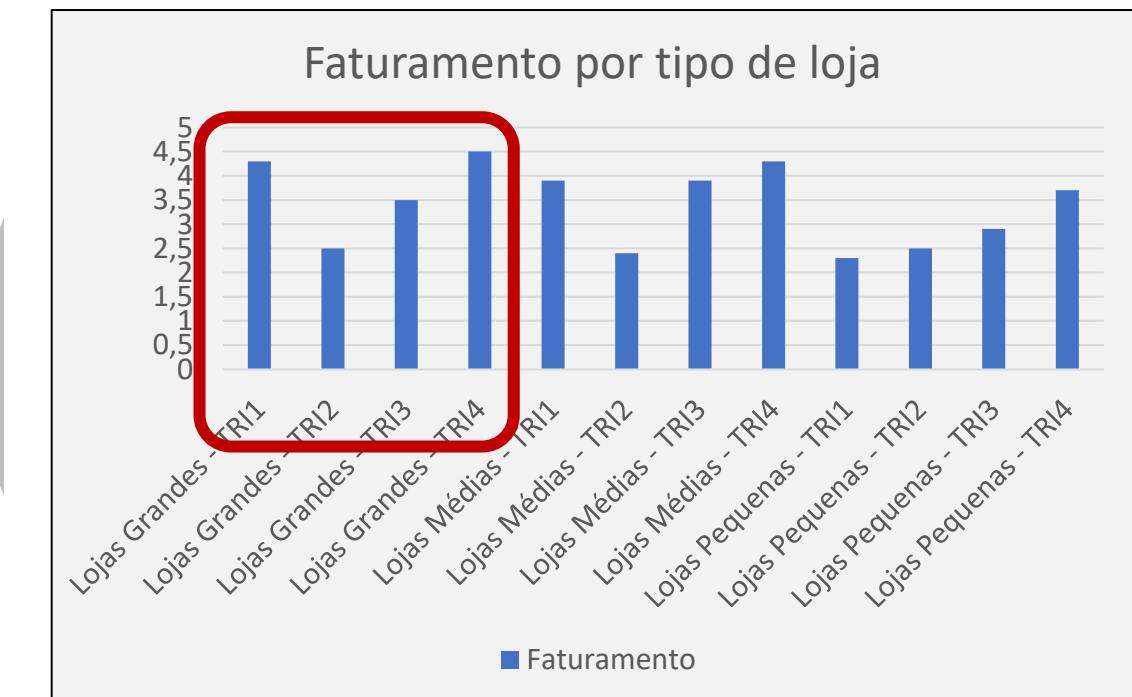
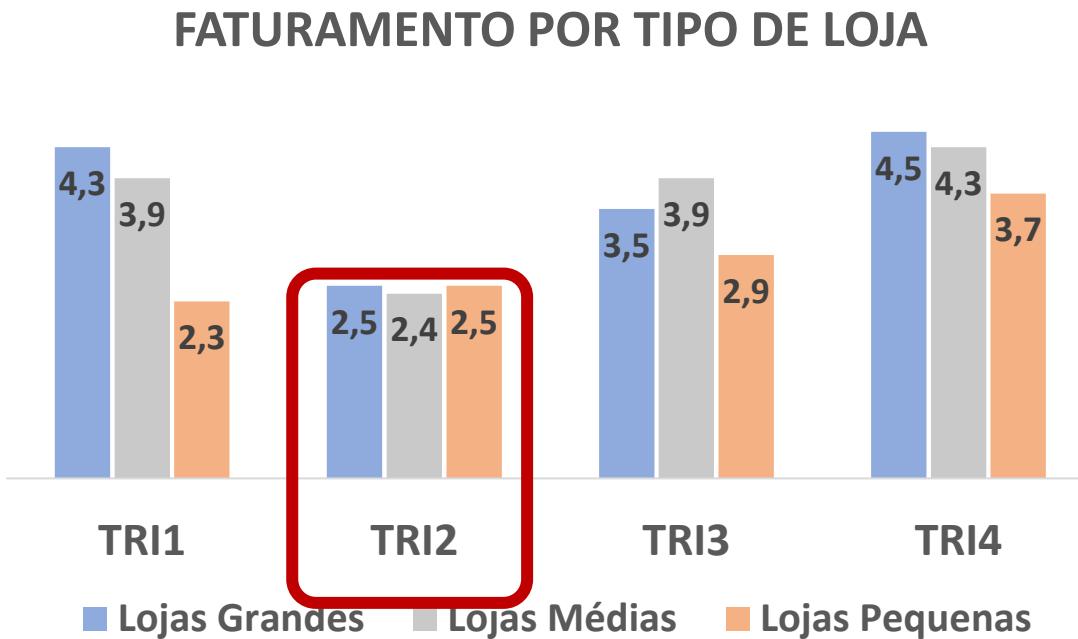
## Módulo 8 – Introdução aos conceitos básicos de apresentação de dados

Repare que quando **ligamos os pontos** fica muito **mais fácil de visualizar** o faturamento dos meses.



## Módulo 8 – Introdução aos conceitos básicos de apresentação de dados

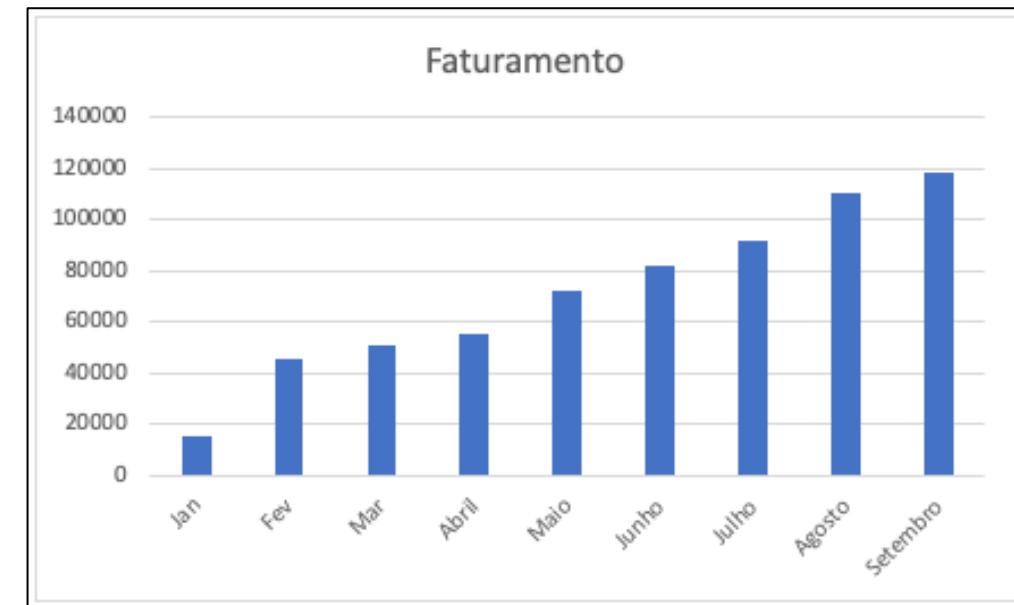
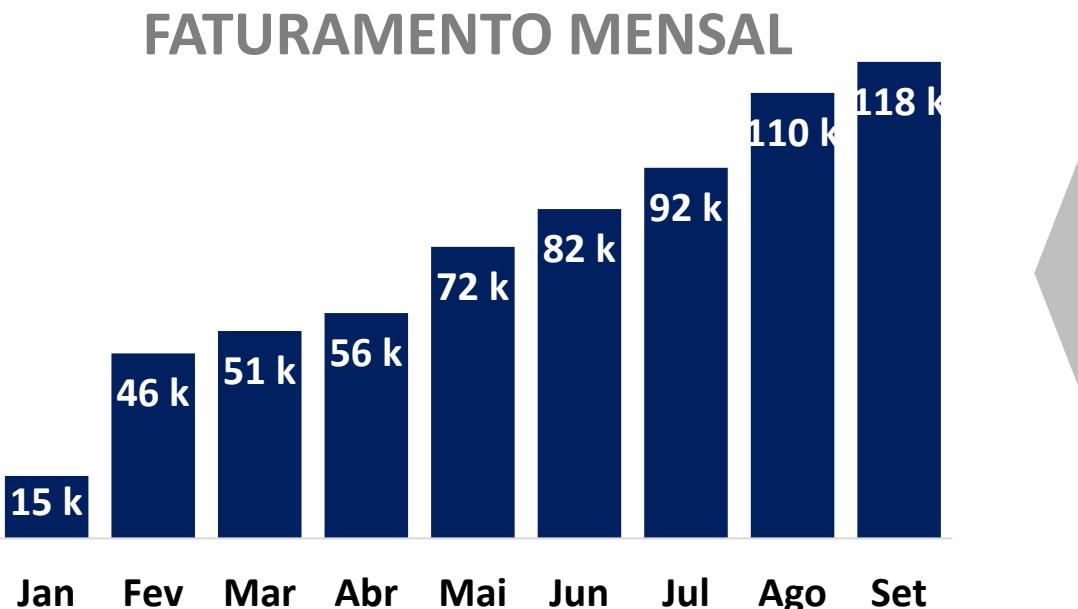
Repare que no gráfico da esquerda **não tem uma sequência cronológica**, enquanto no da direita é **mais nítido visualizar cada trimestre**.



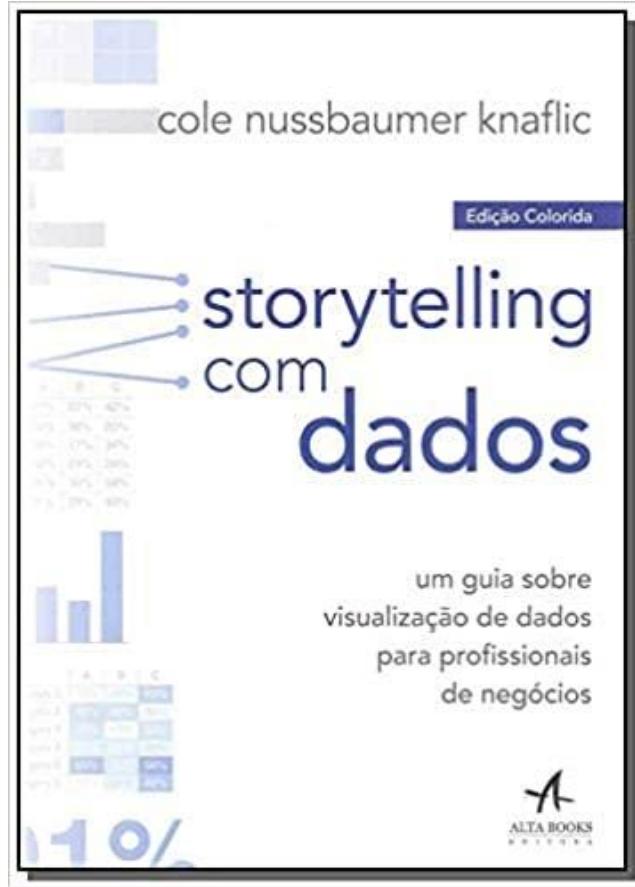
## Módulo 8 – Introdução aos conceitos básicos de apresentação de dados

Repare que agora a informação é a mesma, porém repare na **diferença visual**.

O gráfico fica mais **confortável** para as pessoas.



## Módulo 8 – Introdução aos conceitos básicos de apresentação de dados



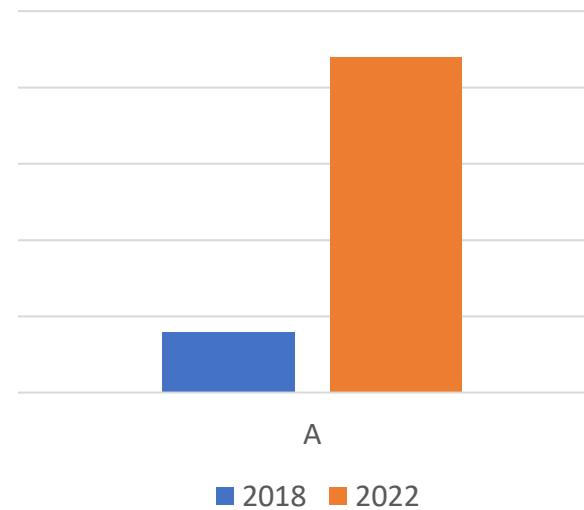
Essas dicas foram retiradas do livro “Storytelling com dados”.

**Storytelling com dados –  
Cole Nussbaumer Knaflic**

## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Vamos falar de coisas que são bem importantes antes de entrar nos detalhes visuais. No gráfico abaixo qual dos dois apresenta um maior crescimento?

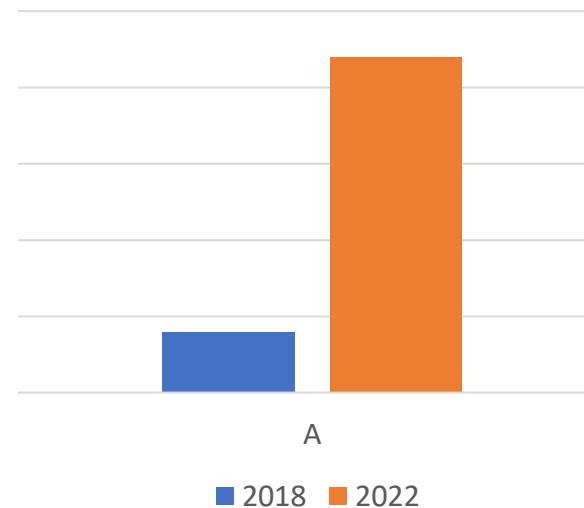
Essa pergunta deve ser respondida muito facilmente por qualquer pessoa que visualize esses gráficos.



## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Se o seu público fica em dúvida, ele pode interpretar de uma maneira errada.

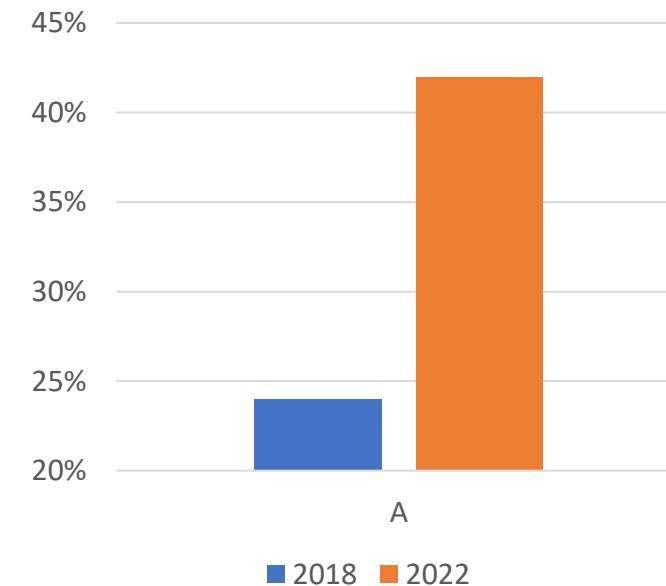
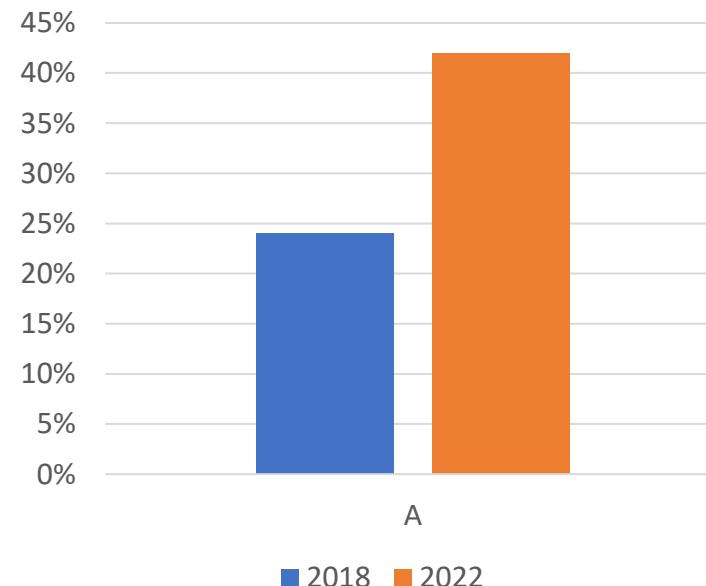
As pessoas podem achar que você está querendo enganá-las ou ocultar uma informação.



## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Observe que ao colocar o valor dos eixos, um dos gráficos começa em 0% e o outro em 20%.

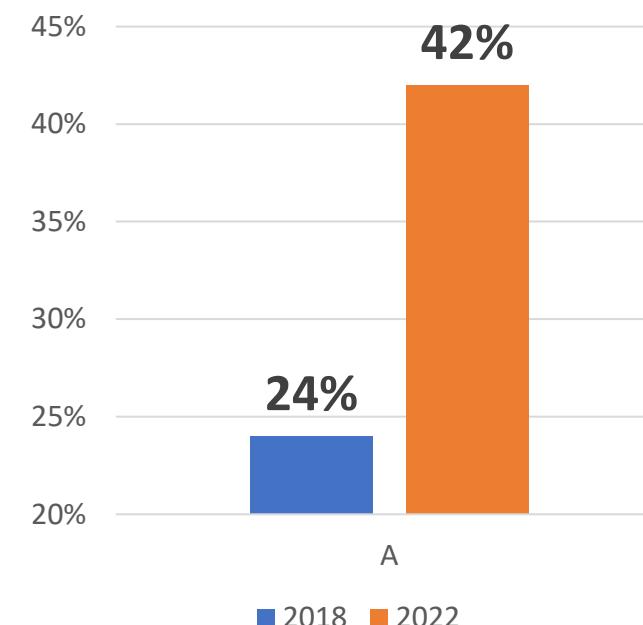
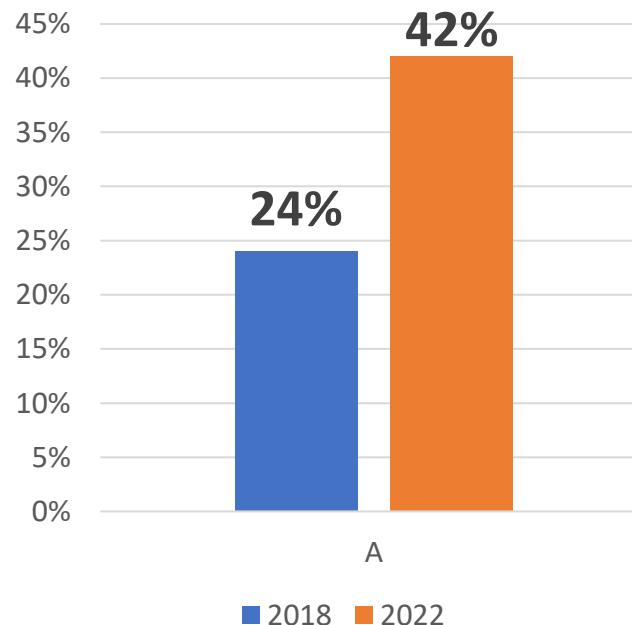
Parecia que estávamos mostrando um comportamento que não é verdade.



## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

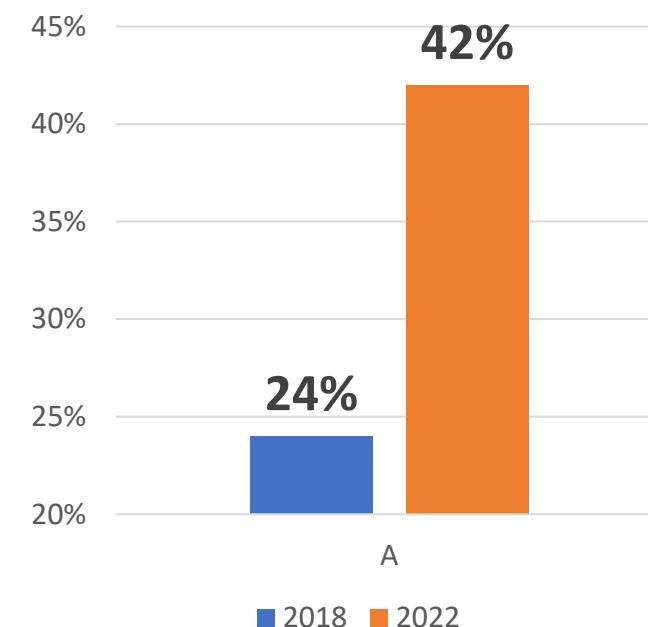
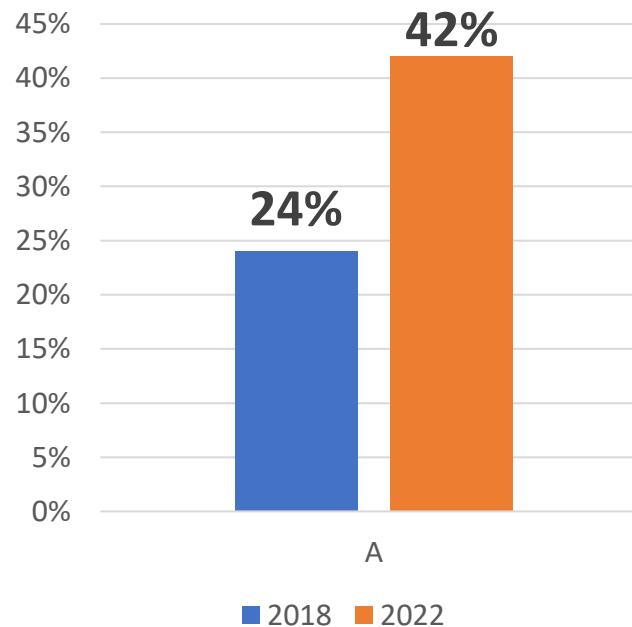
Quando colocamos o rótulo de dados, percebemos que as informações são iguais.

Mas, repare como visualmente não parecem iguais.



## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Como o nosso cérebro vai comparar o tamanho das barras, é importante que a gente sempre **comece do zero** para que essa comparação seja feita da forma correta.

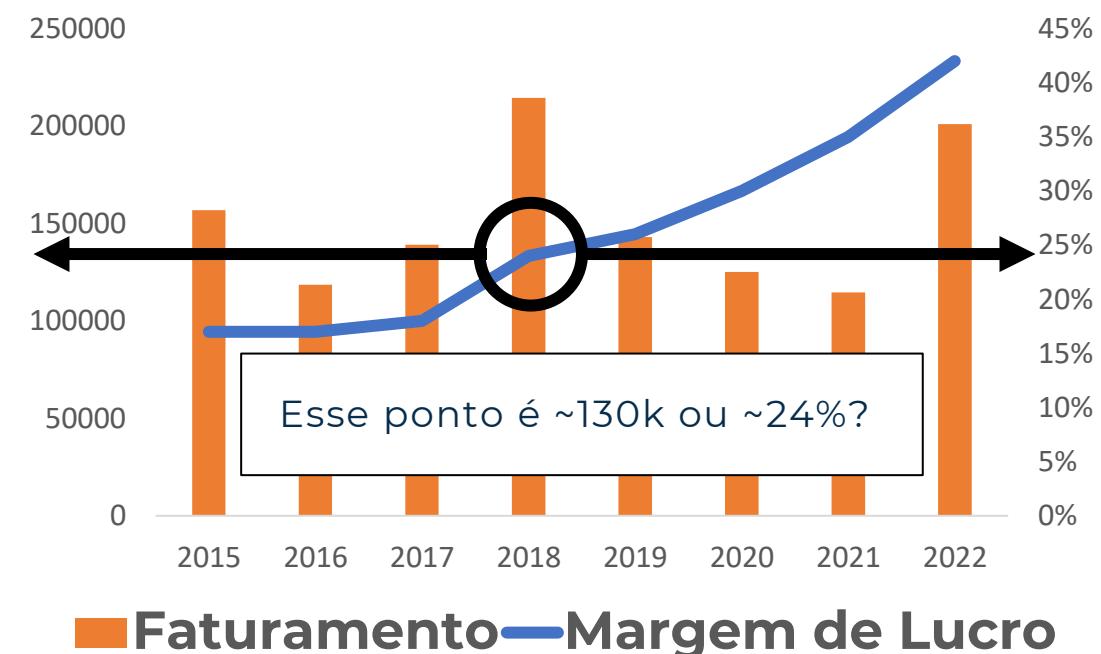


## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Geralmente, o **eixo secundário não é uma boa ideia** pois é difícil identificar a informação.

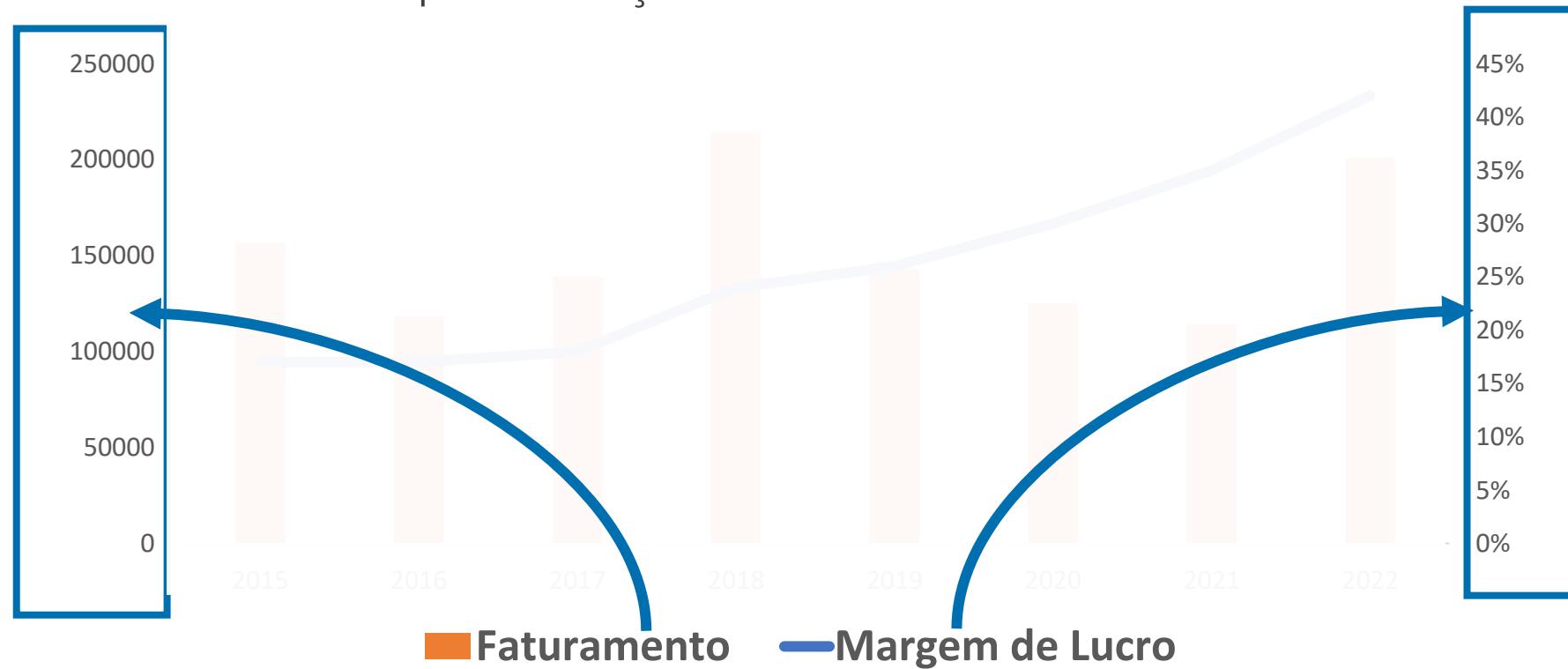
Se não está tão claro para você, vai estar menos ainda pro seu público.

**Dica:** faça a sua apresentação voltada para o seu público, com os gráficos que as pessoas estão acostumadas a ver.



## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Você pode pensar que é muito fácil identificar os eixos, mas será que todas as pessoas que estão assistindo a sua apresentação sabem disso?



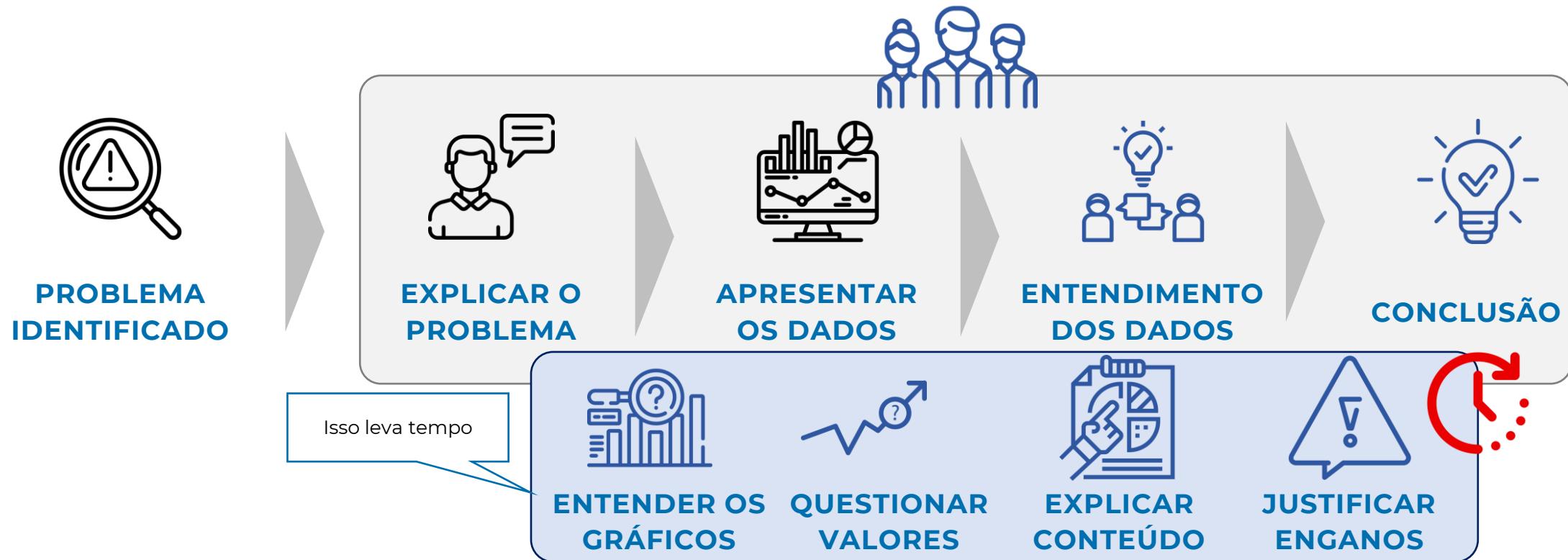
## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Nós começamos a montar uma apresentação identificando e explicando o problema, apresentamos os dados e as pessoas vão ter que entender os dados para chegar em uma conclusão.



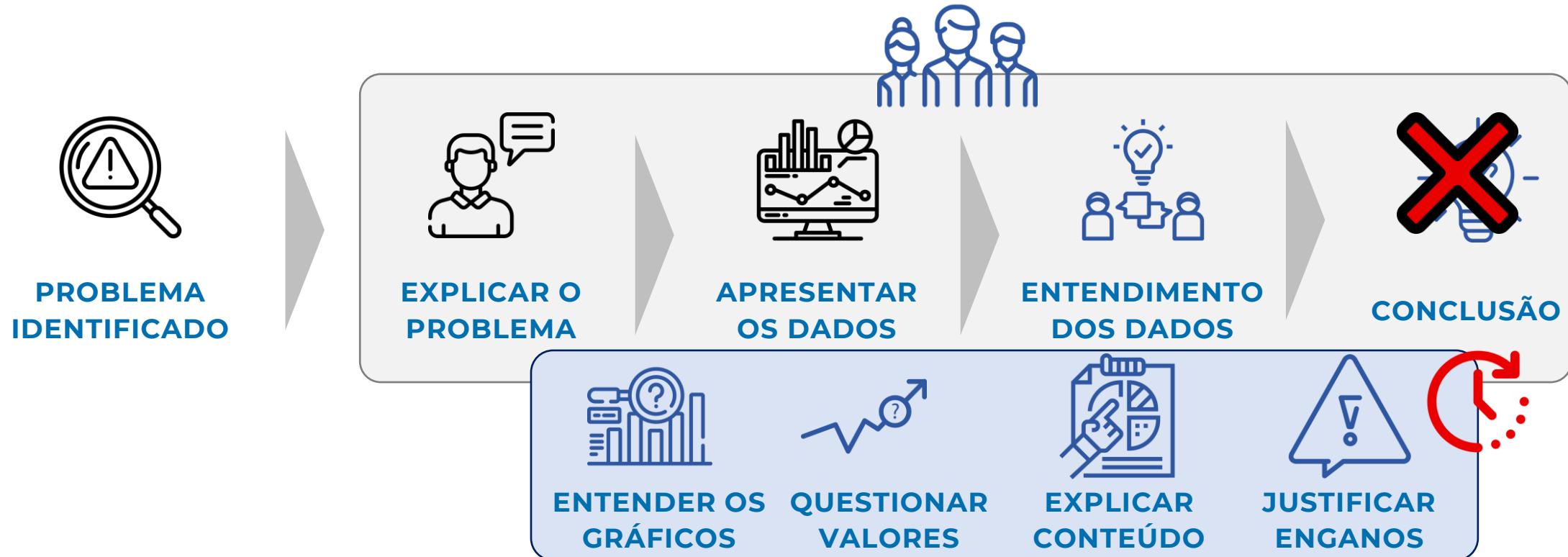
## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Basicamente vamos adicionar etapas no processo de apresentação e entendimento dos dados.



## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Isso pode fazer com que a nossa reunião não tenha uma conclusão.



## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

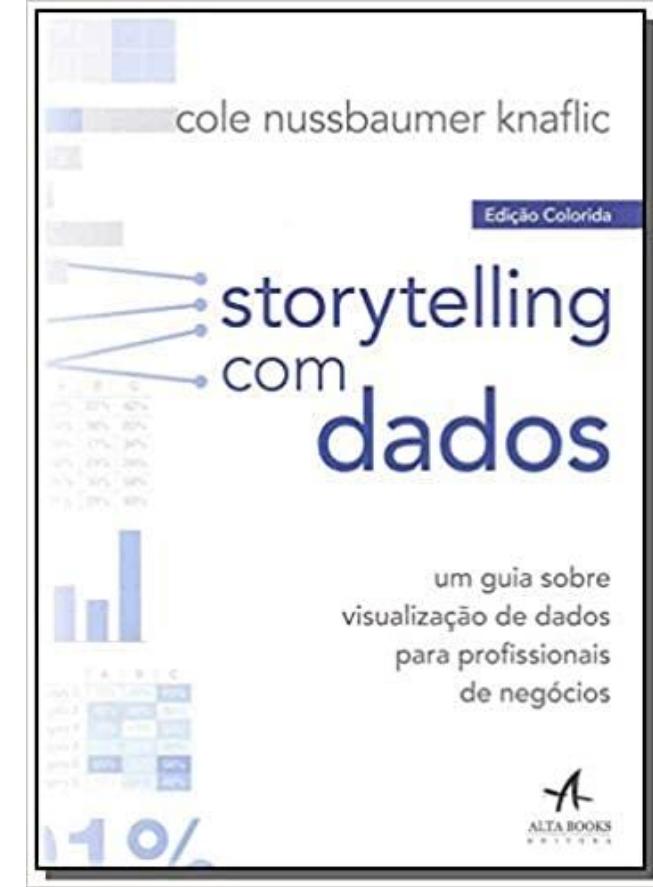
Então façam com que a etapa entre apresentar os dados e o entendimento dos dados seja **a mais tranquila possível para o usuário**.



## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

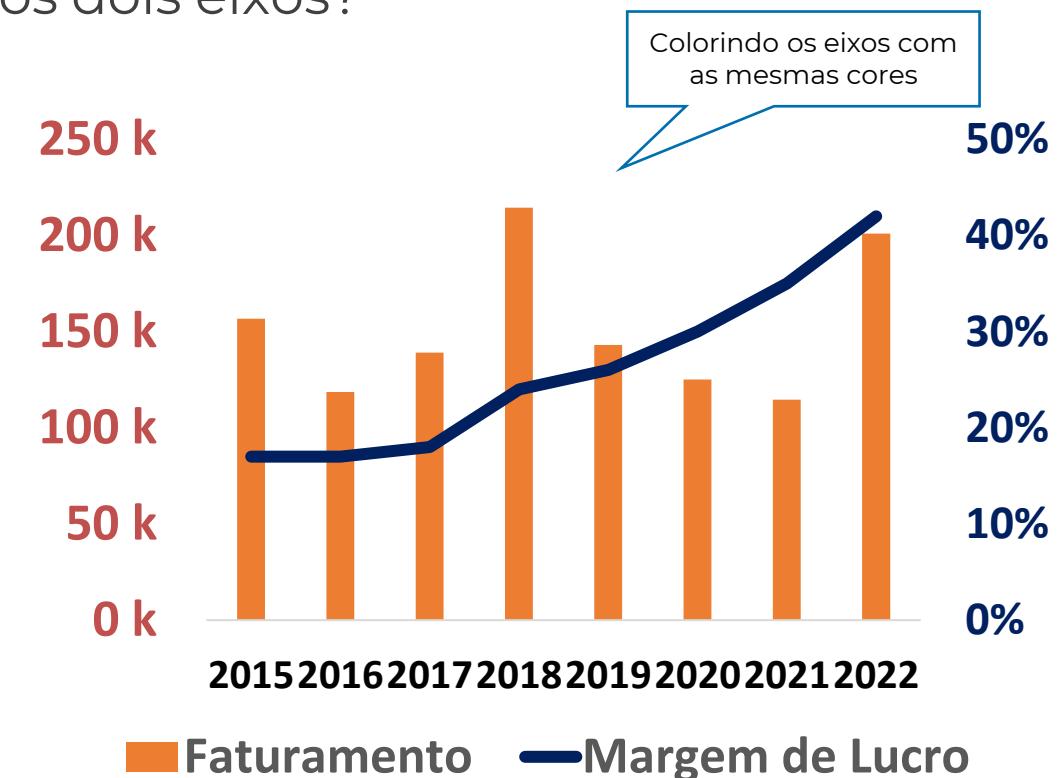
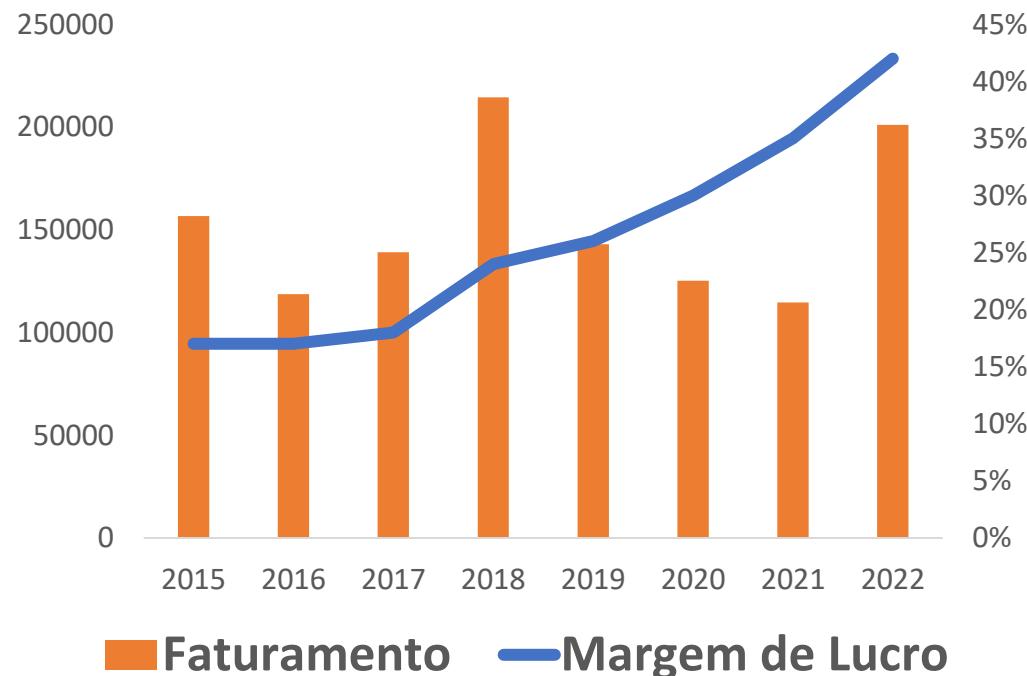
“No que diz respeito às nossas comunicações visuais, o que mais importa é a carga cognitiva **percebida** por parte do nosso público: o quanto ele **acredita que precisará se esforçar** para entender a informação de sua comunicação”

Storytelling com Dados - Cole Nussbaumer Knaflic



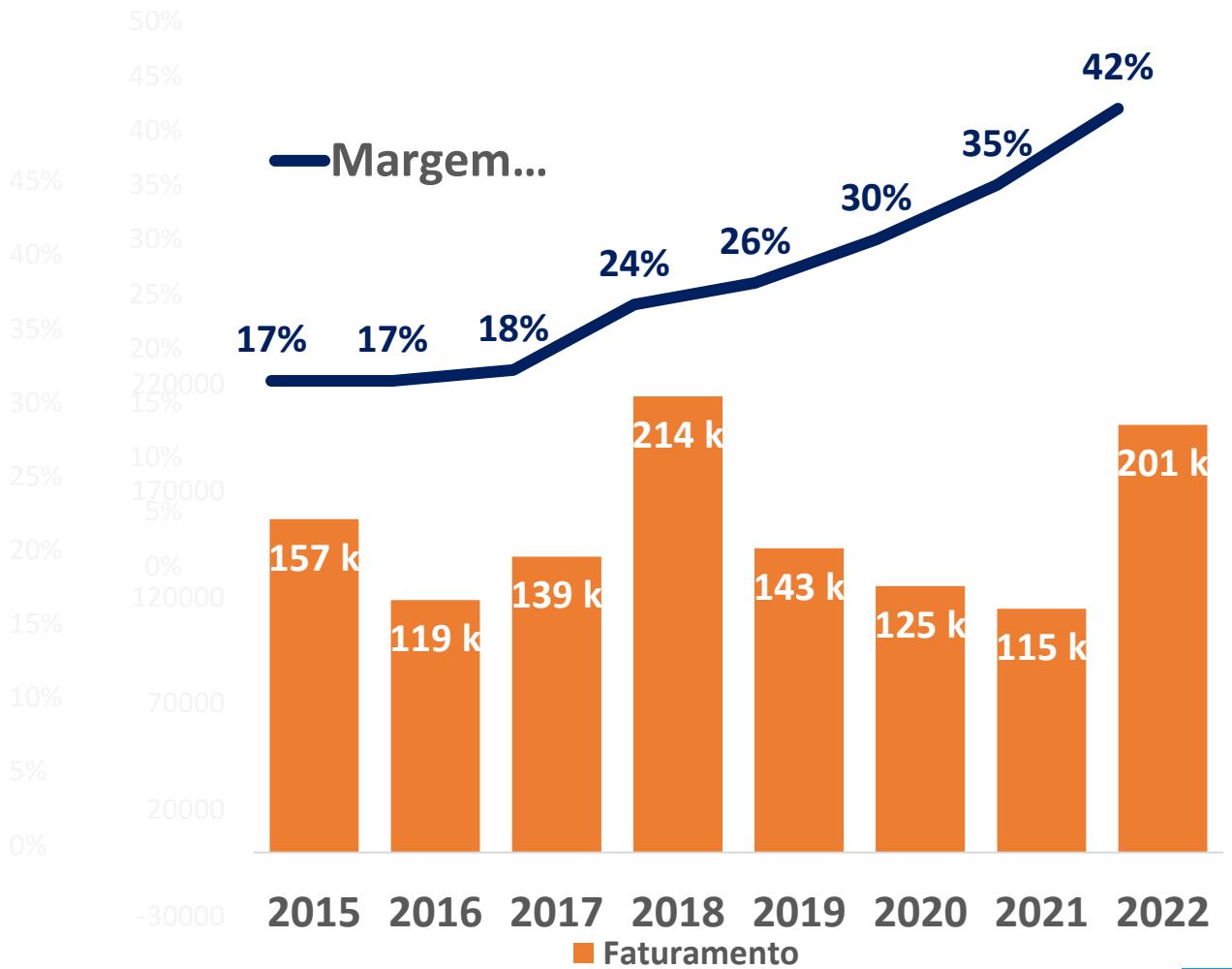
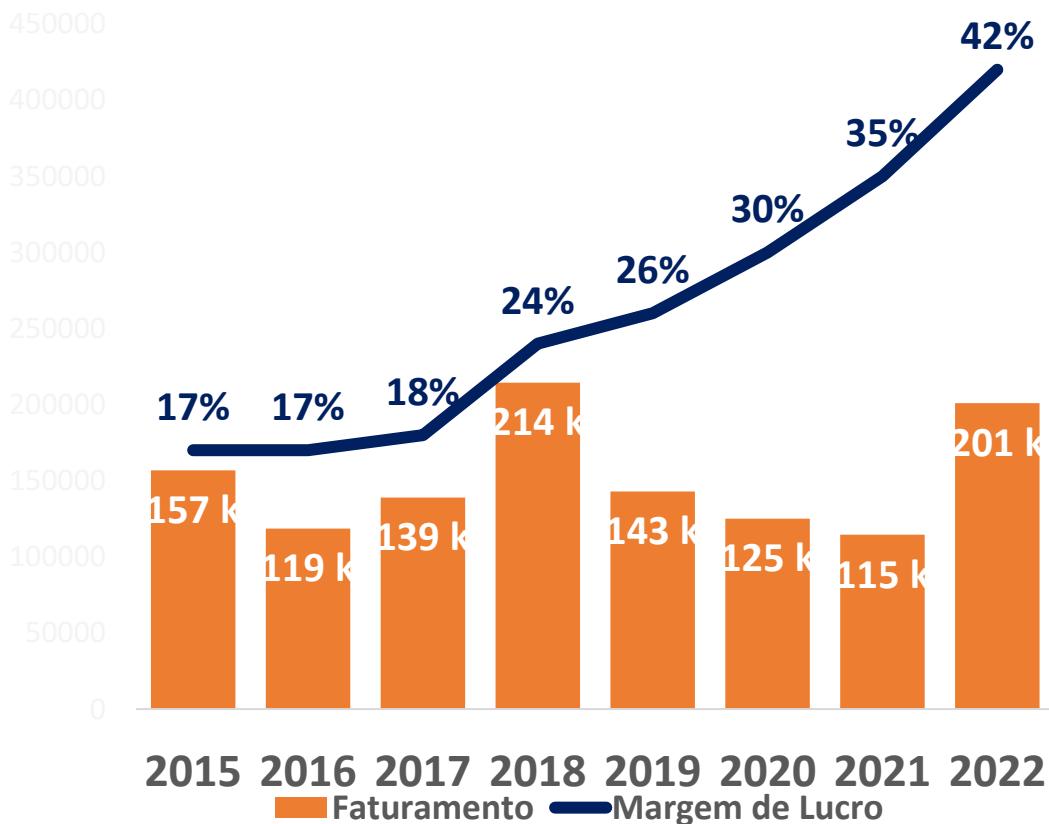
## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Então como poderíamos resolver o problema dos dois eixos?



## Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Ou podemos separar em 2 gráficos:



## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Quando se fala em **melhorar o visual** estamos falando diretamente sobre evitar a saturação, ou seja, tudo o que está no seu gráfico ou imagem que ocupa espaço mas não aumenta o entendimento do leitor.

Por exemplo, ao inserir um texto em um visual que não acrescenta no entendimento para o leitor.

Tudo que estiver na sua **apresentação precisa ter um objetivo** muito claro, você precisa colocar aquilo de forma eficiente.

No próximo slide temos um exemplo de saturações no gráfico.

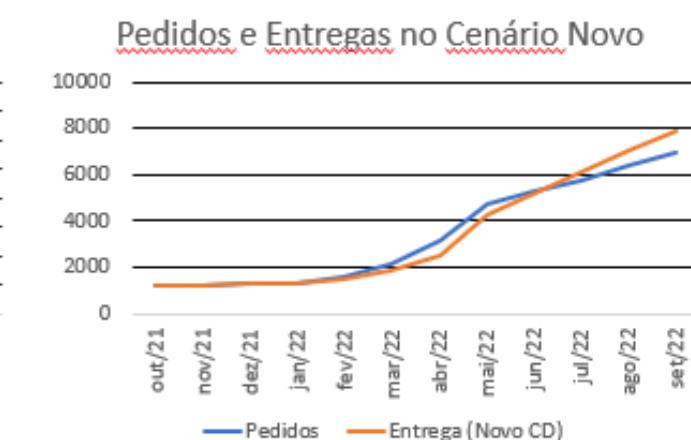
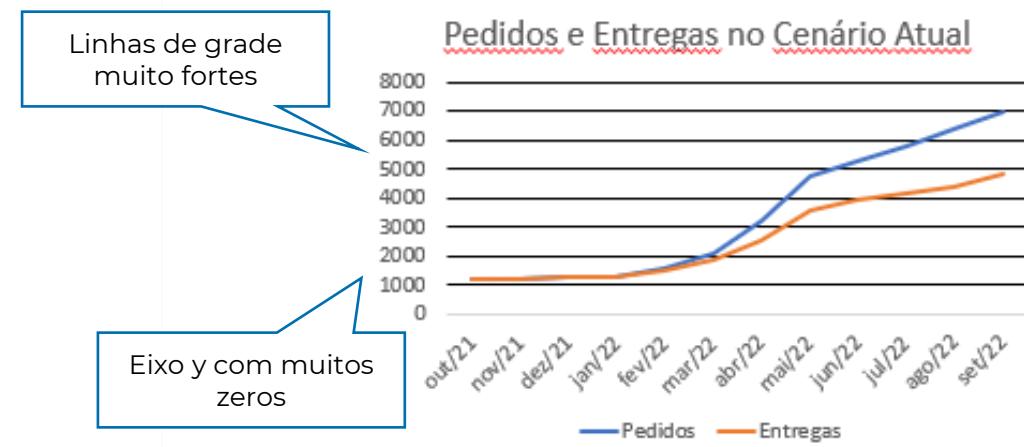
## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

**Saturação** são elementos visuais que **ocupam espaço**, mas não aumentam o entendimento. Por exemplo:

A situação é extremamente crítica pois nossas vendas estão crescendo muito e nossa distribuição não consegue acompanhar. Dessa forma a entrega está ficando muito atrasada e a previsão é que os pedidos de abril possam levar muito mais tempo para serem entregues, prejudicando muito os nossos clientes.

Por isso, é fundamental o investimento no novo Centro de Distribuição para resolver esse problema. Com o novo CD a situação será normalizada até setembro!

Muito texto

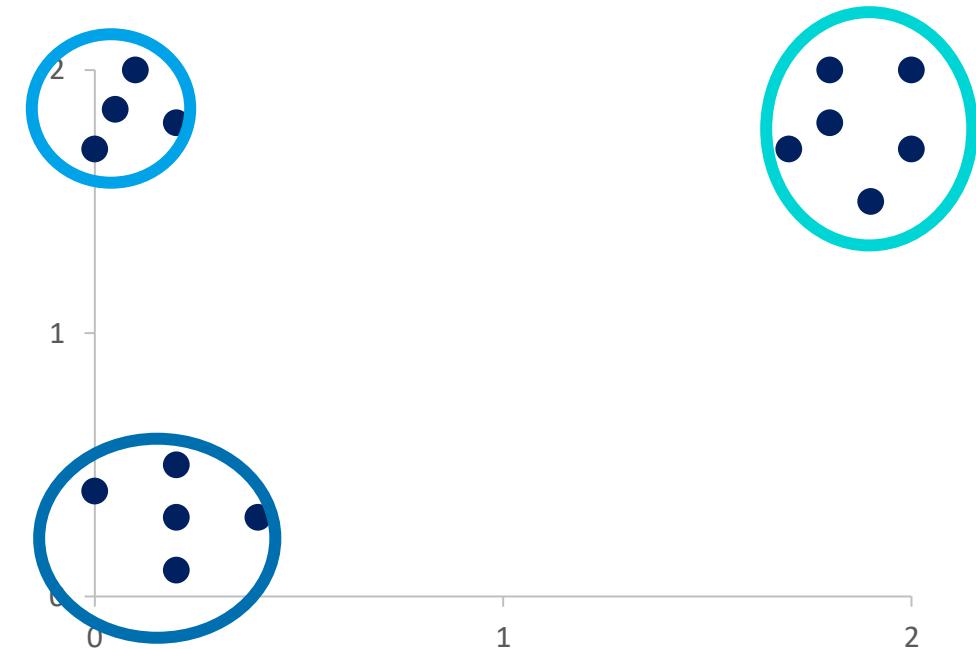


Gráficos com eixos diferentes

## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Existem alguns princípios que são usados para reduzir essa saturação e melhorar a visualização do que está sendo apresentado.

O princípio de **proximidade** diz que o nosso cérebro entende **objetos próximos como parte de um grupo**, por exemplo, quando se fala de cluster de cliente ou quando eu quero separar entre tipos de planta.



## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

O mesmo se aplica à tabela, quando aumentamos o espaçamento entre as colunas, conseguimos focar mais nas colunas e essa é uma forma de direcionar o usuário para que ele leia as informações da forma que queremos.



Produto	Mês	Venda \$	Venda Qtd	Produto	Mês	Venda \$	Venda Qtd
A	Jan	2546	37	A	Jan	2546	37
A	Fev	2027	22	A	Fev	2027	22
A	Mar	2470	31	A	Mar	2470	31
B	Jan	1787	21	B	Jan	1787	21
B	Fev	2664	44	B	Fev	2664	44
B	Mar	2288	46	B	Mar	2288	46
C	Jan	2531	30	C	Jan	2531	30
C	Fev	2173	42	C	Fev	2173	42
C	Mar	2660	38	C	Mar	2660	38

## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Agora, por outro lado, queremos focar nas linhas. O leitor primeiro vai ler as informações do produto A, depois o B e por último o C.

Para fazer essa lógica basta **aumentarmos o espaçoamento das linhas**.

Além disso, a proximidade entre o A, B e o C faz com que o leitor siga essa lógica.

Produto	Mês	Venda \$	Venda Qtd
---------	-----	----------	-----------

A	Jan	2546	37
A	Fev	2027	22
A	Mar	2470	31

B	Jan	1787	21
B	Fev	2664	44
B	Mar	2288	46

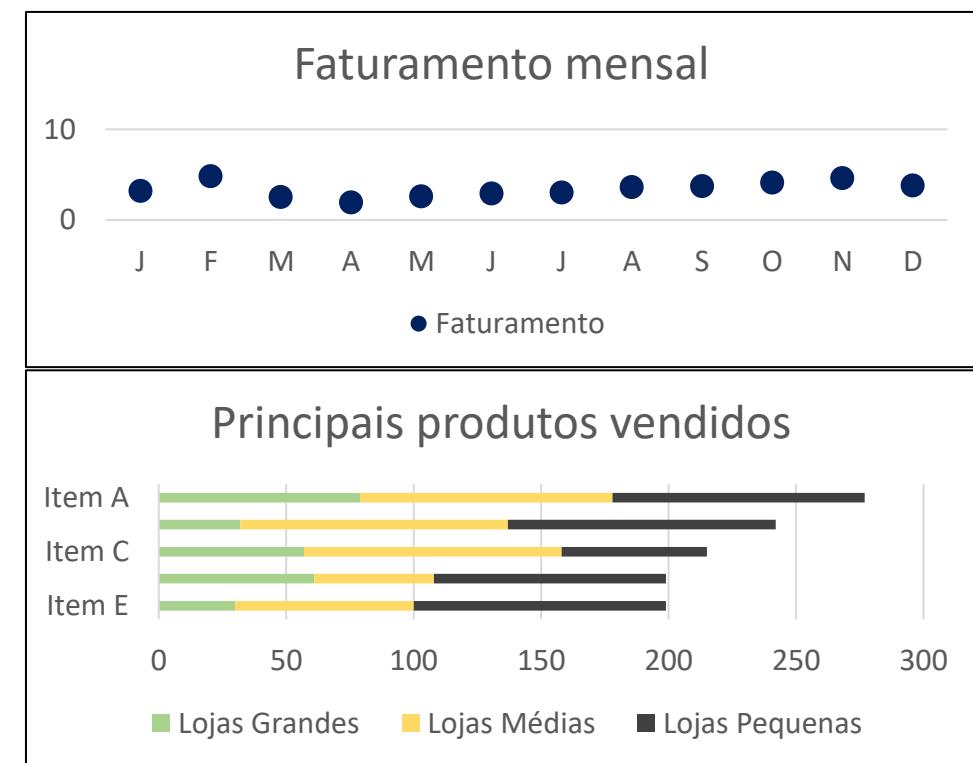
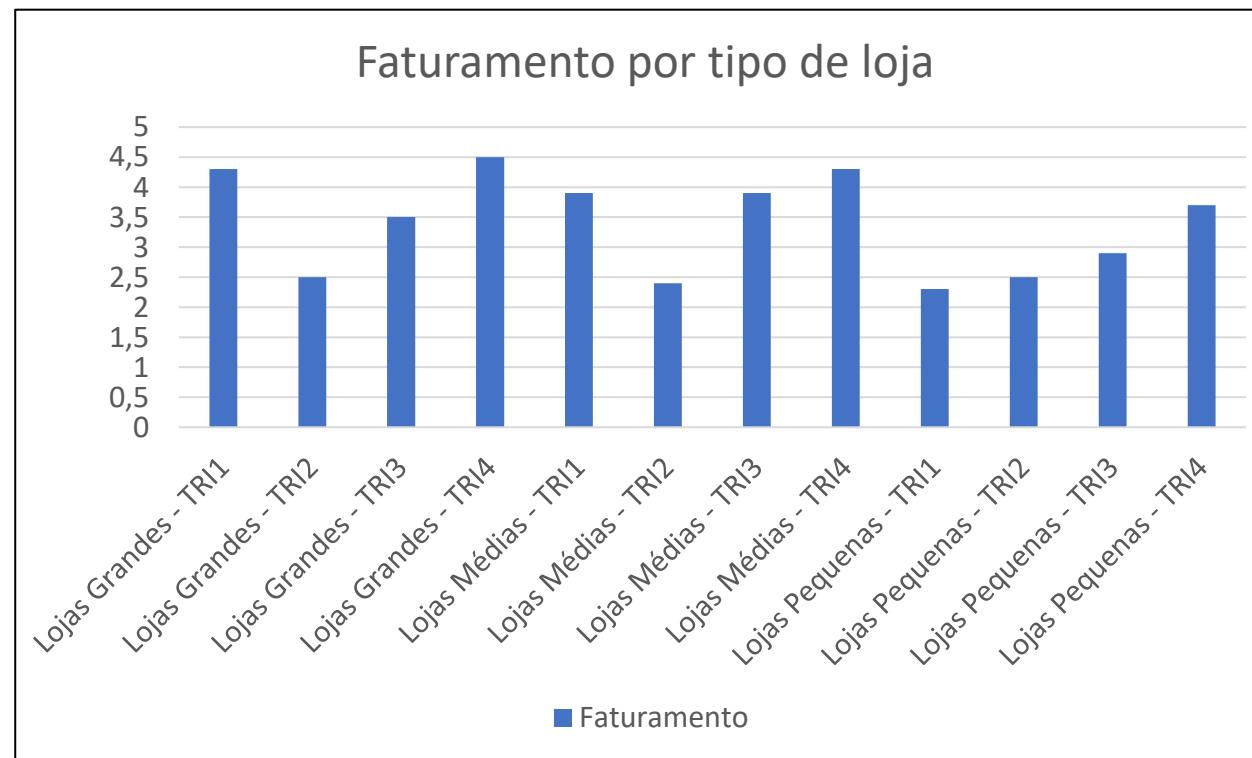
C	Jan	2531	30
C	Fev	2173	42
C	Mar	2660	38

Produto	Mês	Venda \$	Venda Qtd
---------	-----	----------	-----------

A	Jan	2546	37
A	Fev	2027	22
A	Mar	2470	31
B	Jan	1787	21
B	Fev	2664	44
B	Mar	2288	46
C	Jan	2531	30
C	Fev	2173	42
C	Mar	2660	38

## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Entendemos objetos fisicamente próximos como parte de um grupo.



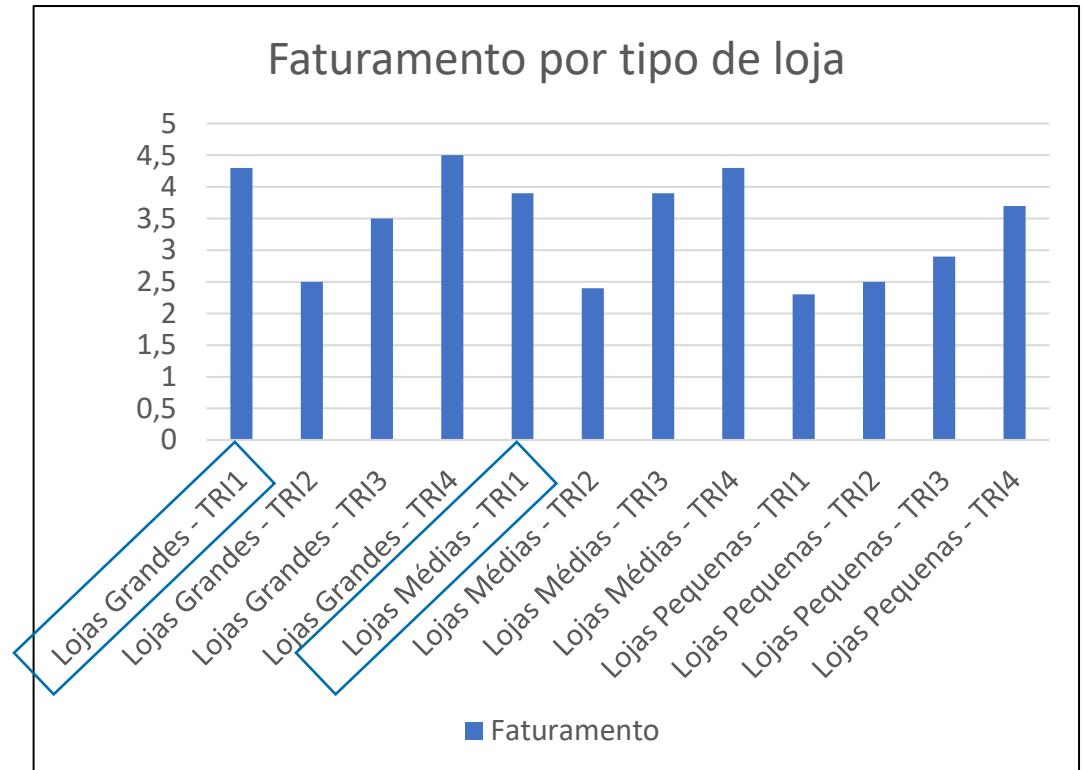
## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Vamos utilizar esse princípio para melhorar o gráfico ao lado.

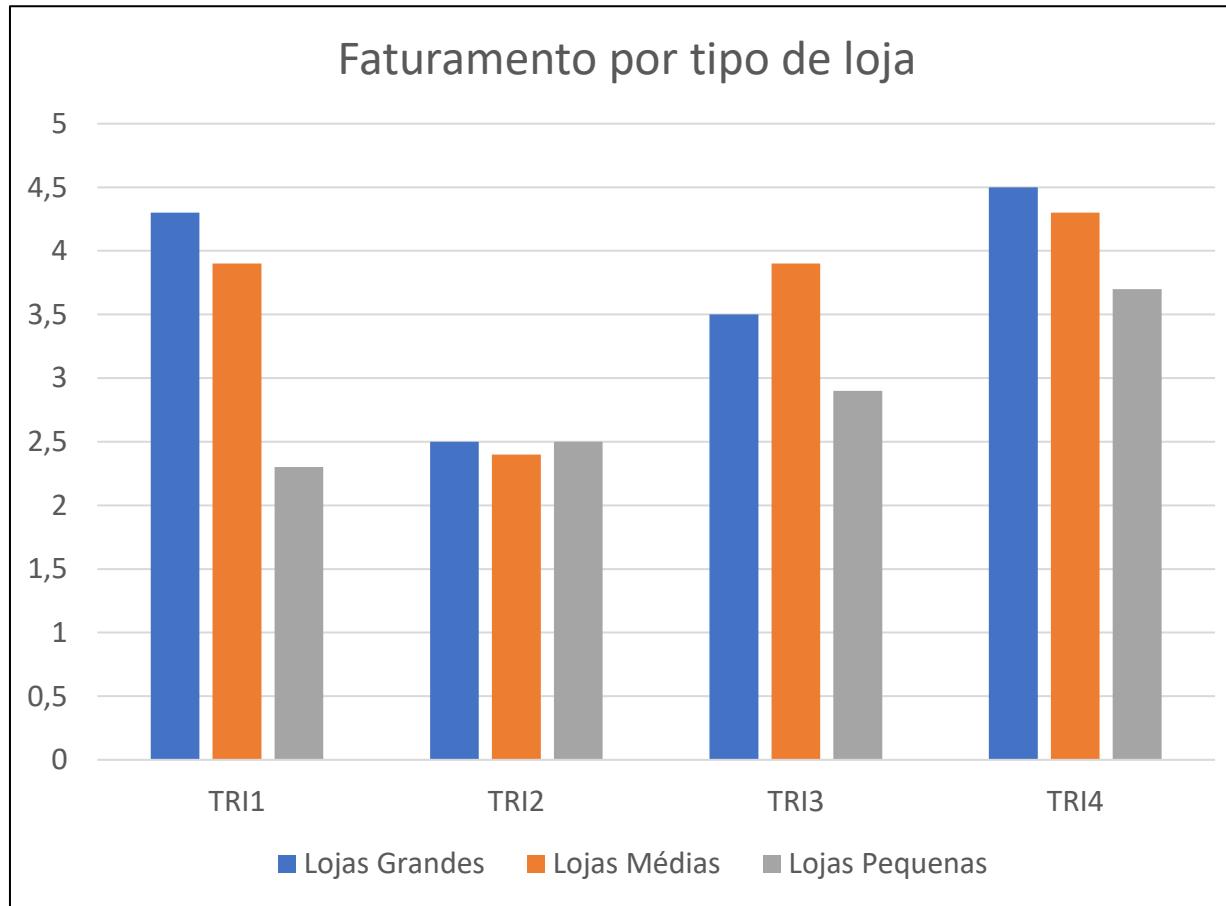
Observem que no eixo x, as informações estão por cada loja e em cada trimestre.

Porém, não é favorável para a leitura ler o 4º Trimestre da Loja grande e em seguida ler o 1º da Loja Média.

Além de não ter uma separação por cor para lojas diferentes.



## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)



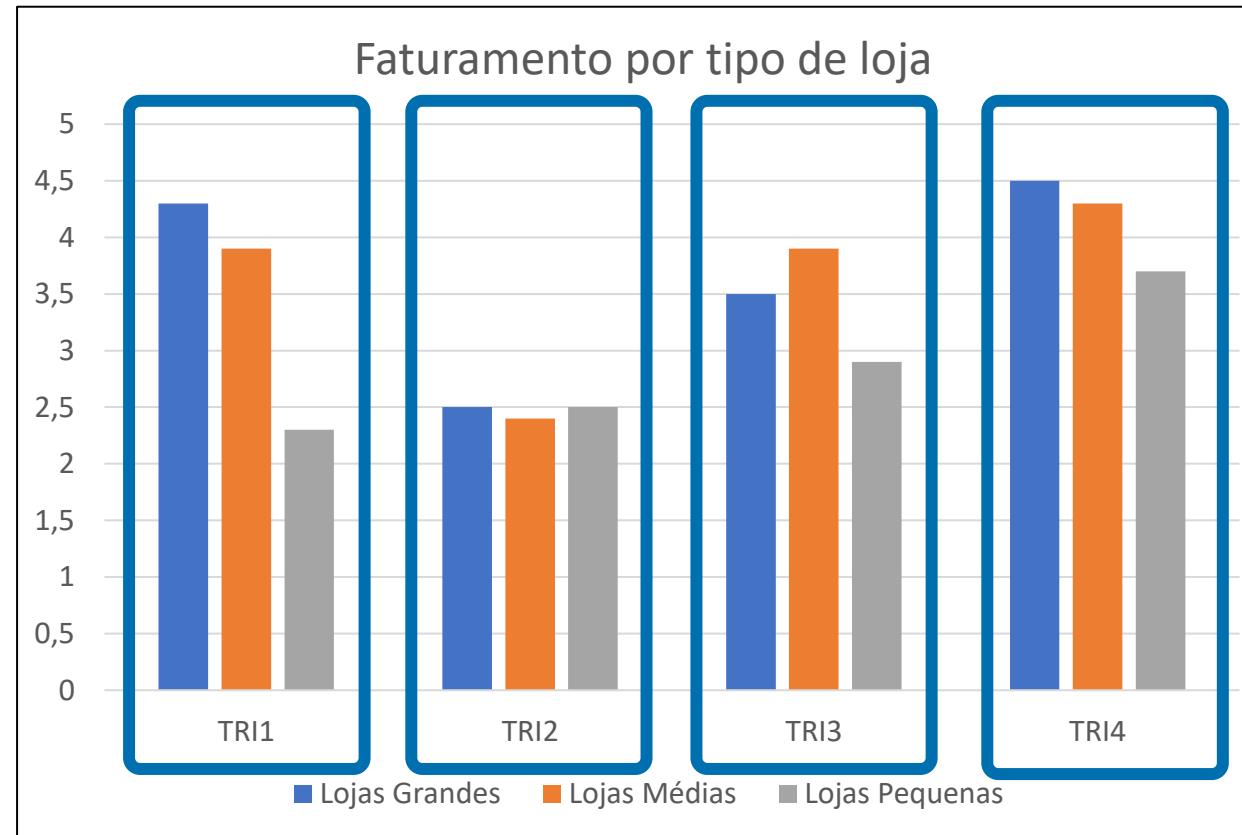
Utilizando o princípio da **proximidade** agora temos uma linha temporal: trimestres 1, 2 e 3.

As informações foram **aproximadas** e colocadas com uma **lógica temporal** entre elas.

Dessa forma fica **mais fácil de entender o gráfico** e com uma visualização mais amigável.

## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Uma outra forma de utilizar esse princípio é agrupando pelas lojas

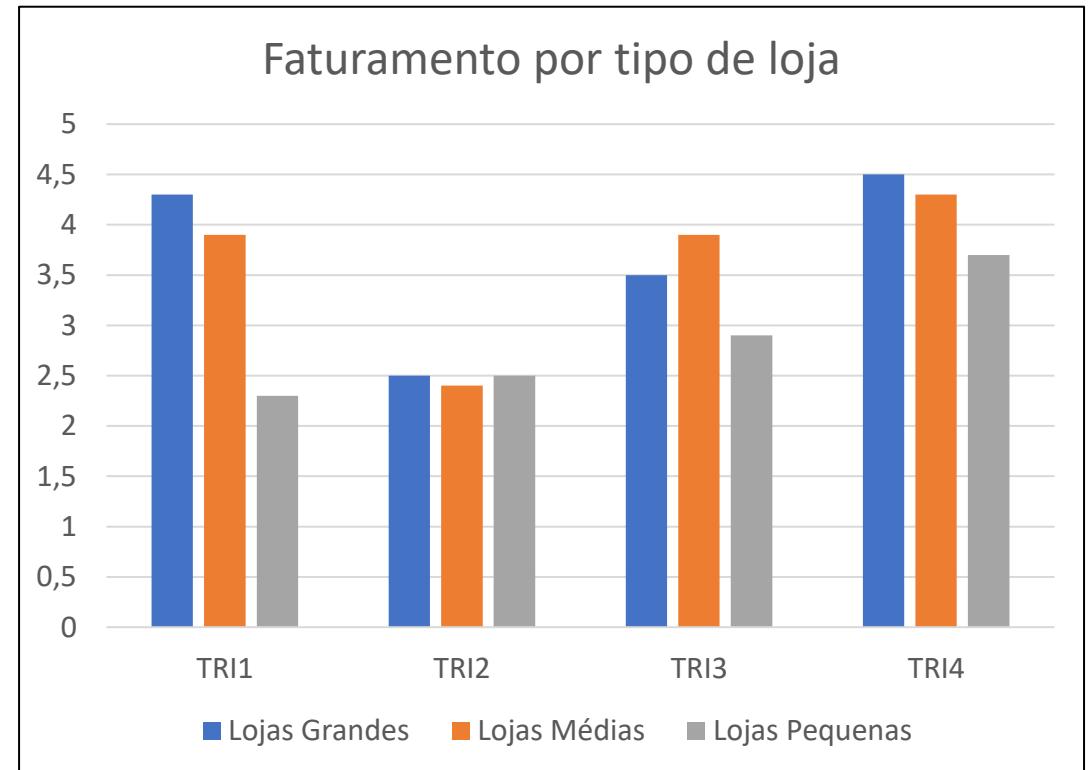


## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

A forma como escolher e separar depende do que queremos mostrar.

Por exemplo, se o foco é evolução no tempo, então provavelmente o uso da linha temporal por trimestre é a melhor opção.

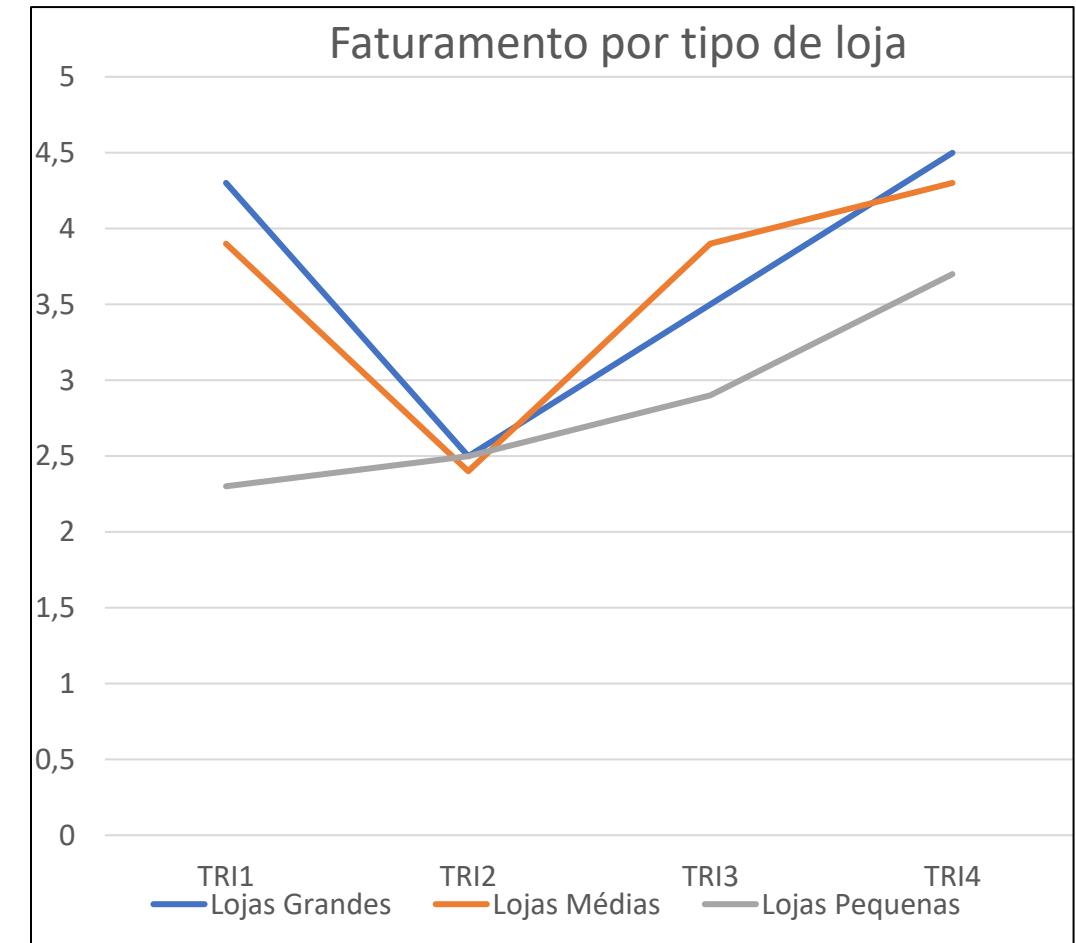
Por outro lado, se o foco é nas lojas, o agrupamento por lojas é a melhor opção.



## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Se o objetivo é **mostrar a evolução**, a melhor forma é a separação por loja utilizando **gráfico de linhas**.

Faça isso de forma consciente e use esse princípio para mostrar o seu objetivo com uma visualização mais amigável.

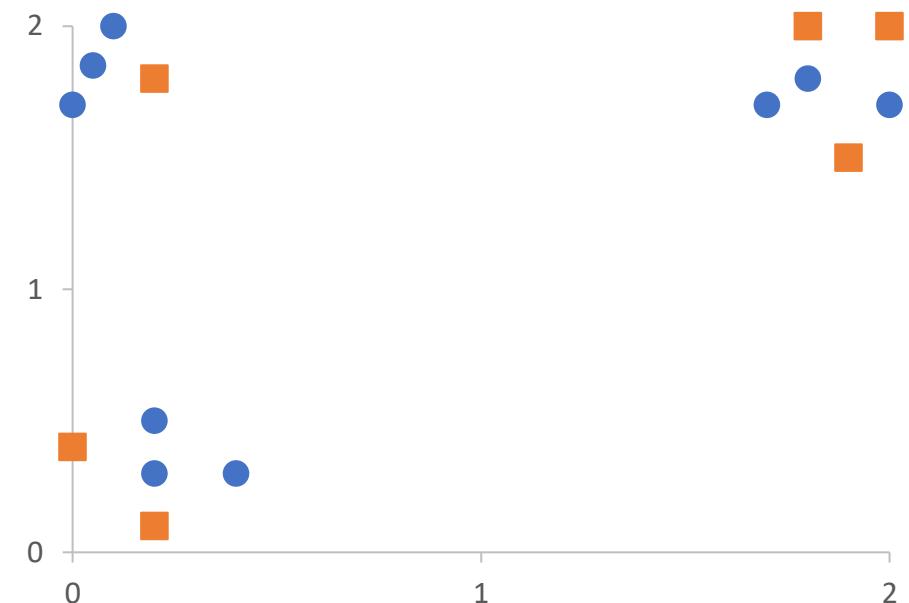


## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Objetos com a mesma cor, tamanho e forma são percebidos como relacionados.

Por mais que os pontos estejam próximos, quando usamos esse princípio da **similaridade**, o nosso cérebro entende que existe uma hierarquia.

Primeiro observamos os **similares** e depois os que estão próximos. Portanto, quando a gente observa o gráfico ao lado, entendemos que o ponto vermelho é uma coisa e o azul outra.



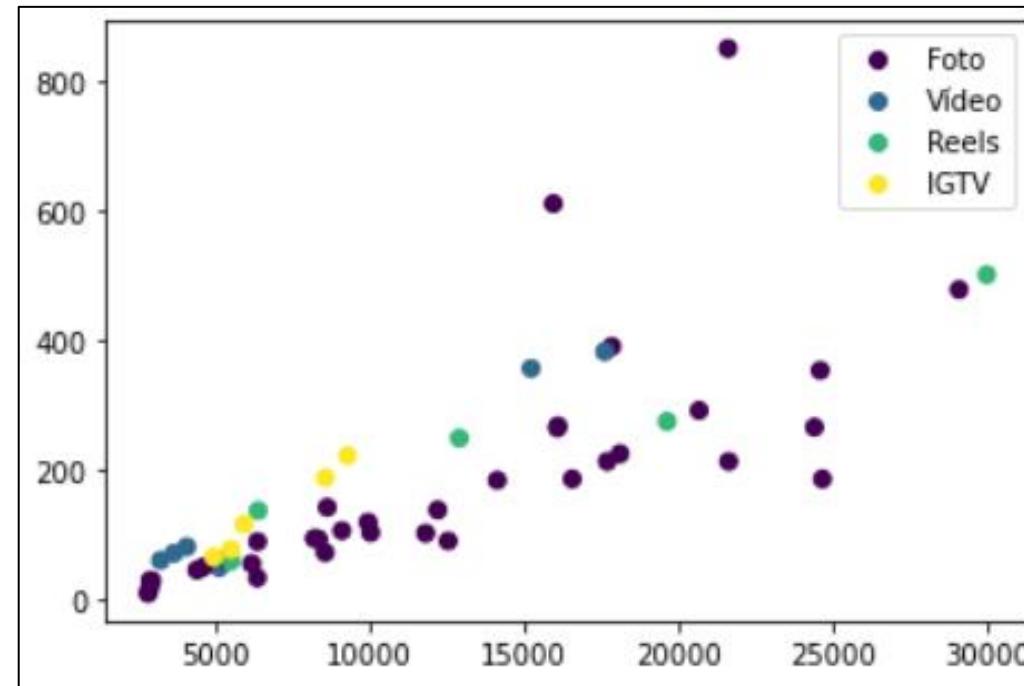
## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Observe como ao organizar as linhas dessa forma fica muito mais fácil de entender e mais confortável para o usuário realizar a leitura.

Produto	Mês	Venda \$	Venda Qtd
A	Jan	2546	37
A	Fev	2027	22
A	Mar	2470	31
B	Jan	1787	21
B	Fev	2664	44
B	Mar	2288	46
C	Jan	2531	30
C	Fev	2173	42
C	Mar	2660	38

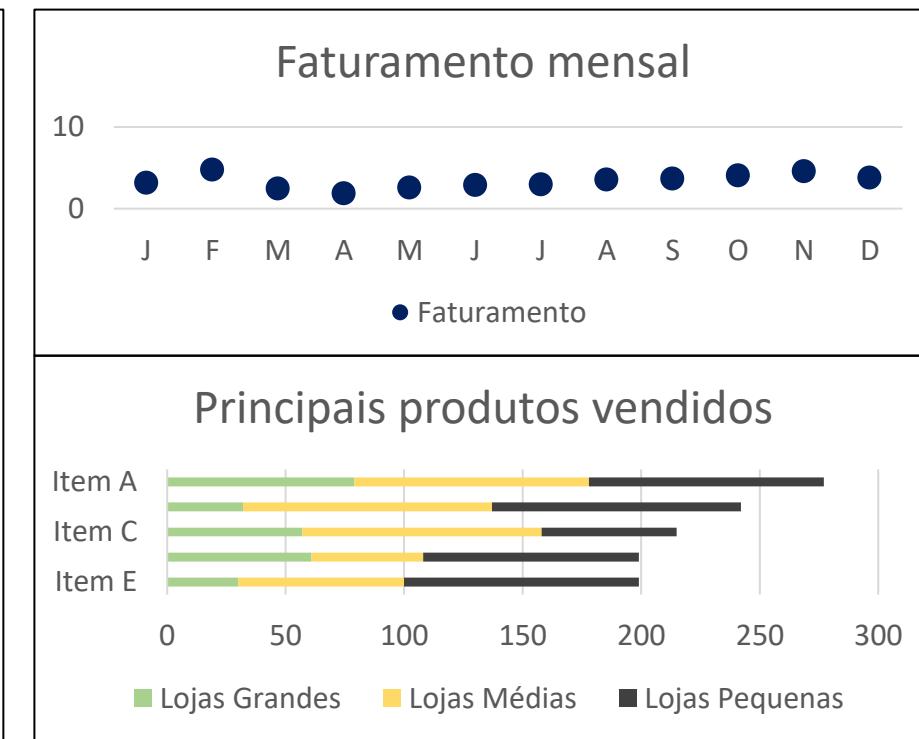
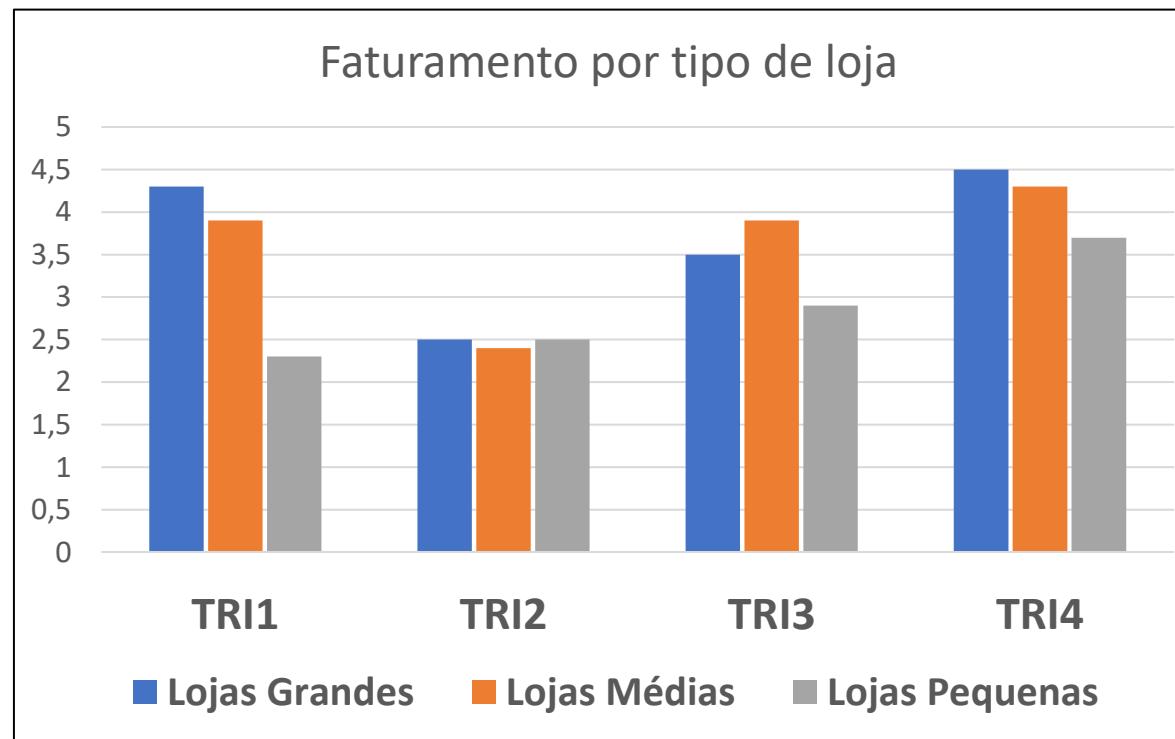
## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

No gráfico abaixo colocamos **cores diferentes para tipos distintos** de mídia e dessa forma **facilita a leitura do gráfico**.



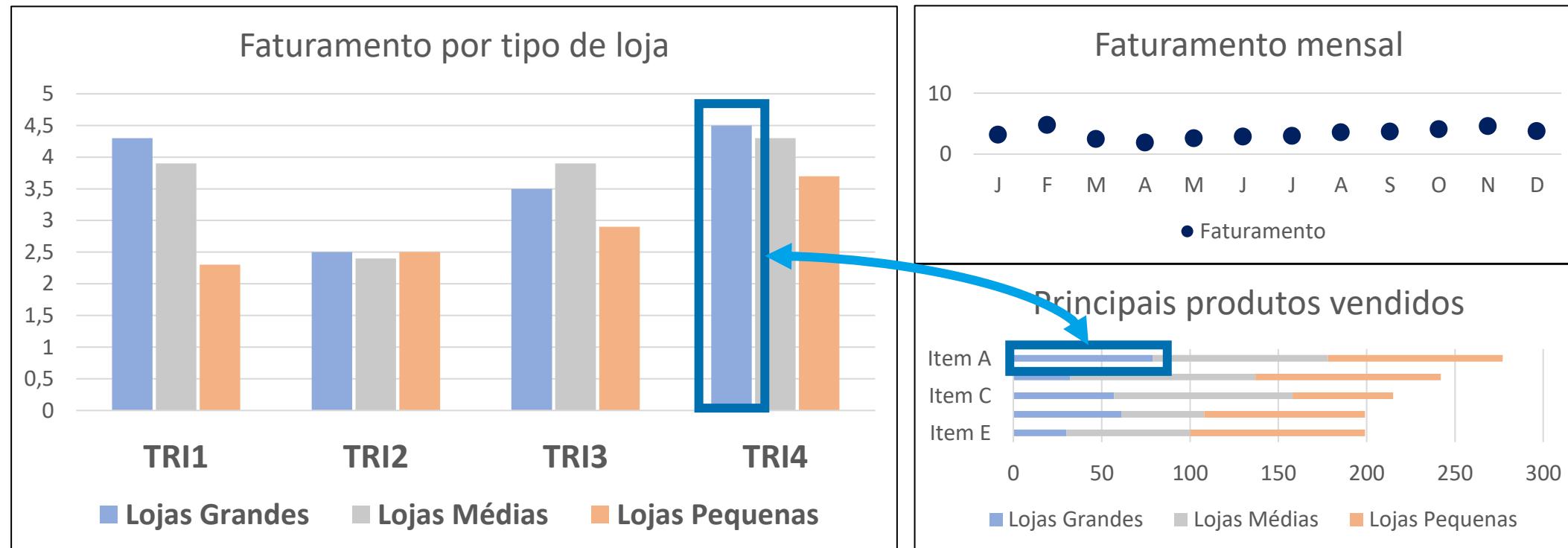
## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Observe que no visual abaixo temos 6 cores para 3 lojas.



## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Uma forma de melhorar a visualização é **padronizar as cores para as lojas** nos gráficos.



## Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Com isso, nós utilizamos 2 princípios muito importantes para melhorar o visual do gráfico:

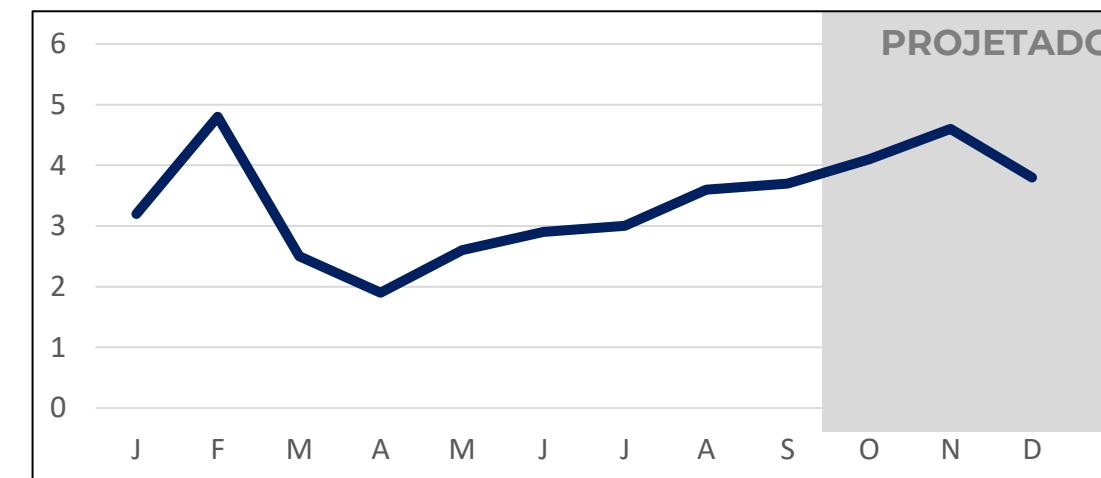
**Proximidade**: entendemos **objetos fisicamente próximos** como parte de um grupo.

**Similaridade**: objetos com a **mesma cor, tamanho, forma** são percebidos como relacionados.

Além de notar que existe uma hierarquia em primeiro vemos a similaridade e depois a proximidade.

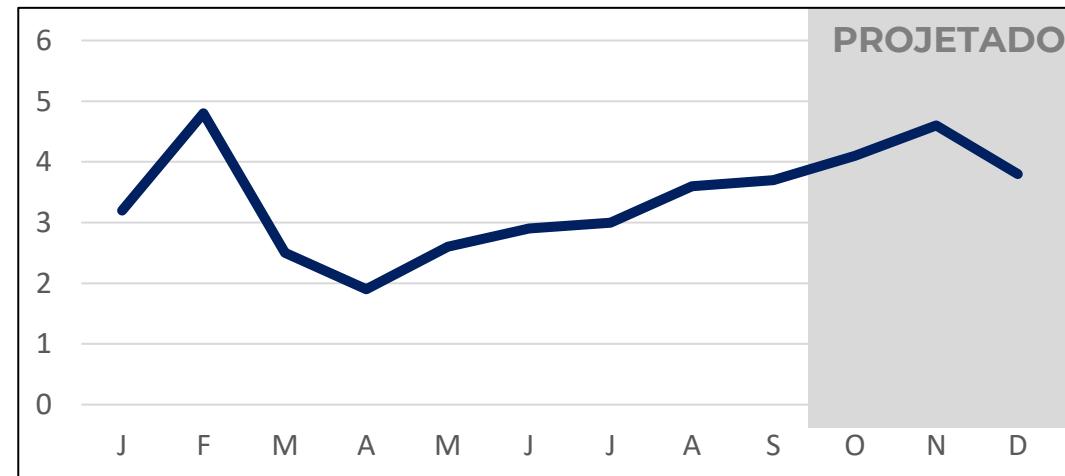
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Um outro princípio é o acercamento que entende objetos fisicamente delimitados como fazendo parte de um grupo.



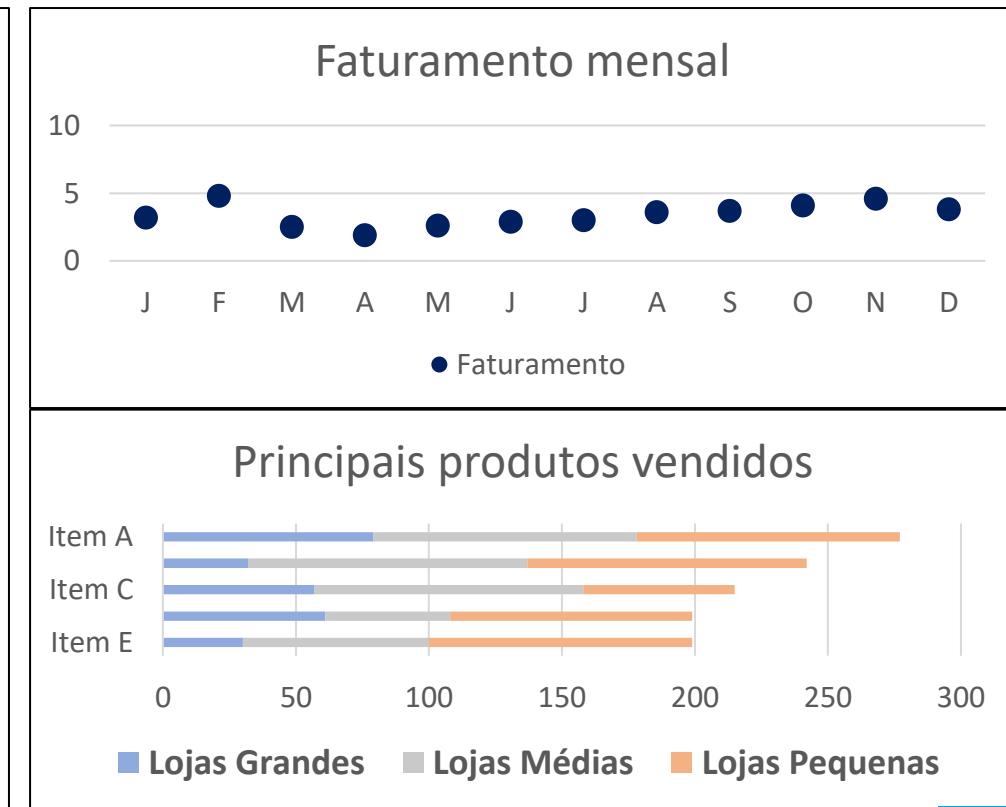
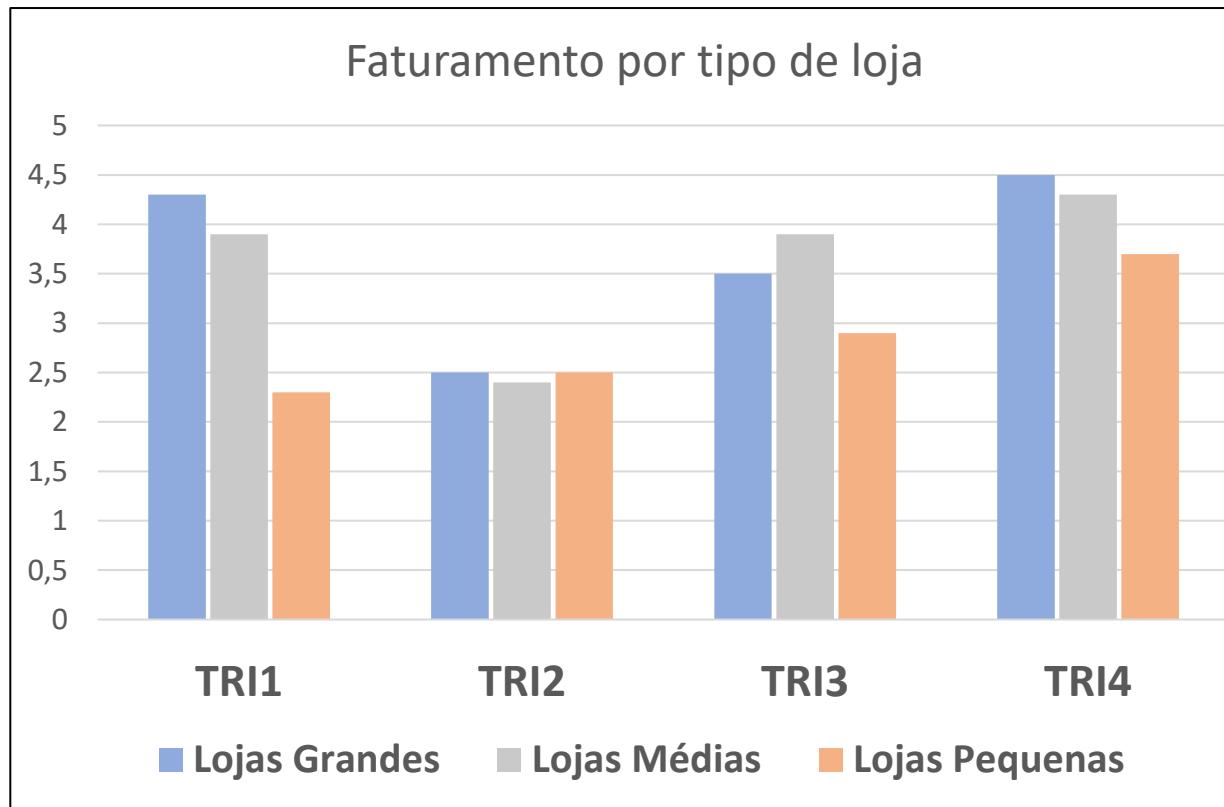
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Utilizamos esse princípio para dar ênfase em alguma área, por exemplo, separar o realizado do projetado.



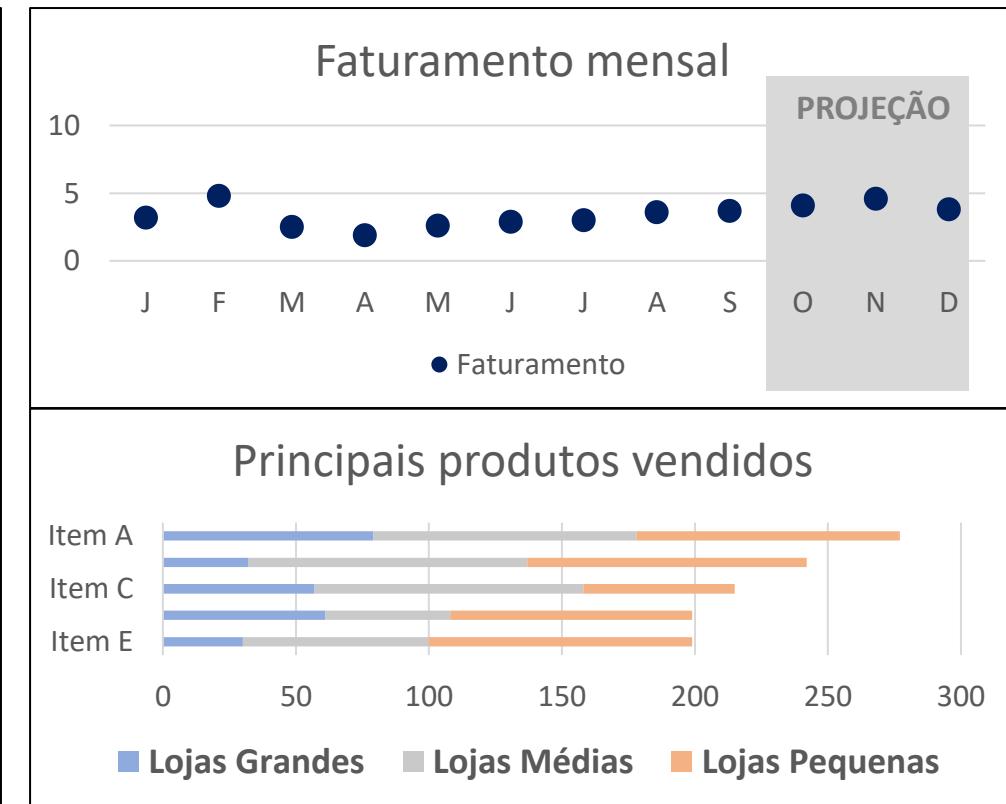
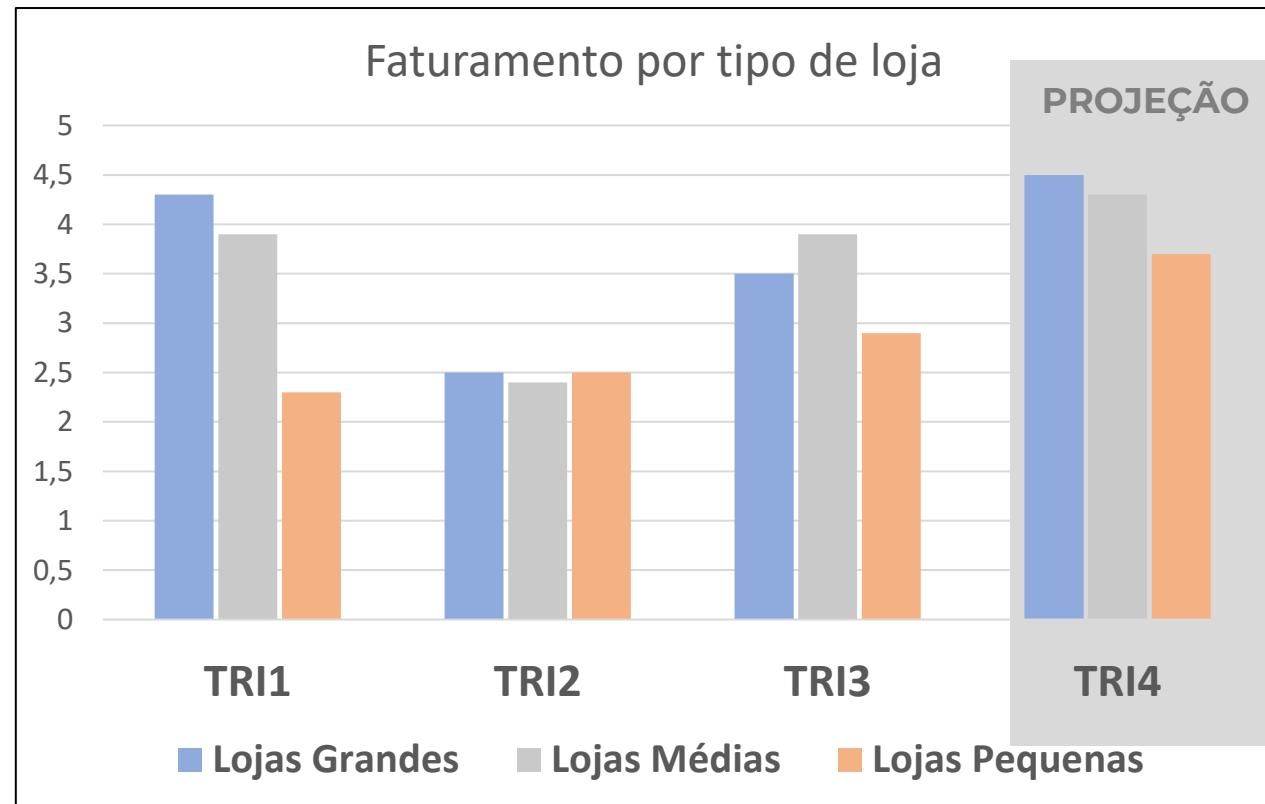
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Como que poderíamos usar o acercamento nesse gráfico?



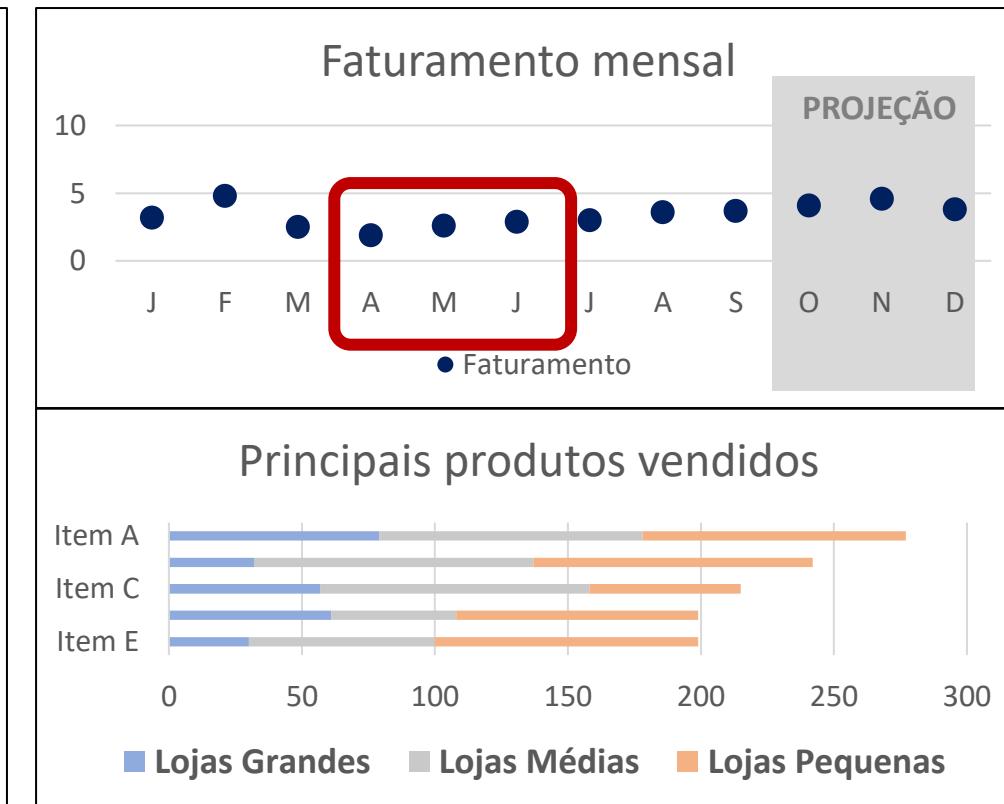
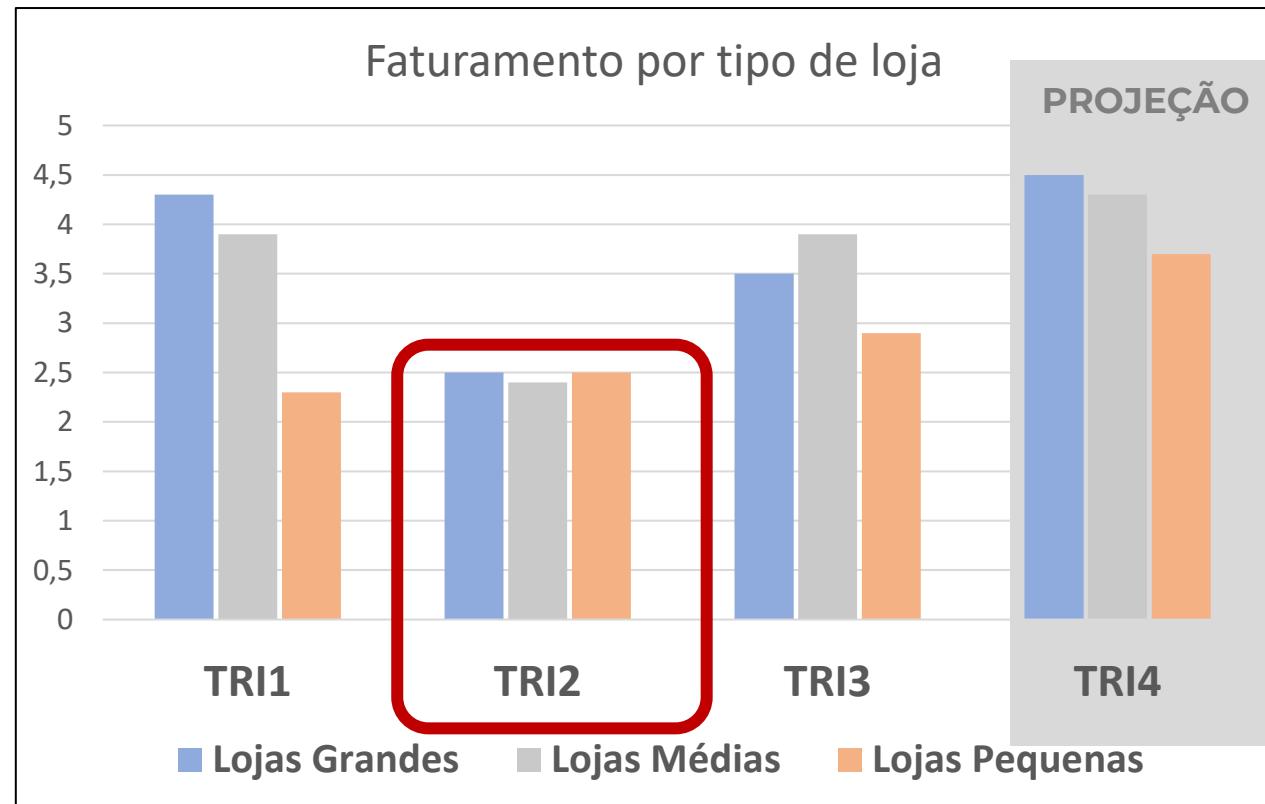
## Módulo 8 – Melhorando o seu visual (Acerramento, Fechamento, Continuidade e Conexão)

Podemos considerar que o 4º trimestre ainda não finalizou



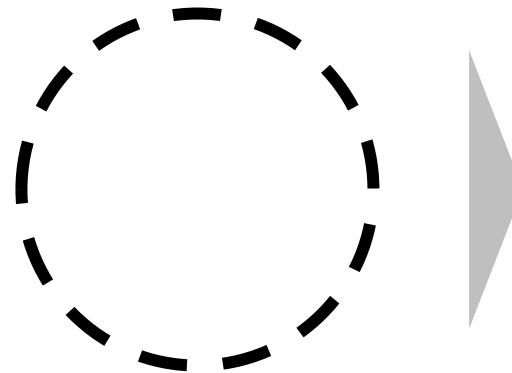
## Módulo 8 – Melhorando o seu visual (Acerramento, Fechamento, Continuidade e Conexão)

E se quisermos destacar o trimestre com menor faturamento?



## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

O princípio do **fechamento** diz que quando partes de um todo estão faltando, nossos **olhos preenchem as lacunas**

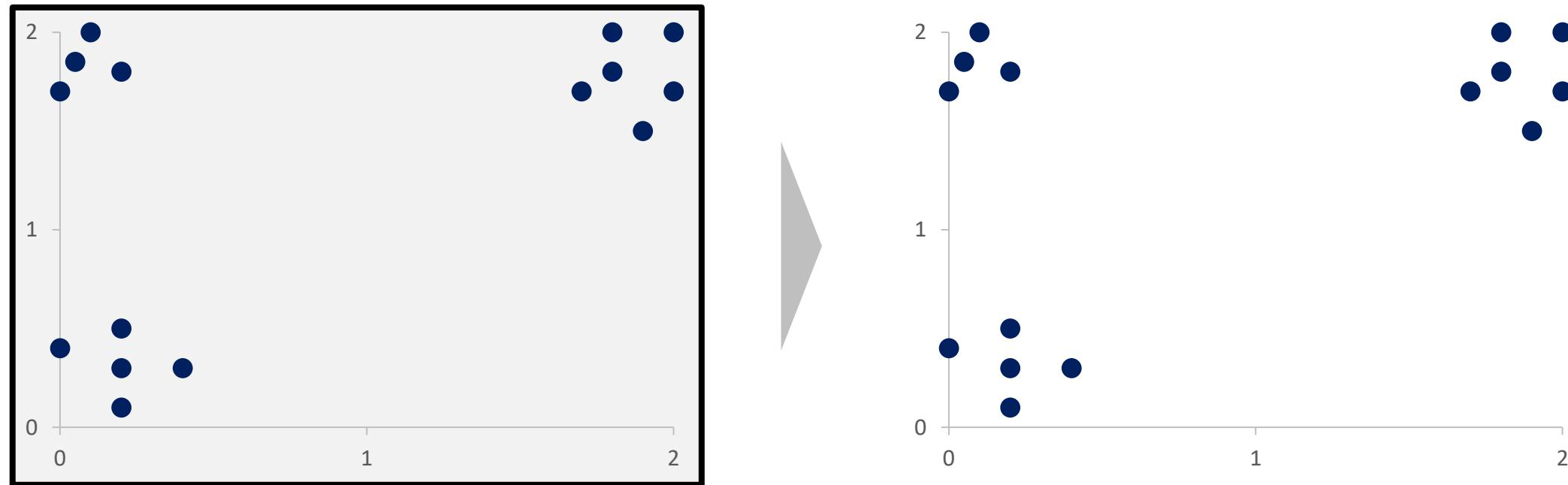


"Nosso cérebro procura simplificar as coisas e tenta **relacionar a algo que já conhecemos**. Tendemos a perceber um conjunto de elementos individuais como uma única forma reconhecível"



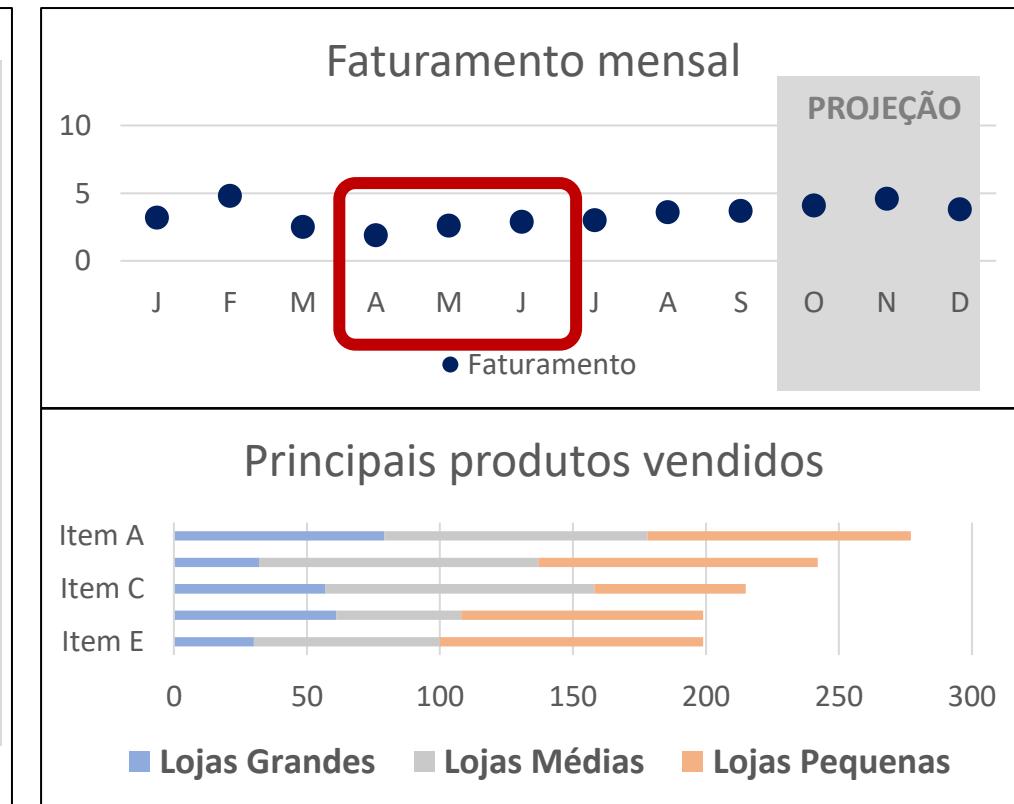
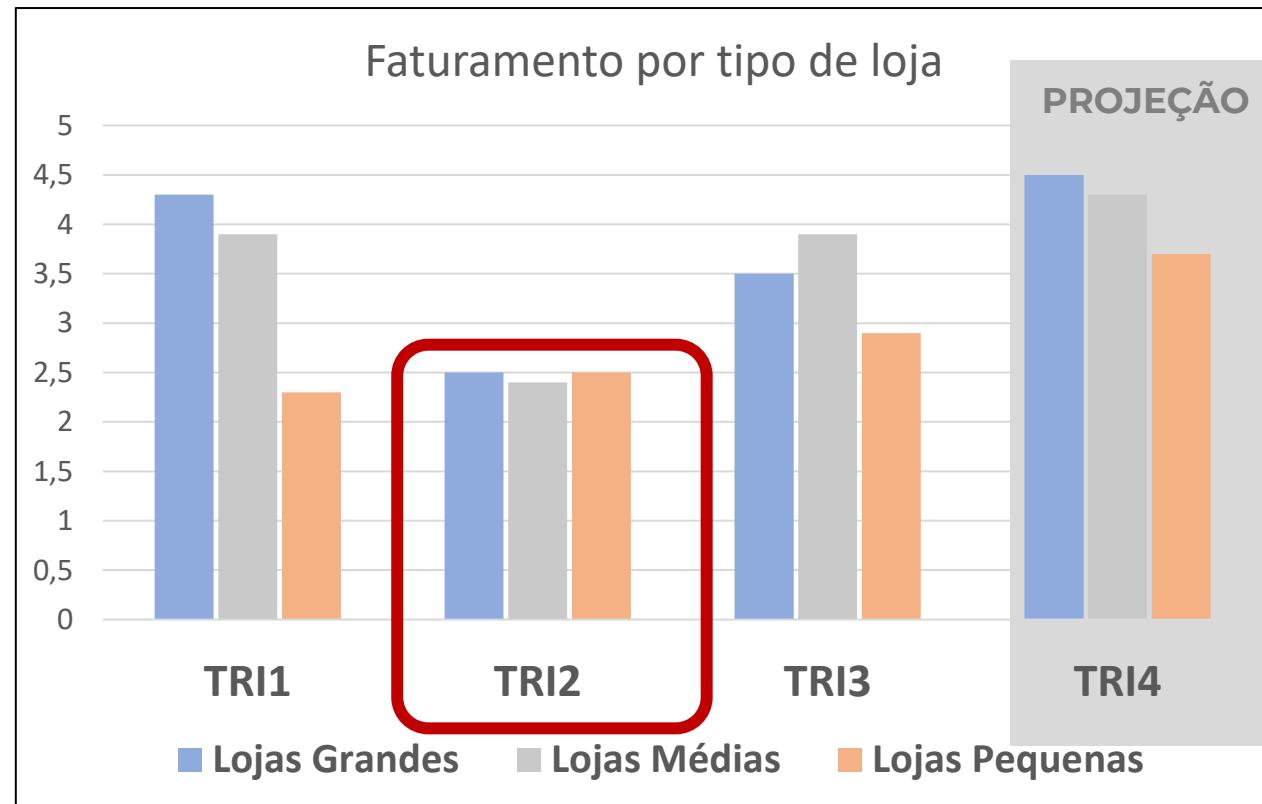
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Vamos usar esse princípio para tirar alguns detalhes, por exemplo a borda.



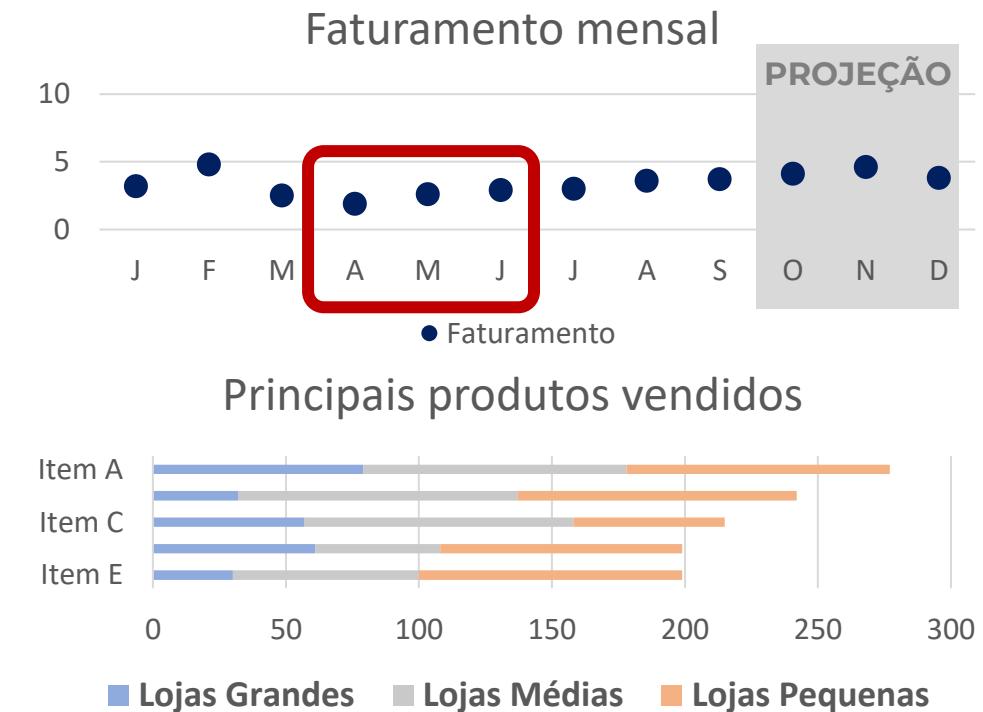
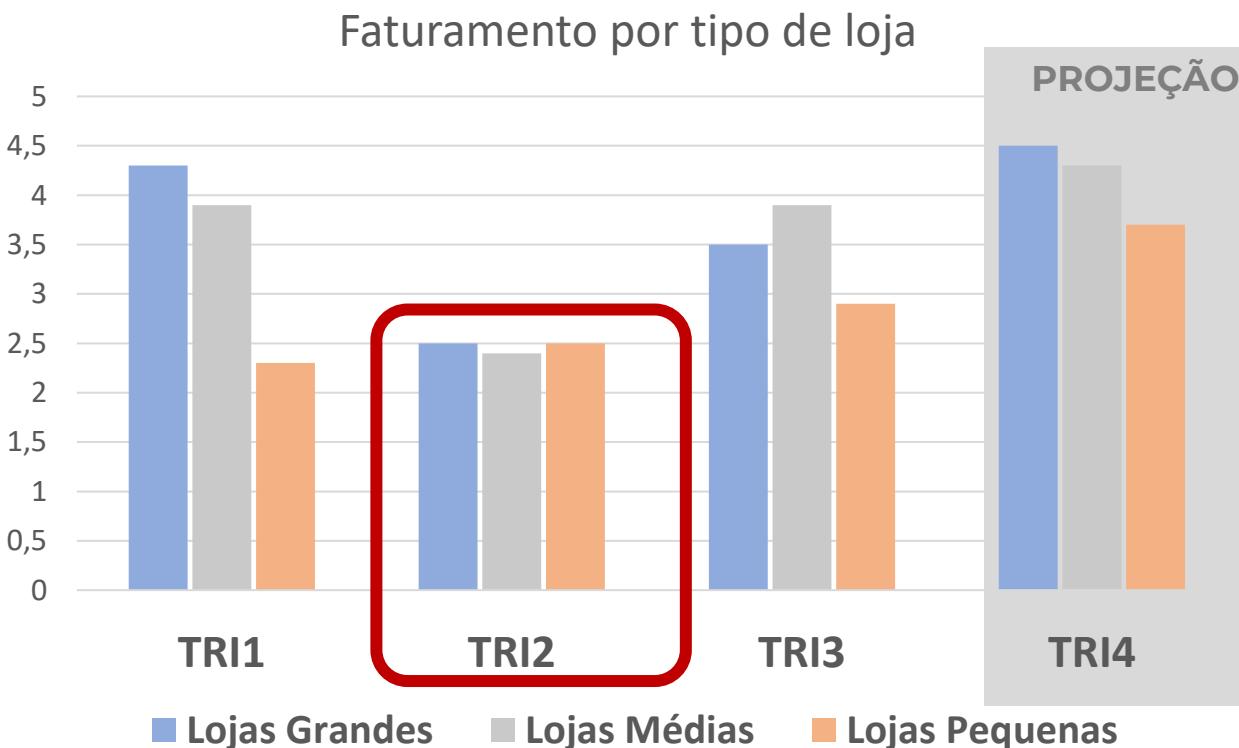
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Como a gente poderia melhorar essa nossa apresentação com o fechamento?



## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

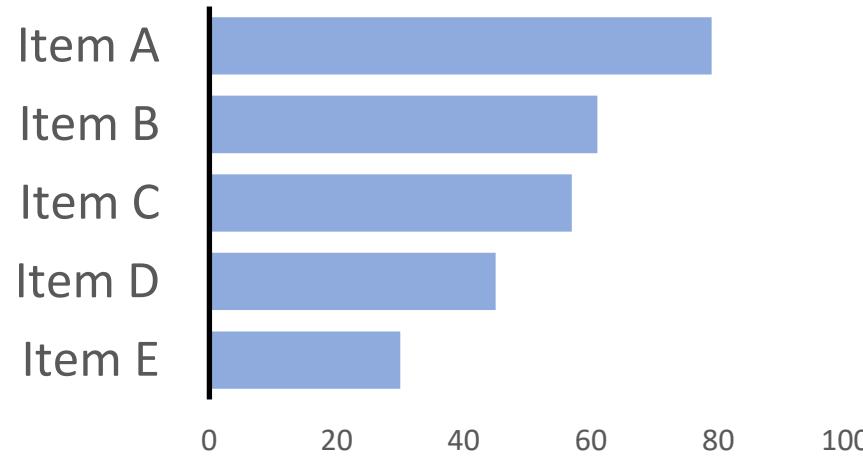
Retirando as bordas, o gráfico fica mais amigável.



## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

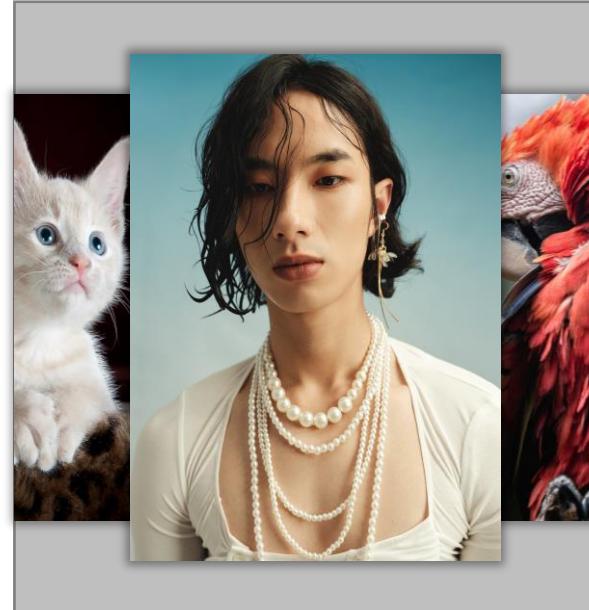
Outro princípio que podemos utilizar para reduzir a saturação é a continuidade.

O princípio da **continuidade** diz que os nossos olhos vão criar **continuidade mesmo que ela não esteja explícita**.



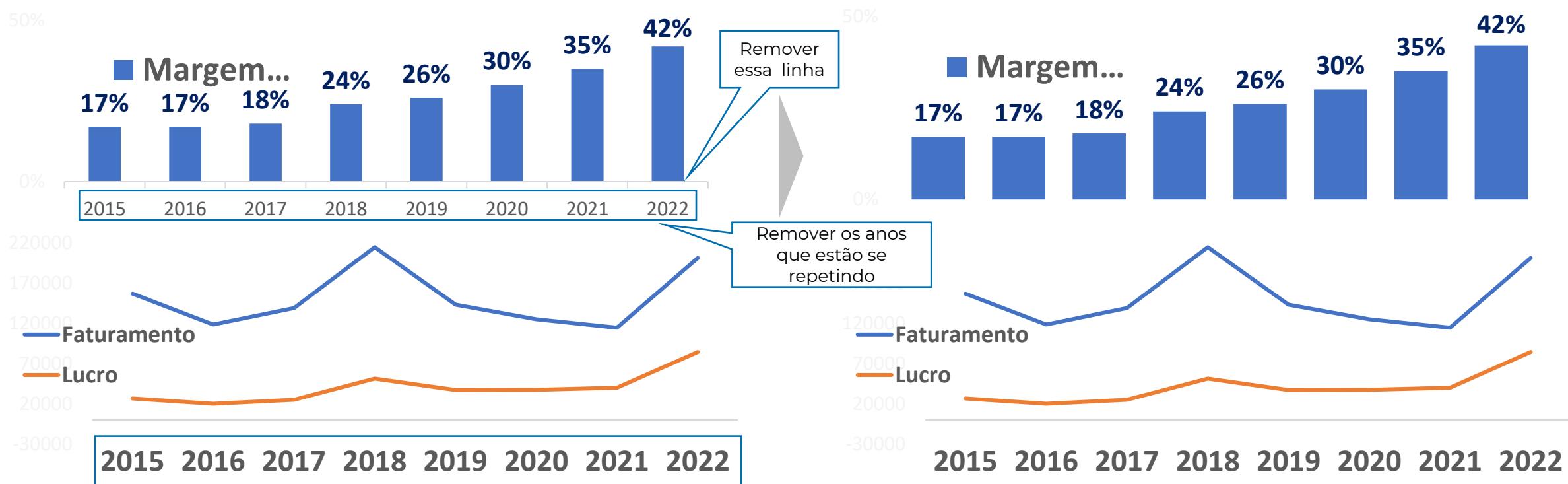
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Esse princípio é muito utilizado em aplicativos, filmes e sites de catálogos de produtos



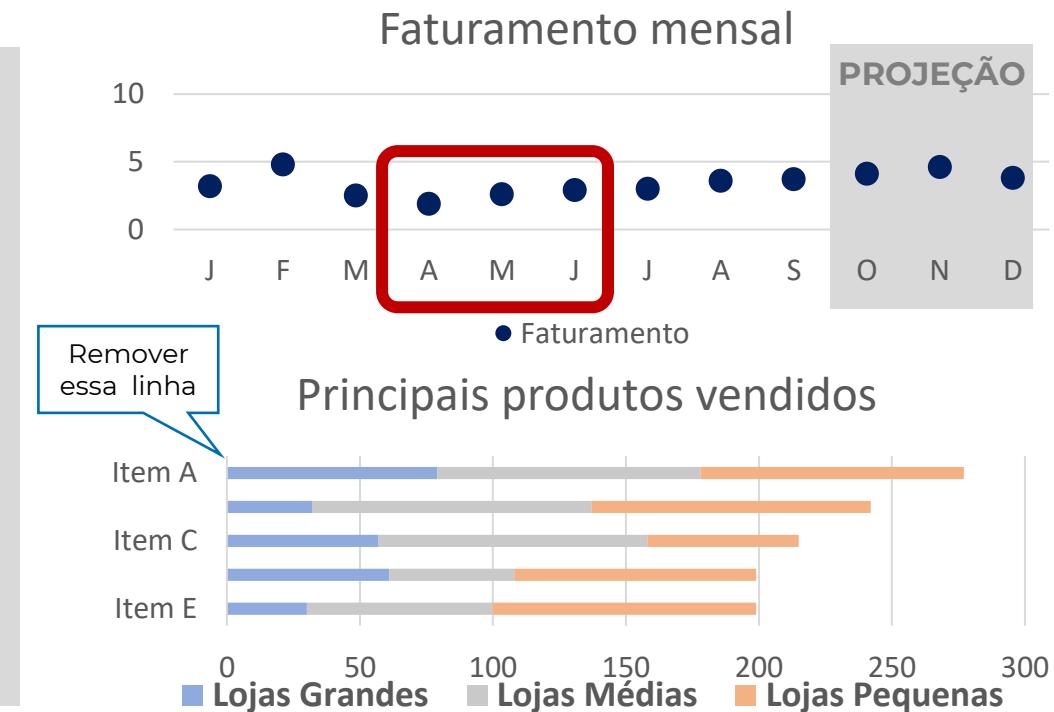
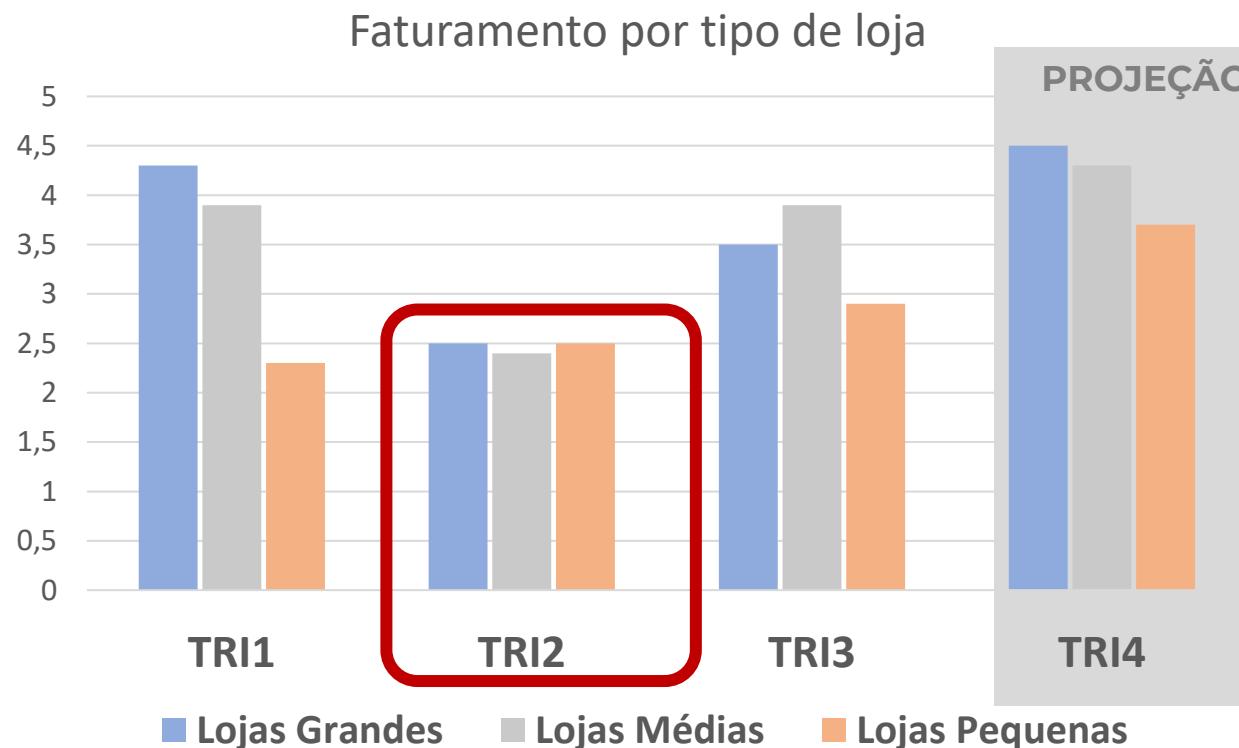
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Pensando em como utilizar isso em nossas apresentações, vamos ver o exemplo abaixo:



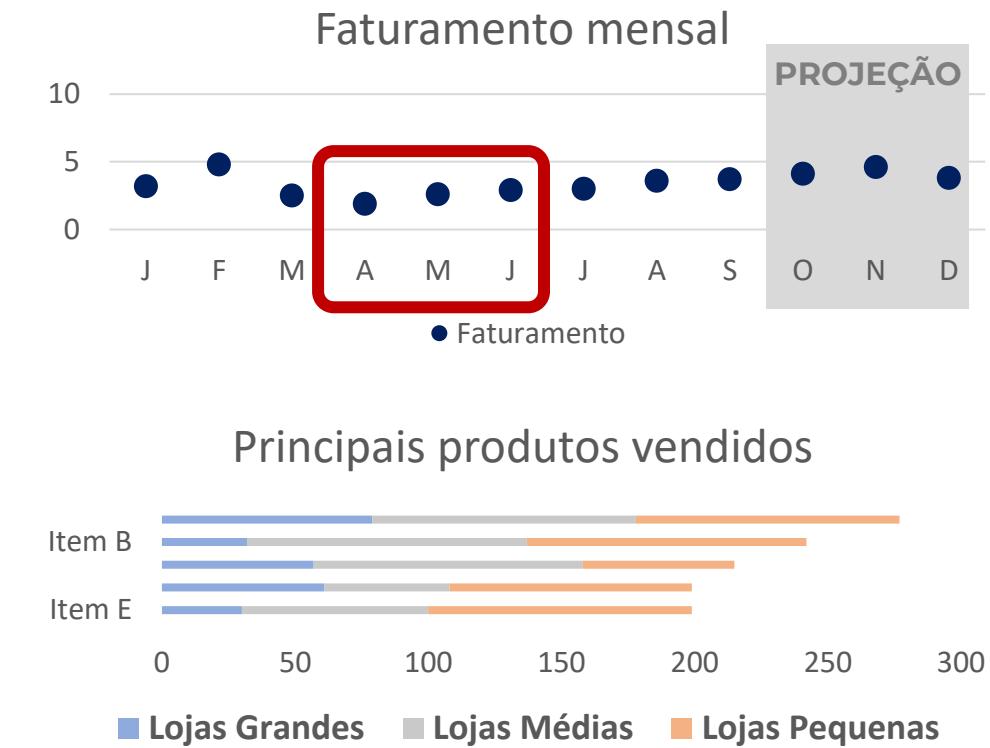
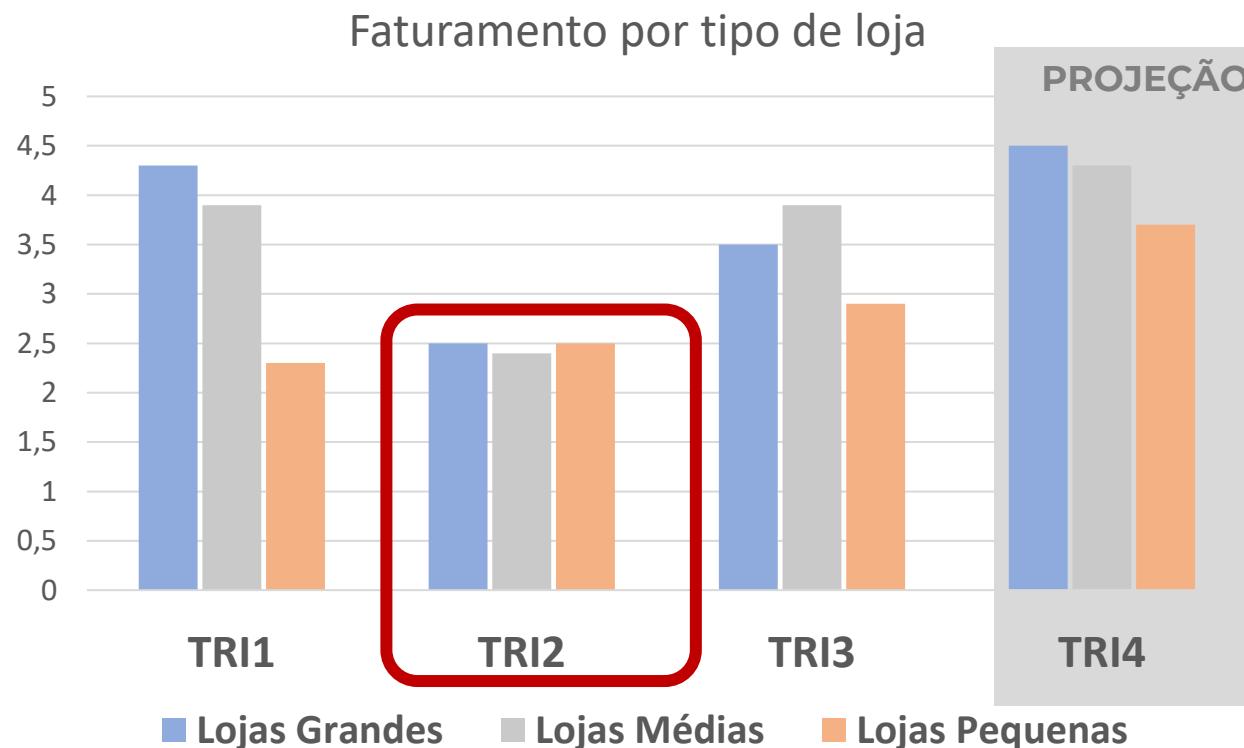
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Como que eu vou utilizar isso nesse modelo que estamos criando?



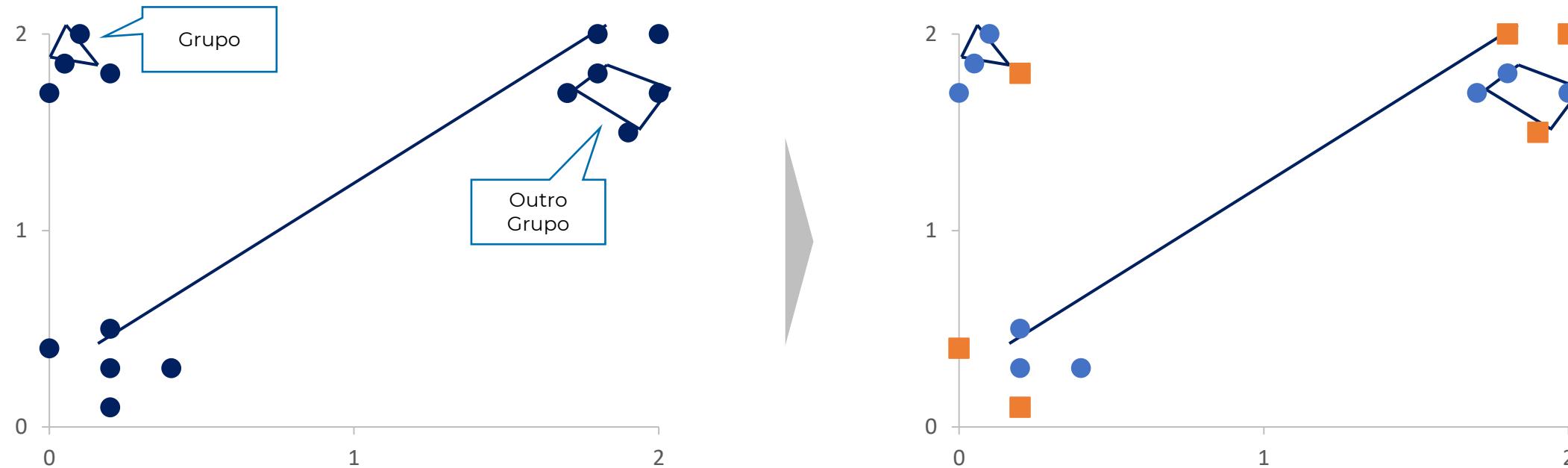
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Então vamos remover a linha do eixo y e continuamos entendendo o gráfico.



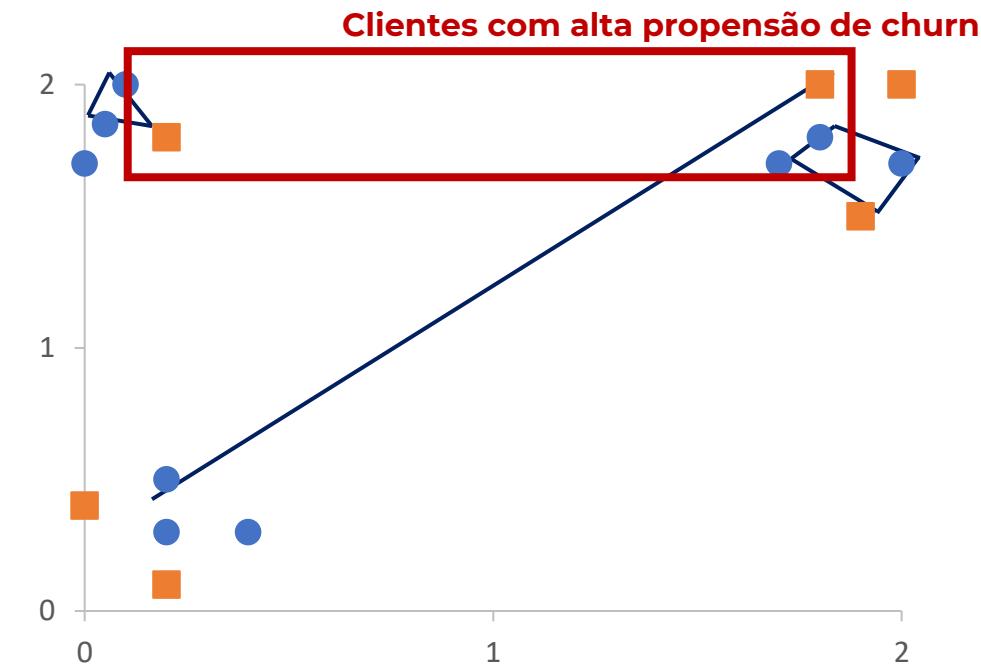
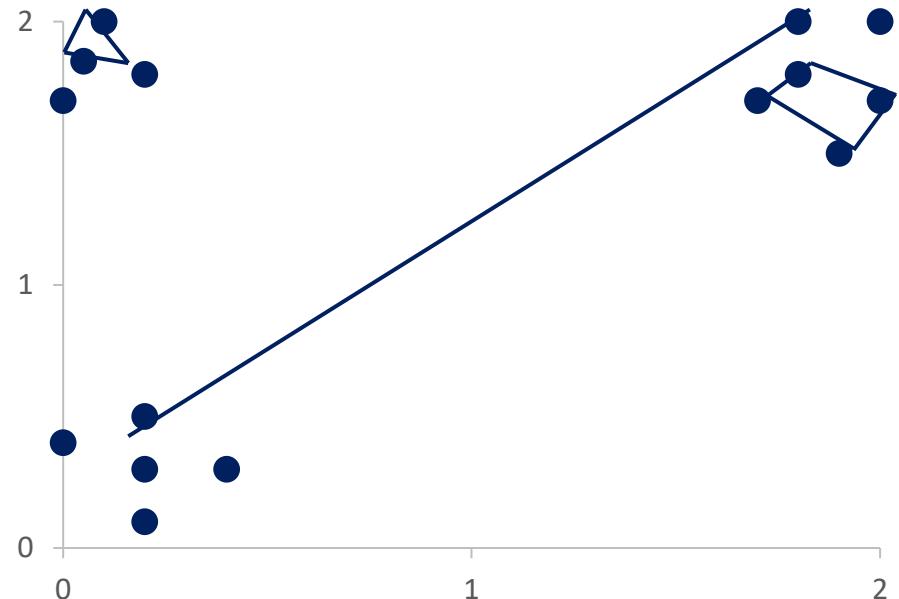
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Um outro princípio que podemos considerar é a conexão: entendemos objetos fisicamente conectados como parte de um grupo.



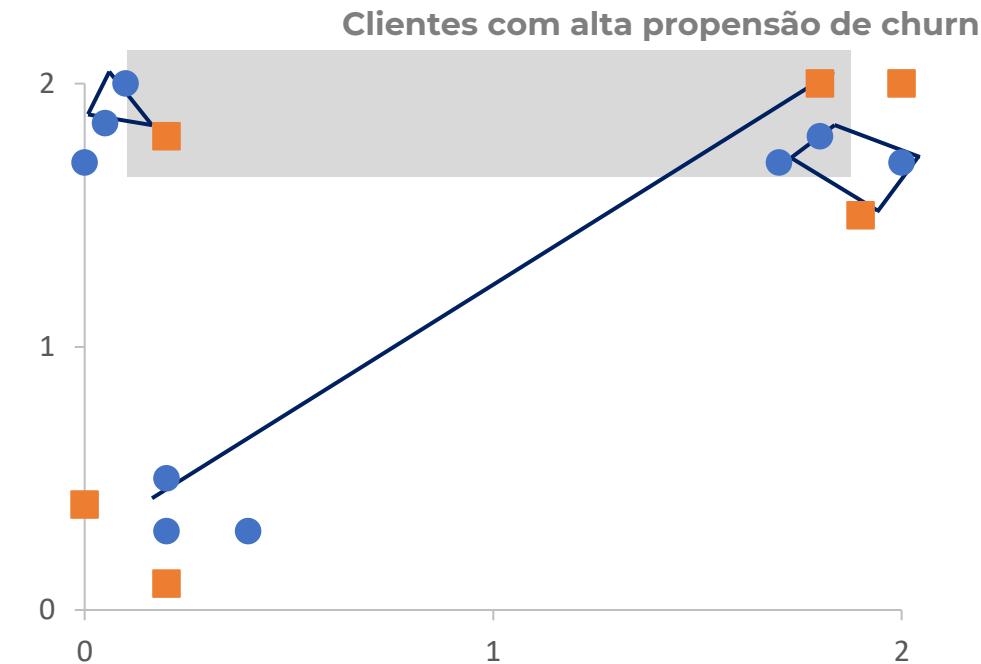
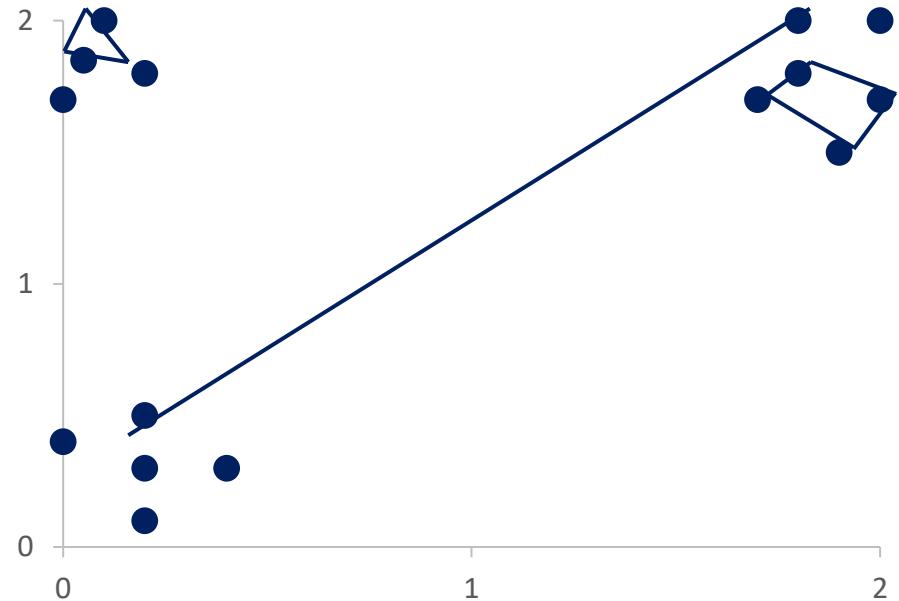
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Podemos ver que os 2 clientes estão relacionados.



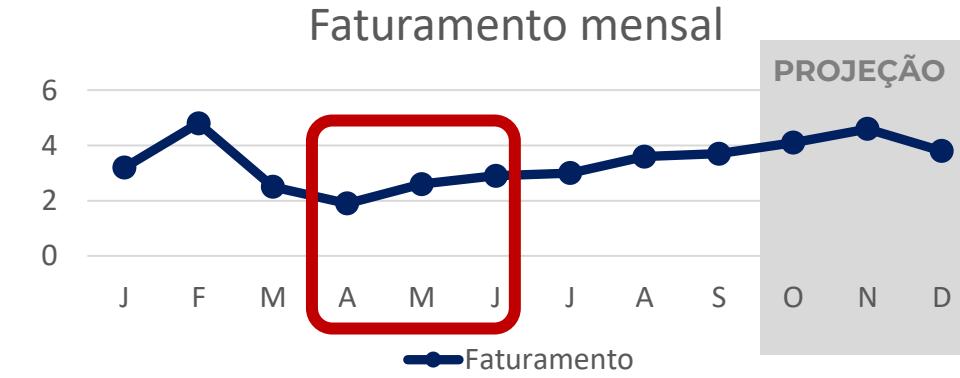
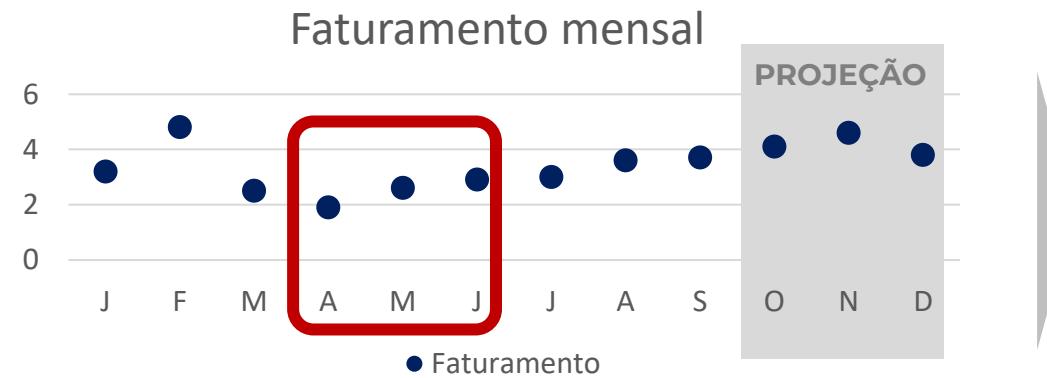
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Podemos utilizar o mesmo princípio, porém, colocando uma cor diferente.



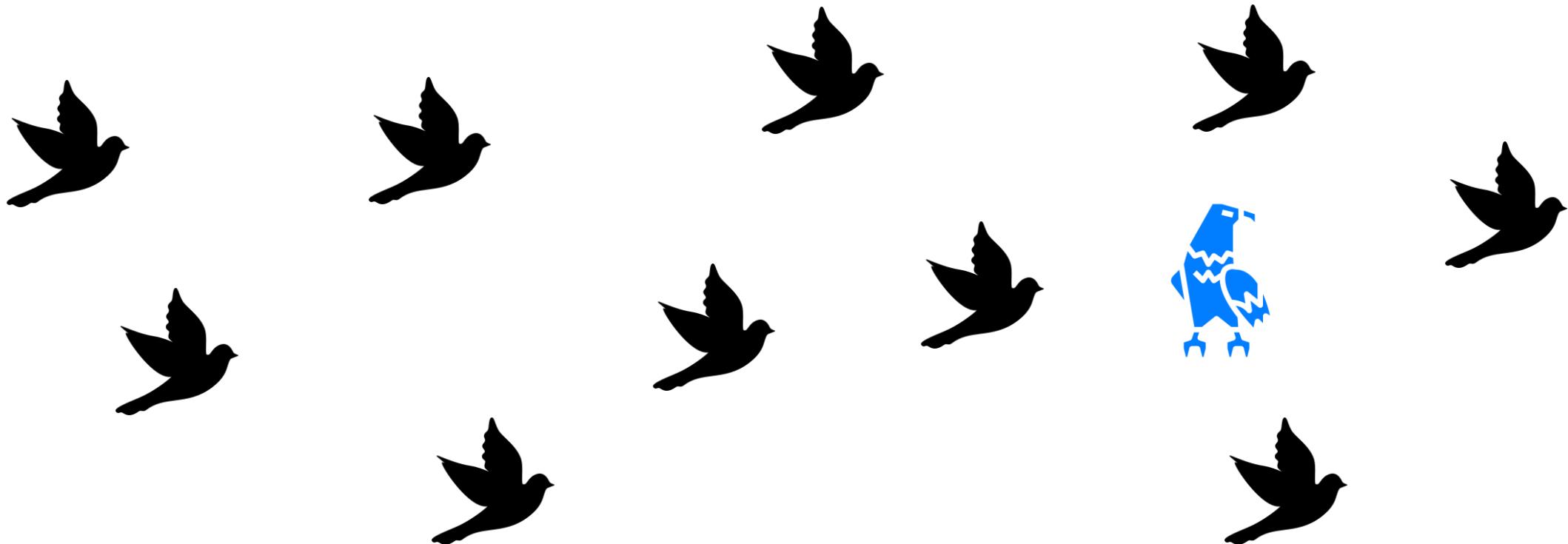
## Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

Em gráficos de pontos, quando traçamos uma reta, fica muito mais claro para interpretar os dados.



## Módulo 8 – Contraste e atributos pré-atentivos

Devemos fazer o uso estratégico do **contraste** para direcionar os olhos do nosso público para o que **ele deve ver**. Repare que no momento é fácil achar o falcão.



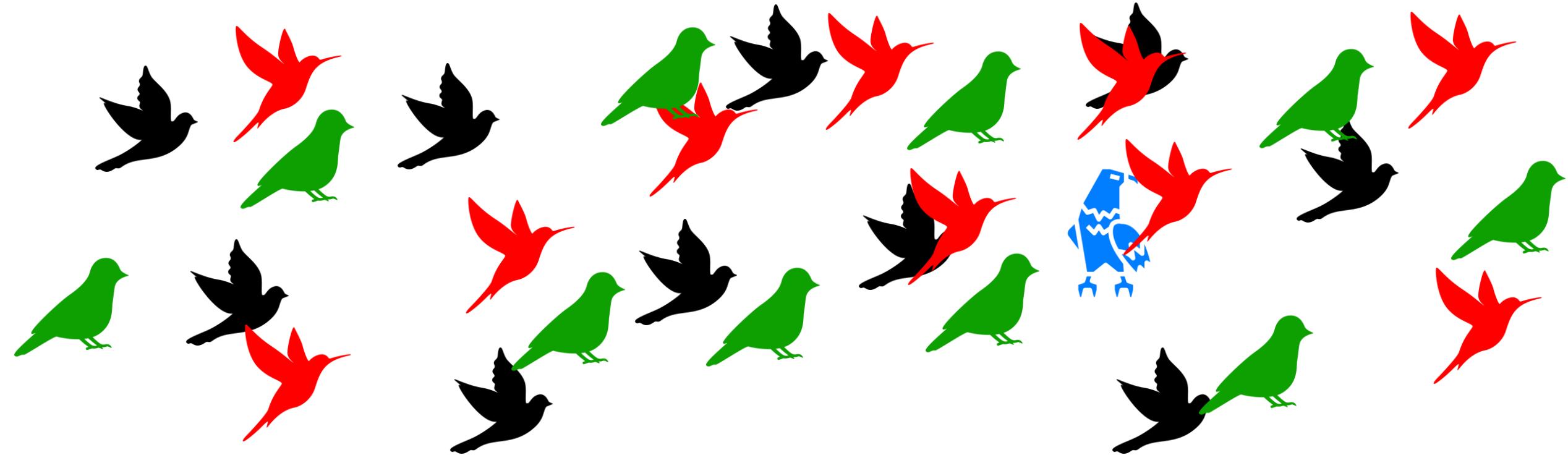
## Módulo 8 – Contraste e atributos pré-atentivos

Porém quando adicionamos outro pássaro já fica mais difícil de visualizar.



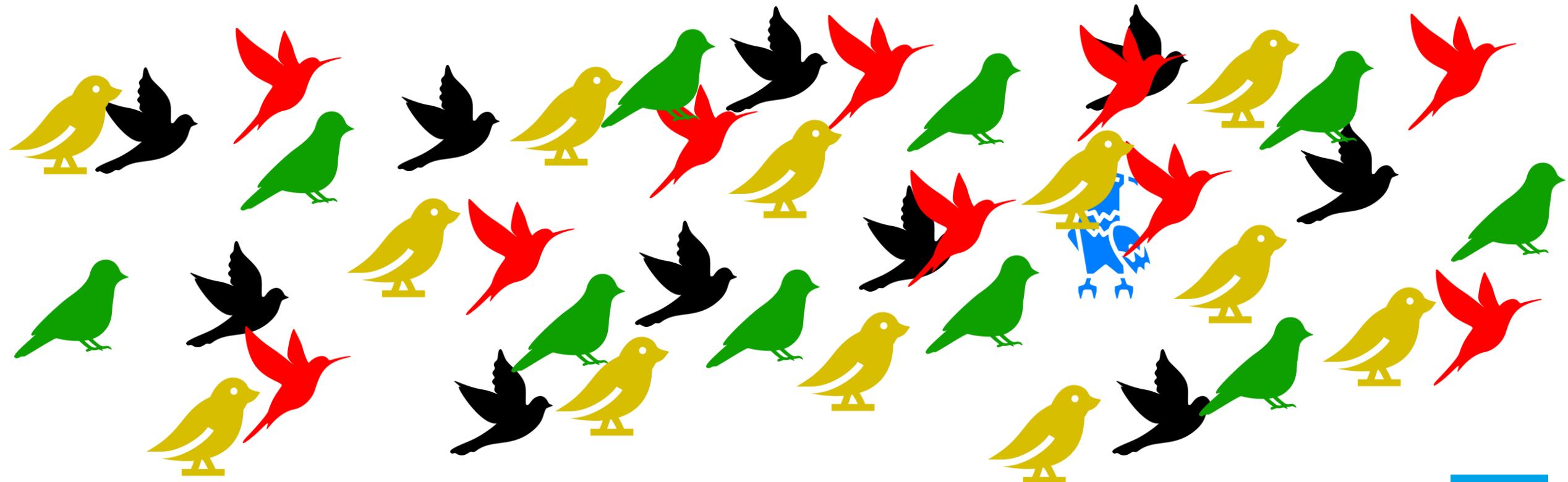
## Módulo 8 – Contraste e atributos pré-atentivos

Quando adicionamos mais um pássaro fica mais difícil de visualizar.



## Módulo 8 – Contraste e atributos pré-atentivos

Quando adicionamos mais um pássaro fica mais difícil de visualizar, o mesmo se aplica aos seus dados.



## Módulo 8 – Contraste e atributos pré-atentivos

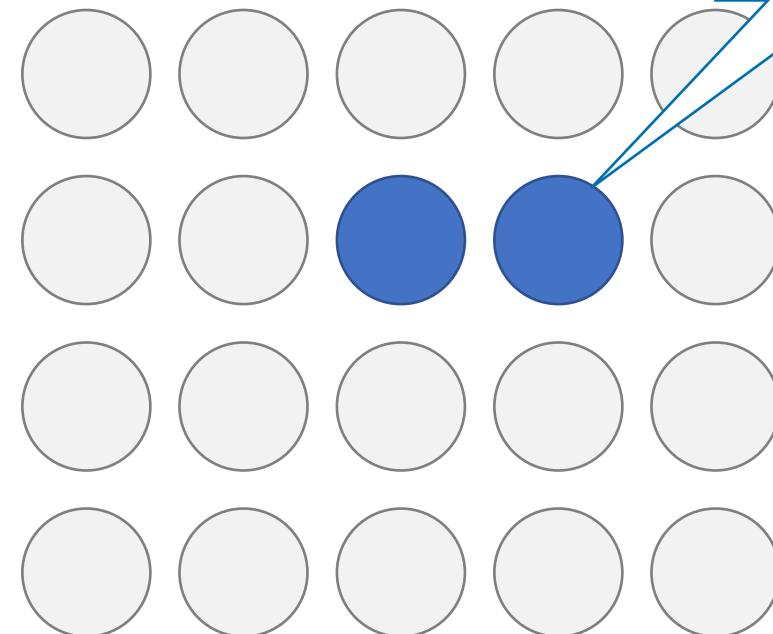
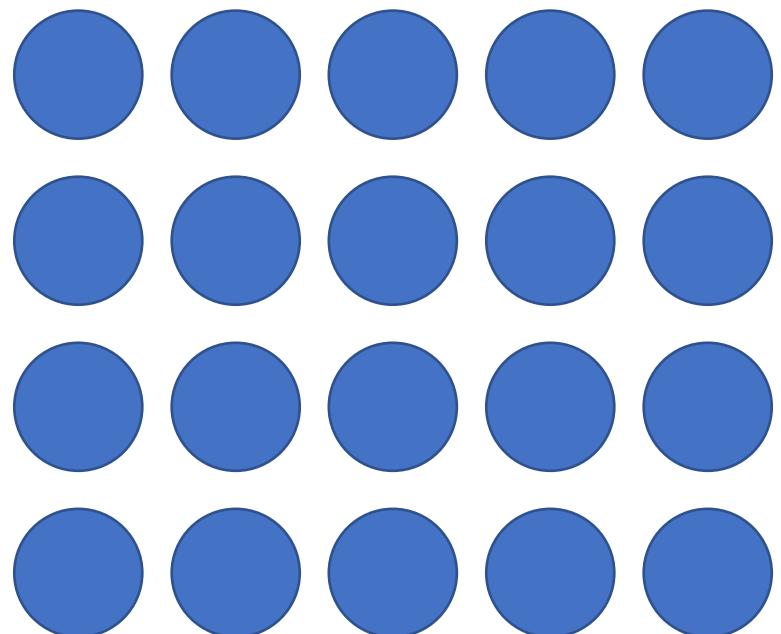
Repare que ao colocarmos o falcão em destaque é muito mais fácil de visualizá-lo.



## Módulo 8 – Contraste e atributos pré-atentivos

Retire o que não for necessário e dê destaque ao que você quer mostrar.

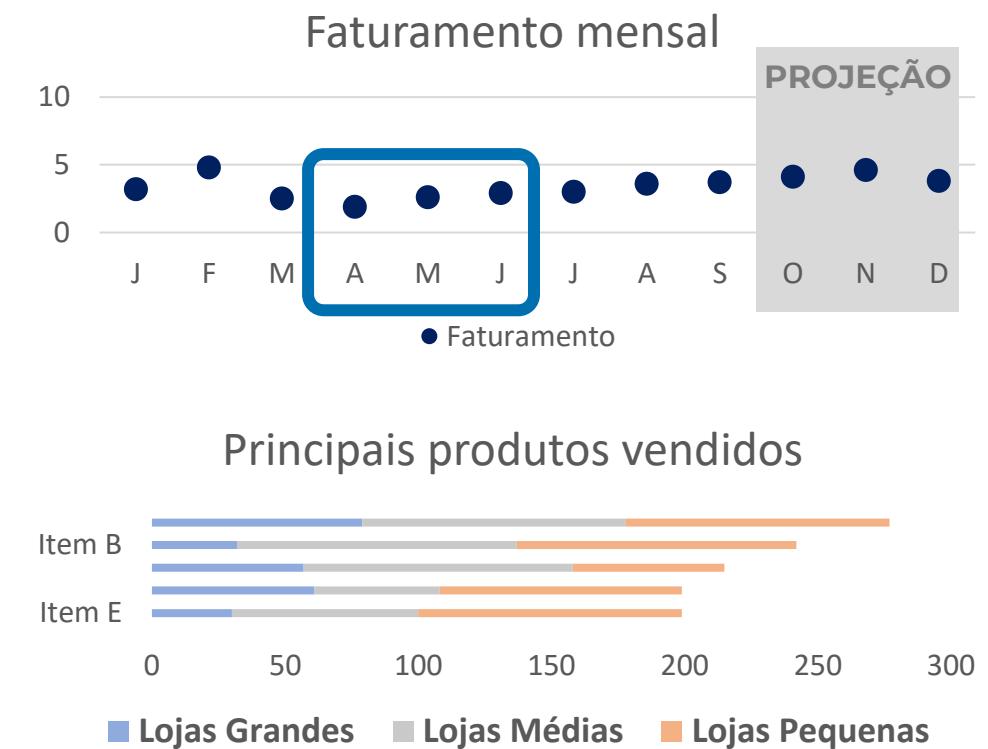
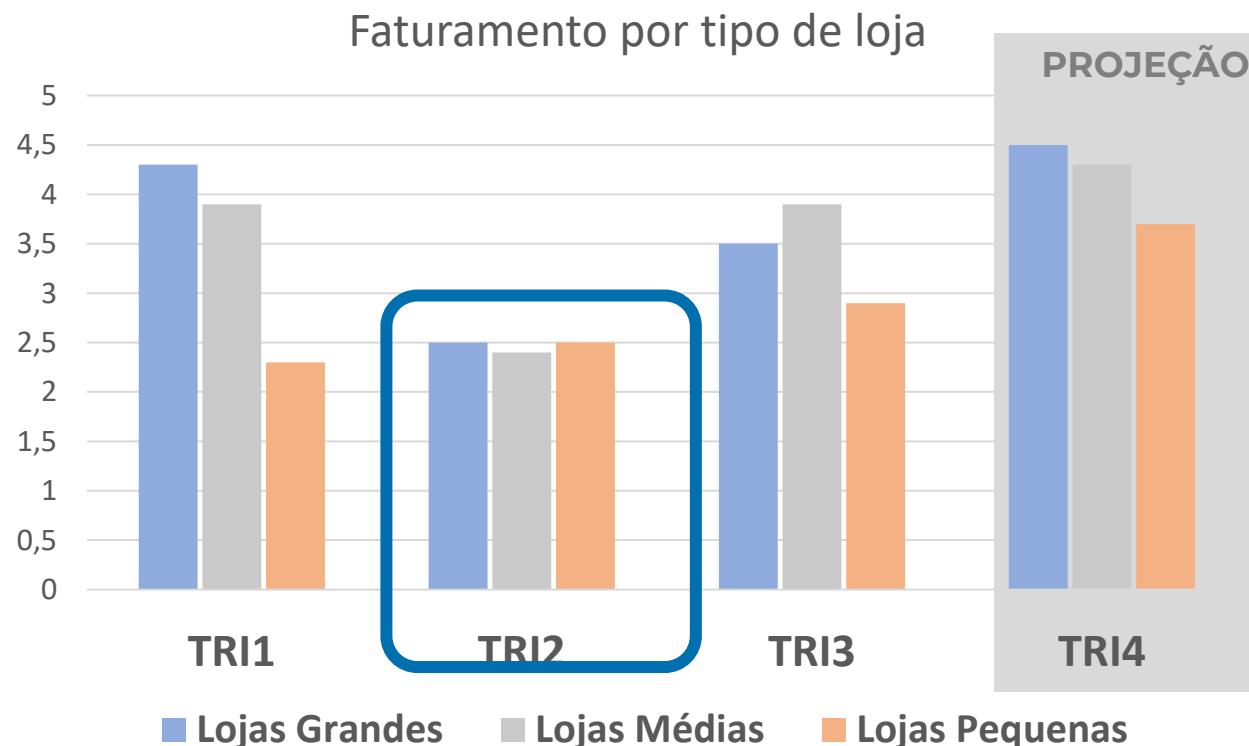
Vamos supor que isso é uma tabela e que apenas 2 dados são importantes



Dar destaque  
apenas ao que é  
importante

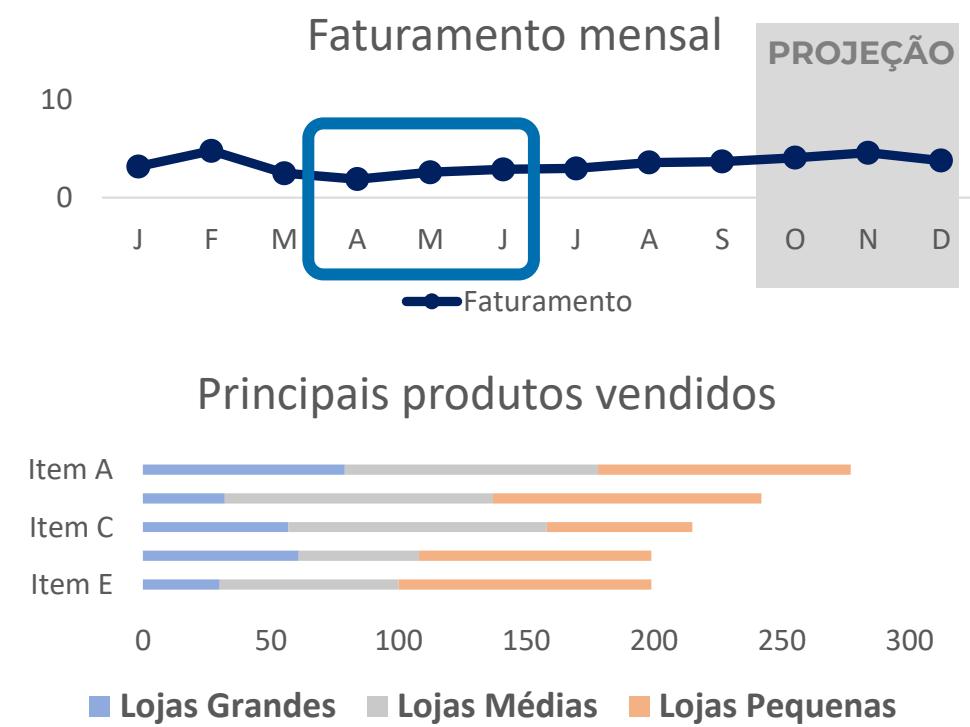
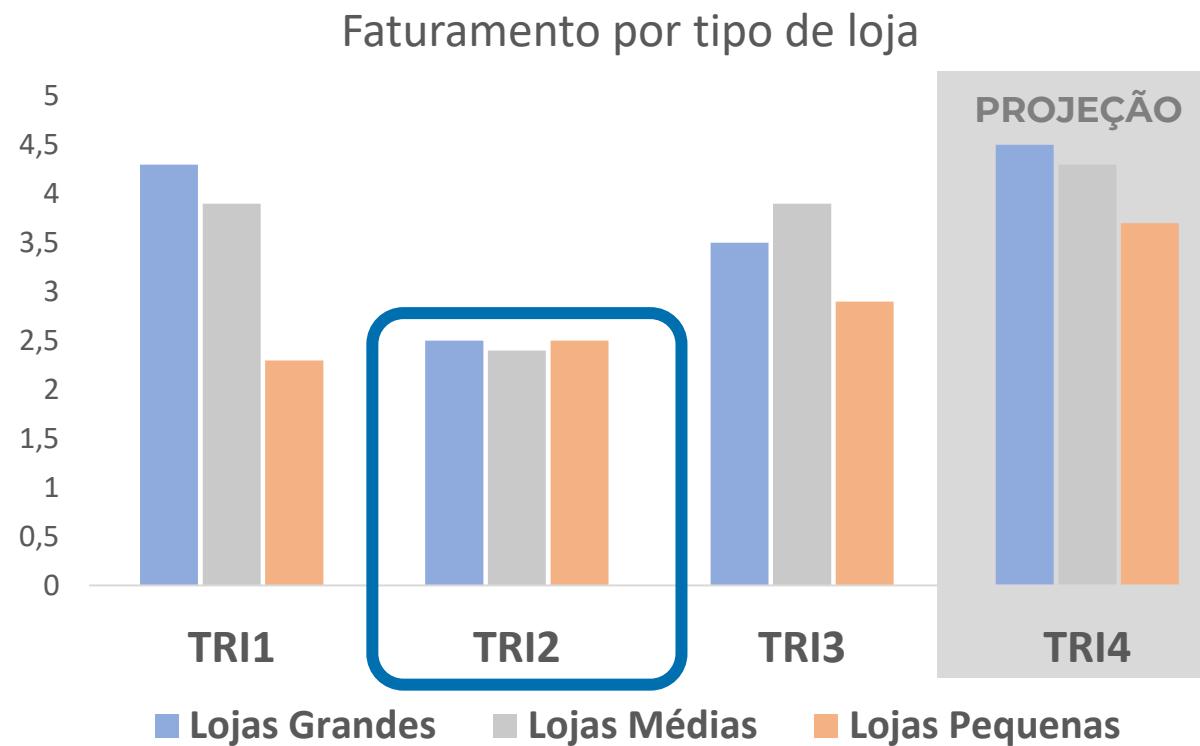
## Módulo 8 – Contraste e atributos pré-atentivos

Como poderíamos melhorar o contraste nesses gráficos? Um exemplo muito prático é retirando a linha de grade.



## Módulo 8 – Contraste e atributos pré-atentivos

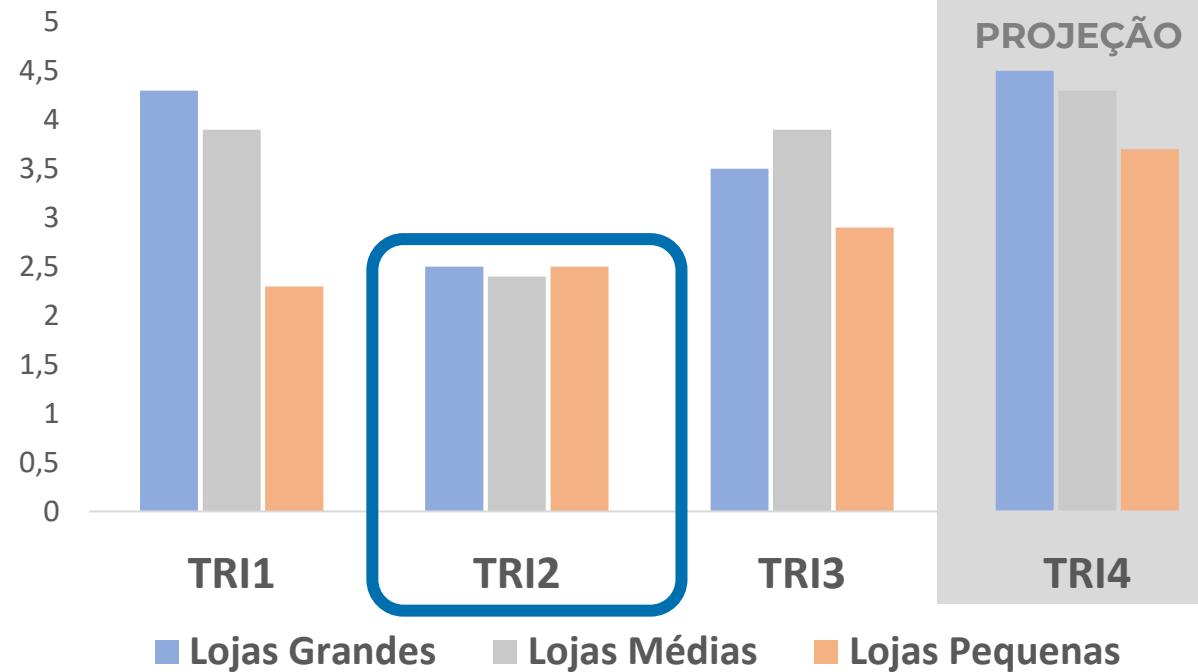
Retirando a linha de grade o nosso gráfico fica **mais limpo**.



## Módulo 8 – Contraste e atributos pré-atentivos

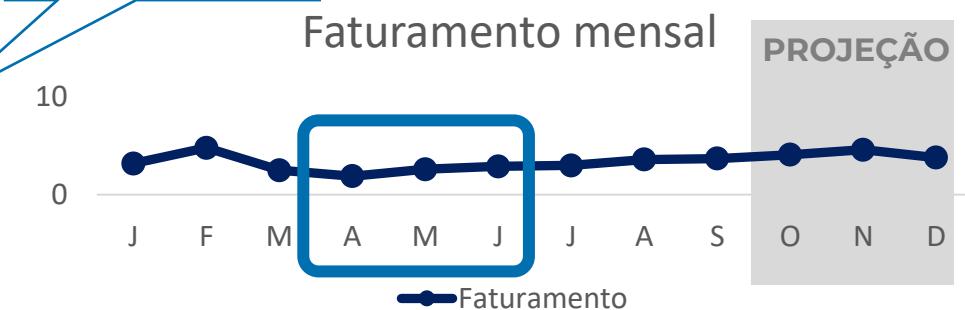
Até em elementos que criamos vamos verificar se todos agregam valor.

Faturamento por tipo de loja

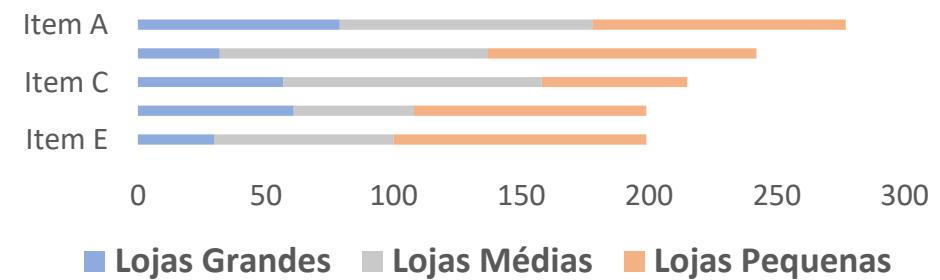


Essa projeção  
agrega valor?

Faturamento mensal

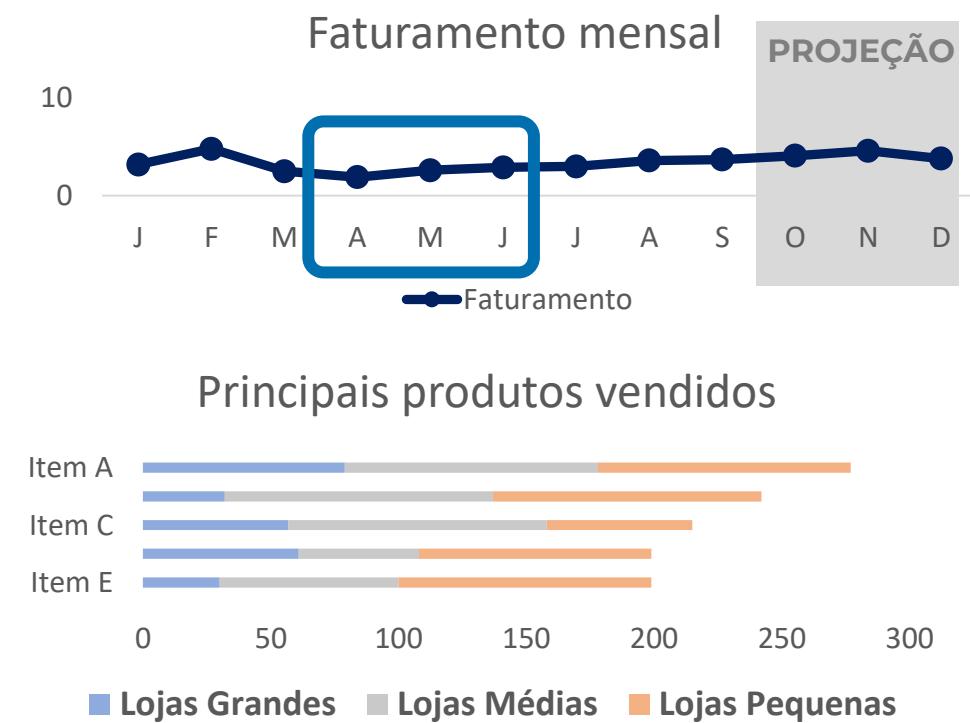
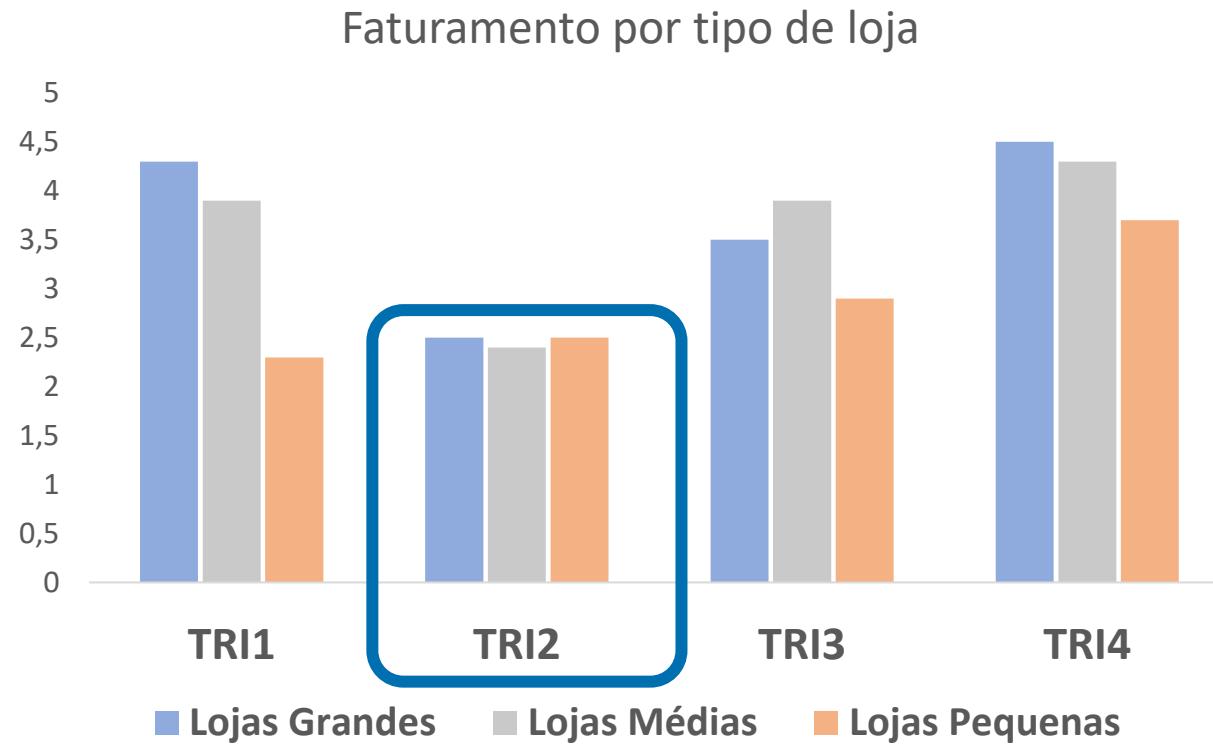


Principais produtos vendidos



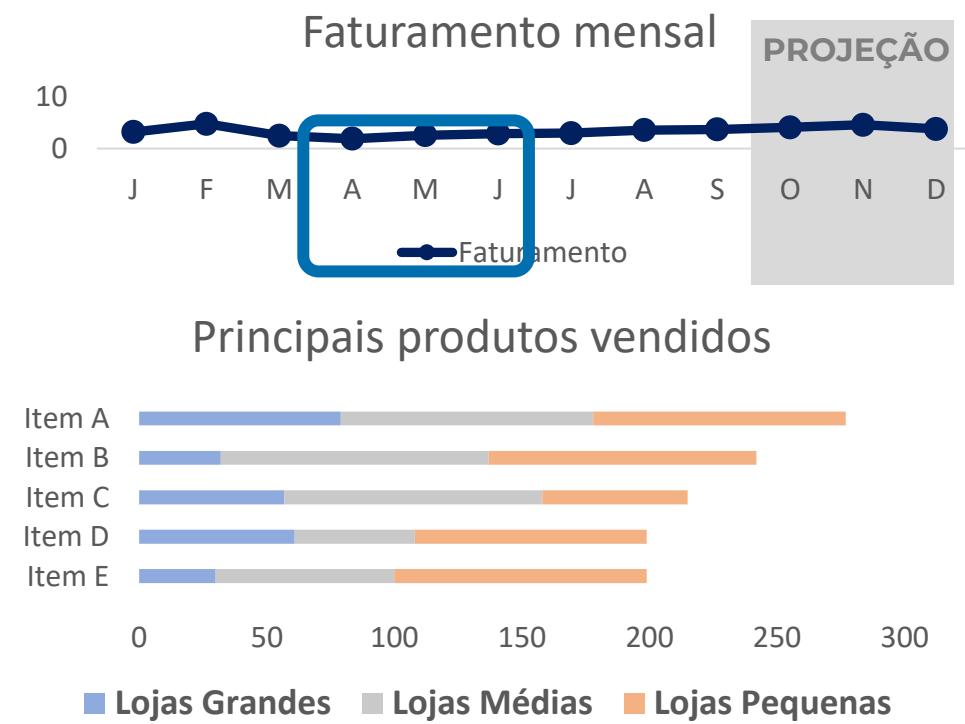
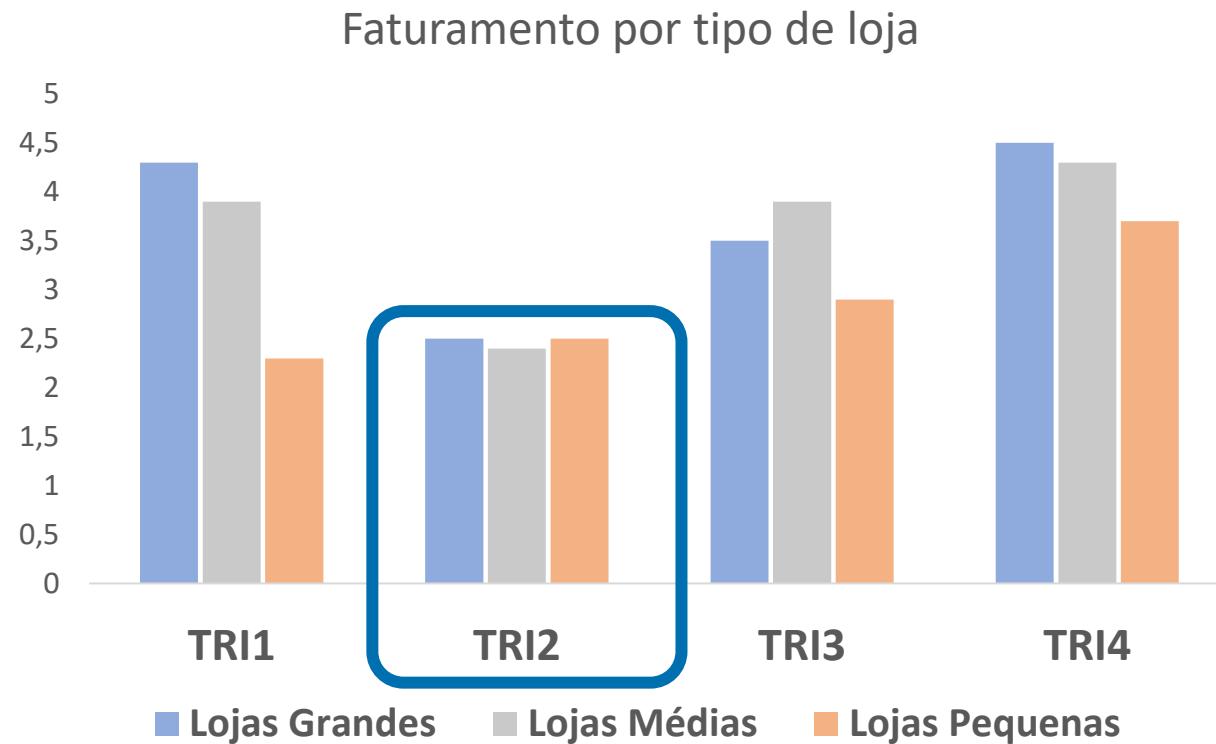
## Módulo 8 – Contraste e atributos pré-atentivos

Aquela projeção não agregava valor, pois em outro gráfico os dados de outubro, novembro e dezembro estão indicados como projeção.



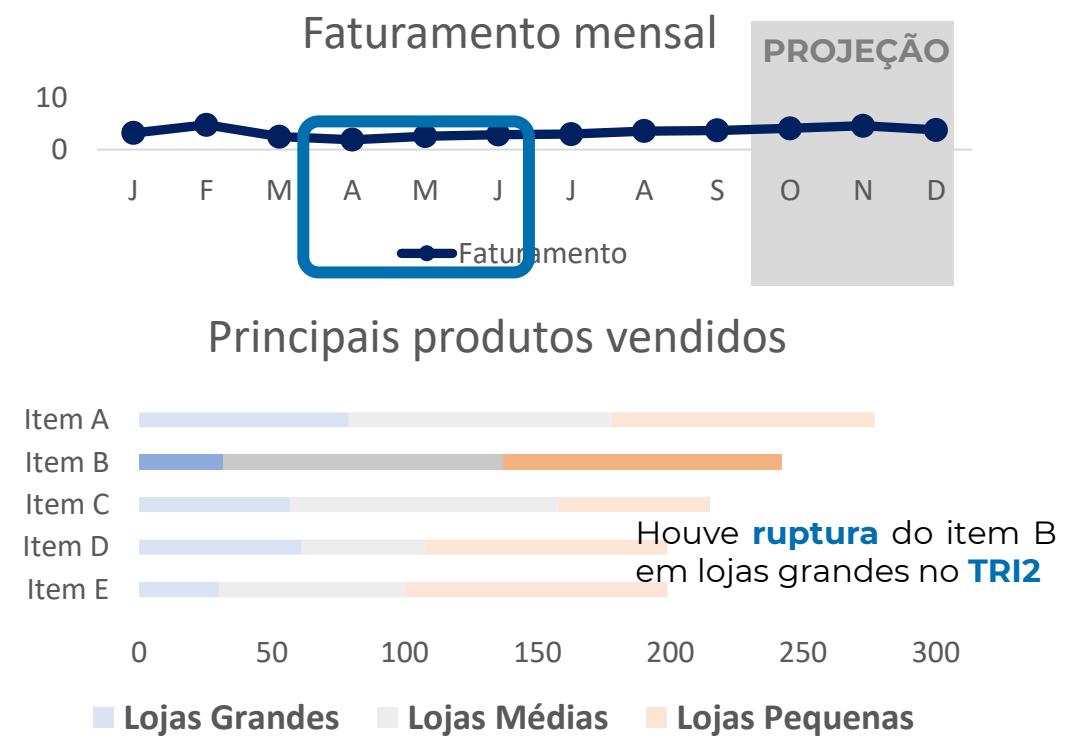
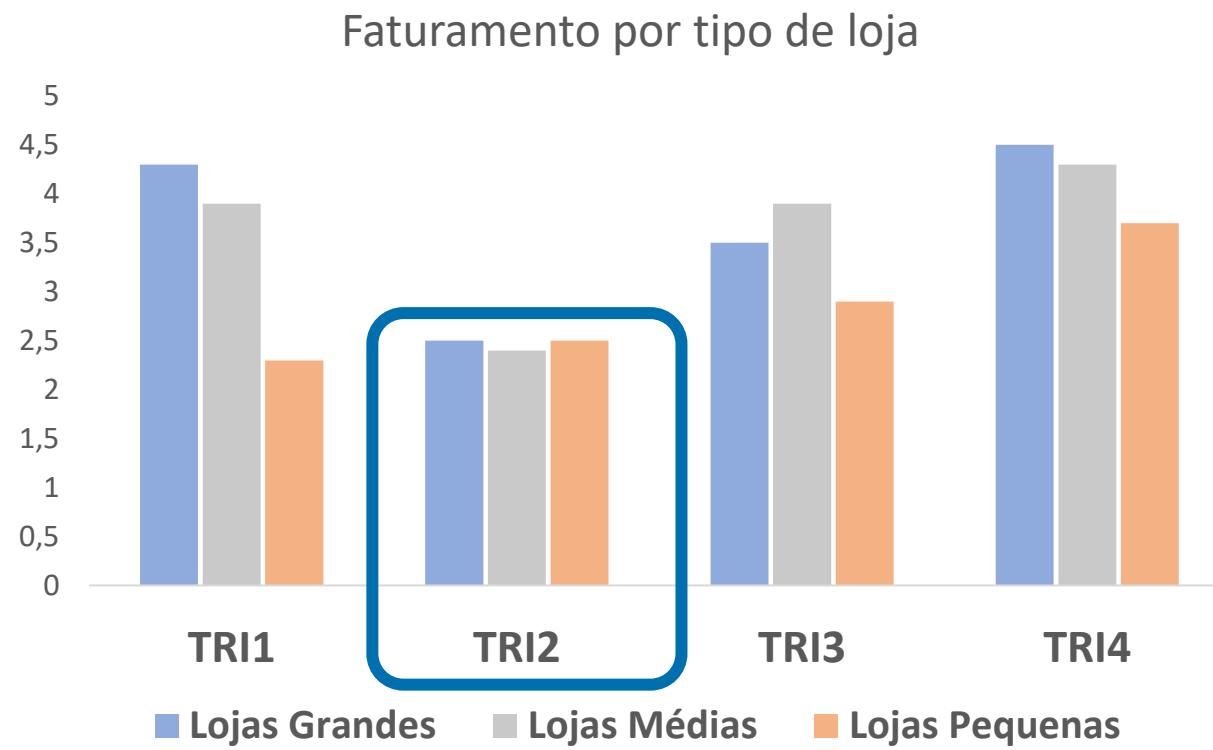
## Módulo 8 – Contraste e atributos pré-atentivos

Nosso olhos tendem a ser atraídos pelo que está diferente no grupo.  
E se quisermos dar um destaque para o item B?



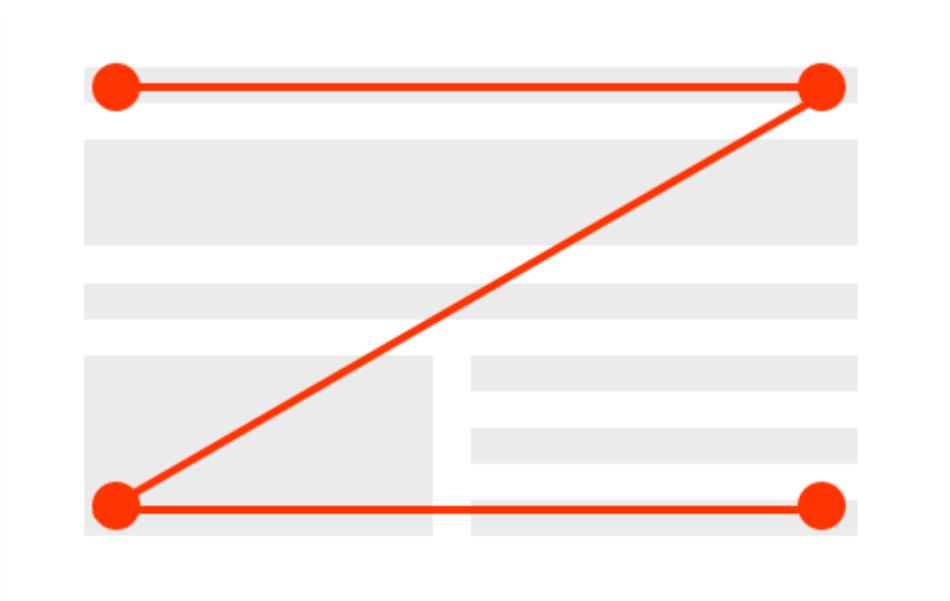
## Módulo 8 – Contraste e atributos pré-atentivos

Podemos colocar a cor do item B mais escuro que as demais.



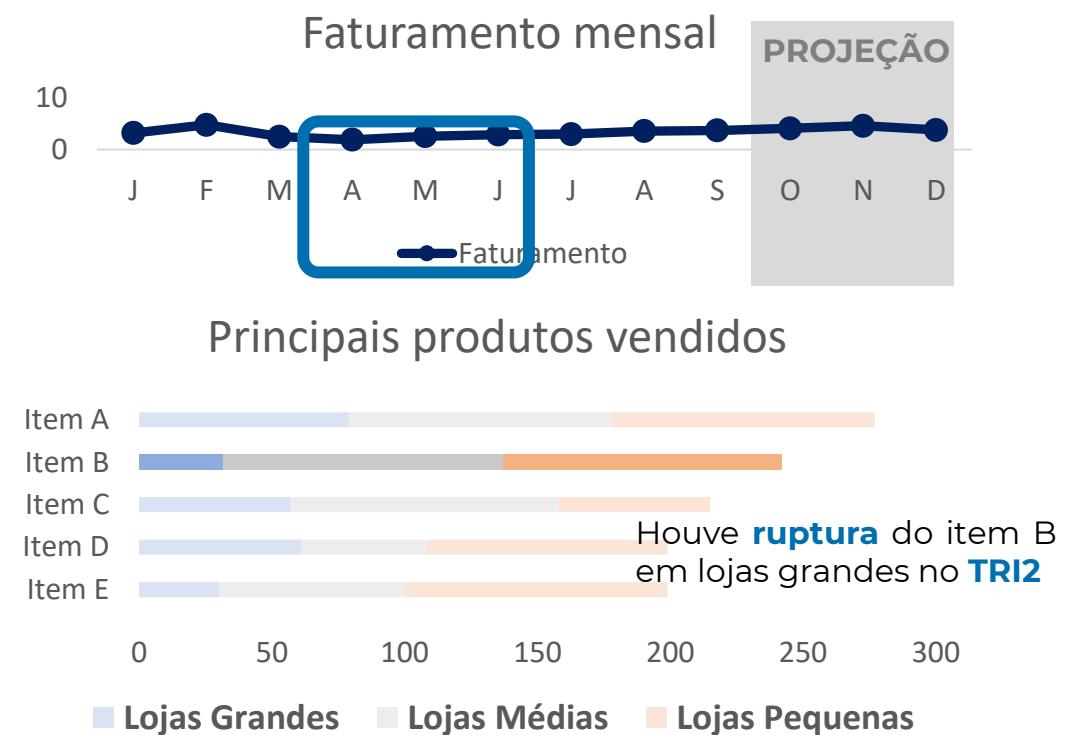
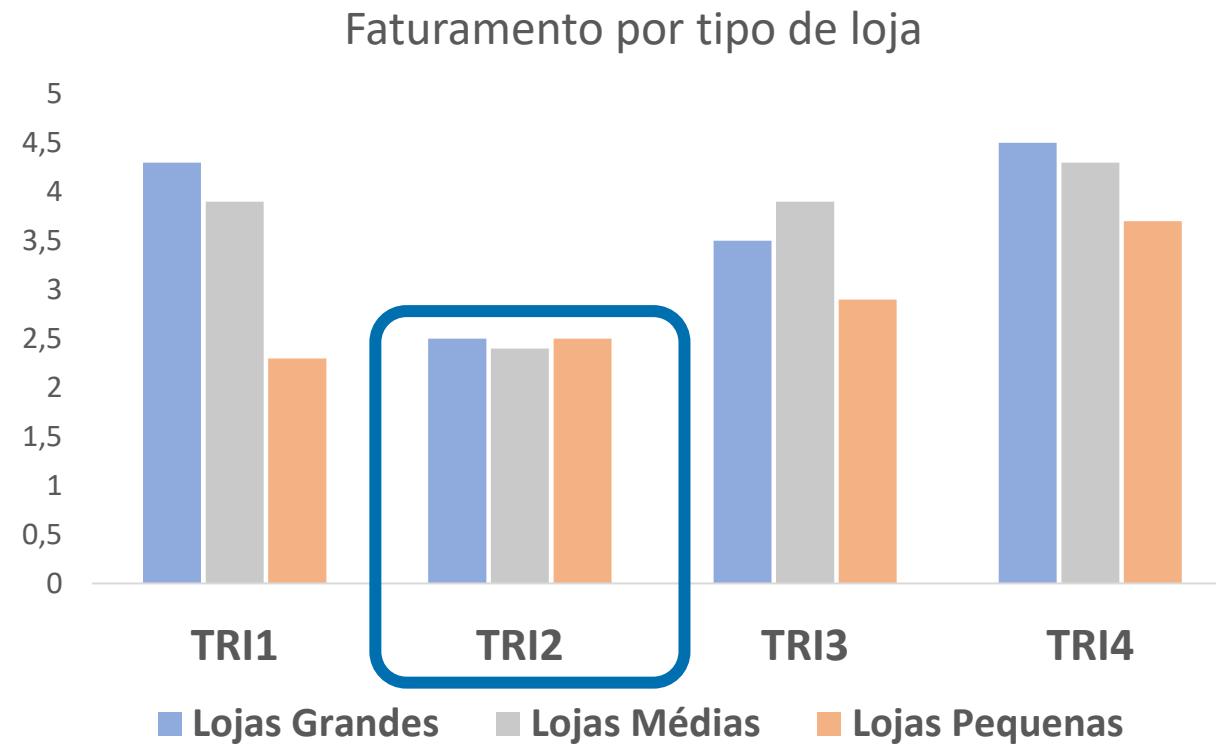
## Módulo 8 – Contraste e atributos pré-atentivos

As pessoas tendem a ler a apresentação em formato Z, então podemos organizar os nossos dados dessa forma, colocando o principal na esquerda.



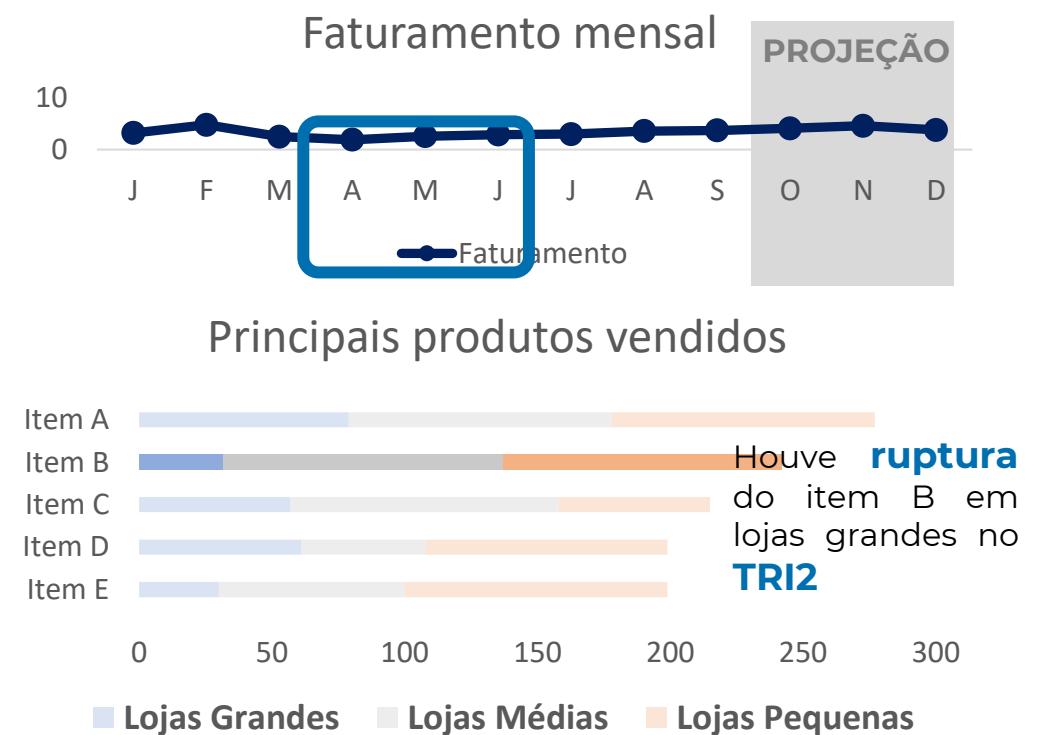
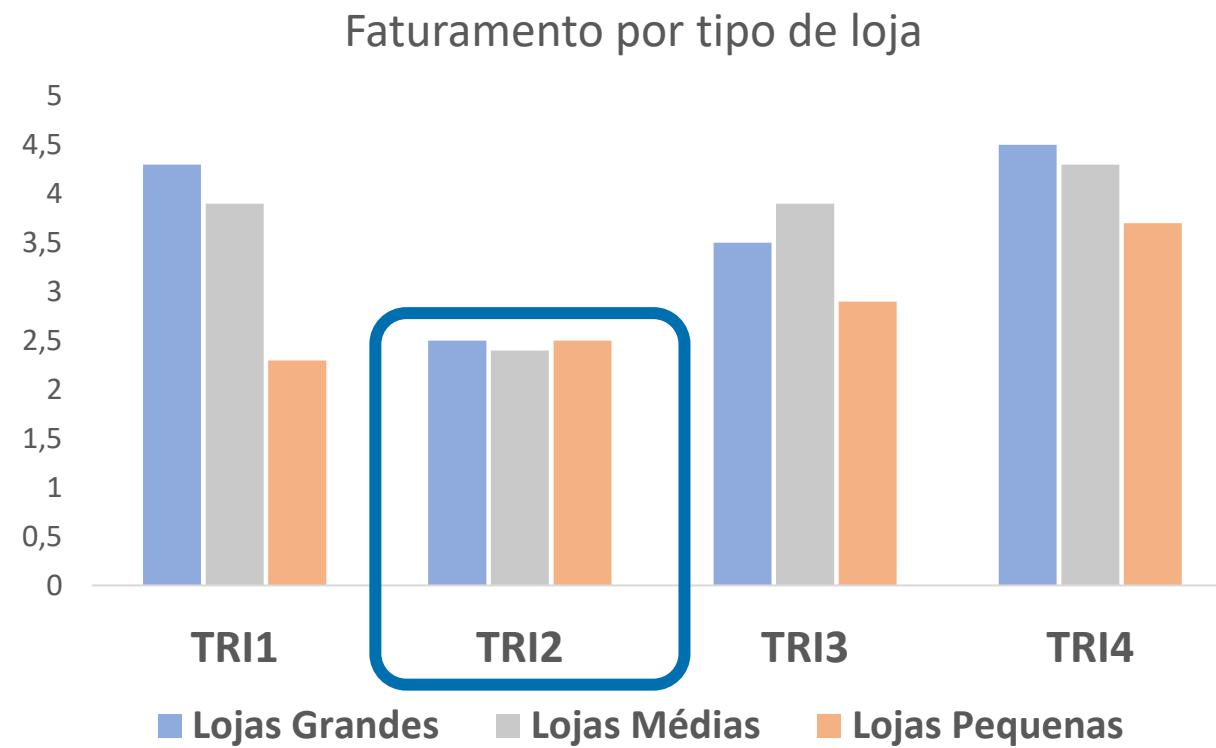
## Módulo 8 – Contraste e atributos pré-atentivos

As cores podem ser usadas para destacar parte do texto / gráfico. Repare que o texto em azul se comunica com o retângulo em azul.



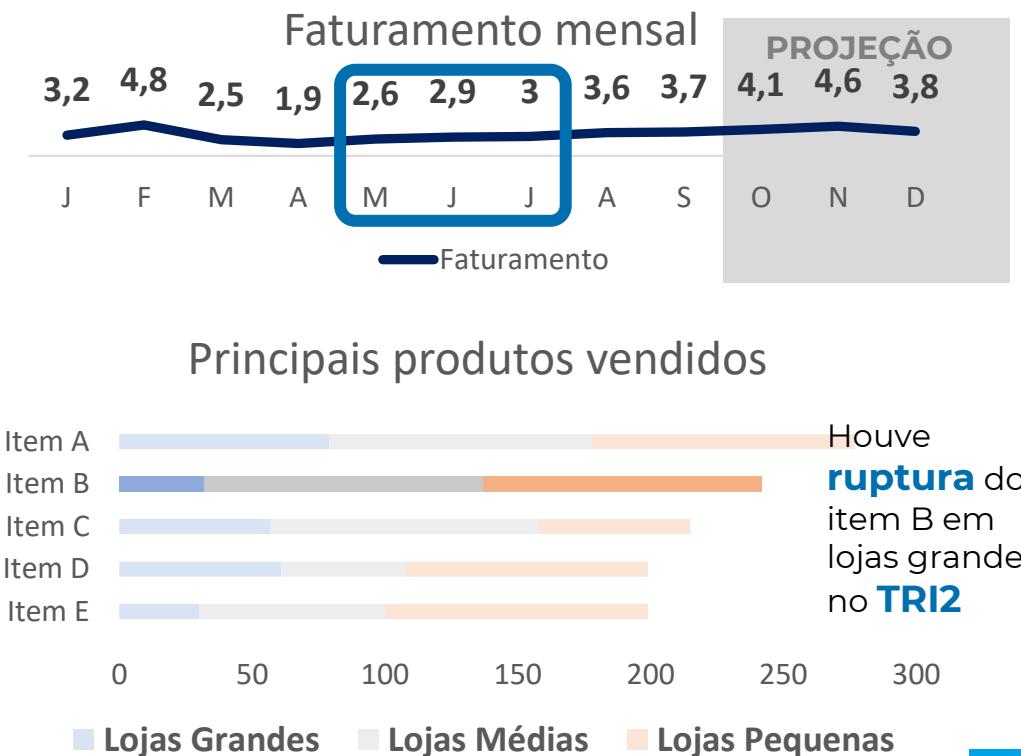
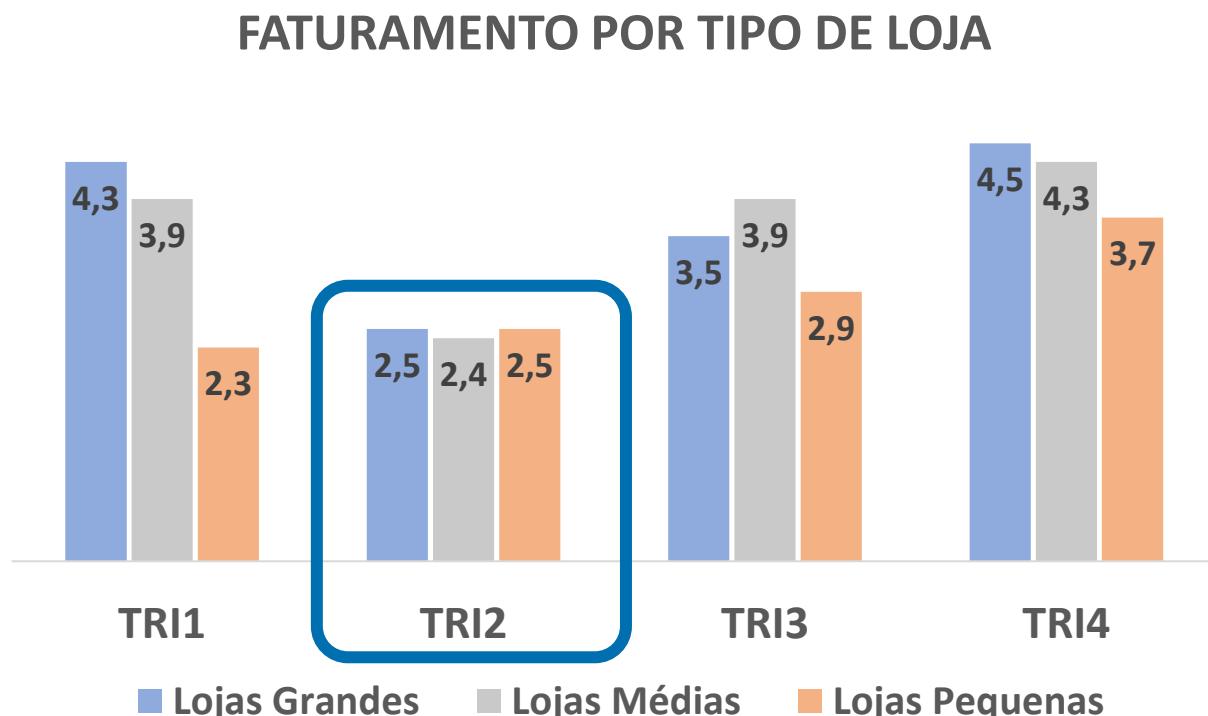
## Módulo 8 – Contraste e atributos pré-atentivos

Da mesma forma que as cores, os tamanhos também dão destaque



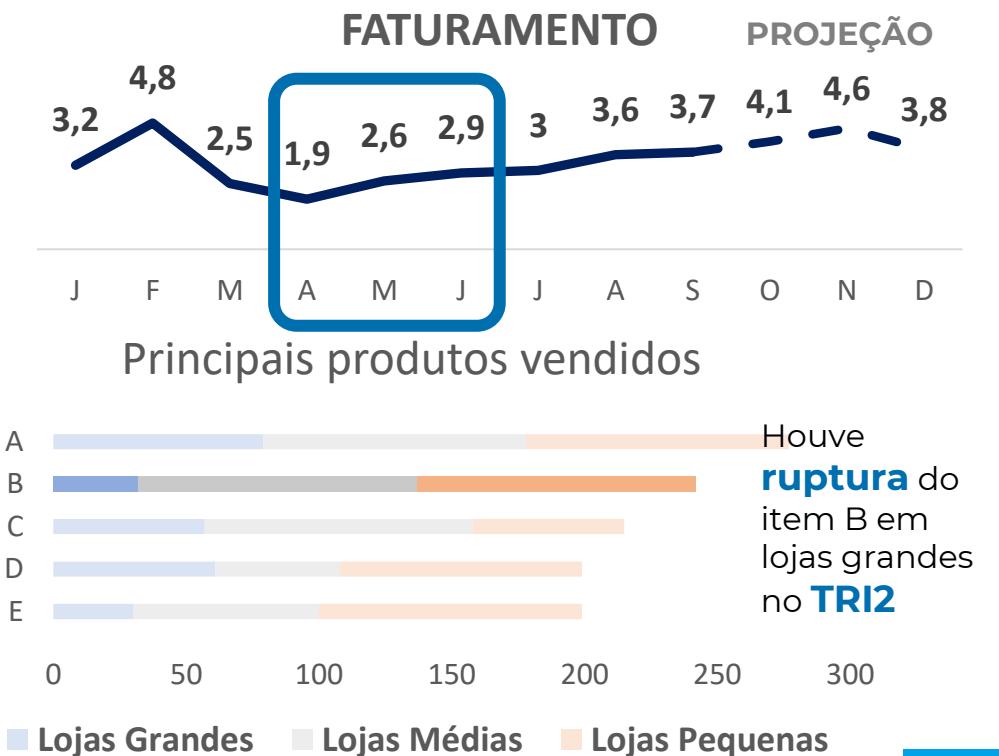
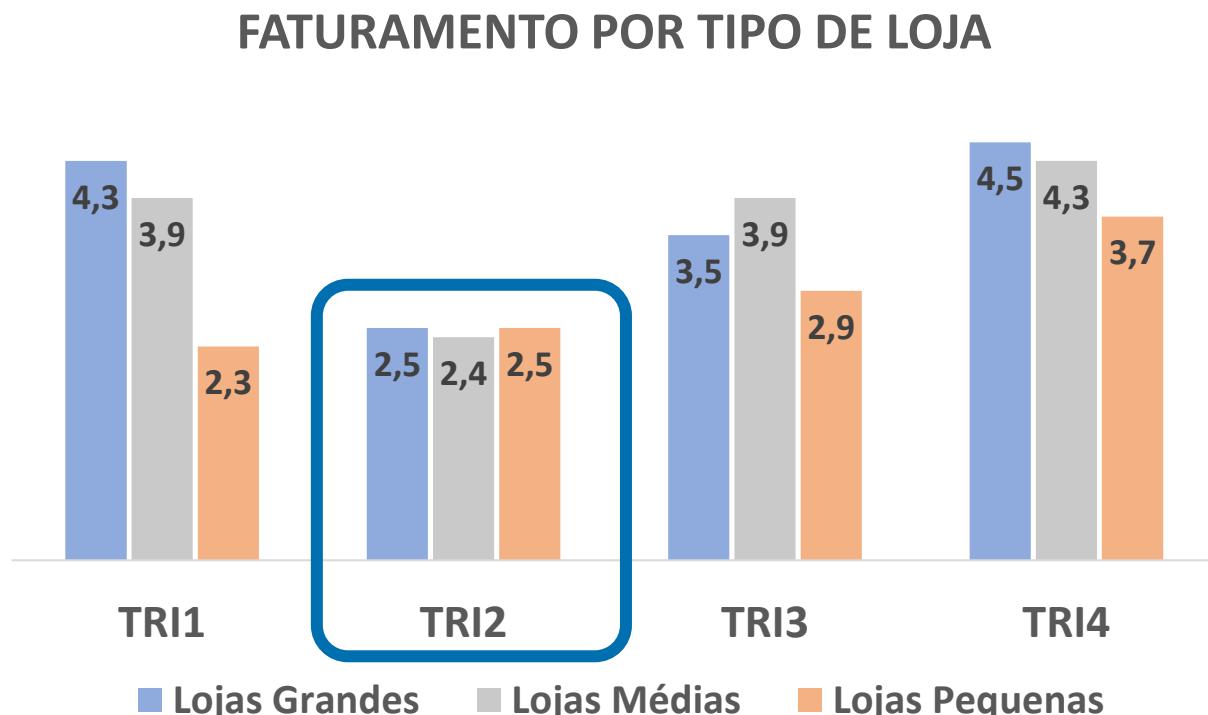
## Módulo 8 – Contraste e atributos pré-atentivos

Faça de tudo para tornar seu visual o mais fácil possível de compreender.



## Módulo 8 – Contraste e atributos pré-atentivos

Podemos também remover esse retângulo da projeção e colocar uma linha tracejada, que indica que é uma formação que ainda está sendo executada.



## Módulo 8 – Visualização de dados no Python: Passo a passo para melhorar seus visuais no matplotlib

Agora vamos aplicar as boas práticas que falamos no PowerPoint e aplicar no matplotlib. Para começar vamos utilizar uma base de dados de x, y e z.

```
import pandas as pd  
import matplotlib.pyplot as plt  
  
dados = {  
    'X': [1,2,3,4,5,6],  
    'Y': [120,110,130,145,118,125],  
    'Z': [95,54,86,77,90,81]  
}  
  
base = pd.DataFrame(dados)  
base.head()
```

`base.head()`

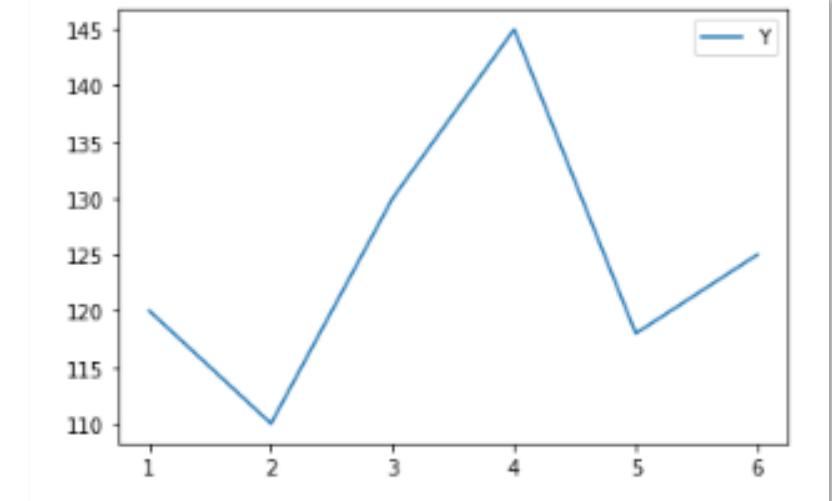
	x	y	z
0	1	120	95
1	2	110	54
2	3	130	86
3	4	145	77
4	5	118	90

## Módulo 8 – Visualização de dados no Python: Passo a passo para melhorar seus visuais no matplotlib

E aí como que criamos um plot?

Podemos consultar na documentação e copiar o código.

```
import matplotlib.pyplot as plt  
  
fig, ax = plt.subplots()  
  
ax.plot(base.X,base.Y, label="Y")  
  
plt.legend  
plt.show()
```



## Módulo 8 – Visualização de dados no Python: Passo a passo para melhorar seus visuais no matplotlib

Como podemos plotar 2 linhas nesse mesmo gráfico?

```
import matplotlib.pyplot as plt
```

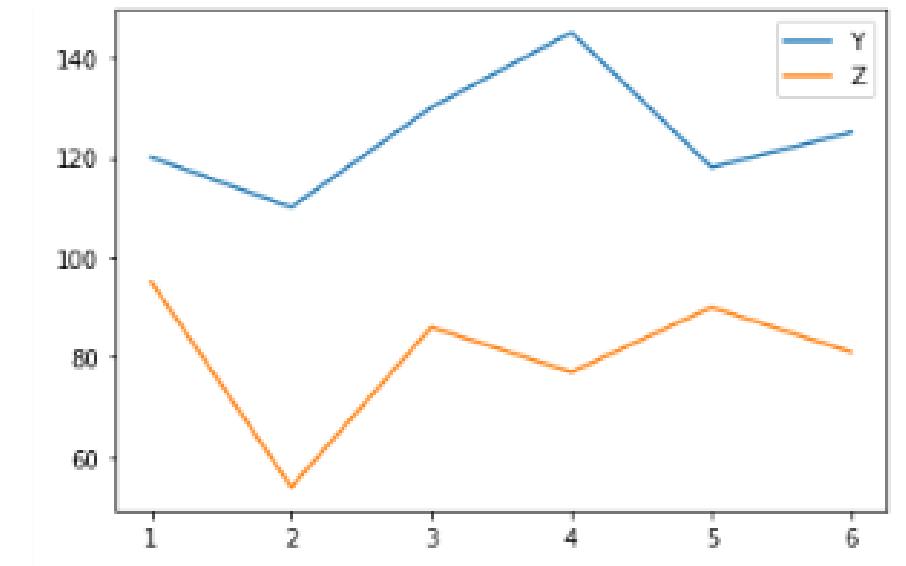
```
fig, ax = plt.subplots()
```

```
ax.plot(base.X,base.Y, label="Y")
```

```
ax.plot(base.X,base.Z, label="Z")
```

```
plt.legend  
plt.show()
```

Adicionamos os valores de Z



## Módulo 8 – Visualização de dados no Python: Passo a passo para melhorar seus visuais no matplotlib

E se quiséssemos separar em 2 gráficos?

Como argumento do **plt.subplots** podemos passar o número de linhas e colunas que queremos no nosso plot.

- `plt.subplots(nrows= ,ncols= )`

Nesse caso vamos precisar passar um índice para o ax (`ax[0]`, `ax[1]`)

## Módulo 8 – Visualização de dados no Python: Passo a passo para melhorar seus visuais no matplotlib

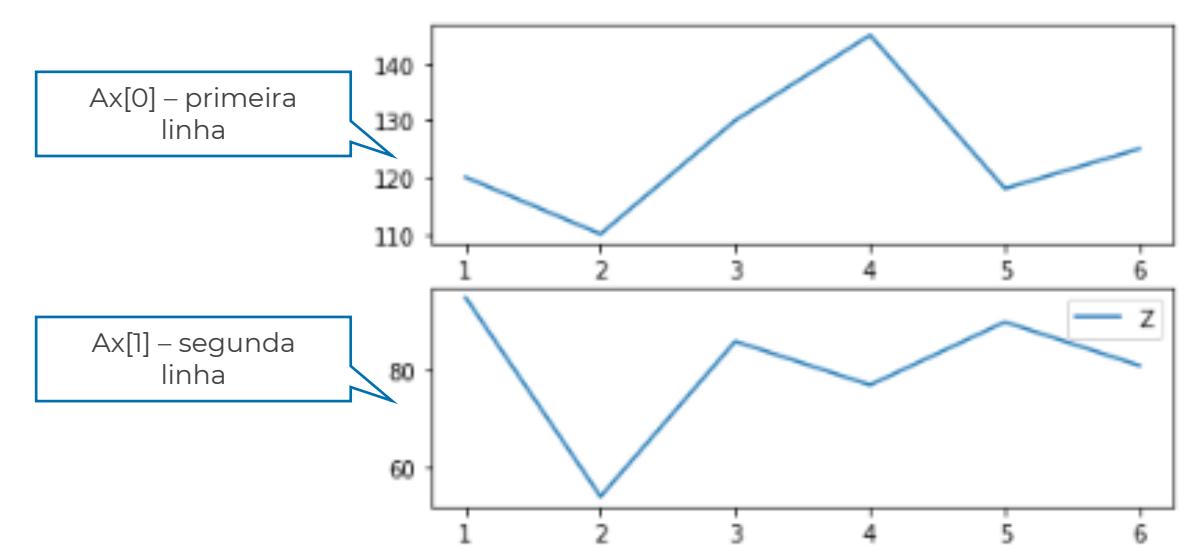
Vamos aplicar essa esses argumentos do **plt.subplots** para o nosso código:

```
fig, ax = plt.subplots(nrows=2,ncols=1)
```

```
ax[0].plot(base.X,base.Y, label="Y")  
ax[1].plot(base.X,base.Z, label="Z")
```

```
plt.legend()
```

```
plt.show()
```



## Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Para melhorar o visual do nosso gráfico podemos:

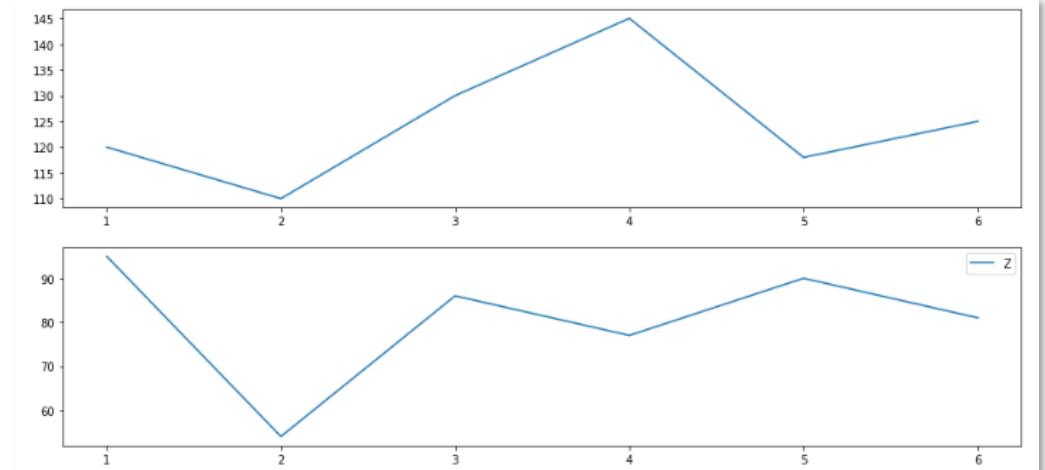
- Aumentar o tamanho do gráfico usando o `figsize=(x,y)`

```
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(15,7))
```

```
ax[0].plot(base.X, base.Y, label="Y")  
ax[1].plot(base.X, base.Z, label="Z")
```

```
plt.legend()
```

```
plt.show()
```



## Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Para melhorar o visual do nosso gráfico podemos:

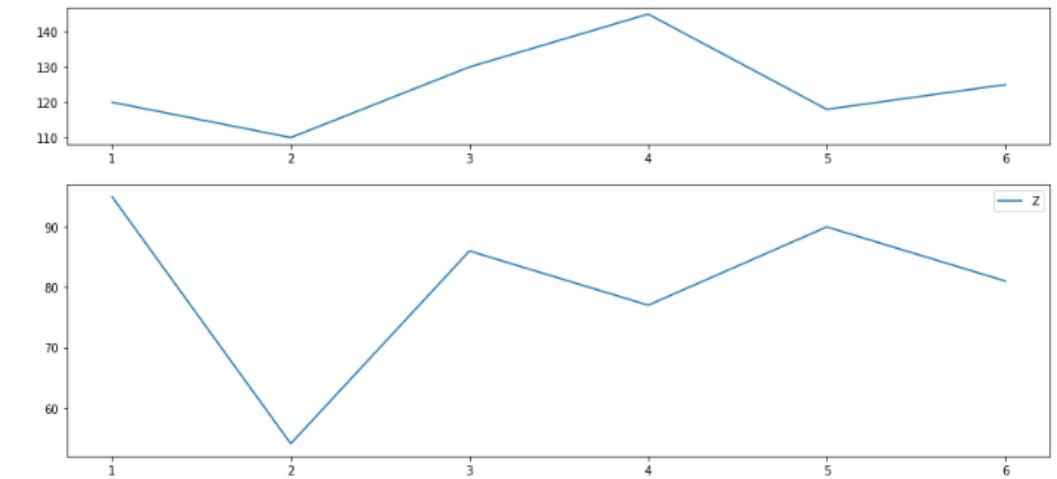
- mudar a proporção dos gráficos, podemos usar o **gridspec\_kw**={'height\_ratios': []}

```
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(15,7), gridspec_kw={'height_ratios': [1,2]})
```

```
ax[0].plot(base.X, base.Y, label="Y")  
ax[1].plot(base.X, base.Z, label="Z")
```

```
plt.legend()
```

```
plt.show()
```



## Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

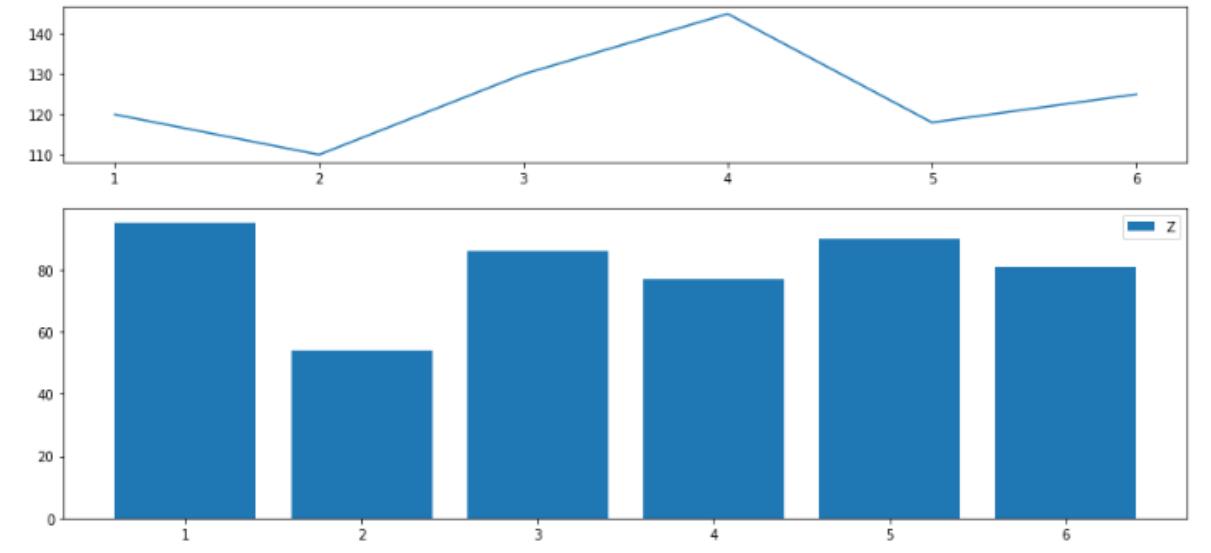
Para melhorar o visual do nosso gráfico podemos:

- Alterar o tipo de gráfico

```
ax[0].plot(base.X, base.Y, label="Y")
ax[1].bar(base.X, base.Z, label="Z")
```

```
plt.legend()
```

```
plt.show()
```



**Atenção:** ao trocar um tipo de gráfico, verifique os argumentos do gráfico.  
Por exemplo, gráfico de linha não possui largura.

## Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

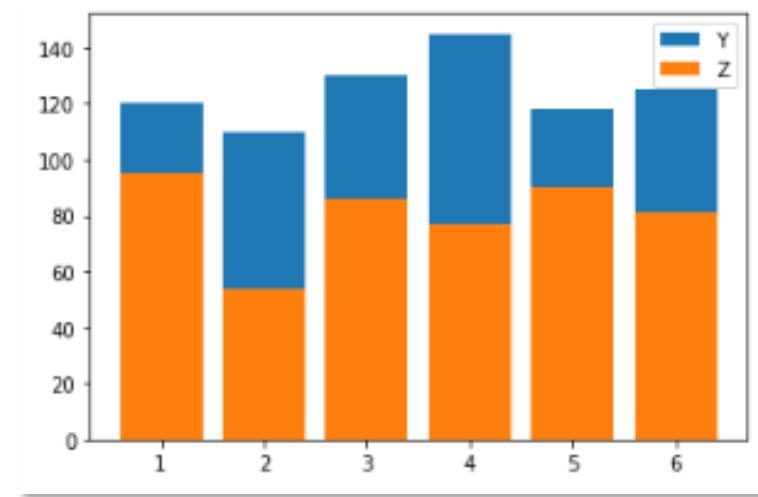
Vamos ajustar o gráfico de barras ao lado para as colunas ficarem lado a lado.

```
fig, ax = plt.subplots()
```

```
ax.bar(base.X,base.Y, label="Y")  
ax.bar(base.X,base.Z, label="Z")
```

```
plt.legend()
```

```
plt.show()
```



## Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Vamos ajustar as barras para ficarem lado a lado, para isso:

- Vamos definir a largura das barras (w)

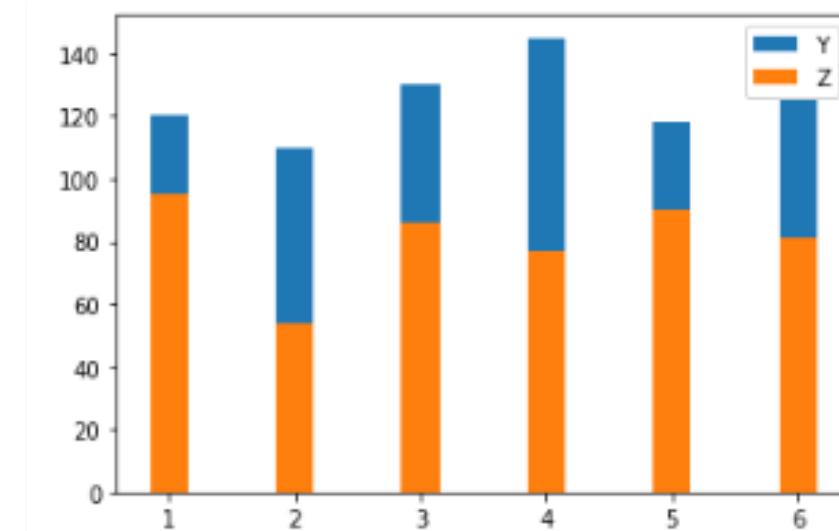
```
fig, ax = plt.subplots()
```

```
w=0.3
```

```
ax.bar(base.X,base.Y, label="Y", width=w)  
ax.bar(base.X,base.Z, label="Z", width=w)
```

```
plt.legend()
```

```
plt.show()
```



## Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Vamos ajustar as barras para ficarem lado a lado., para isso:

- Depois vamos deslocar o gráfico para os lados

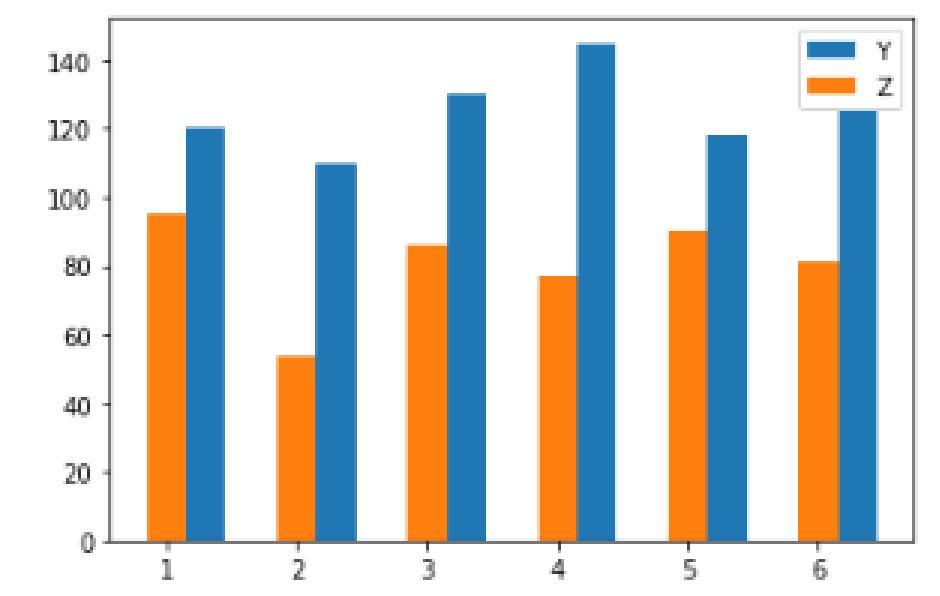
```
fig, ax = plt.subplots()
```

```
w=0.3
```

```
ax.bar(base.X+w,base.Y, label="Y", width=w)  
ax.bar(base.X,base.Z, label="Z", width=w)
```

```
plt.legend()
```

```
plt.show()
```



## Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Vamos ajustar as barras para ficarem lado a lado., para isso:

- Depois vamos deslocar o gráfico para os lados utilizando  $w/2$

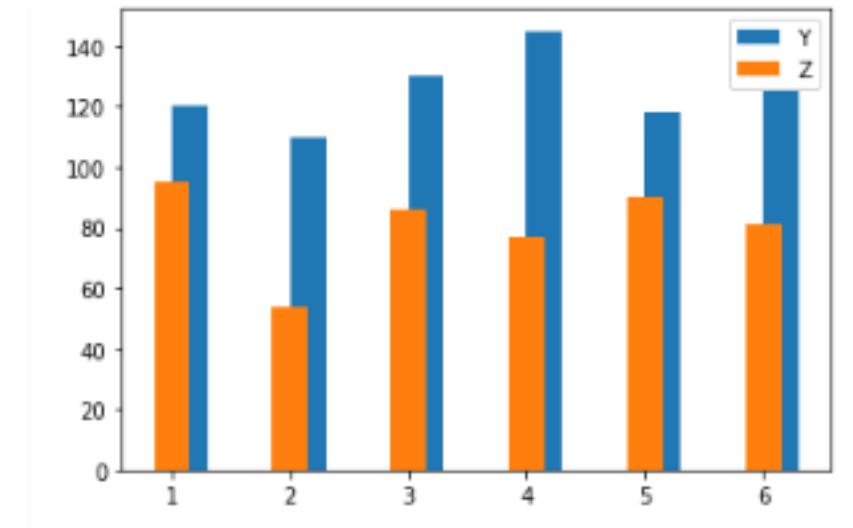
```
fig, ax = plt.subplots()
```

```
w=0.3
```

```
ax.bar(base.X+w/2,base.Y, label="Y", width=w)
ax.bar(base.X,base.Z, label="Z", width=w)
```

```
plt.legend()
```

```
plt.show()
```



## Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Vamos ajustar as barras para ficarem lado a lado., para isso:

- Depois vamos deslocar o gráfico para os lados utilizando  $w/2$

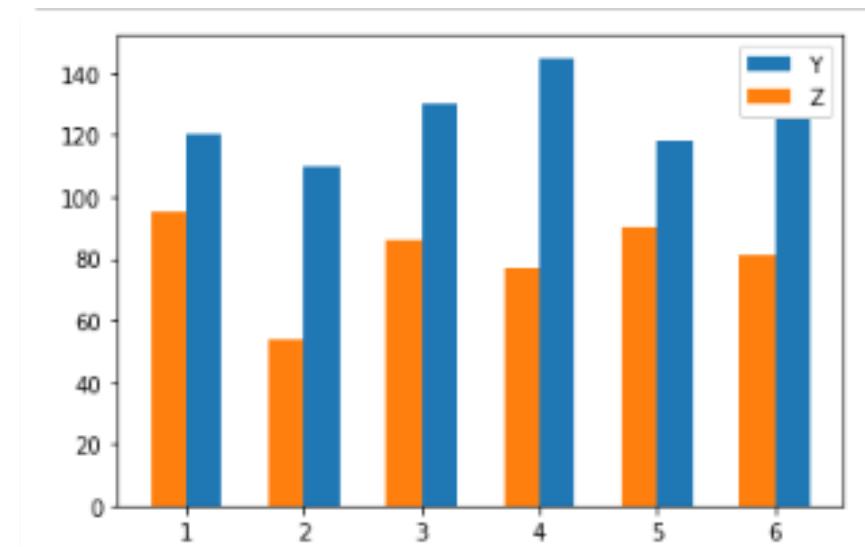
`fig, ax = plt.subplots()`

`w=0.3`

```
ax.bar(base.X+w/2,base.Y, label="Y", width=w)
ax.bar(base.X-w/2,base.Z, label="Z", width=w)
```

`plt.legend()`

`plt.show()`



## Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Agora que já vimos como **separar as barras** e **organizar melhor o nosso visual**, vamos aprender a colocar o rótulo de dados utilizando a função **annotate**.

O **annotate** é utilizado para colocar qualquer texto.

Como funciona essa função ?

A estrutura do **annotate** é:

```
.annotate("<Texto>",<posicao(x,y)>)
```

## Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Vamos fazer o rótulo de dados passo a passo no nosso código do gráfico:

```
fig, ax = plt.subplots()
```

```
w=0.3
```

```
ax.bar(base.X+w/2, base.Y, label="Y", width=w)  
ax.bar(base.X-w/2, base.Z, label="Z", width=w)
```

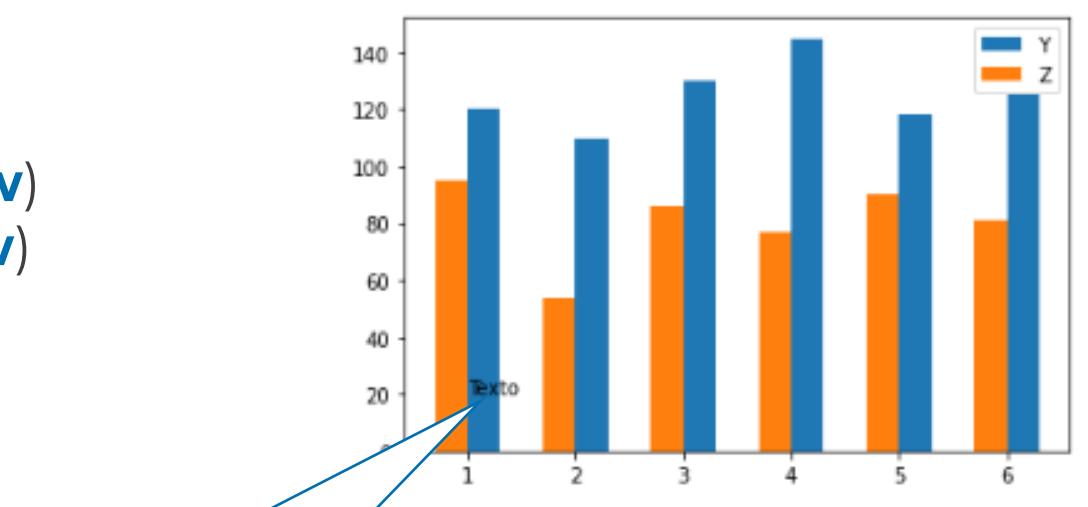
```
ax.annotate("Texto", (1,20))
```

```
plt.legend()
```

```
plt.show()
```

Vamos substituir  
primeiramente por valores  
aleatórios

"Texto" na posição (1,20)



## Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

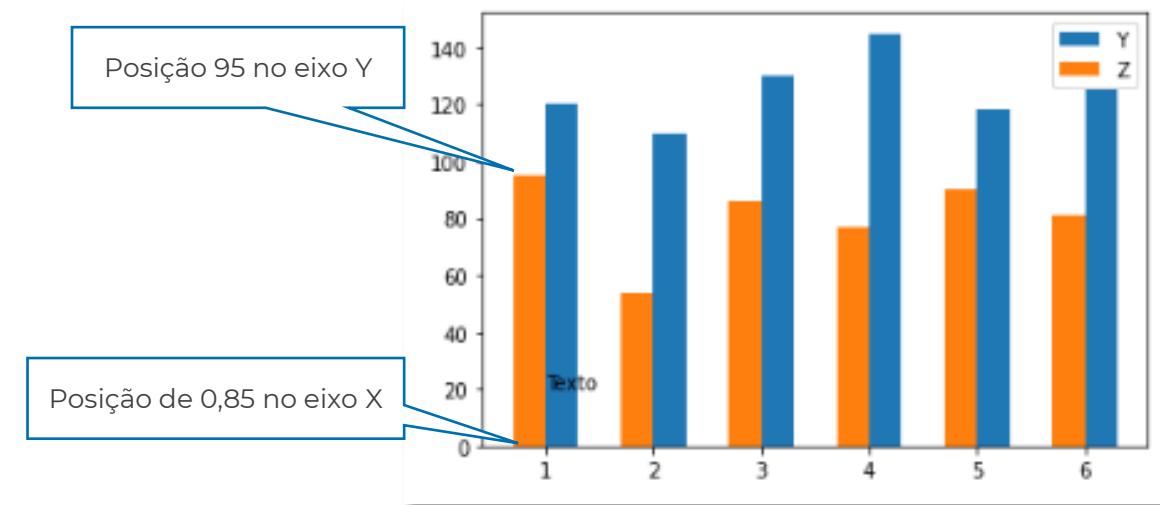
Para descobrir a posição do primeiro ponto laranja basta digitarmos o código:

**base.X[0]-w/2**

```
base.X[0]-w/2  
0.85
```

**base.Z[0]**

```
base.Z[0]  
95
```



Vamos colocar exatamente esses dois pontos como posição X e Y na função **annotate**.

## Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Substituindo os valores de X e Y pela posição do primeiro valor de Z no código:

```
fig, ax = plt.subplots()
```

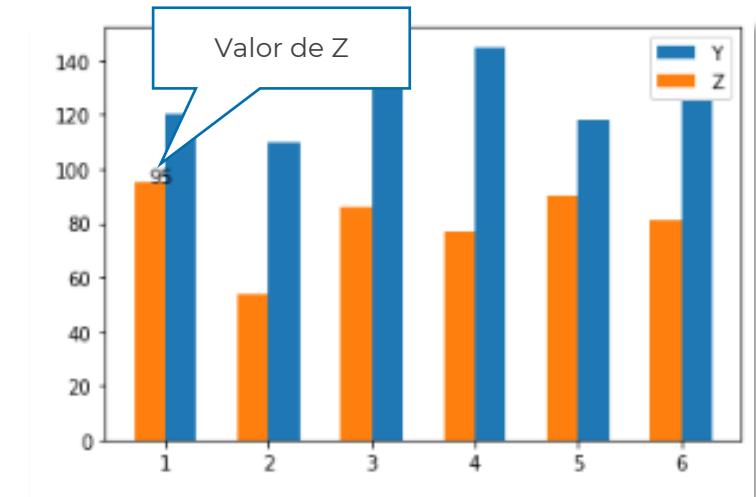
w=0.3

```
ax.bar(base.X+w/2, base.Y, label="Y", width=w)  
ax.bar(base.X-w/2, base.Z, label="Z", width=w)
```

```
ax.annotate(base.Z[0], (base.X[0]-w/2,base.Z[0]))
```

plt.legend()

plt.show()



## Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Usando o **for**, é possível aplicar o **annotate** para todas as barras.

O **for** percorre uma lista, por exemplo:

```
for i in range(0,6):  
    print(i)
```

```
for i in range(0,6):  
    print(i)
```

0  
1  
2  
3  
4  
5

Percorre todos os valores de 0 a 6

## Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Utilizando o for na função annotate

```
fig, ax = plt.subplots()
```

```
w=0.3
```

```
ax.bar(base.X+w/2, base.Y, label="Y", width=w)
```

```
ax.bar(base.X-w/2, base.Z, label="Z", width=w)
```

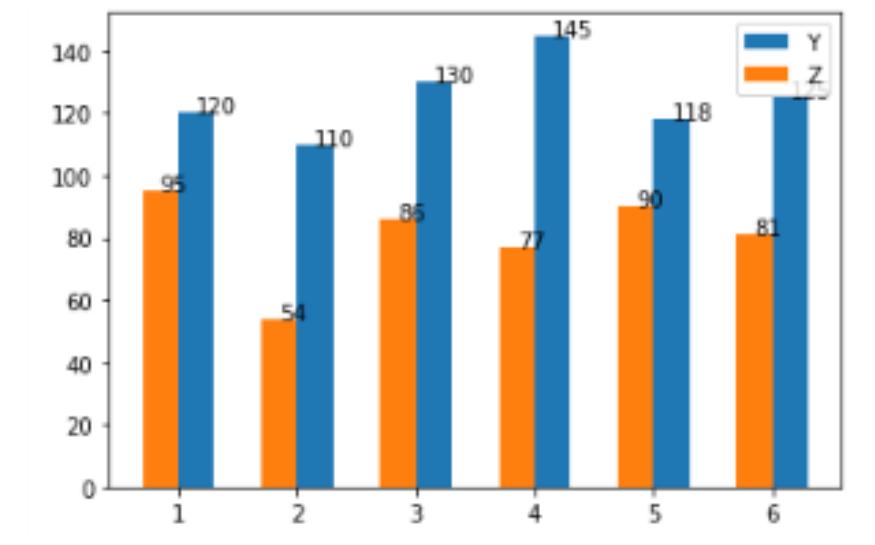
```
for i in range(0,6):
```

```
    ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]))
```

```
    ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]))
```

```
plt.legend()  
plt.show()
```

No eixo Y, ao invés de ser  
 $-w/2$  é  $+w/2$



## Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Por fim, vamos usar os parâmetros do `annotate` para melhorar a visualização:

`ha` → alinhamento horizontal

`xytext` → deslocamento x,y do texto em relação a um referencial

`textcoords` → referencial que vamos fazer o deslocamento acima

`fontsize` → tamanho da fonte

`fontweight` → colocando em negrito

## Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Vamos centralizar os valores utilizando o **ha**:

```
fig, ax = plt.subplots()
```

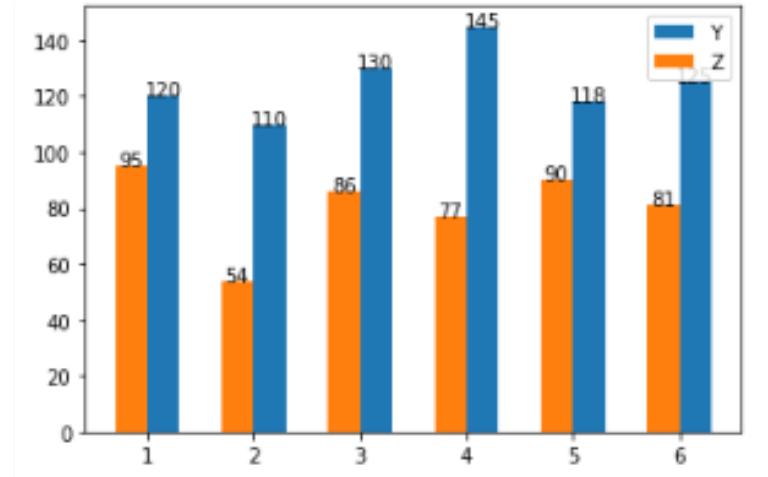
```
w=0.3
```

```
ax.bar(base.X+w/2, base.Y, label="Y", width=w)  
ax.bar(base.X-w/2, base.Z, label="Z", width=w)
```

```
for i in range(0,6):
```

```
    ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center")  
    ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]), ha="center")
```

```
plt.legend()  
plt.show()
```



Centralizando os valores

## Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

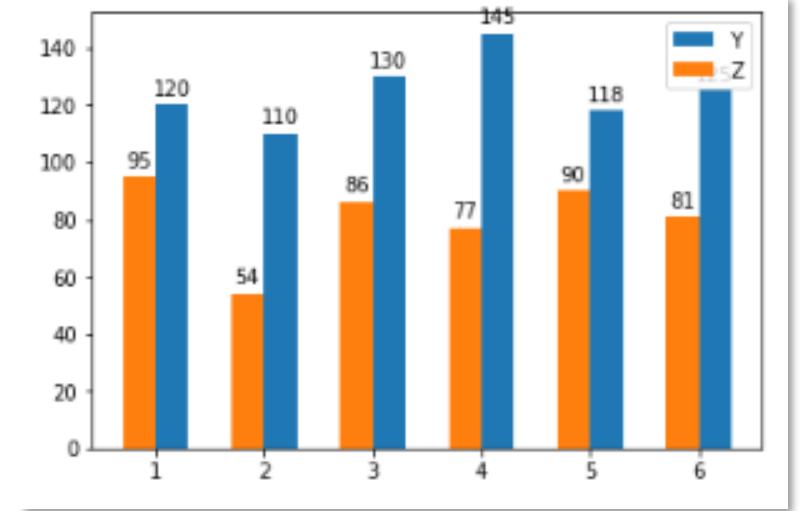
Vamos utilizar o **xytext** para deslocar a posição:

Estrutura: **xytext(x,y),textcoords="offset points"** ou **"offset pixels"**

Aplicando o **xytext** na parte do **annotate** do nosso código:

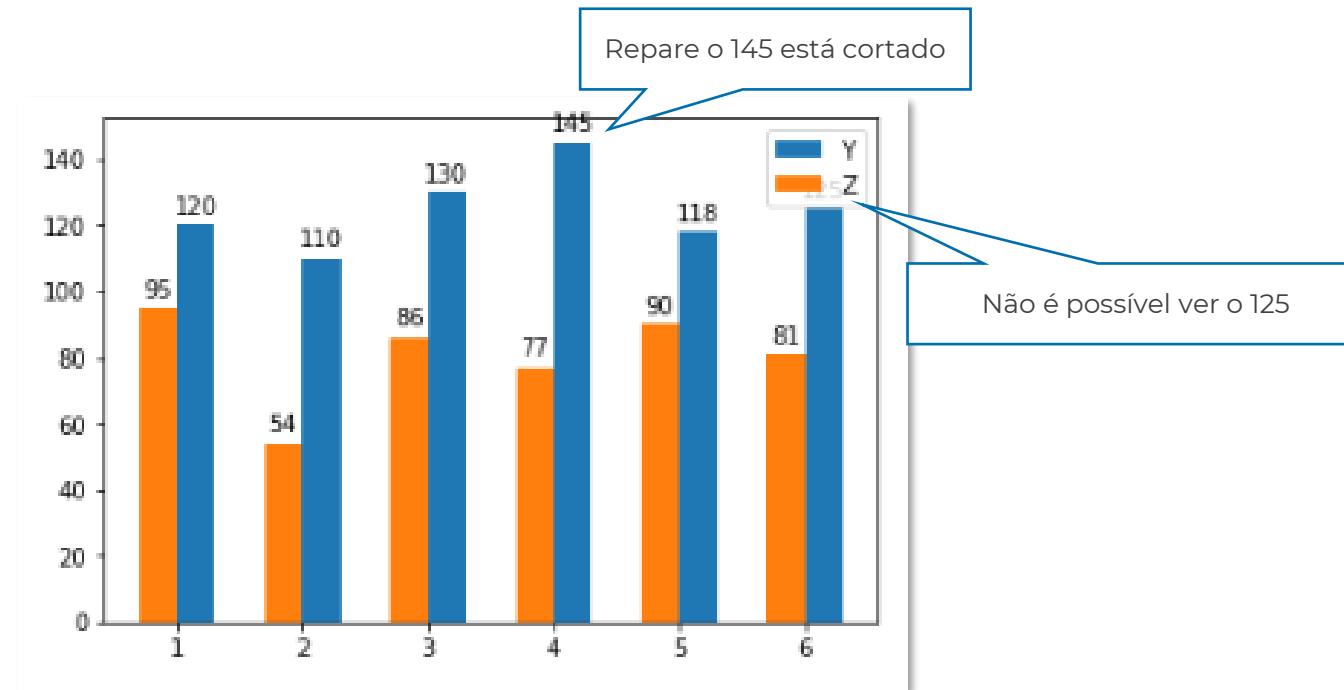
```
ax.annotate(base.Z[i],(base.X[i]-  
w/2,base.Z[i]),ha="center",xytext=(0,5),textcoords="offset  
points")
```

```
ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center",  
xytext=(0,5),textcoords="offset points")
```



## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Agora que colocamos o rótulo de dados, vamos melhorar essa visualização.



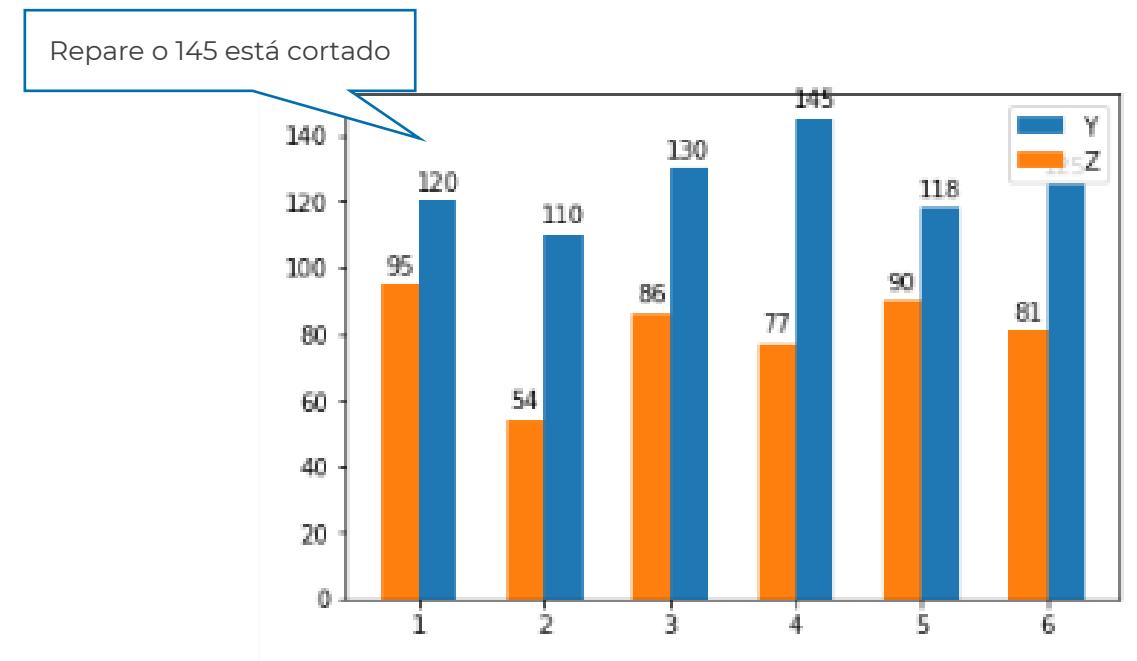
## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Vamos ajustar o limite do eixo para 150 utilizando o código **.set\_yticks** que vai permitir alterar o intervalo do eixo y.

Estrutura:

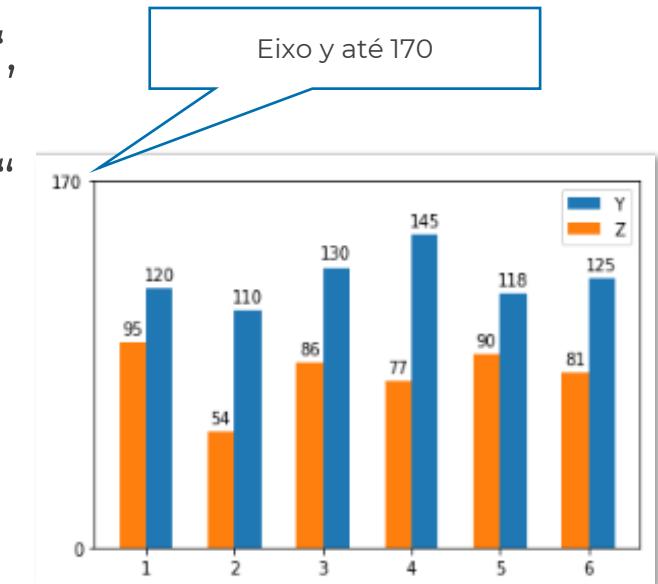
```
ax.set_yticks([início,final])
```

Vamos aplicar o **.set\_yticks** no nosso código:



## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

```
fig, ax = plt.subplots()  
w=0.3  
ax.bar(base.X+w/2, base.Y, label="Y", width=w)  
ax.bar(base.X-w/2, base.Z, label="Z", width=w)  
for i in range(0,6):  
    ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center",  
    xytext=(0,5),textcoords="offset points")  
    ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center"  
,xytext=(0,5),textcoords="offset points")  
  
ax.set_yticks([0,170])  
  
plt.legend()  
plt.show()
```



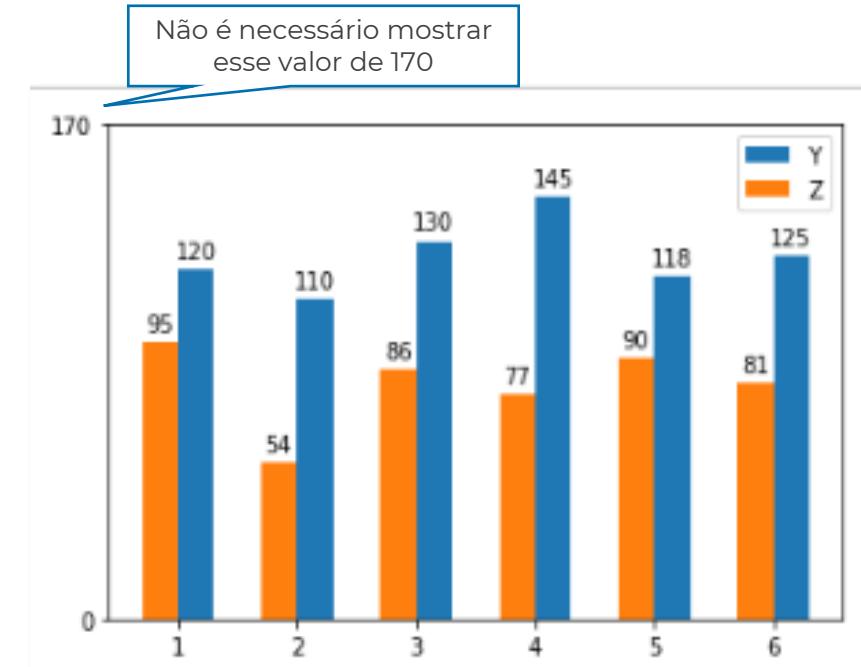
## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Já que estamos com todos os rótulos de dados, esse eixo y torna-se desnecessário. Para isso vamos utilizar a função **.yaxis.set\_visible**

Estrutura:

```
ax.yaxis.set_visible(False)
```

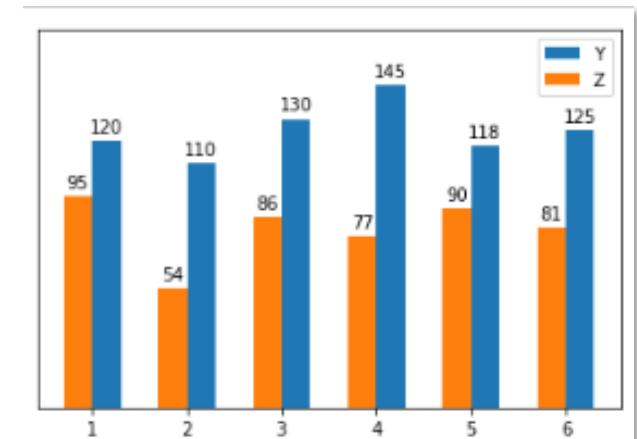
Vamos aplicar o **.yaxis.set\_visible** no nosso código:



## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

```
fig, ax = plt.subplots()  
w=0.3  
ax.bar(base.X+w/2, base.Y, label="Y", width=w)  
ax.bar(base.X-w/2, base.Z, label="Z", width=w)  
for i in range(0,6):  
    ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center",  
    xytext=(0,5),textcoords="offset points")  
    ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center"  
,xytext=(0,5),textcoords="offset points")  
  
ax.set_yticks([0,170])  
ax.yaxis.set_visible(False)  
plt.legend()  
plt.show()
```

Retira o valor do eixo Y



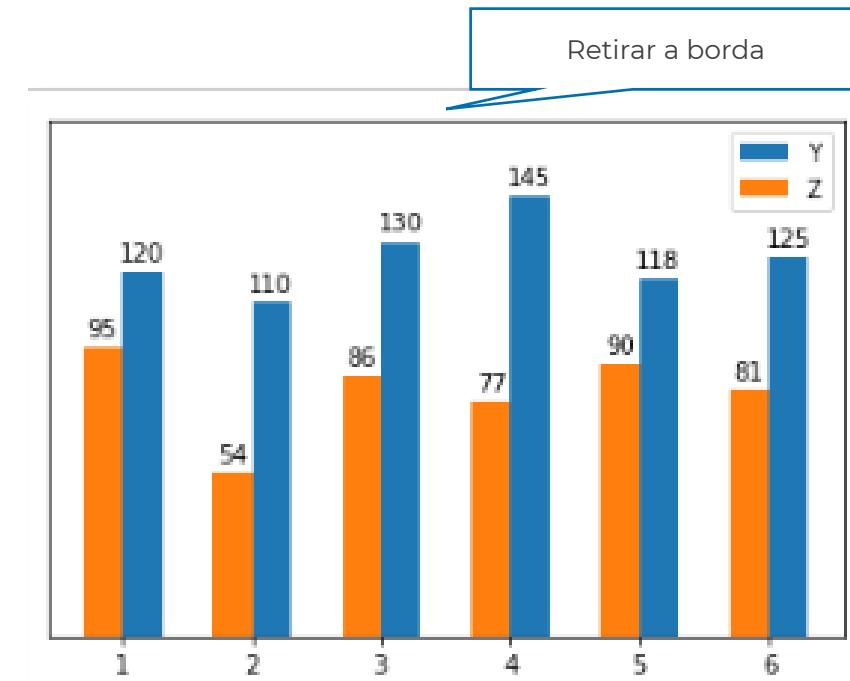
## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Além disso, podemos retirar a borda do gráfico utilizando a função `.spines[].set_visible`

Estrutura:

```
ax.spines[].set_visible(False)
```

Vamos aplicar essa função no nosso código:



## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Além disso, podemos retirar a borda do gráfico utilizando a função `.spines[] .set_visible`

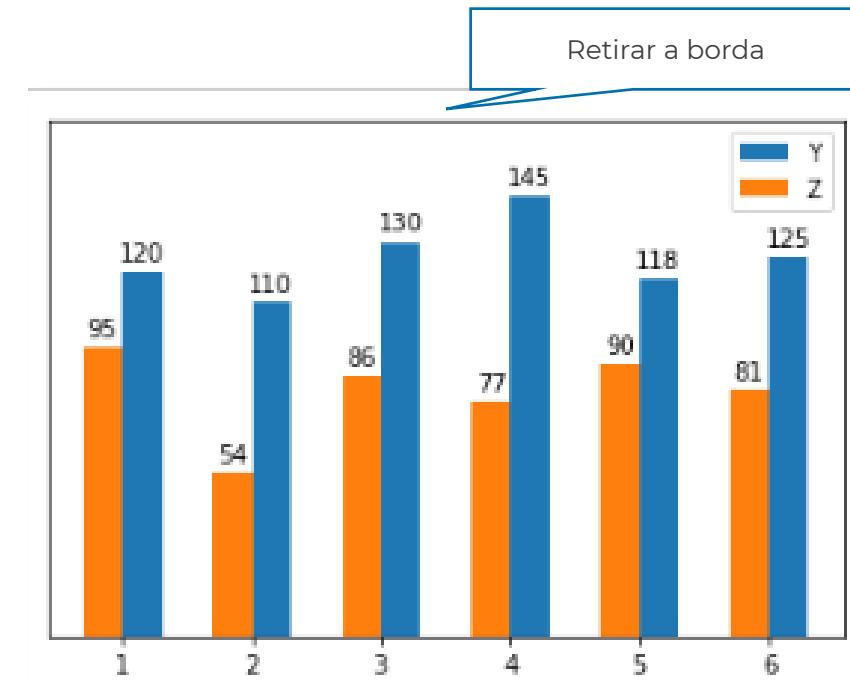
Estrutura:

```
ax.spines[ posição ].set_visible( False )
```

Posições:

- Right
- Top
- Left
- Bottom

Vamos aplicar essa função no nosso código:



## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Aplicando no código abaixo:

```
fig, ax = plt.subplots()

w=0.3

ax.bar(base.X+w/2,base.Y, label="Y",width=w)
ax.bar(base.X-w/2,base.Z, label="Z",width=w)

for i in range(0,6):
    ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center",xytext=(0,5),textcoords="offset points")
    ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center",xytext=(0,5),textcoords="offset points")

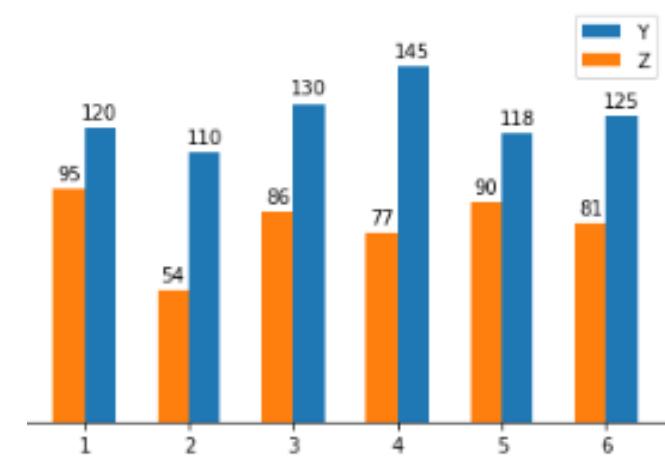
ax.set_yticks([0,170])
ax.yaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)

plt.legend()

plt.show()
```

Retirando a borda superior, direita  
e esquerda

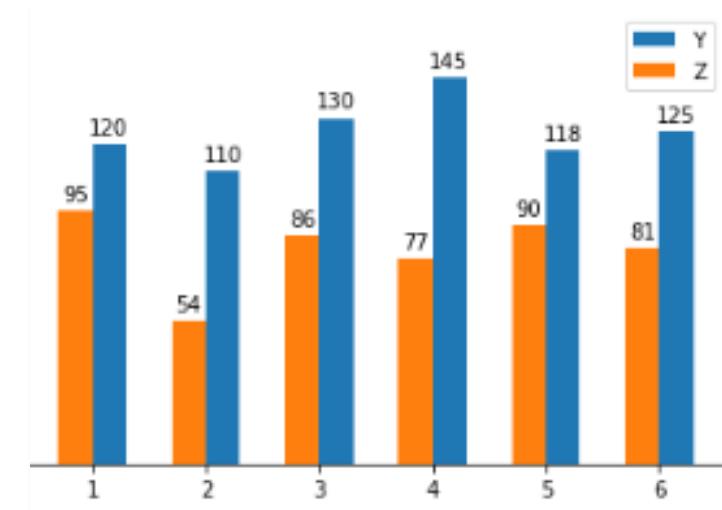


## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Vamos supor que os valores 5 e 6 do gráfico são projetados e o seu chefe pediu para você diferenciá-los, mudando as cores das barras.

Cores: azul (#C6F0F5) e laranja (#FFC174)

Para isso vamos separar a nossa base em duas.



## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Aplicando no código abaixo:

```
fig, ax = plt.subplots()

w=0.3

ax.bar(base.X[0:4]+w/2,base.Y[0:4], label="Y",width=w)
ax.bar(base.X[0:4]-w/2,base.Z[0:4], label="Z",width=w)
ax.bar(base.X[4:6]+w/2,base.Y[4:6],width=w,color="#C6F0F5")
ax.bar(base.X[4:6]-w/2,base.Z[4:6],width=w,color="#FFC174")

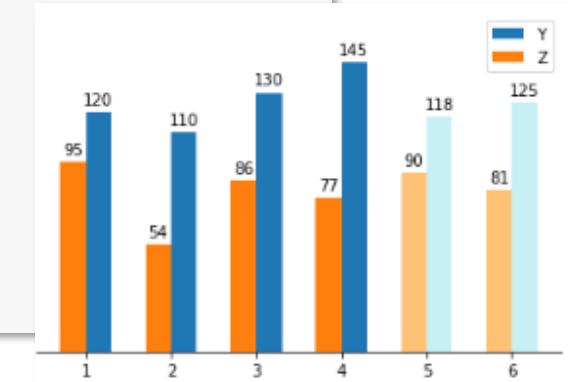
for i in range(0,6):
    ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center",xytext=(0,5),textcoords="offset points")
    ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center",xytext=(0,5),textcoords="offset points")

ax.set_yticks([0,170])
ax.yaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)

plt.legend()

plt.show()
```



## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

**EXTRA:** Adicionando retângulos no gráfico para destacar áreas

```
fig, ax = plt.subplots()
```

[esquerda,fundo,largura,altura]  
em fração da figura

```
ax1 = fig.add_axes([0.5,0.5,0.4,0.3])
```

```
rect = ax1.patch
```

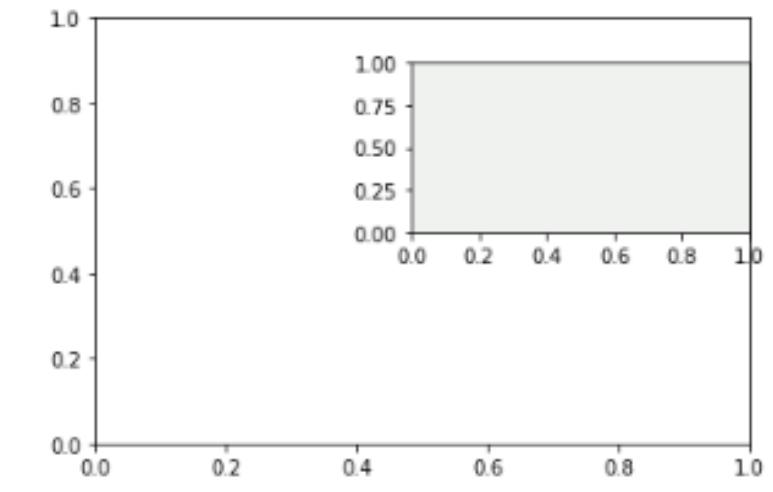
Facecolor altera a  
cor do retângulo

```
rect.set_facecolor('#E0E6DE')
```

```
rect.set_alpha(0.5)
```

```
plt.show()
```

Alpha altera a  
transparência



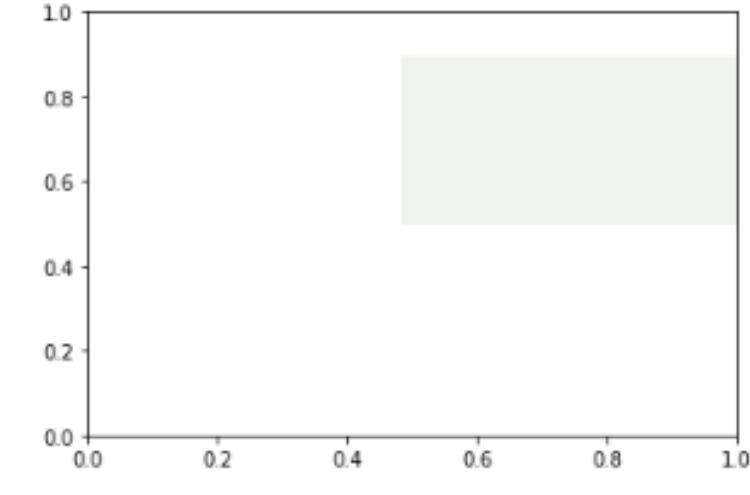
## Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

**EXTRA:** Adicionando retângulos no gráfico para destacar áreas

```
fig, ax = plt.subplots()  
ax1 = fig.add_axes([0.5,0.5,0.4,0.3])  
rect = ax1.patch  
rect.set_facecolor('#EOE6DE')  
rect.set_alpha(0.5)  
ax1.xaxis.set_visible(False)  
ax1.yaxis.set_visible(False)  
ax1.spines['top'].set_visible(False)  
ax1.spines['left'].set_visible(False)  
ax1.spines['right'].set_visible(False)  
ax1.spines['bottom'].set_visible(False)  
plt.show()
```

Retira os valores do retângulo

.set\_visible retira as bordas



## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

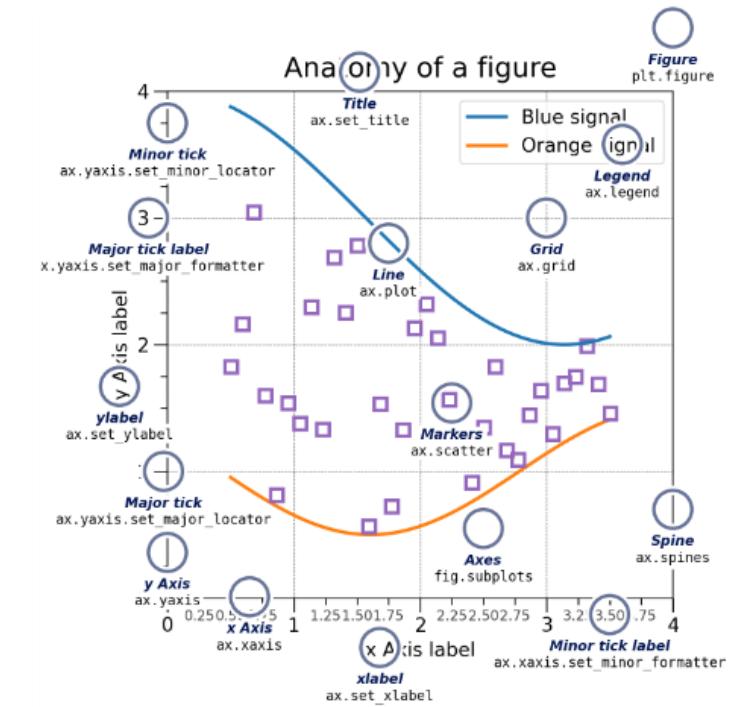
Agora vamos montar um visual de apresentação:

O primeiro passo que vamos fazer é importar as bibliotecas e vamos utilizar essa figura ao lado como base:

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```



## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

Vamos importar a nossa base visualizar utilizando **head**:

```
base = pd.read_excel("Visualização de dados no Python.xlsx")
```

```
base.head()
```

	Data	Pedidos	Entregas	EntregaNovoCD	TMAantigoCD	TMAnovoCD
0	2021-10-01	1230	1230	1230	1	1
1	2021-11-01	1235	1235	1235	1	1
2	2021-12-01	1280	1280	1280	1	1
3	2022-01-01	1310	1310	1310	1	1
4	2022-02-01	1572	1493	1493	2	2

Tempo médio de atendimento

## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

Vamos entender um pouco a nossa base digitando o código abaixo:

**base**

	Data	Pedidos	Entregas	EntregaNovoCD	TMAantigoCD	TMAnovoCD
0	2021-10-01	1230	1230	1230	1	1
1	2021-11-01	1235	1235	1235	1	1
2	2021-12-01	1280	1280	1280	1	1
3	2022-01-01	1310	1310	1310	1	1
4	2022-02-01	1572	1493	1493	2	2
5	2022-03-01	2122	1910	1910	3	3
6	2022-04-01	3183	2546	2546	6	4
7	2022-05-01	4775	3581	4300	8	3
8	2022-06-01	5251	3938	5200	8	3
9	2022-07-01	5779	4136	6100	8	2
10	2022-08-01	6355	4356	7000	7	2
11	2022-09-01	6994	4850	7900	8	1

Tempo médio de atendimento menor

## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

Antes de ir para os gráficos vamos verificar se existe alguma informação vazia na nossa base:

### base.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Data         12 non-null    datetime64[ns]
 1   Pedidos      12 non-null    int64  
 2   Entregas     12 non-null    int64  
 3   EntregaNovoCD 12 non-null    int64  
 4   TMAantigoCD 12 non-null    int64  
 5   TMANovoCD    12 non-null    int64  
dtypes: datetime64[ns](1), int64(5)
memory usage: 704.0 bytes
```

Nenhuma  
informação vazia

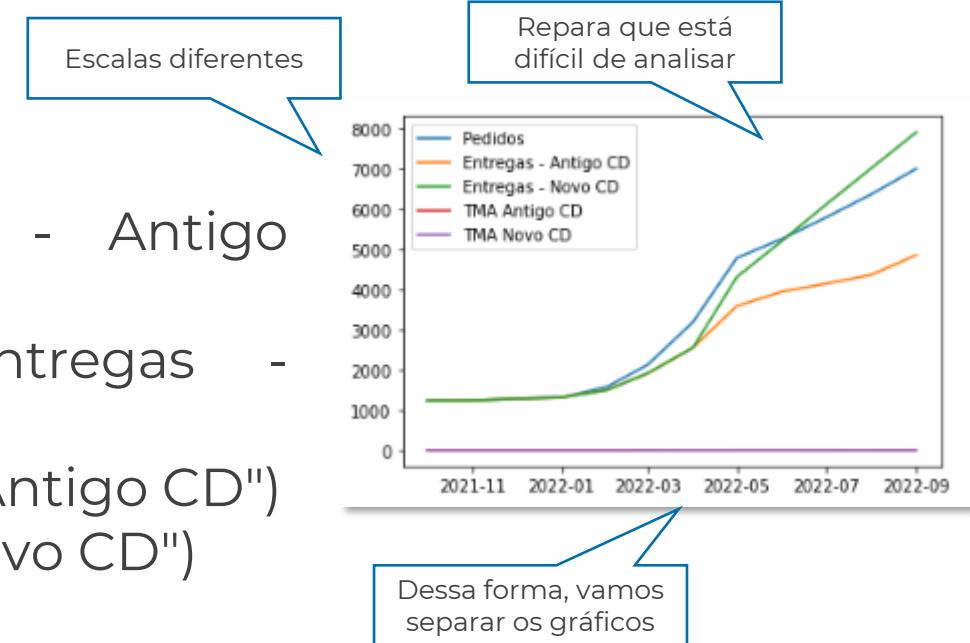
## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

Vamos criar um plot simples de todos os dados:

```
fig, ax = plt.subplots()
```

```
ax.plot(base.Data,base.Pedidos, label="Pedidos")
ax.plot(base.Data,base.Entregas, label="Entregas - Antigo CD")
ax.plot(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")
ax.plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")
ax.plot(base.Data,base.TMAnovoCD, label="TMA Novo CD")
```

```
plt.legend()
plt.show()
```



## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

```
fig, ax = plt.subplots(nrows=2, ncols=1)
```

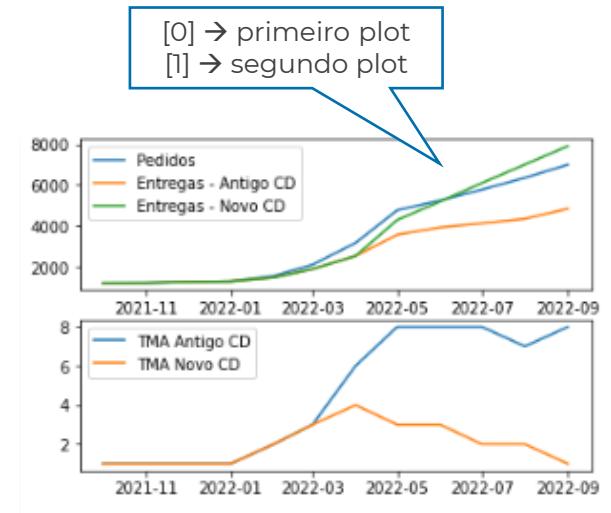
```
ax[0].plot(base.Data,base.Pedidos, label="Pedidos")  
ax[0].plot(base.Data,base.Entregas, label="Entregas - Antigo CD")
```

```
ax[0].plot(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")  
ax[0].legend()
```

```
ax[1].plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")
```

```
ax[1].plot(base.Data,base.TMANovoCD, label="TMA Novo CD")
```

```
ax[1].legend()  
plt.show()
```



## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

Vamos melhorar a visualização fazendo os seguintes itens:

- Vamos alterar a ordem dos gráficos (colocar o de TMA por cima)
- Podemos aumentar o tamanho dos gráficos
- Vamos aumentar proporcionalmente o gráfico de Pedidos x Entregas em relação ao de TMA

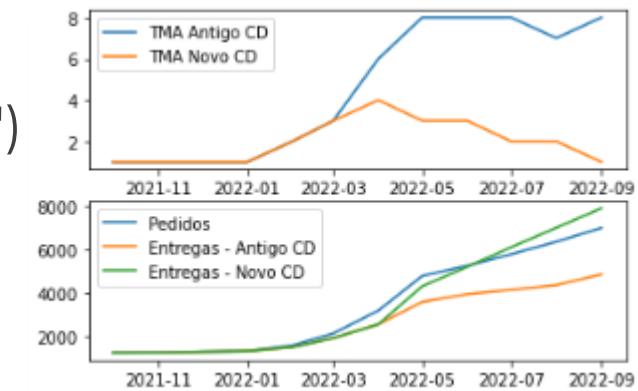
## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

- Para alterar a ordem do gráfico vamos trocar o que estava com [0] por um e vice-versa.

```
fig, ax = plt.subplots(nrows=2, ncols=1)
```

```
ax[1].plot(base.Data,base.Pedidos,label="Pedidos") ax[0].plot(base.Data,base.Entregas,  
label="Entregas - Antigo CD")  
ax[1].plot(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")  
ax[1].legend()
```

```
ax[0].plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")  
ax[0].plot(base.Data,base.TMAnovoCD, label="TMA Novo CD")  
ax[0].legend()  
plt.show()
```



## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

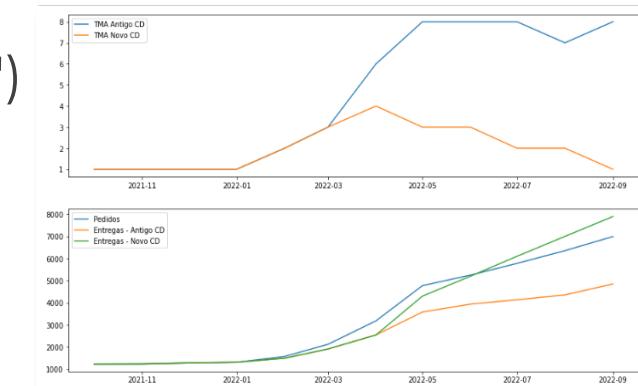
- Para aumentar o tamanho do gráfico:

```
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(15,9))
```

Aumenta o  
tamanho do gráfico

```
ax[1].plot(base.Data,base.Pedidos,label="Pedidos") ax[0].plot(base.Data,base.Entregas,  
label="Entregas - Antigo CD")  
ax[1].plot(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")  
ax[1].legend()
```

```
ax[0].plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")  
ax[0].plot(base.Data,base.TMANovoCD, label="TMA Novo CD")  
ax[0].legend()  
plt.show()
```



## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

- Para aumentar proporcionalmente o tamanho do gráfico:

```
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(15,9), gridspec_kw={'height_ratios': [1,3]})
```

```
ax[1].plot(base.Data,base.Pedidos,label="Pedidos") ax[0].plot(base.Data,base.Entregas,  
label="Entregas - Antigo CD")
```

```
ax[1].plot(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")  
ax[1].legend()
```

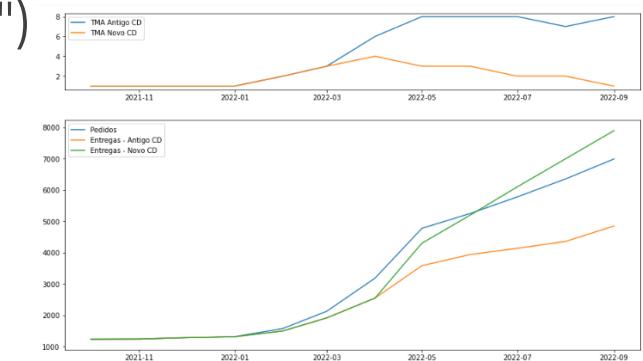
```
ax[0].plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")
```

```
ax[0].plot(base.Data,base.TMANovoCD, label="TMA Novo CD")
```

```
ax[0].legend()
```

```
plt.show()
```

Aumenta o tamanho do  
gráfico proporcionalmente



## Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

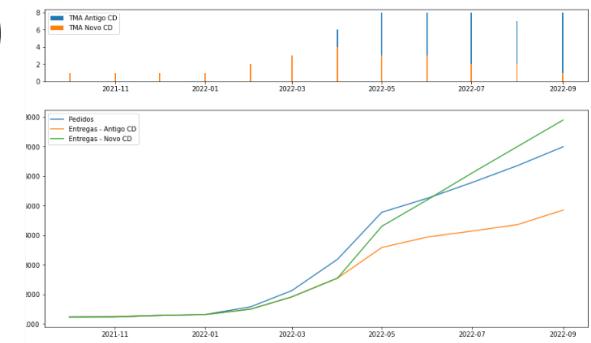
- Para alterar o tipo de gráfico basta trocar **plot** por **bar**:

```
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(15,9), gridspec_kw={'height_ratios': [1,3]})
```

```
ax[1].bar(base.Data,base.Pedidos,label="Pedidos") ax[0].plot(base.Data,base.Entregas,  
label="Entregas - Antigo CD")
```

```
ax[1].bar(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")  
ax[1].legend()
```

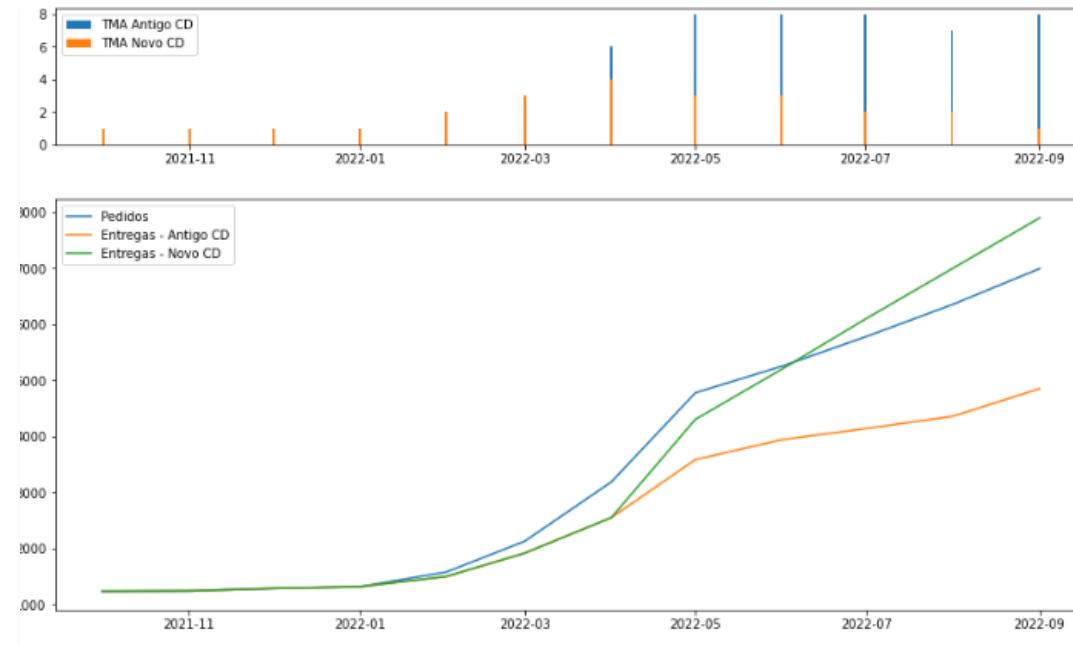
```
ax[0].plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")  
ax[0].plot(base.Data,base.TMANovoCD, label="TMA Novo CD")  
ax[0].legend()  
plt.show()
```



## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Repare que no gráfico cada barra é um dia, como vamos melhorar isso?

Vamos transformar essas datas em valores inteiros.



## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

```
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(15,9),gridspec_kw={'height_ratios': [1,3]})
```

```
ax[0].bar(np.arange(len(base)),base.TMAantigoCD, label="TMA Antigo CD")
```

```
ax[0].bar(np.arange(len(base)),base.TMAnovoCD, label="TMA Novo CD")
```

```
ax[0].legend()
```

Funciona mesmo se a nossa  
base aumentar

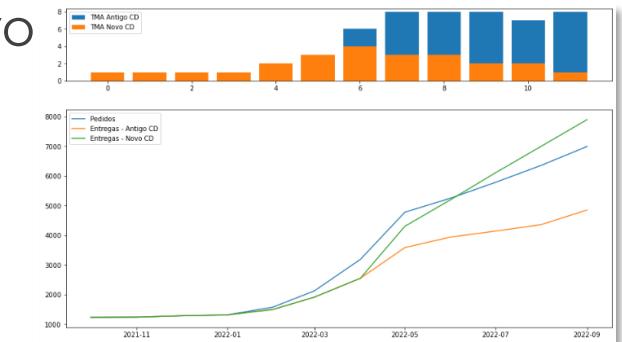
```
ax[1].plot(base.Data,base.Pedidos, label="Pedidos")
```

```
ax[1].plot(base.Data,base.Entregas, label="Entregas - Antigo CD")
```

```
ax[1].plot(base.Data, base.EntregaNovoCD, label="Entregas - Novo")
```

```
ax[1].legend()
```

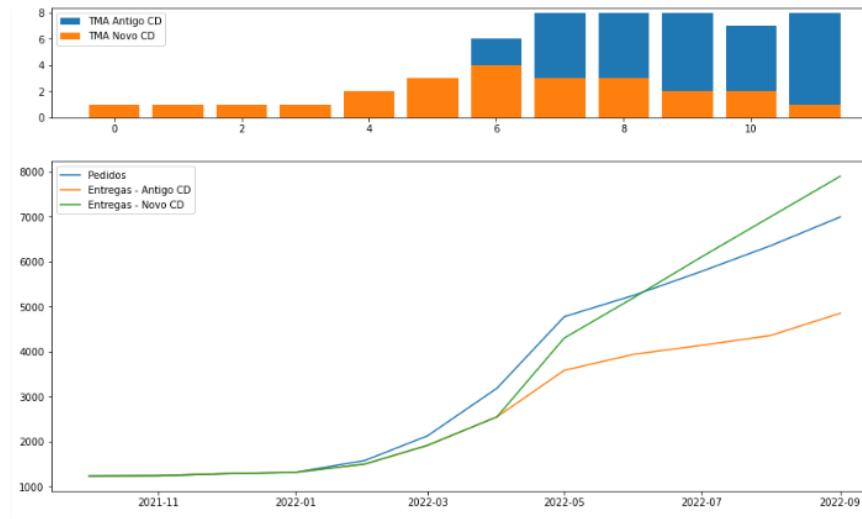
```
plt.show()
```



## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Agora podemos definir o width da coluna e deslocar as colunas em relação a esse width.

Vamos supor um valor de **wid** = 0.4 e aplicar na primeira parte do código.



## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

```
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(15,9),gridspec_kw={'height_ratios': [1,3]})
```

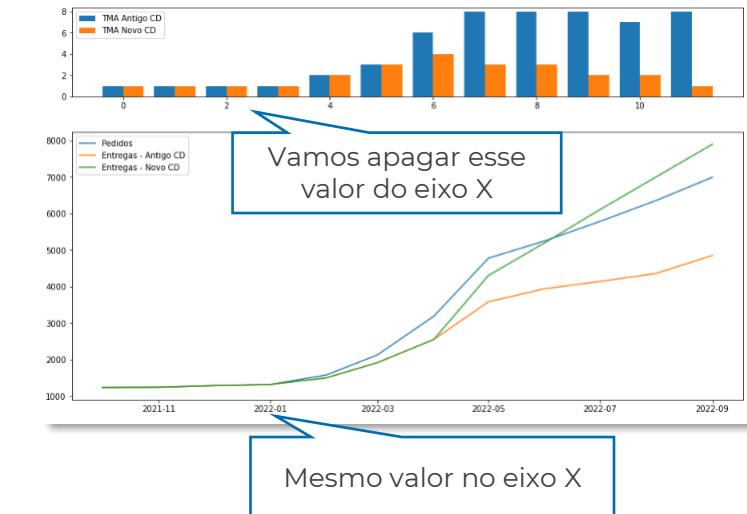
wid = 0.4

```
ax[0].bar(np.arange(len(base))-wid/2,base.TMAantigoCD,width=wid,label="TMA Antigo CD")
```

```
ax[0].bar(np.arange(len(base))+wid/2,base.TMANovoCD,width=wid,label="TMA Novo CD")
```

```
ax[0].legend()
```

```
plt.show()
```



## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

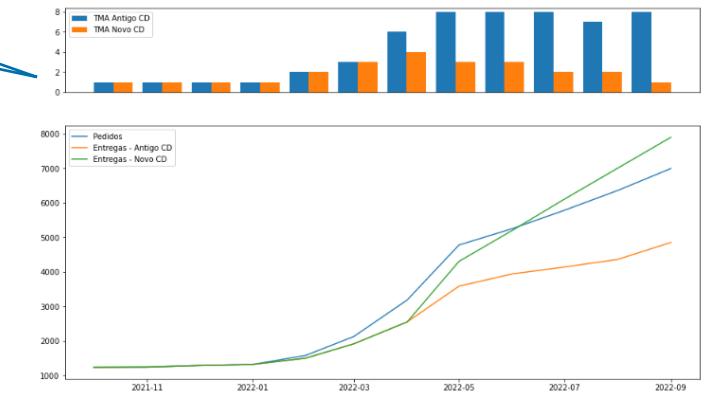
```
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(15,9),gridspec_kw={'height_ratios': [1,3]})
```

**wid** = 0.4

```
ax[0].bar(np.arange(len(base))-  
wid/2,base.TMAantigoCD,width=wid,label="TMA Antigo  
CD")  
ax[0].bar(np.arange(len(base))+wid/2,base.TMANovoCD,  
width=wid,label="TMA Novo CD")  
ax[0].legend()  
ax[0].xaxis.set_visible(False)  
plt.show()
```

Eixo y não está tão claro

Retira o no eixo X



## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados.

- ha → alinhamento horizontal
- xytext → deslocamento x,y do texto em relação a um referencial
- Textcoords → referencial que vamos fazer o deslocamento acima
- fontsize → tamanho da fonte
- fontweight → colocando em negrito

## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados. Para a primeira parte do código:

**for i in np.arange(0,12):**

    ax[0].**annotate('{:.0f}'.format(base.TMAantigoCD[i],(i\*wid)/2,base.TMAantigoCD[i]),**

**ha="center",**

        Alinhamento

        Formatando o número

        Colocando a posição do elemtno

**xytext=(0,3),**

        Deslocamento x,y do texto

**textcoords="offset points",**

        Referencial que vamos  
        fazer o deslocamento

**fontsize=10,**

        Tamanho da fonte

**fontweight='bold')**

        Colocar em negrito

## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados. Para a primeira parte do código:

**for i in np.arange(0,12):**

```
    ax[0].annotate('{}'.format(base.TMAantigoCD[i]),(iwid/2,base.TMAantigoCD[i]),
```

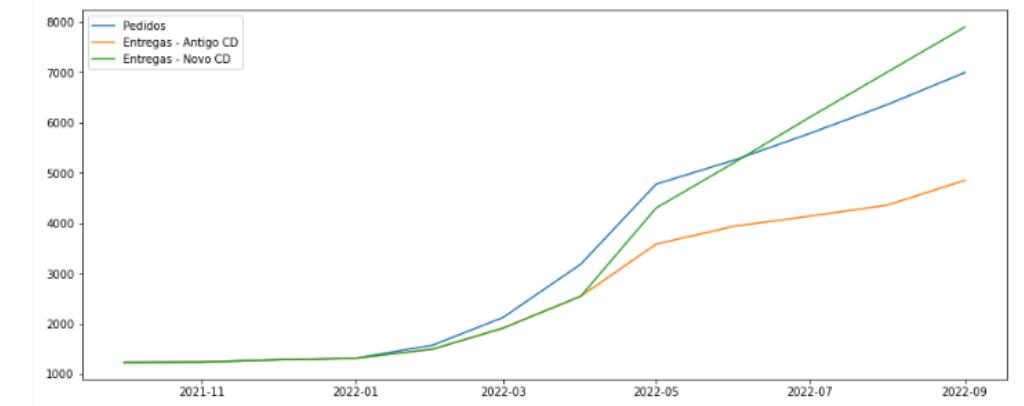
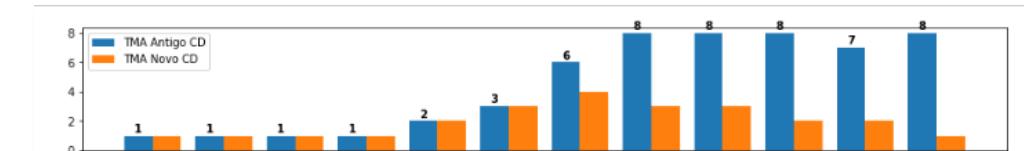
```
        ha="center",
```

```
        xytext=(0,3),
```

```
        textcoords="offset points",
```

```
        fontsize=10,
```

```
        fontweight='bold')
```



## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados. Para a segunda parte do código:

**for i in np.arange(0,12):**

    ax[0].**annotate('{:.0f}'.format(base.TMAnovoCD[i],(i\*wid/2,base.TMAnovoCD[i]),**

**ha="center",**

        Alinhamento

        Formatando o número

        Colocando a posição do elemento

**xytext=(0,3),**

        Deslocamento x,y do texto

**textcoords="offset points",**

        Referencial que vamos  
        fazer o deslocamento

**fontsize=10,**

        Tamanho da fonte

**fontweight='bold')**

        Colocar em negrito

## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados. Para a segunda parte do código:

**for i in np.arange(0,12):**

```
    ax[0].annotate('{}'.format(base.TMAnovoCD[i]),(iwid/2,base.TMAnovoCD[i]),
```

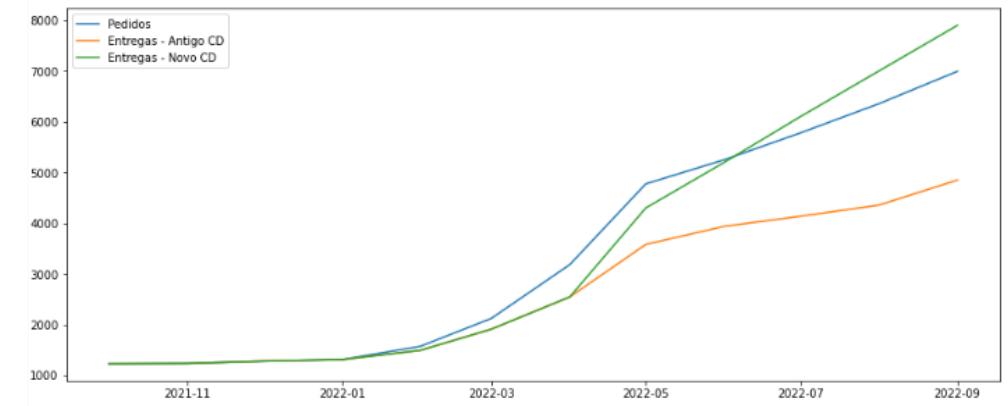
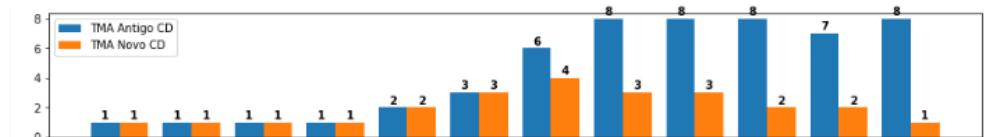
**ha="center",**

**xytext=(0,3),**

**textcoords="offset points",**

**fontsize=10,**

**fontweight='bold')**



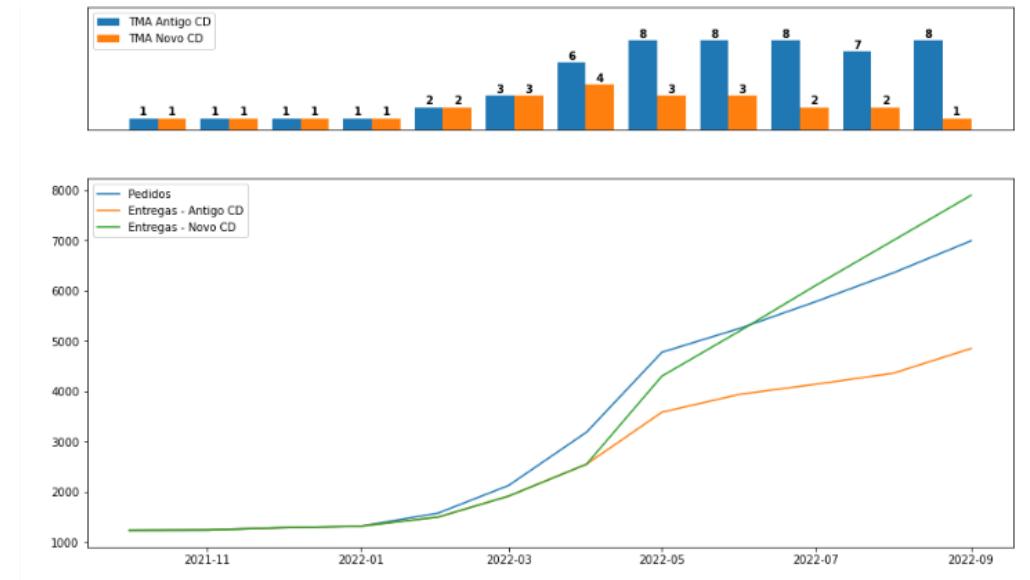
## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar aumentar o tamanho do eixo y para ajustar o gráfico e retirando os valores:

Para isso colocamos no nosso código os seguintes comando:

```
ax[0].set_yticks(np.arange(0,12))
```

```
ax[0].yaxis.set_visible(False)
```



## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

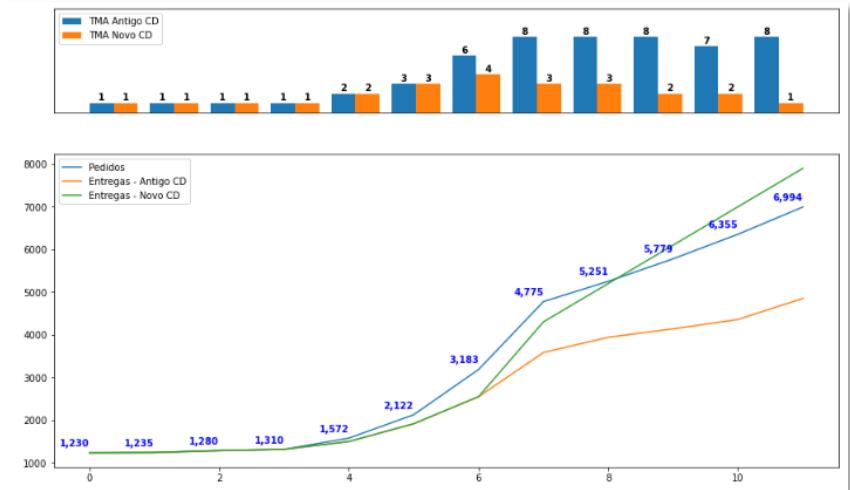
Da mesma forma que utilizamos o `annotate` para o gráfico de barra vamos utilizar para o de linha. Para a primeira linha do gráfico:

```
for i in np.arange(0,12):
```

```
    ax[1].annotate('{:.0f}'.format(base.Pedidos[i]),(i,base.Pedidos[i]),
                   ha="right",
                   va="top",
                   xytext=(0,+15),
                   textcoords="offset points",
                   fontsize=10,
                   fontweight='bold',
                   color="blue")
```

Posição do rótulo em  
relação a linha

Cor do rótulo

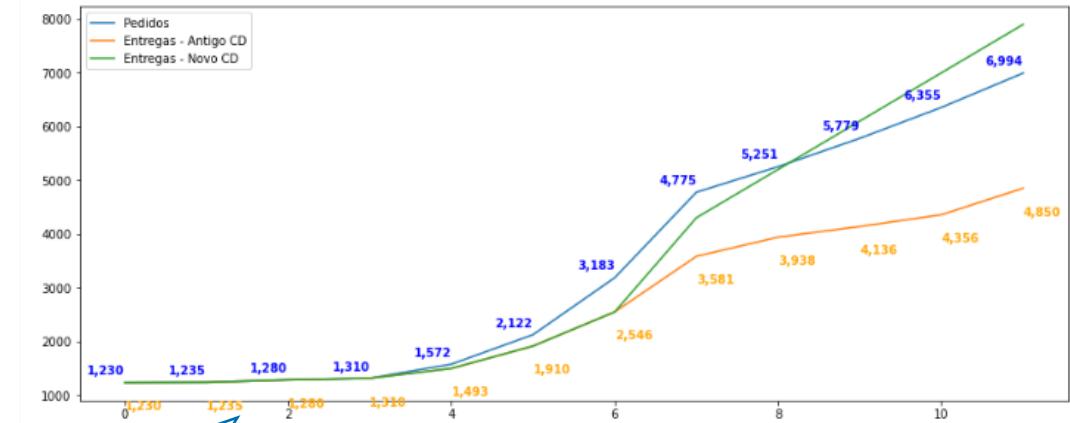
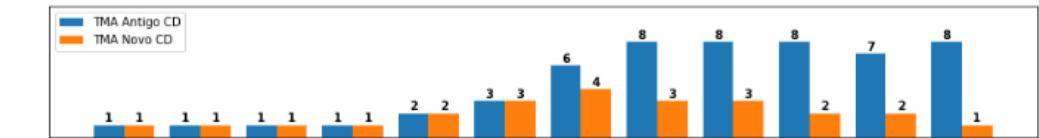


## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para a segunda linha do gráfico:

```
for i in np.arange(0,12):
```

```
    ax[1].annotate('{:.0f}'.format(base.EntregaNovoCD[i]),(i,base.EntregaNovoCD[i]),
                  ha="left",
                  va="top",
                  xytext=(0,-15),
                  textcoords="offset points",
                  fontsize=10,
                  fontweight='bold',
                  color="orange")
```



Rpare que esses números  
estão poluindo o gráfico

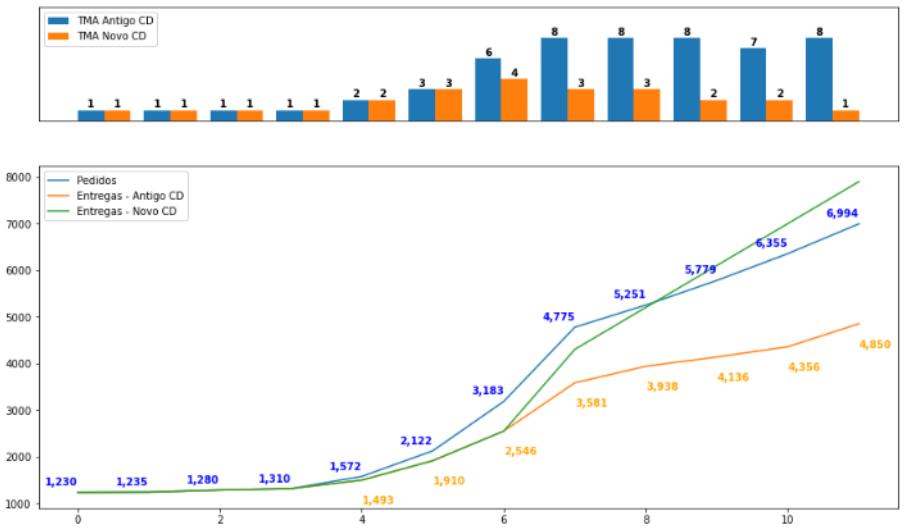
## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para a segunda linha do gráfico:

**for i in np.arange(4,12):**

Vamos alterar a primeira posição do range

```
ax[1].annotate('{:.0f}'.format(base.EntregaNovoCD[i]),
(i,base. EntregaNovoCD[i]),
ha="left",
va="top",
xytext=(0,-15),
textcoords="offset points",
fontsize=10,
fontweight='bold',
color="orange")
```

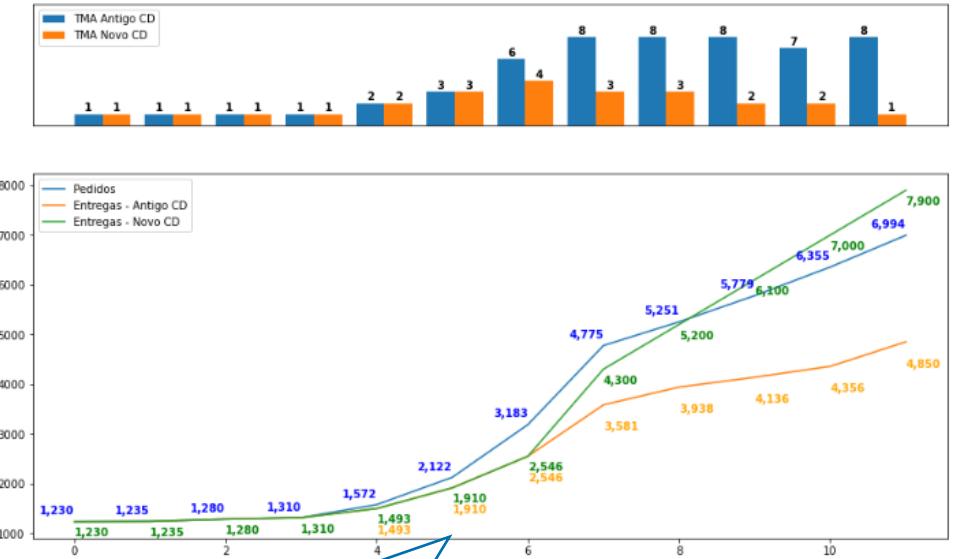


## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para a terceira linha do gráfico:

**for i in np.arange(0,12):**

```
ax[1].annotate('{:.0f}'.format(base.EntregaNovoCD  
[i]),(i,base. EntregaNovoCD[i]),  
ha="left",  
va="bottom",  
xytext=(0,-15),  
textcoords="offset points",  
fontsize=10,  
fontweight='bold',  
color="green")
```



Repare que esses números  
são iguais ao da linha verde

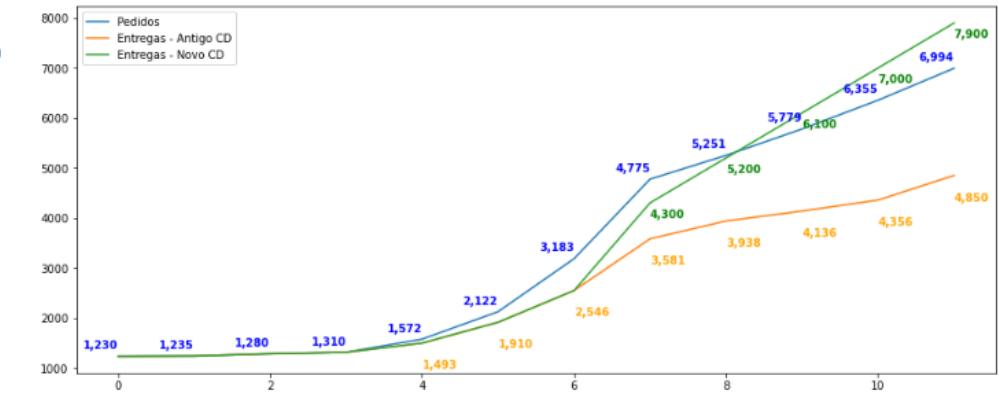
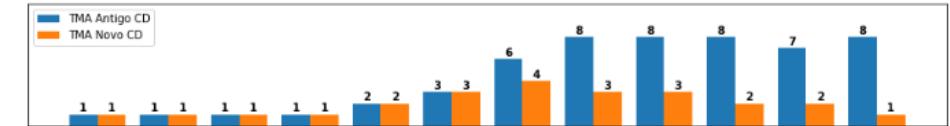
## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para a terceira linha do gráfico:

**for i in np.arange(7,12):**

Vamos alterar a primeira posição do range

```
ax[1].annotate('{:.0f}'.format(base.EntregaNovoCD  
[i]),(i,base. EntregaNovoCD[i]),  
ha="left",  
va="bottom",  
xytext=(0,-15),  
textcoords="offset points",  
fontsize=10,  
fontweight='bold',  
color="green")
```

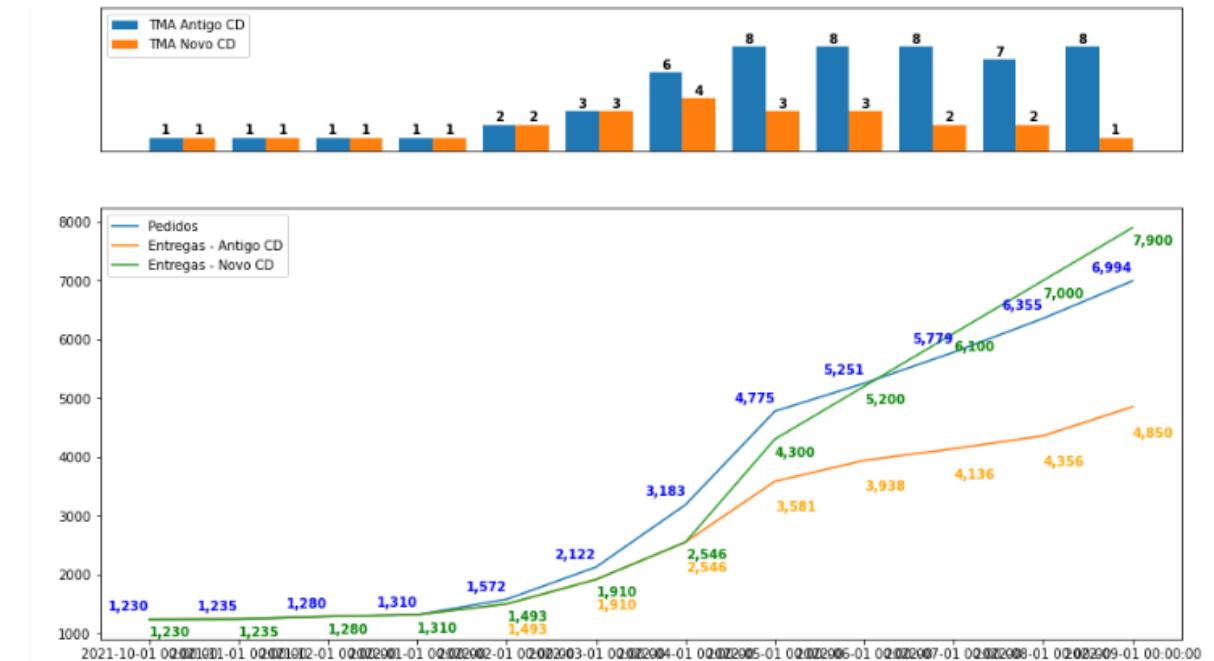


## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Como transformamos a data para número, para ela voltar a aparecer no nosso gráfico como data precisamos utilizar o código abaixo:

```
ax[1].xaxis.set_ticks(np.arange(0,12))
```

```
ax[1].set_xticklabels(base.Data)
```

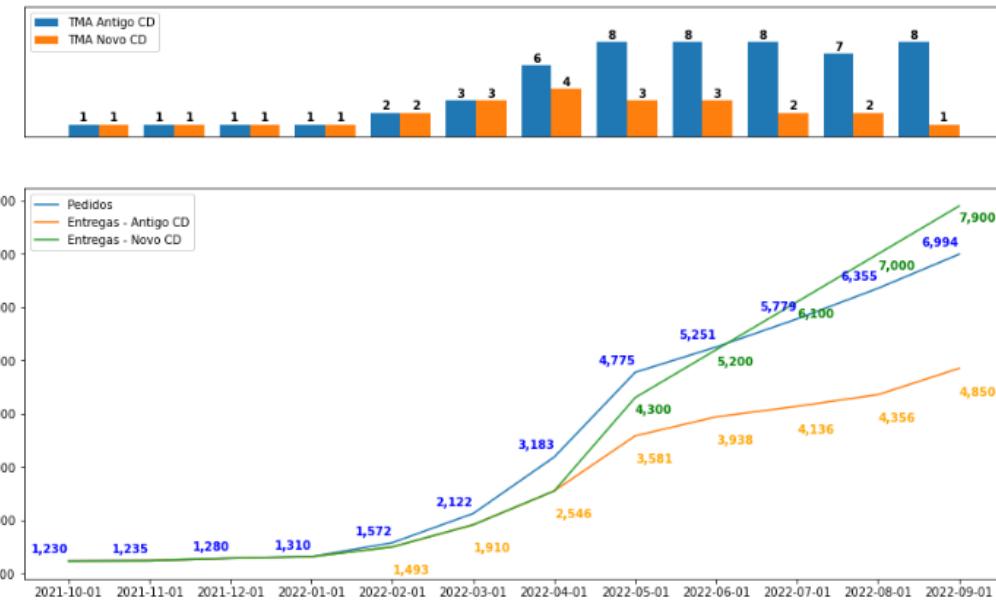


Dessa forma não é possível visualizar as datas

## Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para melhorar a visualização das datas vamos transformar ela em texto.

```
ax[1].set_xticklabels(np.datetime_as_string(base.Data.values,unit='D'))
```



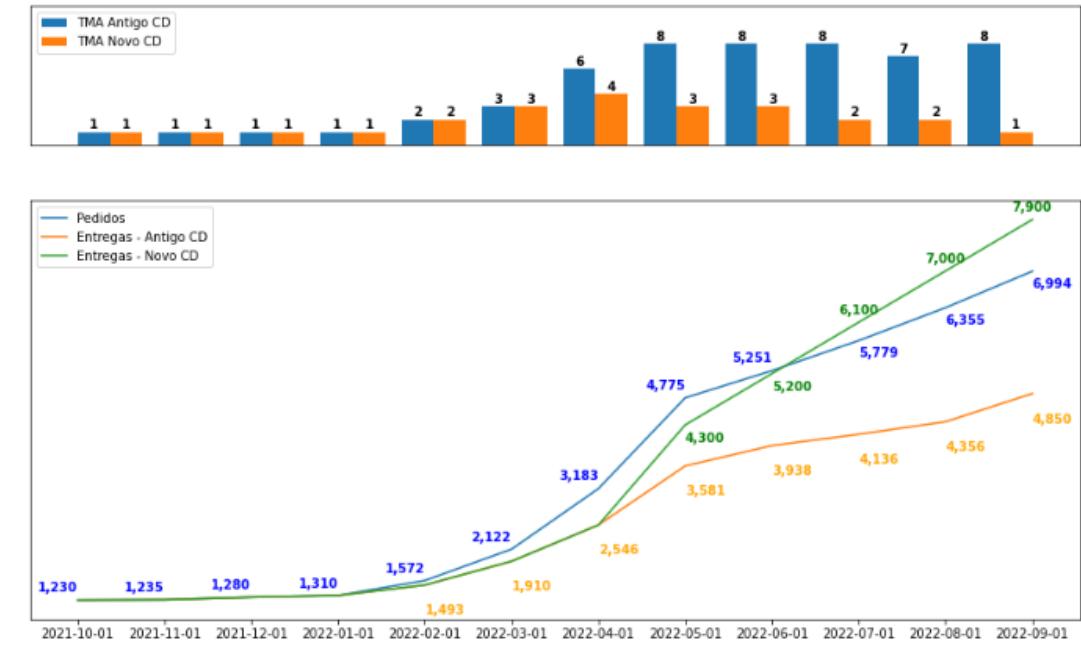
Unida D para ficar ano, dia e mês

## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

Agora que já colocamos os rótulos de dados para o gráfico de linha podemos melhorar o visual dele:

- Retirar o eixo y utilizando o código abaixo:

```
ax[1].yaxis.set_visible(False)
```



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

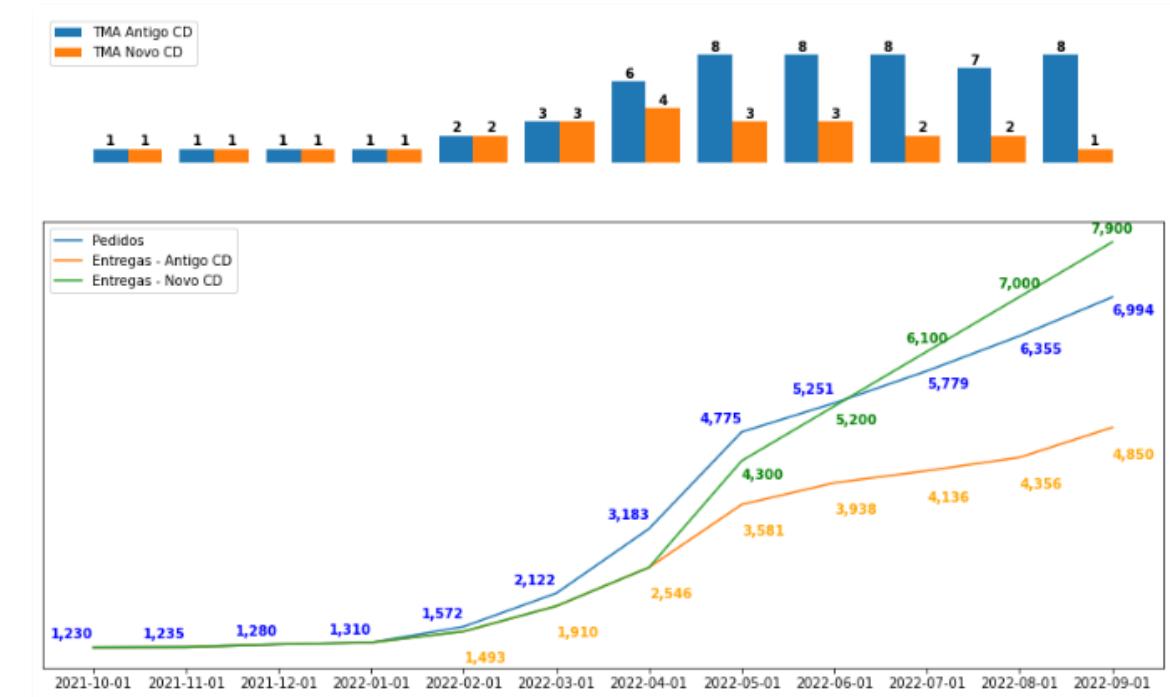
- Retirando a borda do gráfico de barra:

```
ax[0].spines['top'].set_visible(False)
```

```
ax[0].spines['left'].set_visible(False)
```

```
ax[0].spines['right'].set_visible(False)
```

```
ax[0].spines['bottom'].set_visible(False)
```



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

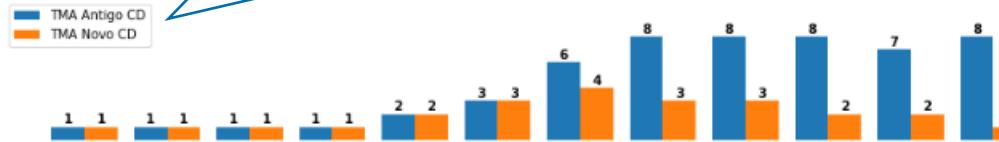
- Retirando a borda do gráfico de linha:

```
ax[1].spines['top'].set_visible(False)
```

```
ax[1].spines['left'].set_visible(False)
```

```
ax[1].spines['right'].set_visible(False)
```

Vamos colocar essa legenda no eixo



Pedidos  
Entregas - Antigo CD  
Entregas - Novo CD

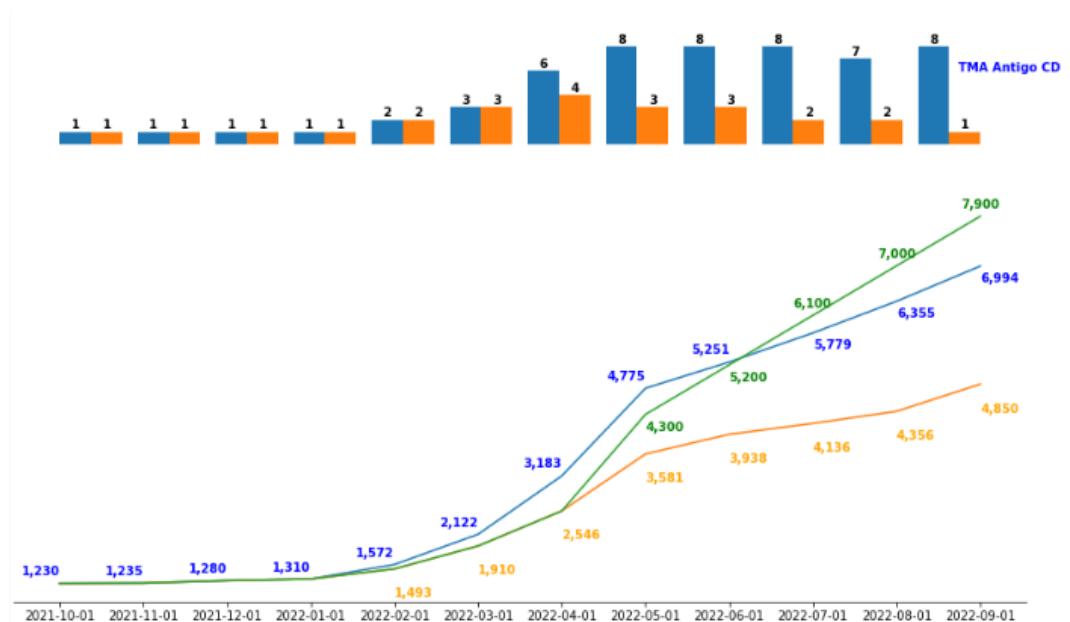
Vamos colocar essa legenda no eixo



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados:

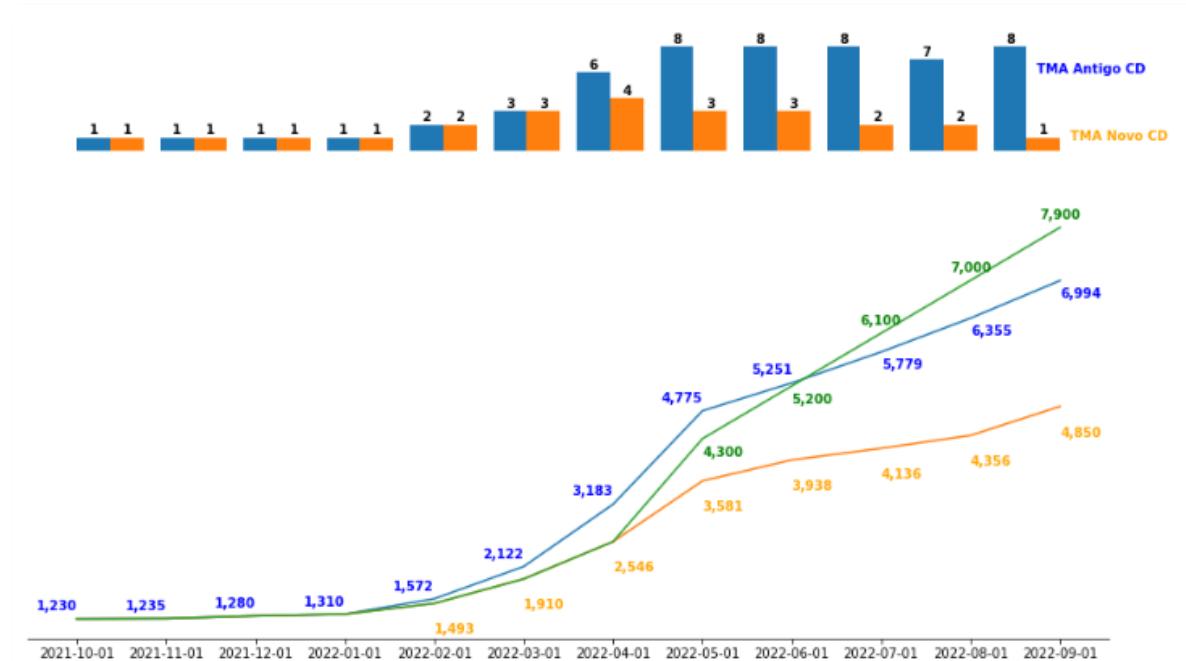
```
ax[0].annotate("TMA Antigo CD",
                (i,base.TMAantigoCD[11]),
                ha="left",
                xytext=(8,-20),
                textcoords="offset points",
                fontsize=10,
                fontweight='bold',
                color="blue"
            )
```



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados:

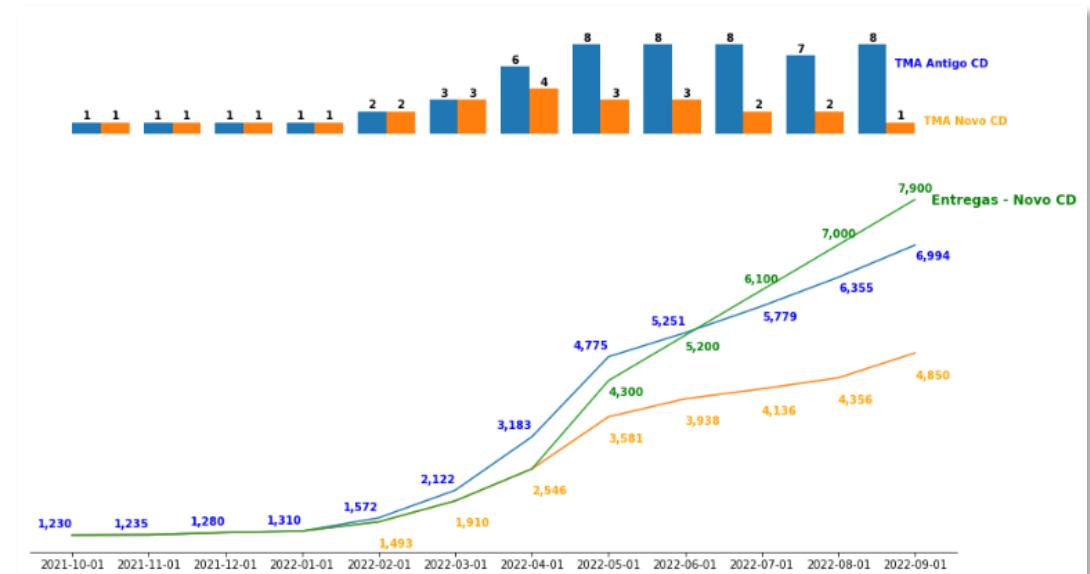
```
ax[0].annotate("TMA Novo CD",
    (i,base.TMAnovoCD[11]),
    ha="left",
    xytext=(8,-20),
    textcoords="offset points",
    fontsize=10,
    fontweight='bold',
    color="orange"
)
```



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados do gráfico das retas:

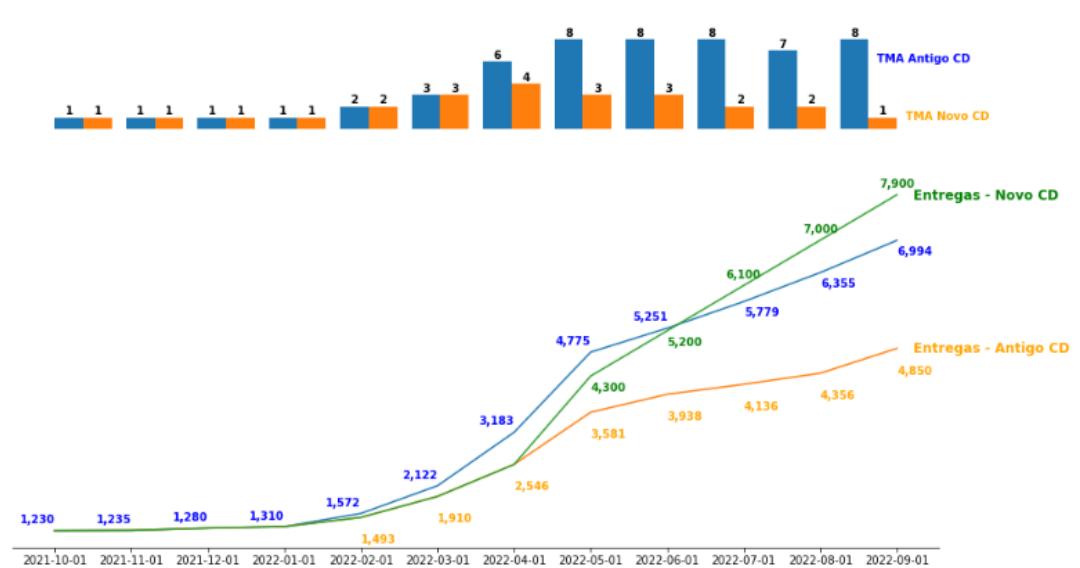
```
ax[1].annotate("Entregas - Novo CD",
                (i,base.EntregaNovoCD[11]),
                ha="left",
                va="center",
                xytext=(+15,0),
                textcoords="offset points",
                fontsize=12,
                fontweight='bold',
                color="green")
```



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados do gráfico das retas:

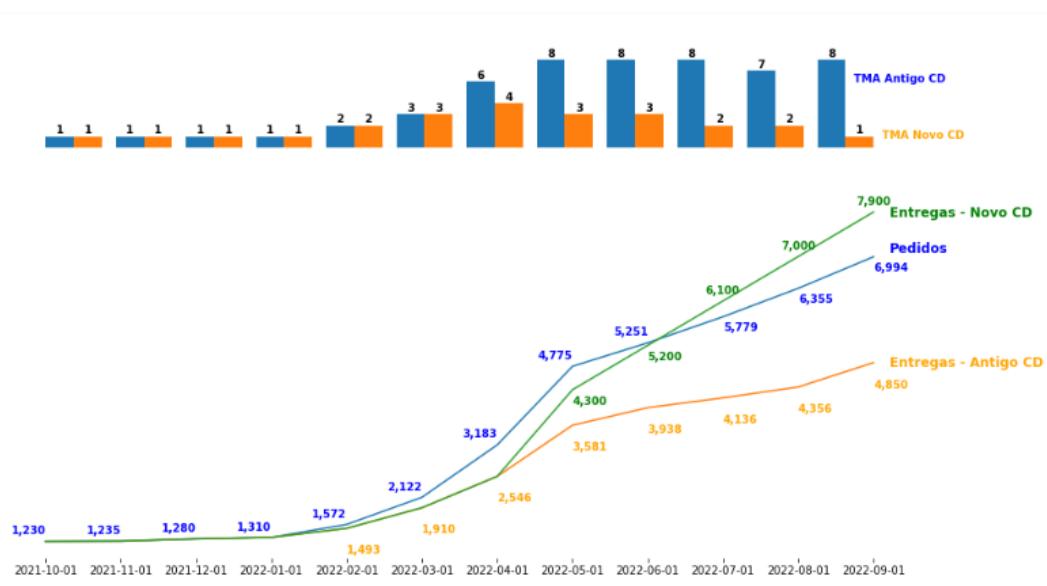
```
ax[1].annotate("Entregas - Antigo CD",
                (i,base.Entregas[11]),
                ha="left",
                va="center",
                xytext=(+15,0),
                textcoords="offset points",
                fontsize=12,
                fontweight='bold',
                color="orange")
```



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados do gráfico das retas:

```
ax[1].annotate("Pedidos",
    (i,base.Pedidos[11]),
    ha="left",
    va="center",
    xytext=(+15,+8),
    textcoords="offset points",
    fontsize=12,
    fontweight='bold',
    color="blue")
```



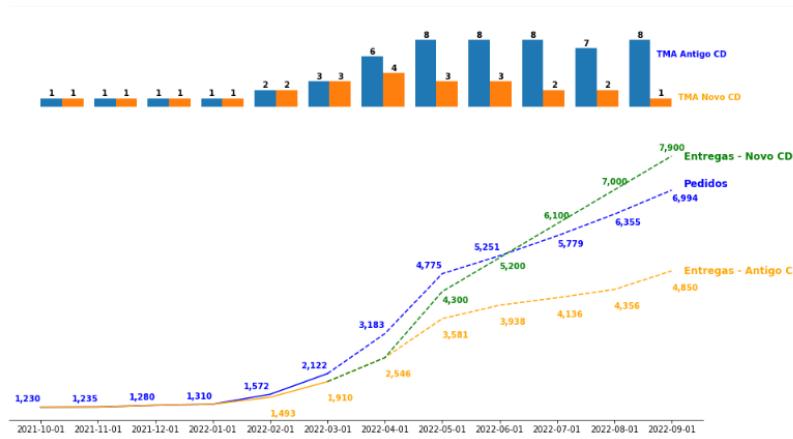
## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Por fim, podemos separar entre realizado e projetado:

```
ax[1].plot(np.arange(len(base))[:6],base.Pedidos[:6],color="blue")
ax[1].plot(np.arange(len(base))[-7:],base.Pedidos[-7:],"--",color="blue")
ax[1].plot(np.arange(len(base))[:6],base.Entregas[:6],color="orange")
ax[1].plot(np.arange(len(base))[-7:],base.Entregas[-7:],"--",color="orange")
ax[1].plot(np.arange(len(base))[-7:],base.EntregaNovoCD[-7:],"--",color="green")
```

Diferenciar o projetado colocando o tracejado “—”

Separando em 2 gráficos pelo comprimento da base



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Por fim, podemos separar entre realizado e projetado:

```
ax[0].bar(np.arange(len(base))[:7]-wid/2,  
         base.TMAantigoCD[:7],  
         width=wid,  
         label="TMA Antigo CD",  
         color="orange")
```

Definindo divisão do que é projetado

```
ax[0].bar(np.arange(len(base))[-6:]-wid/2,  
         base.TMAantigoCD[-6:],  
         width=wid,  
         label="TMA Antigo CD",  
         color="#FFB973")
```

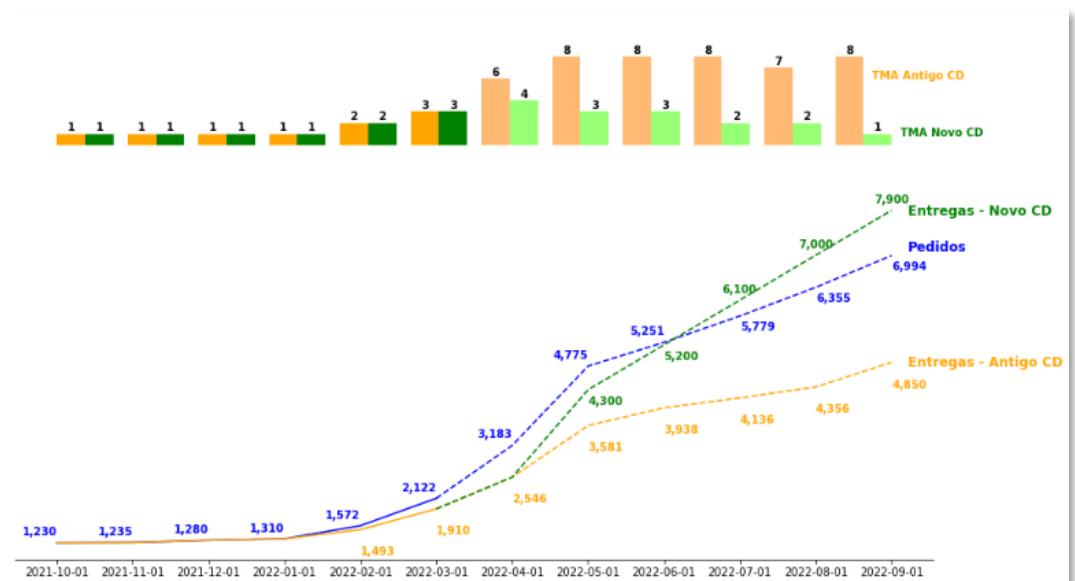
Definindo nova cor do projetado

## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Por fim, podemos separar entre realizado e projetado:

```
ax[0].bar(np.arange(len(base))[:7]+wid/2,  
         base.TMANovoCD[:7],  
         width=wid,  
         label="TMA Novo CD", color="green")
```

```
ax[0].bar(np.arange(len(base))[-6:]+wid/2,  
         base.TMANovoCD[-6:],  
         width=wid,  
         label="TMA Novo CD",  
         color="#97FF73")
```



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Por fim, podemos destacar a área do projetado com o retângulo:

```
Ax1 = fig.add_axes([0.51,0.125,0.4,0.755])
```

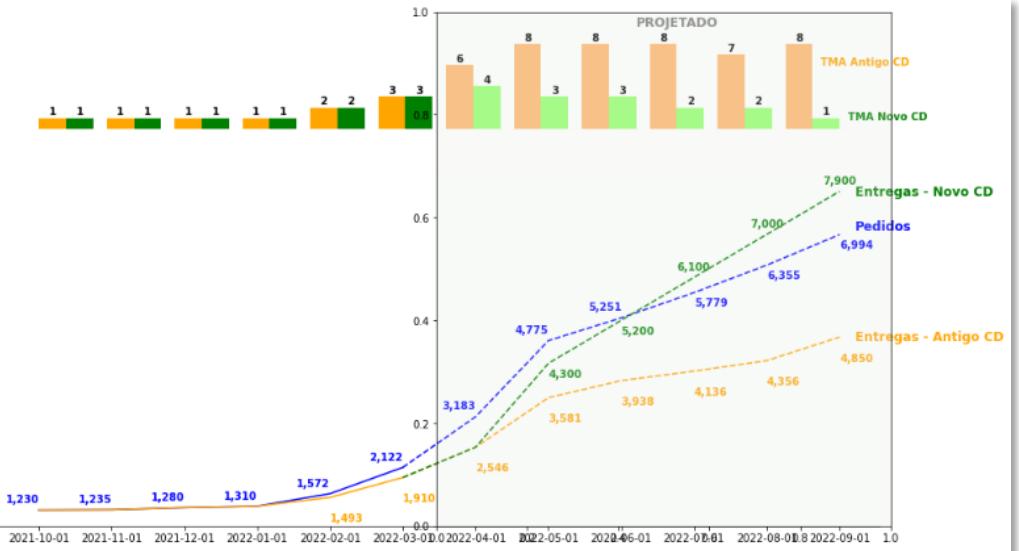
```
rect = ax1.patch
```

```
rect.set_edgecolor(None)
```

```
rect.set_facecolor('#E0E6DE')
```

```
rect.set_alpha(0.2)
```

```
ax[0].annotate("PROJETADO", (9,10), ha="center", va="center", xytext=(0,0),  
textcoords="offset points", fontsize=12, fontweight='bold', color="gray")
```



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Por fim, podemos retirar as bordas e os valores dos eixos:

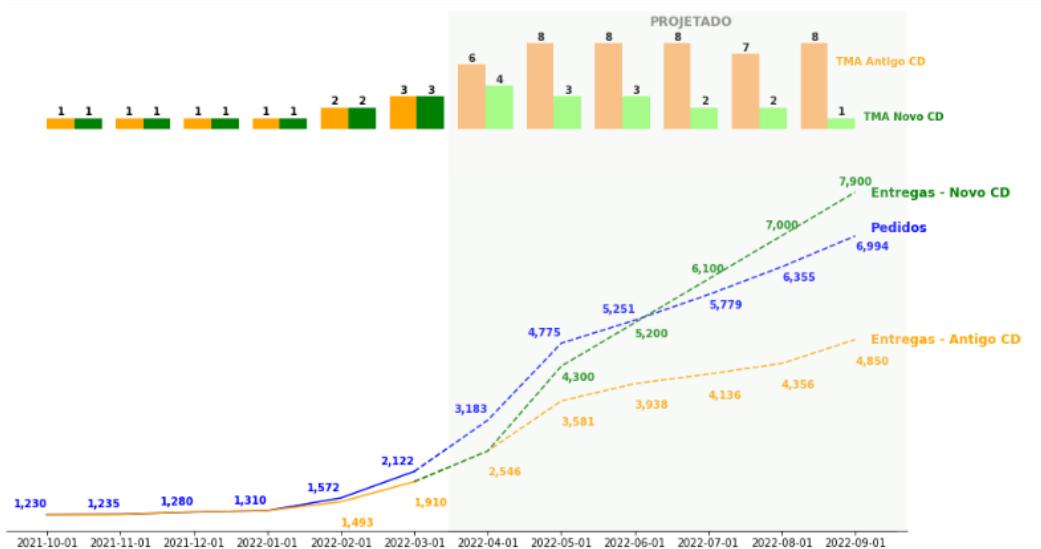
```
ax1.xaxis.set_visible(False)
```

```
ax1.yaxis.set_visible(False)
```

```
ax1.spines['top'].set_visible(False)
```

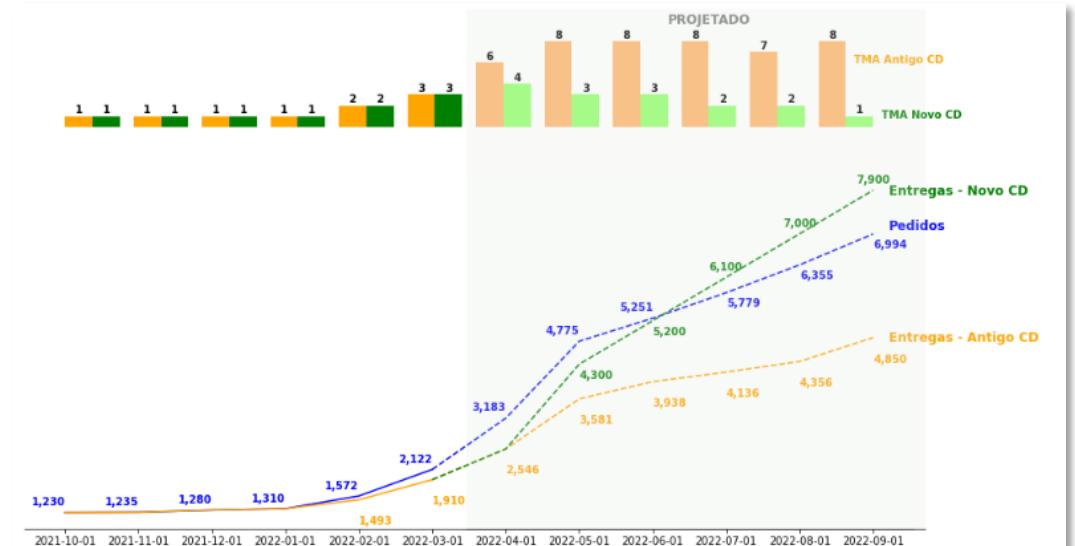
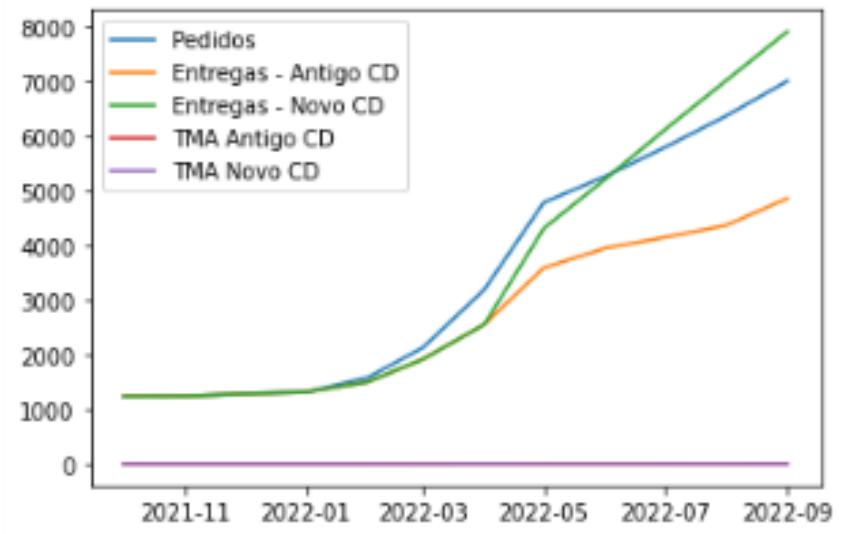
```
ax1.spines['left'].set_visible(False)
```

```
ax1.spines['right'].set_visible(False)
```



## Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

Observem a diferença do primeiro gráfico da aula para o último.



Mais fácil de visualizar os dados

## MÓDULO 9

# Criando uma apresentação executiva

## Módulo 9 – Apresentando o projeto

Agora que já vimos como usar o Matplotlib e as boas práticas de apresentação, já podemos fazer uma apresentação executiva, que seja realmente **bonita, visual** e que traga valor para a nossa empresa.

Nesse projeto vamos fazer as discussões sobre os dados tratados.

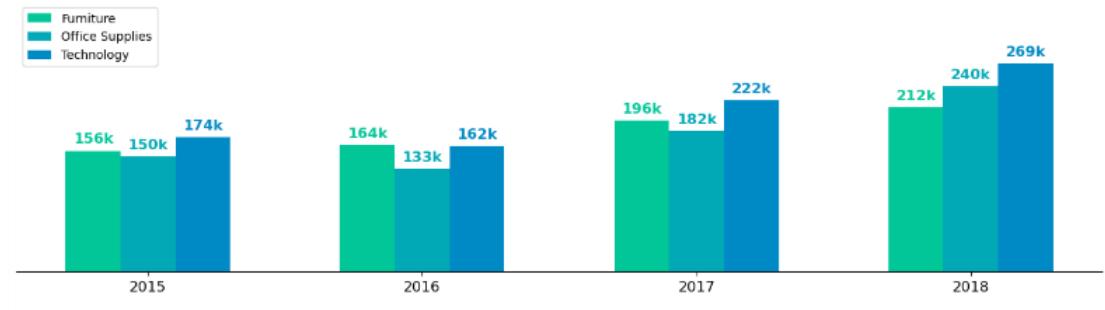


## Módulo 9 – Importando e analisando a base

Começando o nosso projeto, nós vamos utilizar a [base do kaggle](#) de vendas.

O que queremos responder nesse projeto?

- Como foi a **venda** nesse período?
- Qual foi a **categoria mais vendida**?
- Qual foi o **item mais vendido**?



## Módulo 9 – Importando e analisando a base

O primeiro passo para começar o nosso projeto é importar as bibliotecas com os código abaixo:

```
import pandas as pd
```

```
import numpy as np
```

Apelido da biblioteca

```
# Importando as bibliotecas e a base
import pandas as pd
import numpy as np
base = pd.read_csv("Criando uma apresentação executiva.csv")
```

## Módulo 9 – Importando e analisando a base

Para começar o nosso projeto vamos importar a base de dados.

Estrutura:

**base = pd.read\_** tipodoarquivo("Nome do arquivo.tipodoarquivo")

**base = pd.read\_csv("Criando uma apresentação executiva.csv")**

```
# Importando as bibliotecas e a base
import pandas as pd
import numpy as np
base = pd.read_csv("Criando uma apresentação executiva.csv")
```

## Módulo 9 – Importando e analisando a base

Uma forma de verificar se a base foi importada é utilizando o comando **.head** para visualizar as 5 primeiras linhas.

Estrutura:

**base.head()**

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240	Office Supplies
3	4	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577	Furniture
4	5	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000780	Office Supplies

## Módulo 9 – Importando e analisando a base

Podemos também utilizar o **.tail** para visualizar as 5 últimas linhas.

Estrutura:

**base.tail()**

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID
9795	9796	CA-2017-125920	21/05/2017	28/05/2017	Standard Class	SH-19975	Sally Hughsby	Corporate	United States	Chicago	Illinois	60610.0	Central OFF-BI-10003429
9796	9797	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East OFF-AR-10001374
9797	9798	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East TEC-PH-10004977
9798	9799	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East TEC-PH-10000912
9799	9800	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East TEC-AC-10000487

# Módulo 9 – Importando e analisando a base

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798 Furniture
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454 Furniture
2	3	CA-2017-139688	12/06/2017	16/06/2017	Second Class	DV-13045	Darin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240 Office Supplies
3	4	US-2016-105966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577 Furniture
4	5	US-2016-105966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000760 Office Supplies
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9795	9796	CA-2017-125920	21/05/2017	28/05/2017	Standard Class	SH-19975	Sally Hughesby	Corporate	United States	Chicago	Illinois	60610.0	Central	OFF-BI-10003429 Office Supplies
9796	9797	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	OFF-AR-10001374 Office Supplies
9797	9798	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	TEC-PH-10004977 Technology
9798	9799	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	TEC-PH-10000912 Technology
9799	9800	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	TEC-AC-10000487 Technology

9800 rows x 18 columns

Número de linhas e colunas

Além disso, podemos usar o comando **display** para visualizar o formato da base, as 5 primeiras e últimas linhas

Estrutura:

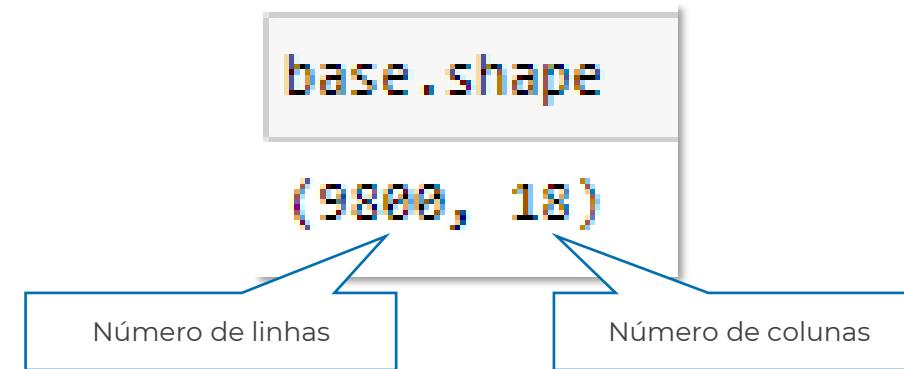
**display(base)**

## Módulo 9 – Importando e analisando a base

Outra forma de visualizar o formato da base é utilizando o comando **shape**.

Estrutura:

**base.shape()**

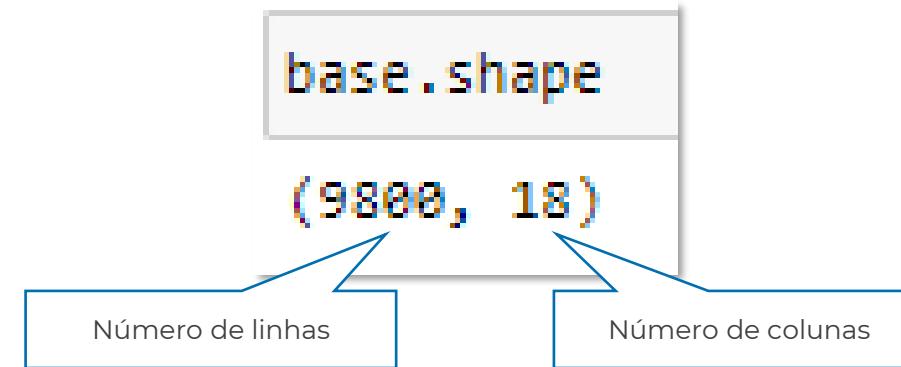


## Módulo 9 – Importando e analisando a base

Outra forma de visualizar o formato da base é utilizando o comando **shape**.

Estrutura:

**base.shape()**



## Módulo 9 – Importando e analisando a base

Vamos verificar se tem alguma informação faltando, algum dado no formato errado ou uma informação nula.

Estrutura:

**base.info()**

Vamos ajustar as informações de data que não estão com o formato de data.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Row ID            9800 non-null   int64  
 1   Order ID          9800 non-null   object  
 2   Order Date        9800 non-null   object  
 3   Ship Date         9800 non-null   object  
 4   Ship Mode         9800 non-null   object  
 5   Customer ID      9800 non-null   object  
 6   Customer Name    9800 non-null   object  
 7   Segment           9800 non-null   object  
 8   Country           9800 non-null   object  
 9   City               9800 non-null   object  
 10  State              9800 non-null   object  
 11  Postal Code       9789 non-null   float64 
 12  Region             9800 non-null   object  
 13  Product ID        9800 non-null   object  
 14  Category           9800 non-null   object  
 15  Sub-Category       9800 non-null   object  
 16  Product Name       9800 non-null   object  
 17  Sales              9800 non-null   float64 
dtypes: float64(2), int64(1), object(15)
memory usage: 1.3+ MB
```

Order date e ship date estão como objeto

Observe que temos 1 valor nulo

## Módulo 9 – Importando e analisando a base

Apenas código postal possui valores nulos, podemos visualizar quais linhas não possuem esse valor.

Verificando os registros com Postal Code vazio com o código abaixo:

```
base[base['Postal Code'].isnull()]
```

Filtrando a base pela coluna Postal Code

Todos os registros sem código postal são da cidade de Burlington, em Vermont.

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
2234	2235	CA-2018-104066	05/12/2018	10/12/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	Nan	East	TEC-AC-10001013 Technology
5274	5275	CA-2018-162887	07/11/2016	09/11/2016	Second Class	SV-20785	Stewart Visinsky	Consumer	United States	Burlington	Vermont	Nan	East	FUR-CH-1000595 Furniture
8798	8799	US-2017-150140	06/04/2017	10/04/2017	Standard Class	VM-21685	Valerie Mitchum	Home Office	United States	Burlington	Vermont	Nan	East	TEC-PH-10002555 Technology
9146	9147	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	Nan	East	TEC-AC-10002926 Technology
9147	9148	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	Nan	East	OFF-AR-10003477 Office Supplies
9148	9149	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	Nan	East	OFF-ST-10001526 Office Supplies
9386	9387	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-PA-1000157 Office Supplies
9387	9388	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-PA-10001970 Office Supplies
9388	9389	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-AP-10000828 Office Supplies
9389	9390	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-EN-10001509 Office Supplies
9741	9742	CA-2016-117086	08/11/2016	12/11/2016	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	Nan	East	FUR-BO-10004834 Furniture

Valor nulo quando City é Burlington

## Módulo 9 – Importando e analisando a base

Vamos verificar se existe algum registro de Burlington, Vermont com Postal Code.

```
base[(base.City == 'Burlington')]
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
683	684	US-2018-168116	04/11/2018	04/11/2018	Same Day	GT-14635	Grant Thornton	Corporate	United States	Burlington	North Carolina	27217.0	South	TEC-MA-10004125 Technology
684	685	US-2018-168116	04/11/2018	04/11/2018	Same Day	GT-14635	Grant Thornton	Corporate	United States	Burlington	North Carolina	27217.0	South	OFF-AP-10002457 Office Supplies
1008	1009	US-2018-106705	26/12/2018	01/01/2019	Standard Class	PO-18850	Patrick O'Brill	Consumer	United States	Burlington	Iowa	52601.0	Central	OFF-PA-10001509 Office Supplies
1038	1039	CA-2018-121818	20/11/2018	21/11/2018	First Class	JH-15430	Jennifer Halladay	Consumer	United States	Burlington	North Carolina	27217.0	South	OFF-AR-10000203 Office Supplies
1039	1040	CA-2018-121818	20/11/2018	21/11/2018	First Class	JH-15430	Jennifer Halladay	Consumer	United States	Burlington	North Carolina	27217.0	South	OFF-AR-10004790 Office Supplies
1393	1394	CA-2018-124828	03/07/2018	04/07/2018	First Class	YS-21880	Yana Sorensen	Corporate	United States	Burlington	North Carolina	27217.0	South	OFF-AR-10003514 Office Supplies
2234	2235	CA-2018-104066	05/12/2018	10/12/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	5401.0	East	TEC-AC-10001013 Technology
2928	2929	US-2018-120390	19/10/2018	26/10/2018	Standard Class	TH-21550	Tracy Hopkins	Home Office	United States	Burlington	North Carolina	27217.0	South	OFF-BI-10004995 Office Supplies
5065	5066	CA-2018-142090	30/11/2018	07/12/2018	Standard Class	SC-20380	Shahid Collister	Consumer	United States	Burlington	North Carolina	27217.0	South	TEC-AC-10002001 Technology
5066	5067	CA-2018-142090	30/11/2018	07/12/2018	Standard Class	SC-20380	Shahid Collister	Consumer	United States	Burlington	North Carolina	27217.0	South	FUR-TA-10001889 Furniture

## Módulo 9 – Importando e analisando a base

Vamos verificar se existe algum registro de Burlington, Vermont com Postal Code.

`base[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal Code'].notnull())]:`

Concatenando dois filtros, se fosse OU utilizariam o sinal |

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales
--------	----------	------------	-----------	-----------	-------------	---------------	---------	---------	------	-------	-------------	--------	------------	----------	--------------	--------------	-------

## Módulo 9 – Tratando valores vazios

Agora que já identificamos que a nossa base possui valores vazios, nós vamos tratar eles.

Então, primeiro vamos observar os valores que são nulos:

```
base[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal Code'].isnull())]
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
2234	2235	CA-2018-104066	05/12/2018	10/12/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	Nan	East	TEC-AC-10001013 Technology
5274	5275	CA-2018-162887	07/11/2018	09/11/2018	Second Class	SV-20785	Stewart Visinsky	Consumer	United States	Burlington	Vermont	Nan	East	FUR-CH-10000595 Furniture
8798	8799	US-2017-150140	06/04/2017	10/04/2017	Standard Class	VM-21685	Valerie Mitchum	Home Office	United States	Burlington	Vermont	Nan	East	TEC-PH-10002555 Technology
9146	9147	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	Nan	East	TEC-AC-10002926 Technology
9147	9148	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	Nan	East	OFF-AR-10003477 Office Supplies
9148	9149	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	Nan	East	OFF-ST-10001526 Office Supplies
9386	9387	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-PA-10000157 Office Supplies
9387	9388	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-PA-10001970 Office Supplies
9388	9389	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-AP-10000828 Office Supplies
9389	9390	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-EN-10001509 Office Supplies
9741	9742	CA-2018-117086	08/11/2018	12/11/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	Nan	East	FUR-BO-10004834 Furniture

## Módulo 9 – Tratando valores vazios

Existe um forma que podemos buscar só pelo Postal Code.

```
base[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal  
Code'].isnull())] ['Postal Code']
```

```
2234    NaN  
5274    NaN  
8798    NaN  
9146    NaN  
9147    NaN  
9148    NaN  
9386    NaN  
9387    NaN  
9388    NaN  
9389    NaN  
9741    NaN  
Name: Postal Code, dtype: float64
```

## Módulo 9 – Tratando valores vazios

Então vamos utilizar o **loc** para localizar esses valores. = 5401

**base.loc[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal Code'].isnull())]**

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
2234	2235	CA-2018-104066	05/12/2018	10/12/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	Nan	East	TEC-AC-10001013 Technology
5274	5275	CA-2018-162887	07/11/2018	09/11/2018	Second Class	SV-20785	Stewart Visinsky	Consumer	United States	Burlington	Vermont	Nan	East	FUR-CH-10000595 Furniture
8798	8799	US-2017-150140	08/04/2017	10/04/2017	Standard Class	VM-21685	Valerie Mitchum	Home Office	United States	Burlington	Vermont	Nan	East	TEC-PH-10002555 Technology
9146	9147	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	Nan	East	TEC-AC-10002626 Technology
9147	9148	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	Nan	East	OFF-AR-10003477 Office Supplies
9148	9149	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	Nan	East	OFF-ST-10001526 Office Supplies
9386	9387	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-PA-10000157 Office Supplies
9387	9388	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-PA-10001970 Office Supplies
9388	9389	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-AP-10000828 Office Supplies
9389	9390	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	Nan	East	OFF-EN-10001509 Office Supplies
9741	9742	CA-2018-117086	08/11/2018	12/11/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	Nan	East	FUR-BO-10004834 Furniture

## Módulo 9 – Tratando valores vazios

Agora se quisermos apenas da Coluna Postal Code basta digitar:

```
base.loc[(base.City == 'Burlington') & (base.State  
== 'Vermont') & (base['Postal Code'].isnull()),  
['Postal Code']]
```

```
2234    NaN  
5274    NaN  
8798    NaN  
9146    NaN  
9147    NaN  
9148    NaN  
9386    NaN  
9387    NaN  
9388    NaN  
9389    NaN  
9741    NaN  
Name: Postal Code, dtype: float64
```

## Módulo 9 – Tratando valores vazios

Vamos atribuir o código postal de 05401 para essa coluna:

```
base.loc[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal  
Code'].isnull()), ['Postal Code']] = 05401
```

Vamos procurar esses valores para verificar se foi atualizado:

```
base.loc[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal  
Code'].isnull())]
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales
																Nenhum valor em branco	

## Módulo 9 – Tratando valores vazios

Vamos utilizar o comando .info para verificar essa base:

### base.info

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column        Non-Null Count  Dtype  
--- 
 0   Row ID        9800 non-null    int64  
 1   Order ID      9800 non-null    object  
 2   Order Date    9800 non-null    object  
 3   Ship Date     9800 non-null    object  
 4   Ship Mode     9800 non-null    object  
 5   Customer ID   9800 non-null    object  
 6   Customer Name 9800 non-null    object  
 7   Segment       9800 non-null    object  
 8   Country       9800 non-null    object  
 9   City          9800 non-null    object  
 10  State         9800 non-null    object  
 11  Postal Code   9800 non-null    float64 
 12  Region        9800 non-null    object  
 13  Product ID    9800 non-null    object  
 14  Category       9800 non-null    object  
 15  Sub-Category  9800 non-null    object  
 16  Product Name   9800 non-null    object  
 17  Sales          9800 non-null    float64 
dtypes: float64(2), int64(1), object(15)
memory usage: 1.3+ MB
```

Nenhum valor nulo

## Módulo 9 – Tratando valores vazios

Se fosse necessário excluir a coluna, poderíamos utilizar o comando **drop**

```
base = base.drop('Postal Code',  
axis = 1)
```

```
base.head()
```

Porém não vamos utilizar isso no nosso projeto

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798 Furniture
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454 Furniture
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240 Office Supplies
3	4	US-2016-108986	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577 Furniture
4	5	US-2016-108986	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000760 Office Supplies

## Módulo 9 – Usando o datetime para tratar datas

Agora vamos para etapa de visualização. Aqui temos 2 questões que precisamos pensar:

Quais **colunas** vamos usar?

- Podemos verificar a venda na coluna "**Sales**"
- O período nós podemos usar a coluna "**Order Date**", mas precisamos entender melhor essa coluna

Qual o período queremos exibir?

## Módulo 9 – Usando o datetime para tratar datas

Observe que Order Date está como objeto

**Atenção:** Se a coluna de data estiver como texto e não como data, a ordenação vai ser pelo texto e não pela data!

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
 ---  -- 
 0   Row ID             9800 non-null   int64  
 1   Order ID           9800 non-null   object 
 2   Order Date          9800 non-null   object 
 3   Ship Date           9800 non-null   object 
 4   Ship Mode            9800 non-null   object 
 5   Customer ID         9800 non-null   object 
 6   Customer Name        9800 non-null   object 
 7   Segment              9800 non-null   object 
 8   Country              9800 non-null   object 
 9   City                 9800 non-null   object 
 10  State                9800 non-null   object 
 11  Postal Code          9800 non-null   float64
 12  Region               9800 non-null   object 
 13  Product ID           9800 non-null   object 
 14  Category              9800 non-null   object 
 15  Sub-Category          9800 non-null   object 
 16  Product Name          9800 non-null   object 
 17  Sales                 9800 non-null   float64
dtypes: float64(2), int64(1), object(15)
memory usage: 1.3+ MB
```

Order date está  
como objeto

## Módulo 9 – Usando o datetime para tratar datas

Podemos converter uma coluna para data usando o **to\_datetime** do pandas

Estrutura:

```
base["coluna"] = pd.to_datetime(base["coluna"])
```

No **datetime**, caso tenha apenas a data, a hora ficaria como 00:00:00

Se quisermos considerar apenas a data, podemos usar o **.date** da biblioteca datetime.

## Módulo 9 – Usando o datetime para tratar datas

Vamos aplicar no nosso projeto

**Import** datetime **as** dt

```
base["Order Date"] = pd.to_datetime(base["Order Date"])
```

```
base["Ship Date"] = pd.to_datetime(base["Ship Date"])
```

```
base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Row ID      9800 non-null   int64  
 1   Order ID    9800 non-null   object  
 2   Order Date  9800 non-null   datetime64[ns]
 3   Ship Date   9800 non-null   datetime64[ns]
 4   Ship Mode   9800 non-null   object  
 5   Customer ID 9800 non-null   object  
 6   Customer Name 9800 non-null   object  
 7   Segment     9800 non-null   object  
 8   Country     9800 non-null   object  
 9   City         9800 non-null   object  
 10  State        9800 non-null   object  
 11  Postal Code 9800 non-null   float64 
 12  Region       9800 non-null   object  
 13  Product ID  9800 non-null   object  
 14  Category     9800 non-null   object  
 15  Sub-Category 9800 non-null   object  
 16  Product Name 9800 non-null   object  
 17  Sales        9800 non-null   float64 
dtypes: datetime64[ns](2), float64(2), int64(1), object(13)
memory usage: 1.3+ MB
```

Formato de data

## Módulo 9 – Usando o datetime para tratar datas

Agora podemos verificar qual a data mínima e a máxima:

**base['Order Date'].min()**

```
base['Order Date'].min()
```

```
'01/01/2018'
```

**base['Order Date'].max()**

```
base['Order Date'].max()
```

```
'31/12/2017'
```

## Módulo 9 – Usando o datetime para tratar datas

Como estamos falando de 4 anos de análise, começando em jan/2015 e indo até dez/2018.

Estamos trabalhando com um período grande de anos, podemos começar a visualização pelos anos.

Para isso, podemos usar o **.year** da biblioteca datetime para criar uma nova coluna apenas com o ano dessa base

## Módulo 9 – Usando o datetime para tratar datas

Podemos criar uma coluna com o ano:

**base['Ano'] = base['Order Date'].dt.year**



Para visualizar:

**base.head()**

Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales	Ano
2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	261.9600	2017
2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...	731.9400	2017
2017-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240	Office Supplies	Labels	Self-Adhesive Address Labels for Typewriters b...	14.6200	2017
2016-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table	957.5775	2016
2016-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000780	Office Supplies	Storage	Eldon Fold 'N Roll Cart System	22.3680	2016

## Módulo 9 – Usando o datetime para tratar datas

Vamos gerar um gráfico de quando a compra foi feita, mas para isso precisamos agrregar as informações utilizando o **group by**.

`base.groupby('Ano')[‘Sales’].sum()`

Qual coluna vamos usar para fazer a função de agregação

Função de agregação

Ano	Sales
2015	479856.2081
2016	459436.0054
2017	600192.5500
2018	722052.0192

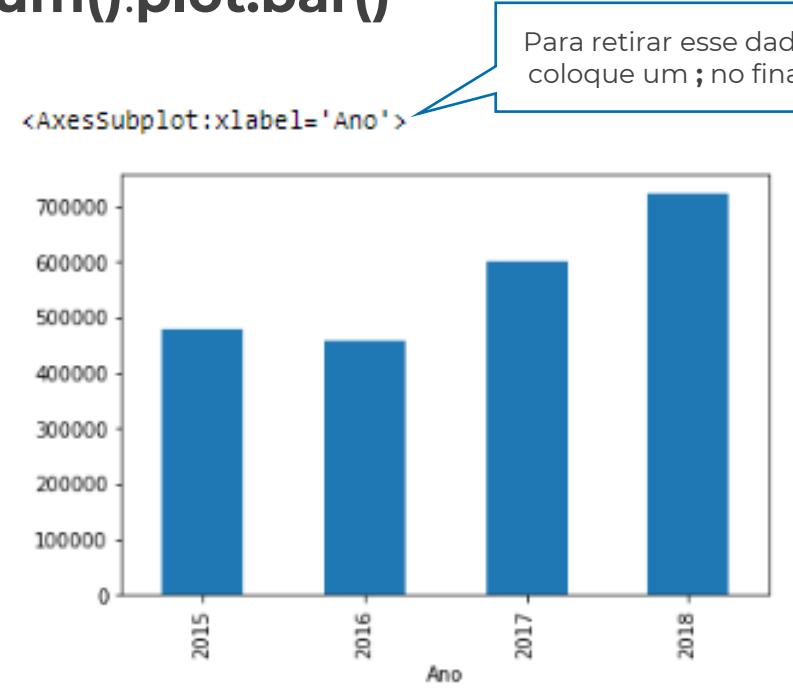
Name: Sales, dtype: float64

Então o resultado é a soma das vendas por cada ano.

## Módulo 9 – Usando o datetime para tratar datas

Agora vamos transformar o dado anterior em um gráfico.

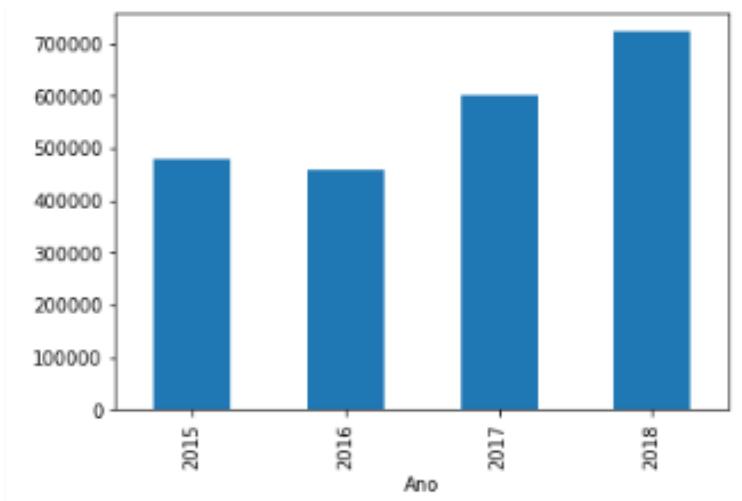
**base.groupby('Ano')[‘Sales’].sum().plot.bar()**



## Módulo 9 – Usando o datetime para tratar datas

Agora vamos retirar aquela informação antes do gráfico.

```
base.groupby('Ano')[‘Sales’].sum().plot.bar();
```



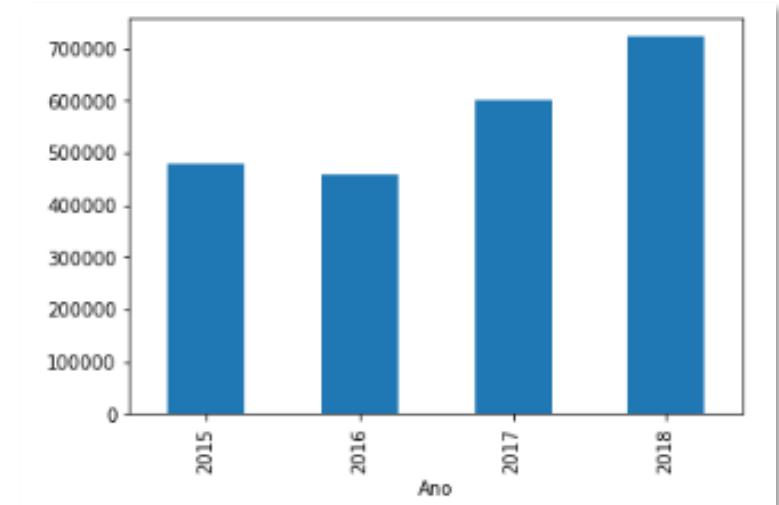
## Módulo 9 – Criando um gráfico de barras no matplotlib

A visualização ainda não está tão boa, porque não conseguimos visualizar os valores exatos de cada ano.

Vamos melhorar esse visual utilizando o **matplotlib**, para isso vamos importar as bibliotecas:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```



## Módulo 9 – Criando um gráfico de barras no matplotlib

Vamos utilizar o código padrão da documentação do matplotlib:

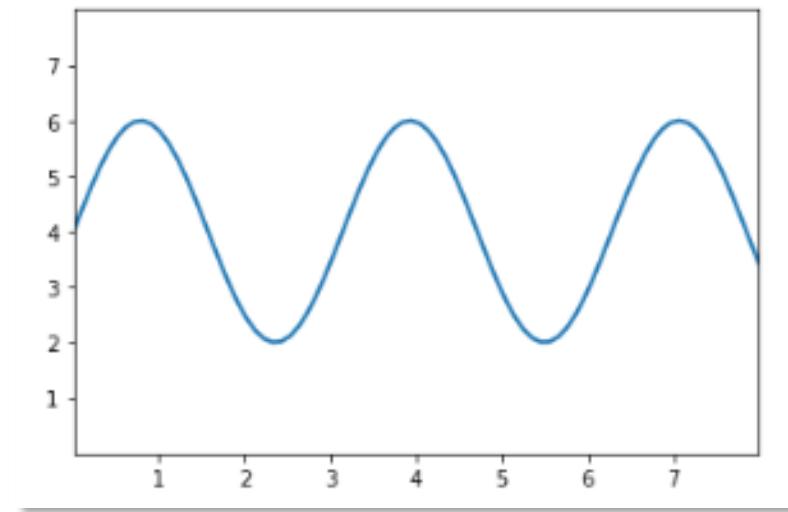
```
x = np.linspace(0, 10, 100)
y = 4 + 2 * np.sin(2 * x)

fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



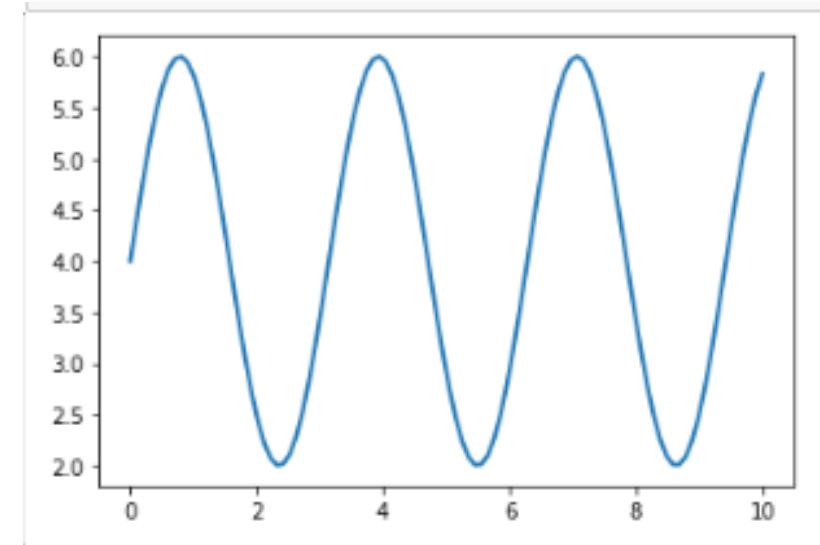
## Módulo 9 – Criando um gráfico de barras no matplotlib

Na verdade para realizar um gráfico basta o código abaixo:

```
fig, ax = plt.subplots()
```

```
ax.plot(x, y, linewidth=2.0)
```

```
plt.show()
```



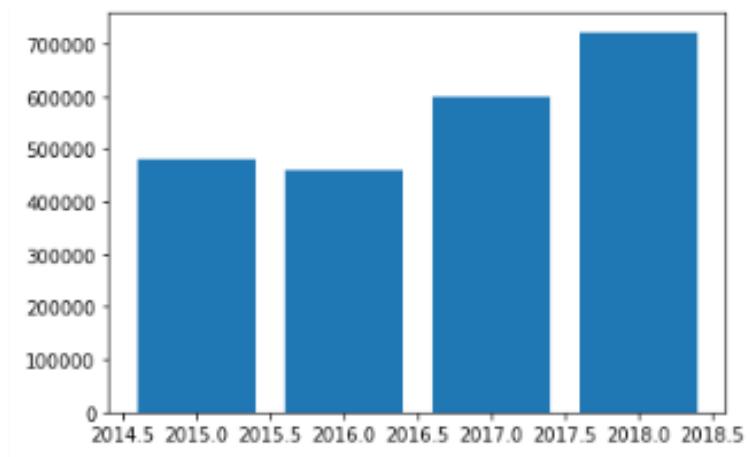
## Módulo 9 – Criando um gráfico de barras no matplotlib

Agora vamos substituir os nossos valores em x e y:

```
fig, ax = plt.subplots()
```

```
ax.bar(base.groupby("Ano")["Sales"].sum().index,base.groupby("Ano")["Sales"].sum().values,linewidth=2.0)
```

```
plt.show()
```



## Módulo 9 – Criando um gráfico de barras no matplotlib

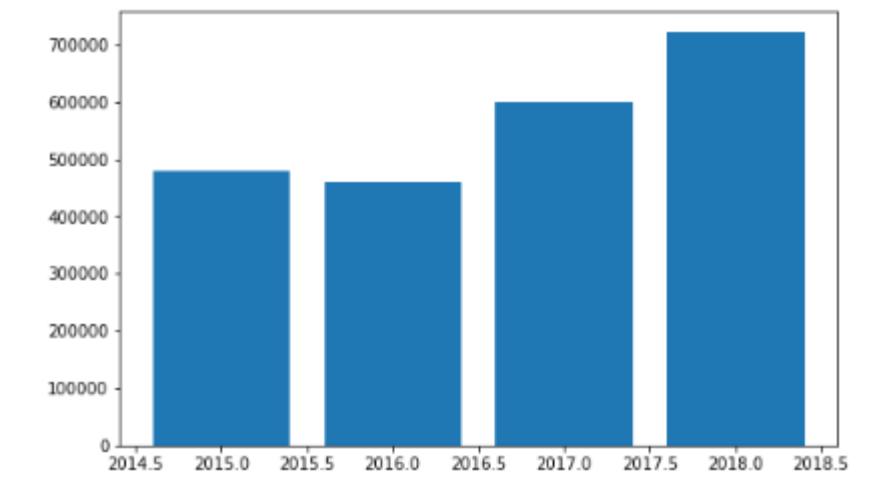
Agora vamos mudar o tamanho da figura:

```
fig, ax = plt.subplots(figsize=(8,5))
```

Tamanho do gráfico

```
ax.bar(base.groupby("Ano")["Sales"].sum().index,base.groupby("Ano")["Sales"].sum().v  
alues, linewidth=2.0)
```

```
plt.show()
```



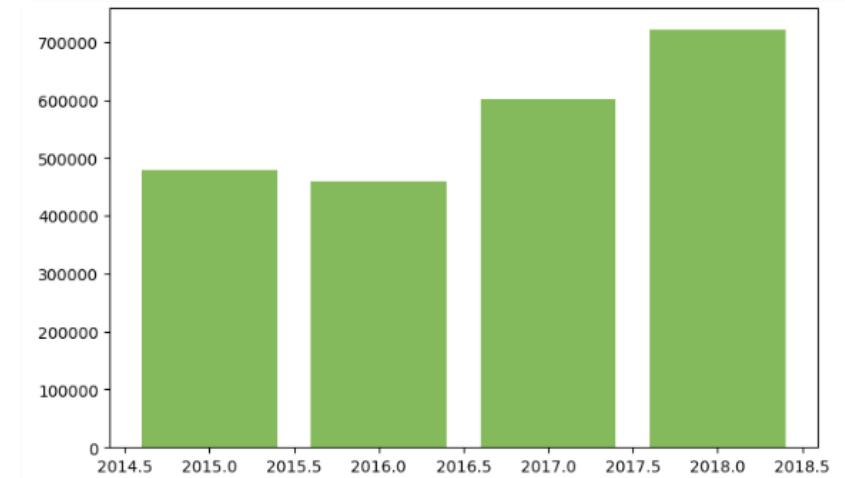
## Módulo 9 – Criando um gráfico de barras no matplotlib

Agora vamos alterar a cor do gráfico:

```
fig, ax = plt.subplots(figsize=(8,5))
```

```
ax.bar(base.groupby("Ano")["Sales"].sum().index,base.groupby("Ano")["Sales"].sum().v  
alues,linewidth=2.0, color="#84ba5b")
```

```
plt.show()
```



## Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

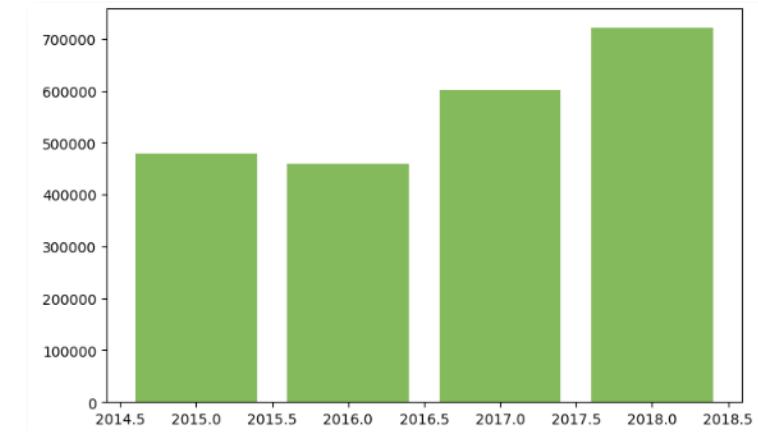
Para melhorar o nosso código vamos colocar o groupby dentro de uma variável:

```
fig, ax = plt.subplots(figsize=(8,5))
```

```
soma_ano = base.groupby("Ano")["Sales"].sum()
```

```
ax.bar(soma_ano .index, soma_ano .values,linewidth=2.0, color="#84ba5b")
```

```
plt.show()
```



## Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

Vamos adicionar o título:

- O `.set_title` vai permitir colocar título e ajustar os parâmetros desse títulos:

**fontsize**: Tamanho da fonte

**fontweight**: Espessura da fonte

**color**: Cor da fonte

## Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

Vamos adicionar melhorar o eixo x utilizando os códigos:

- O **xaxis.set\_ticks** vai definir os valores mostrados no eixo x
- O **.tick\_params(axis='x')** vai ajustar o parâmetro do eixo x

**labelsize**: tamanho do rótulo de dados

## Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

```
fig, ax = plt.subplots(figsize=(8,5))  
soma_ano = base.groupby("Ano")["Sales"].sum()
```

```
ax.set_title("VENDAS POR ANO",  
            fontsize=20,  
            fontweight="bold",  
            color="#84ba5b")
```

Ajustando os  
parâmetros do título  
do gráfico

```
ax.bar(soma_ano .index, soma_ano .values, linewidth=2.0, color='#84ba5b')
```

```
plt.show()
```



## Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

Vamos ajustar os valores dos anos:

```
fig, ax = plt.subplots(figsize=(8,5))
soma_ano = base.groupby("Ano")["Sales"].sum()
```

```
ax.set_title("VENDAS POR ANO",
             fontsize=20,
             fontweight="bold",
             color="#84ba5b")
```

```
ax.bar(soma_ano .index, soma_ano .values,linewidth=2.0, color='#84ba5b')
```

```
ax.xaxis.set_ticks([2015,2016,2017,2018])
plt.show()
```



Ajustando os valores  
dos anos

## Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

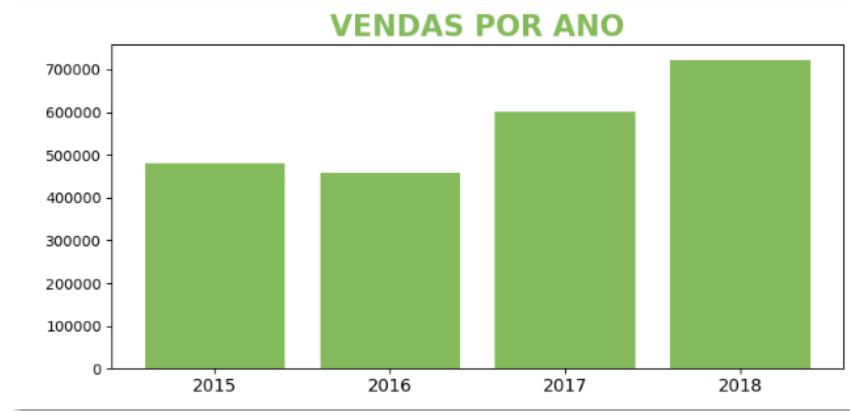
```
fig, ax = plt.subplots(figsize=(8,5))
soma_ano = base.groupby("Ano")["Sales"].sum()

ax.set_title("VENDAS POR ANO",
             fontsize=20,
             fontweight="bold",
             color="#84ba5b")

ax.bar(soma_ano .index, soma_ano .values, linewidth=2.0, color='#84ba5b')

ax.xaxis.set_ticks([2015,2016,2017,2018])

ax.tick_params(axis='x',labelsize=14)
plt.show()
```



Ajustando o tamanho  
do rótulo de dados dos  
anos

## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Para adicionar rótulo de dados, vamos usar o **.annotate** como vimos nas outras aulas  
fig, ax = plt.subplots(figsize=(8,5))

```
soma_ano = base.groupby("Ano")["Sales"].sum()
```

```
ax.set_title("VENDAS POR ANO",
```

```
    fontsize=20,
```

```
    fontweight="bold",
```

```
    color="#84ba5b")
```

```
ax.bar(soma_ano .index, soma_ano .values, linewidth=2.0, color='#84ba5b')
```

```
ax.xaxis.set_ticks([2015,2016,2017,2018])
```

```
ax.tick_params(axis='x',labelsize=14)
```

```
ax.annotate("TESTE", (soma_ano.index[0],100000))
```

```
plt.show()
```



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Porém, no caso anterior só tínhamos o valor de 1 ano no rótulo de dados.

Para fazer com todos os valores vamos utilizar o **for**.

Relembrando:

**For i in range (0,4):**  
**print (i)**

```
for i in range (0,4):
    print (i)
```

0  
1  
2  
3

## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos focar no código do **annotate** utilizando o for:

**For i in range (0,4):**

```
ax.annotate('TESTE',(soma_ano.index[i],100000))
```

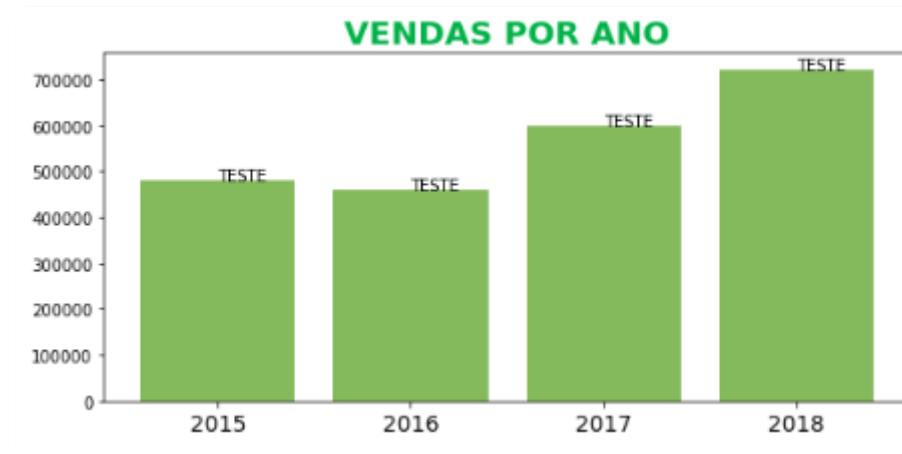


## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos alterar os valores do eixo y no **annotate**:

**For i in range (0,4):**

```
ax.annotate('TESTE',(soma_ano.index[i], soma_ano.values[i]))
```



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos alterar o texto TESTE para exatamente o valor de y no **annotate**:

**For i in range (0,4):**

```
ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]))
```



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos ajustar os valores dos rótulos de dados para que eles fiquem centralizados, para isso vamos utilizar o **ha = "center"**:

**For i in range (0,4):**

```
    ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),  
               ha="center",  
               )
```



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos ajustar a posição dos valores dos rótulos de dados, utilizando o **xytext** e o **textcoords**:

**For i in range (0,4):**

```
    ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),  
               ha="center",  
               xytext=(0,5),  
               textcoords="offset points",  
               )
```

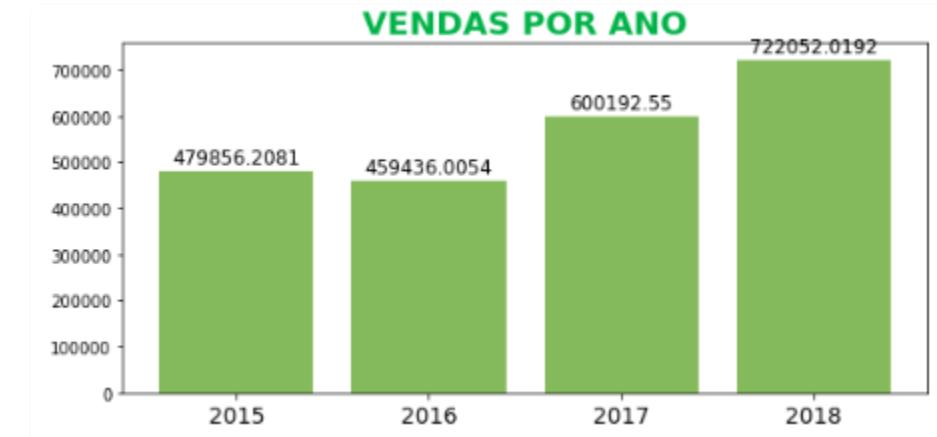


## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos aumentar o tamanho da fonte com o **fontsize**:

**For i in range (0,4):**

```
    ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),  
    ha="center",  
    xytext=(0,5),  
    textcoords="offset points",  
    fontsize=12,  
    )
```



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos ajustar alterar a cor do rótulo de dados utilizando o **color**:

**For i in range (0,4):**

```
    ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),  
               ha="center",  
               xytext=(0,5),  
               textcoords="offset points",  
               fontsize=12,  
               color="green",  
)
```

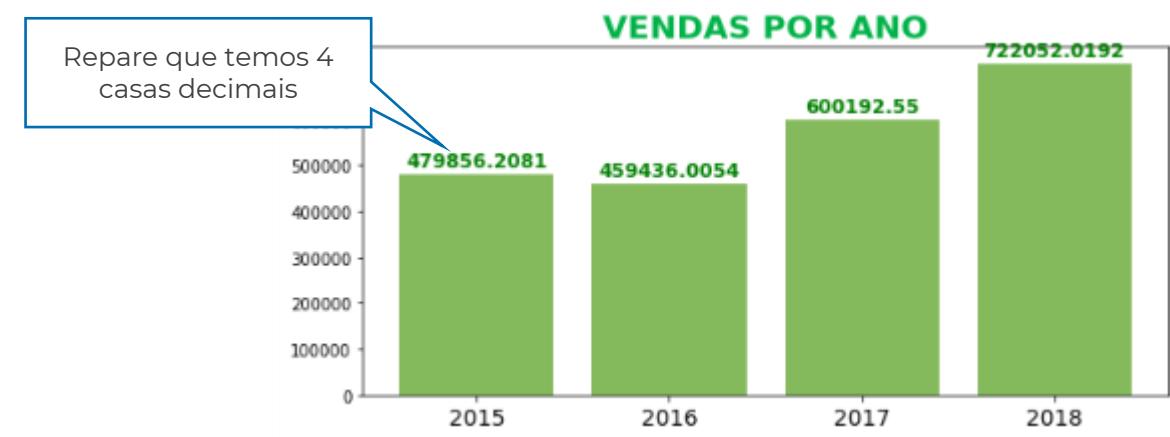


## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Por último, vamos colocar o rótulo de dados em negrito utilizando o **fontweight**

**For i in range (0,4):**

```
    ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),  
               ha="center",  
               xytext=(0,5),  
               textcoords="offset points",  
               fontsize=12,  
               color="green",  
               fontweight="bold")
```



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Para melhorar a visualização vamos formatar as casas decimais utilizando o **format**

```
For i in range (0,4):  
    ax.annotate(' {:.2f}'.format(soma_ano.values[i]),  
                (soma_ano.index[i], soma_ano.values[i]),  
  
                ha="center",  
                xytext=(0,5),  
                textcoords="offset points",  
                fontsize=12,  
                color="green",  
                fontweight="bold"  
)
```

Quantidade de casas decimais após o ponto



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Para melhorar a visualização vamos formatar as casas decimais utilizando o **format**

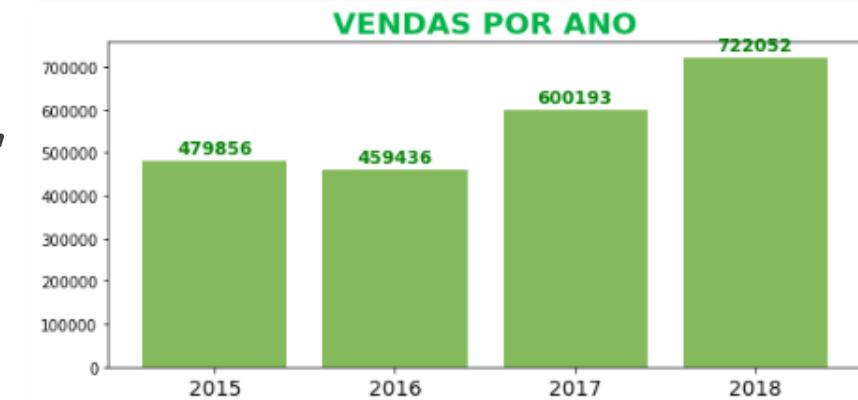
```
For i in range (0,4):
```

Quantidade de casas decimais após o ponto

```
    ax.annotate(' {:.0f}'.format(soma_ano.values[i]),  
                (soma_ano.index[i], soma_ano.values[i]),
```

```
        ha="center",  
        xytext=(0,5),  
        textcoords="offset points",  
        fontsize=12,  
        color="green",  
        fontweight="bold"
```

```
)
```



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Vamos utilizar um separador decimal

For i in range (0,4):

Incluir a vírgula como  
separador decimal

```
    ax.annotate('{:,.0f}'.format(soma_ano.values[i]),  
               (soma_ano.index[i], soma_ano.values[i]),
```

```
        ha="center",  
        xytext=(0,5),  
        textcoords="offset points",  
        fontsize=12,  
        color="green",  
        fontweight="bold"
```

)



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Vamos trocar o separador decimal para ponto utilizando a função **replace**

For i in range (0,4):

```
    ax.annotate('{:.0f}'.format(soma_ano.values[i]).replace(',', '.'),  
              (soma_ano.index[i], soma_ano.values[i]),  
              ha="center",  
              xytext=(0,5),  
              textcoords="offset points",  
              fontsize=12,  
              color="green",  
              fontweight="bold")
```

Substitui a vírgula por ponto

Repare que esse valor está cortado



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Para o eixo y, podemos fazer igual fizemos para o eixo x

- Com o `.set_yticks` vamos ajustar os valores do eixo y

```
ax.set_yticks([0,780000])
```



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

O **.yaxis.set\_visible(False)** permite retirar os valores do eixo y, tornando nosso visual mais claro

```
ax.set_yticks([0,780000])
```

```
ax.yaxis.set_visible(False)
```



## Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Já para retirar as bordas do gráfico, basta usar o **.spines['posição'].set\_visible(False)**

```
ax.set_yticks([0,780000])
```

```
ax.yaxis.set_visible(False)
```

```
ax.spines['top'].set_visible(False)
```

```
ax.spines['left'].set_visible(False)
```

```
ax.spines['right'].set_visible(False)
```



## Módulo 9 – Vendas por mês e transformando índices em colunas com o reset\_index

Da mesma forma que fizemos as vendas por ano, também podemos mostrar essa visão por meses.

- Para isso, podemos usar o **.month** da biblioteca datetime para criar uma nova coluna apenas com o mês dessa base

Então vamos criar uma coluna com o mês:

**base['Mes'] = base['Order Date'].dt.month**

**base.head()**

Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales	Ano	Mes
Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	261.9600	2017	8
Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...	731.9400	2017	8
Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240	Office Supplies	Labels	Self-Adhesive Address Labels for Typewriters b...	14.6200	2017	12
Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table	957.5775	2016	11
Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000760	Office Supplies	Storage	Eldon Fold 'N Roll Cart System	22.3680	2016	11

## Módulo 9 – Vendas por mês e transformando índices em colunas com o reset\_index

Agora vamos utilizar a função group by para agrupar os valores:

```
base.groupby('Mes')['Sales'].sum()
```

Mes	Sales
1	155990.9154
2	131153.2594
3	212256.1344
4	142984.2481
5	166000.7467
6	142034.7713
7	159472.5865
8	207601.7939
9	245155.0671
10	183851.9562
11	268768.7885
12	246266.5152

Valores de todos os anos

Name: Sales, dtype: float64

## Módulo 9 – Vendas por mês e transformando índices em colunas com o reset\_index

Agora vamos utilizar a função group by para agrupar os valores:

```
base.groupby('Mes')['Sales'].sum()
```

Para corrigir esse problema vamos utilizar duas colunas no groupby:

```
base.groupby(['Mes','Ano'])['Sales'].sum()
```

Mes	Ano	Sales
1	2015	28828.2540
	2016	29347.3864
	2017	38048.1840
	2018	59767.0910
	2015	12588.4840
	2016	20728.3520
	2017	49907.5900
	2018	48928.8334
	2015	54027.6920
	2016	34489.6776
	2017	48990.1410
	2018	74748.6238
4	2015	24710.0160
	2016	38056.9685
	2017	42368.0480
	2018	37849.2156
	2015	29520.4900
	2016	30761.5585
	2017	64836.2518
	2018	40882.4464
	2015	29181.3346
	2016	28515.9082
	2017	37424.6810
	2018	46912.8475
7	2015	35194.5580
	2016	28573.3100
	2017	41761.9430
	2018	53942.7755
	2015	37349.2655
	2016	49076.9300
	2017	45766.8144
	2018	75408.7840
	2015	65956.3998
	2016	65352.9970
	2017	40692.3063
	2018	73153.3640
10	2015	34561.9470
	2016	31631.8890
	2017	52156.9580
	2018	65581.1622
	2015	64369.4565
	2016	50009.1450
	2017	66392.5470
	2018	87997.6400
	2015	63568.3107
	2016	52891.8832
	2017	72847.0855
	2018	56959.2358

Name: Sales, dtype: float64

## Módulo 9 – Vendas por mês e transformando índices em colunas com o reset\_index

```
MultiIndex([( 1, 2015),  
           ( 1, 2016),  
           ( 1, 2017),  
           ( 1, 2018),  
           ( 2, 2015),  
           ( 2, 2016),  
           ( 2, 2017),  
           ( 2, 2018),  
           ( 3, 2015),  
           ( 3, 2016),  
           ( 3, 2017),  
           ( 3, 2018),  
           ( 4, 2015),  
           ( 4, 2016),  
           ( 4, 2017),  
           ( 4, 2018),  
           ( 5, 2015),  
           ( 5, 2016),  
           ( 5, 2017),  
           ( 5, 2018),  
           ( 6, 2015),  
           ( 6, 2016),  
           ( 6, 2017),  
           ( 6, 2018),  
           ( 7, 2015),  
           ( 7, 2016),  
           ( 7, 2017),  
           ( 7, 2018),  
           ( 8, 2015),  
           ( 8, 2016),  
           ( 8, 2017),  
           ( 8, 2018),  
           ( 9, 2015),  
           ( 9, 2016),  
           ( 9, 2017),  
           ( 9, 2018),  
           (10, 2015),  
           (10, 2016),  
           (10, 2017),  
           (10, 2018),  
           (11, 2015),  
           (11, 2016),  
           (11, 2017),  
           (11, 2018),  
           (12, 2015),  
           (12, 2016),  
           (12, 2017),  
           (12, 2018)],  
          names=['Mes', 'Ano'])
```

Para facilitar o nosso código podemos colocar esses valores agrupados em uma variável:

**soma\_mes = base.groupby(['Mes', 'Ano'])['Sales'].sum()**

Agora, o **soma\_mes.index** é uma dupla de valores

## Módulo 9 – Vendas por mês e transformando índices em colunas com o `reset_index`

Para corrigir esse índice vamos transformar ele em **dataframe**.

Com o `.reset_index()` conseguimos transformar os índices em colunas e então aplicar todas as propriedades que conhecemos para o **DataFrame**.

```
soma_mes = soma_mes.reset_index()
```

```
soma_mes.head()
```

	level_0	index	Mes	Ano	Sales
0	0	0	1	2015	28828.2540
1	1	1	1	2016	29347.3864
2	2	2	1	2017	38048.1840
3	3	3	1	2018	59767.0910
4	4	4	2	2015	12588.4840

## Módulo 9 – Vendas por mês e transformando índices em colunas com o reset\_index

Supondo que queremos filtrar apenas o ano de 2015, como poderíamos fazer?

```
soma_mes[soma_mes.Ano == 2015]
```

soma_mes[soma_mes.Ano == 2015]					
	level_0	index	Mes	Ano	Sales
0	0	0	1	2015	28828.2540
4	4	4	2	2015	12588.4840
8	8	8	3	2015	54027.6920
12	12	12	4	2015	24710.0160
16	16	16	5	2015	29520.4900
20	20	20	6	2015	29181.3346
24	24	24	7	2015	35194.5580
28	28	28	8	2015	37349.2655
32	32	32	9	2015	65956.3998
36	36	36	10	2015	34561.9470
40	40	40	11	2015	64369.4565
44	44	44	12	2015	63568.3107

## Módulo 9 – Vendas por mês e transformando índices em colunas com o reset\_index

Supondo que queremos filtrar apenas o mês 1, como poderíamos fazer?

```
soma_mes[soma_mes.Mes == 1]
```

soma_mes[soma_mes.Mes == 1]					
	level_0	index	Mes	Ano	Sales
0	0	0	1	2015	28828.2540
1	1	1	1	2016	29347.3864
2	2	2	1	2017	38048.1840
3	3	3	1	2018	59767.0910

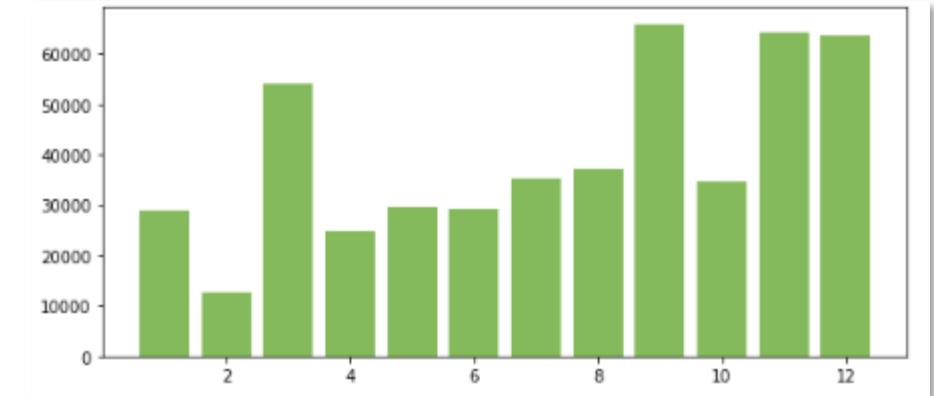
## Módulo 9 – Entendendo o deslocamento das barras em um único gráfico de barras horizontais

Vamos realizar um gráfico, para isso vamos usar como base o nosso gráfico anterior substituindo os valores de x e y:

```
fig,ax = plt.subplots(figsize = (9,4))
```

```
ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values,  
       soma_mes[soma_mes.Ano == 2015].Sales.values,  
       color="#84ba5b", label=2015)
```

```
plt.show()
```



## Módulo 9 – Entendendo o deslocamento das barras em um único gráfico de barras horizontais

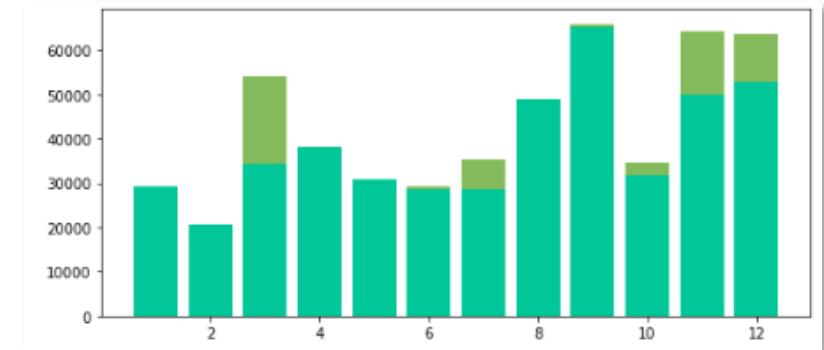
Como poderíamos fazer o mesmo gráfico para 2016?

```
fig,ax = plt.subplots(figsize = (9,4))

ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values,
       soma_mes[soma_mes.Ano == 2015].Sales.values,
       color="#84ba5b", label=2015)

ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values,
       soma_mes[soma_mes.Ano == 2016].Sales.values,
       color="#00c698", label=2016)

plt.show()
```



## Módulo 9 – Entendendo o deslocamento das barras em um único gráfico de barras horizontais

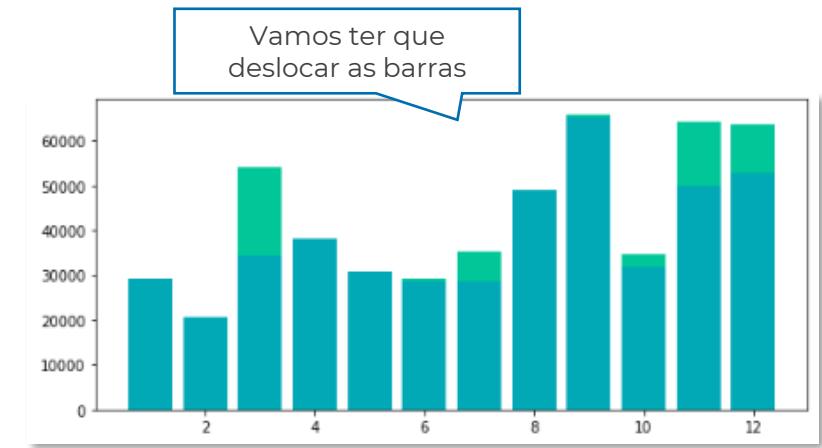
Como poderíamos fazer o mesmo gráfico para 2016?

```
fig,ax = plt.subplots(figsize = (9,4))

ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values,
       soma_mes[soma_mes.Ano == 2015].Sales.values,
       color="#00c698", label=2015)

ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values,
       soma_mes[soma_mes.Ano == 2016].Sales.values,
       color="#00a9b5", label=2016)

plt.show()
```



## Módulo 9 – Entendendo o deslocamento das barras em um único gráfico de barras horizontais

Vamos ajustar essa parte do código adicionando as partes do tamanho das barras

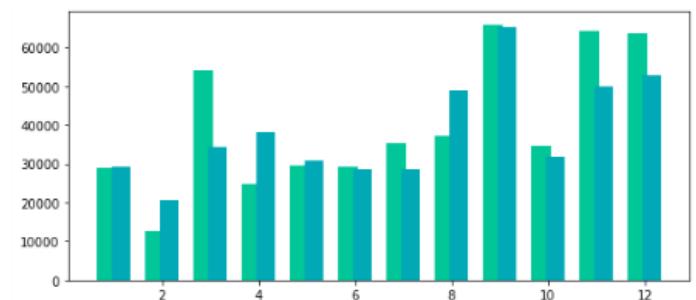
```
fig,ax = plt.subplots(figsize = (9,4))
```

```
ax.bar(soma_mes[soma_mes.Ano== 2015].Mes.values-0.4/2, soma_mes[soma_mes.Ano== 2015].Sales.values, color="#00c698", label=2015, width = 0.4)
```

```
ax.bar(soma_mes[soma_mes.Ano==2016].Mes.values+0.4/2, soma_mes[soma_mes.Ano== 2016].Sales.values, color="#00a9b5", label=2016, width = 0.4)
```

```
plt.show()
```

Deslocando a barra para esquerda



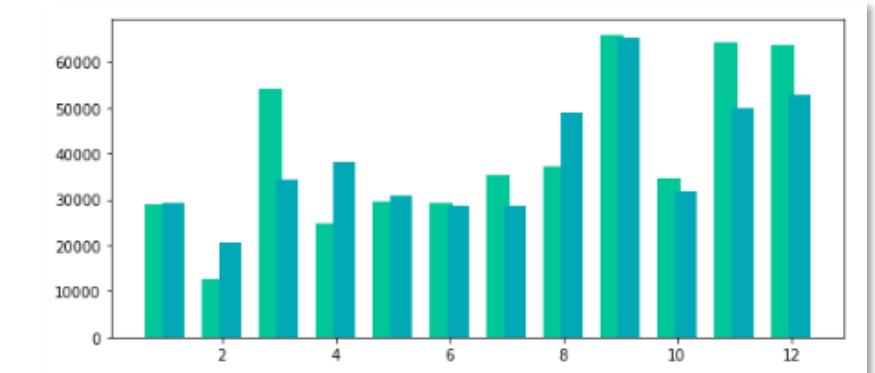
## Módulo 9 – Entendendo o deslocamento das barras em um único gráfico de barras horizontais

```
fig,ax = plt.subplots(figsize = (9,4))  
wid = 0.4
```

```
ax.bar(soma_mes[soma_mes.Ano==2015].Mes.values-wid/2,soma_mes[soma_mes.Ano  
== 2015].Sales.values, color="#00c698", label=2015, width = wid)
```

```
ax.bar(soma_mes[soma_mes.Ano==2016].Mes.values+wid/2,soma_mes[soma_mes.An  
o == 2016].Sales.values, color="#00a9b5", label=2016, width = wid)
```

```
plt.show()
```



## Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

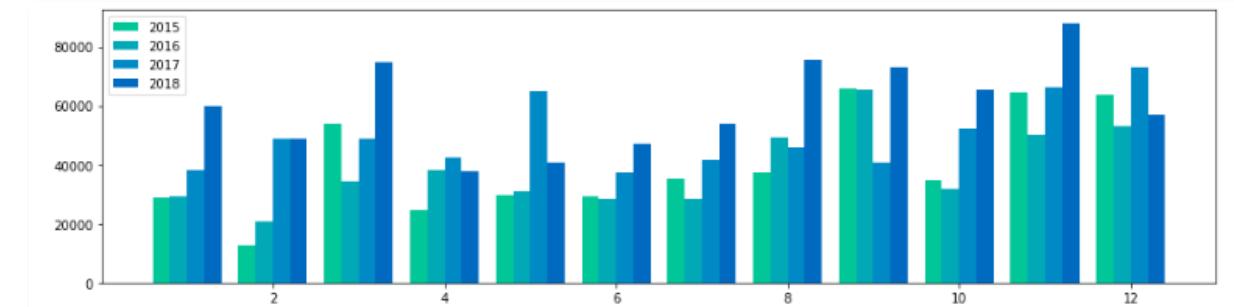
Agora vamos fazer esse mesmo gráfico para os anos de 2017 e 2018:

```
fig,ax = plt.subplots(  
    figsize = (16,4) # Tamanho da figura  
)  
  
wid = 0.2 # Tamanho das barras  
  
ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values-wid-wid/2, # Ajuste da posição das barras  
       soma_mes[soma_mes.Ano == 2015].Sales.values,  
       color="#00c698", # Cor das barras  
       label=2015, # Label dos dados  
       width=wid # Largura das barras  
)  
ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values-wid/2, # Ajuste da posição das barras  
       soma_mes[soma_mes.Ano == 2016].Sales.values,  
       color="#00a9b5",  
       label=2016,  
       width=wid  
)  
  
# Adicionando o restante das barras  
ax.bar(soma_mes[soma_mes.Ano == 2017].Mes.values+wid/2, # Ajuste da posição das barras  
       soma_mes[soma_mes.Ano == 2017].Sales.values,  
       color="#008ac5",  
       label=2017,  
       width=wid  
)  
ax.bar(soma_mes[soma_mes.Ano == 2018].Mes.values+wid+wid/2, # Ajuste da posição das barras  
       soma_mes[soma_mes.Ano == 2018].Sales.values,  
       color="#0069c0",  
       label=2018,  
       width=wid  
)  
ax.legend()  
plt.show()
```

Ajuste da largura

Colocar o 2016 e 2017  
no centro

Ajuste da largura



## Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

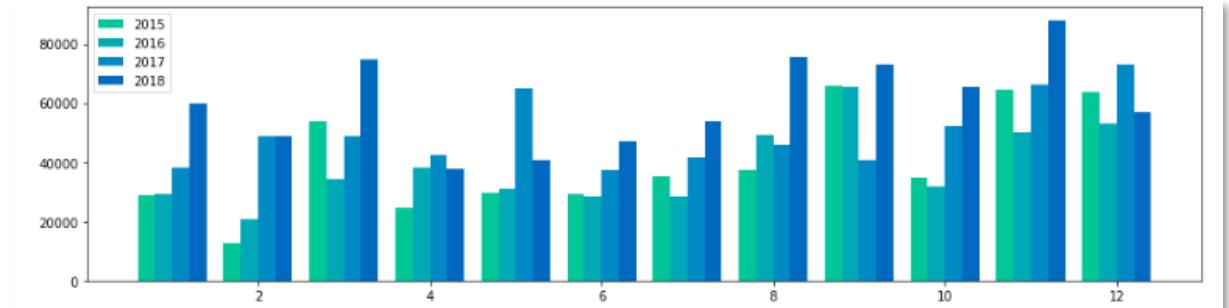
Agora vamos fazer esse mesmo gráfico para os anos de 2017 e 2018:

```
fig,ax = plt.subplots(  
    figsize = (16,4) # Tamanho da figura  
)  
  
wid = 0.2 # Tamanho das barras  
  
ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values-wid-wid/2, # Ajuste da posição das barras  
       soma_mes[soma_mes.Ano == 2015].Sales.values,  
       color="#00c698", # Cor das barras  
       label=2015, # Label dos dados  
       width=wid # Largura das barras  
)  
ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values-wid/2, # Ajuste da posição das barras  
       soma_mes[soma_mes.Ano == 2016].Sales.values,  
       color="#00a9b5",  
       label=2016,  
       width=wid  
)  
  
# Adicionando o restante das barras  
ax.bar(soma_mes[soma_mes.Ano == 2017].Mes.values+wid/2, # Ajuste da posição das barras  
       soma_mes[soma_mes.Ano == 2017].Sales.values,  
       color="#008ac5",  
       label=2017,  
       width=wid  
)  
ax.bar(soma_mes[soma_mes.Ano == 2018].Mes.values+wid+wid/2, # Ajuste da posição das barras  
       soma_mes[soma_mes.Ano == 2018].Sales.values,  
       color="#0069c0",  
       label=2018,  
       width=wid  
)  
ax.legend()  
plt.show()
```

Ajuste da largura

Colocar o 2016 e 2017  
no centro

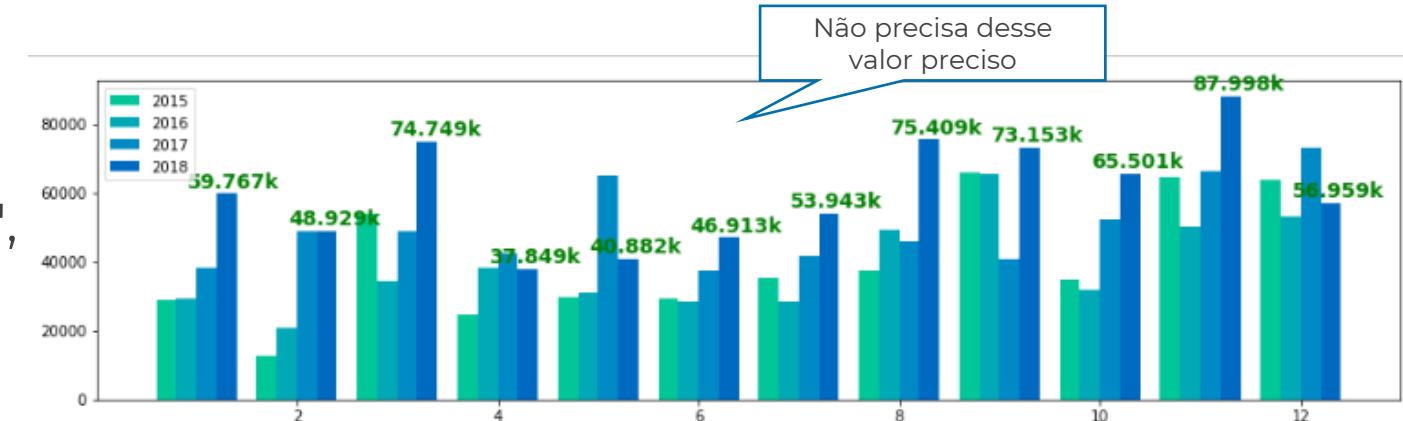
Ajuste da largura



## Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

Então vamos utilizar o código que já utilizamos do **annotate** apenas em 2018, para o gráfico não ficar muito poluído.

```
for i in np.arange(0,12):
    ax.annotate('{:.0f}k'.format(soma_mes[soma_mes.Ano==2018].Sales.values[i])
.replace('.','.'), (soma_mes[soma_mes.Ano==2018].Mes.values[i]+wid+wid/2,
                    soma_mes[soma_mes.Ano == 2018].Sales.values[i]),
                ha="center",
                va="top",
                xytext=(0,+15),
                textcoords="offset points",
                fontsize=14,
                fontweight='bold',
                color="green")
```



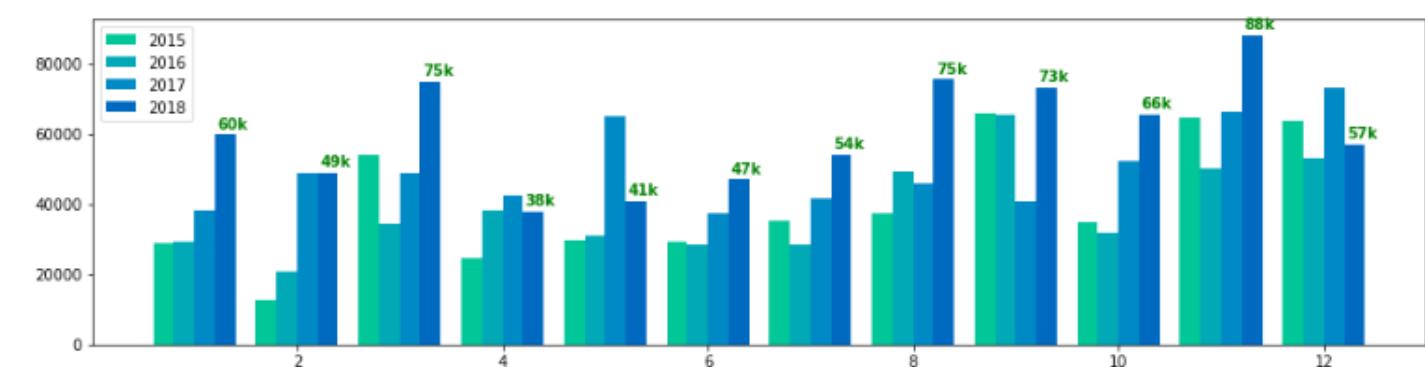
## Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

```
for i in np.arange(0,12):
```

Colocar o K de mil

```
    ax.annotate('{:.0f}k'.format(soma_mes[soma_mes.Ano==2018].Sales.values[i]/1000),  
              (soma_mes[soma_mes.Ano==2018].Mes.values[i]+wid+wid/2,  
               soma_mes[soma_mes.Ano == 2018].Sales.values[i]),  
              ha="center",  
              va="top",  
              xytext=(0,+15),  
              textcoords="offset points",  
              fontsize=14,  
              fontweight='bold',  
              color="green")
```

Dividindo o valor de vendas por 1000



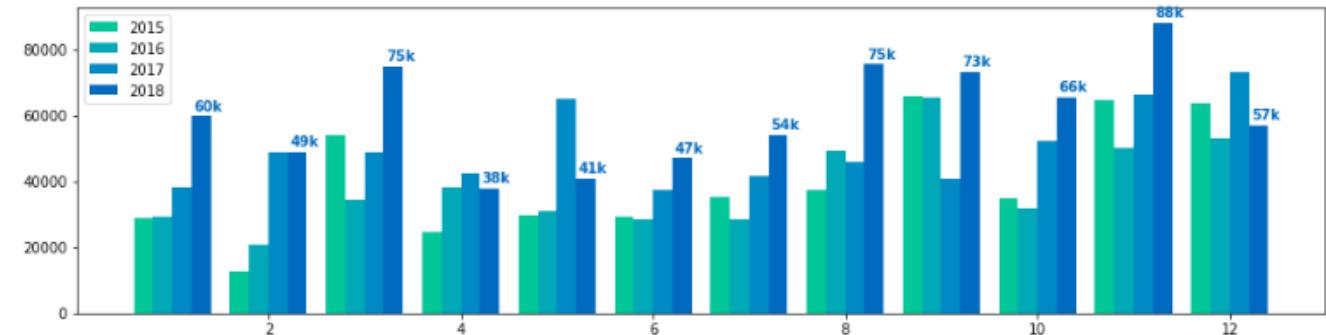
## Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

```
for i in np.arange(0,12):
```

```
    ax.annotate('{:.0f}k'.format(soma_mes[soma_mes.Ano==2018].Sales.values[i]/1000),
                (soma_mes[soma_mes.Ano==2018].Mes.values[i]+wid+wid/2,
                 soma_mes[soma_mes.Ano == 2018].Sales.values[i]),
                ha="center",
                va="top",
                xytext=(5,12),
                textcoords="offset points",
                fontsize=10,
                fontweight='bold',
                color="#0069c0")
```

Alteração da posição

Alteração da cor

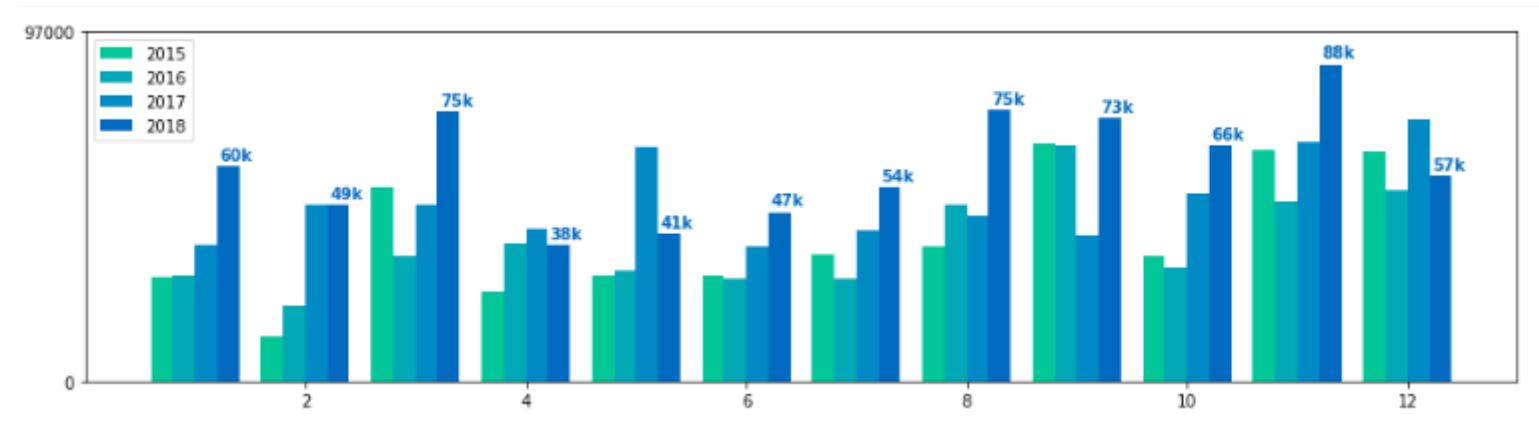


## Módulo 9 – Mudando os rótulos do eixo x e finalizando o visual de venda do mês

Agora vamos finalizar o visual, para isso vamos aproveitar o código que fizemos para o ano:

Vamos aumentar o eixo y:

```
ax.set_yticks(np.array([0,97000]))
```



## Módulo 9 – Mudando os rótulos do eixo x e finalizando o visual de venda do mês

Agora vamos retirar as bordas:

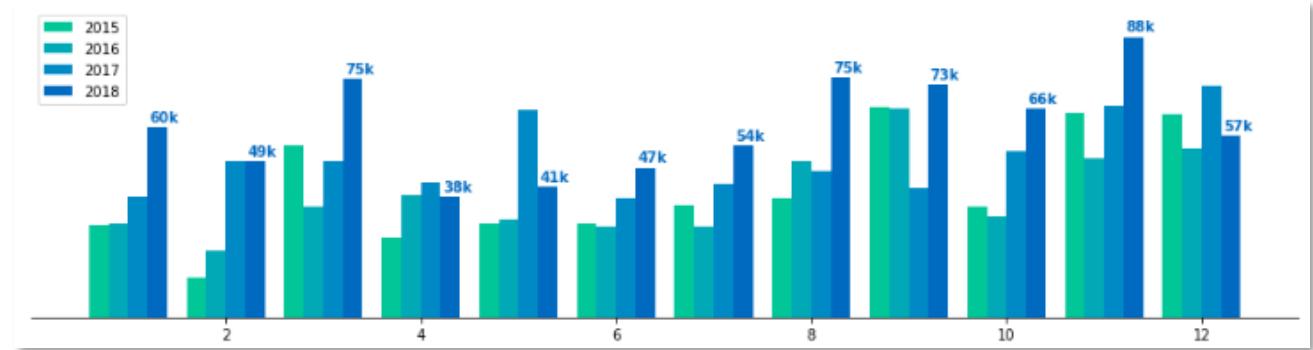
```
ax.set_yticks(np.array([0,97000]))
```

```
ax.yaxis.set_visible(False)
```

```
ax.spines['top'].set_visible(False)
```

```
ax.spines['left'].set_visible(False)
```

```
ax.spines['right'].set_visible(False)
```

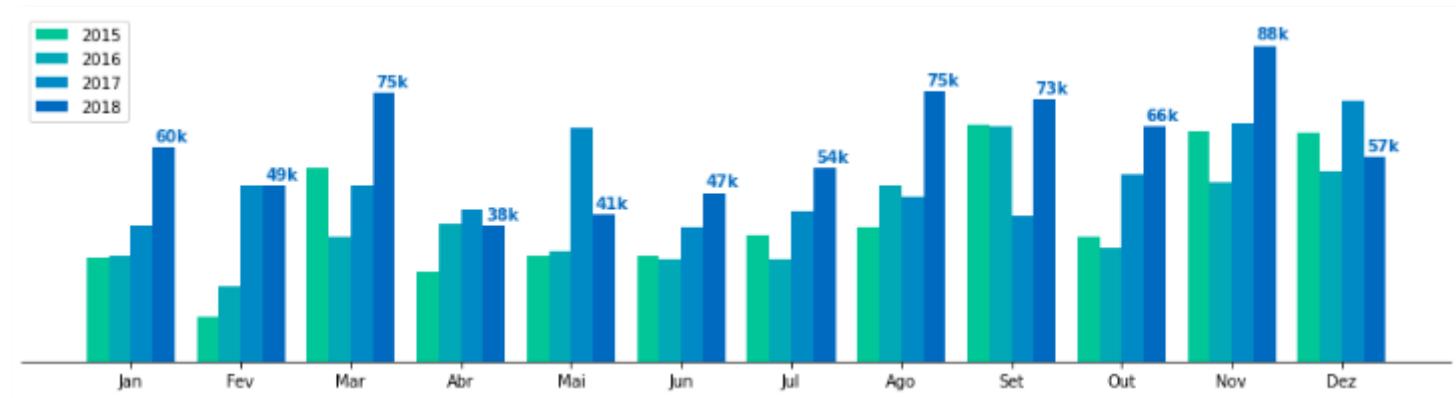


## Módulo 9 – Mudando os rótulos do eixo x e finalizando o visual de venda do mês

Agora vamos ajustar os valores de x:

```
ax.xaxis.set_ticks(np.arange(1,13))
```

```
ax.set_xticklabels(['Jan','Fev','Mar','Abr','Mai','Jun','Jul','Ago','Set','Out','Nov','Dez'])
```



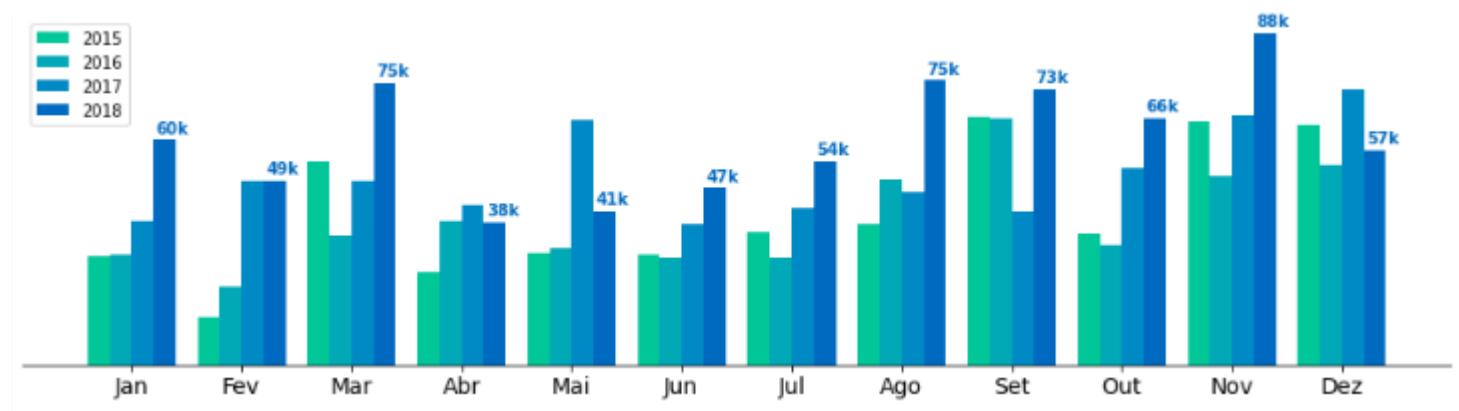
## Módulo 9 – Mudando os rótulos do eixo x e finalizando o visual de venda do mês

Aumentando a fonte do eixo x:

```
ax.xaxis.set_ticks(np.arange(1,13))
```

```
ax.set_xticklabels(['Jan','Fev','Mar','Abr','Mai','Jun','Jul','Ago','Set','Out','Nov','Dez'])
```

```
ax.tick_params(axis='x',labelsize=14)
```



## Módulo 9 – Respondendo qual foi a categoria mais vendida

O primeiro passo para responder qual a categoria mais vendida é verificar na base qual coluna poderíamos utilizar:

**base.head(2)**

Então, podemos ver que já existe uma coluna para categoria

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
1	CA-152156	2017-08-11	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
2	CA-152156	2017-08-11	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture

## Módulo 9 – Respondendo qual foi a categoria mais vendida

O primeiro passo para responder qual a categoria mais vendida é verificar na base qual coluna poderíamos utilizar:

**base.head(2)**

Então, podemos ver que já existe uma coluna para categoria

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
1	CA-2017-152156	2017-08-11	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
2	CA-2017-152156	2017-08-11	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture

Coluna que vamos utilizar

## Módulo 9 – Respondendo qual foi a categoria mais vendida

Então, o segundo passo é utilizar novamente o group by:

```
base.groupby("Category")["Sales"].sum()
```

Dessa forma, ainda não conseguimos passar a informação de categoria por ano, para isso vamos utilizar um group by com o ano também

```
Category
Furniture      728658.5757
Office Supplies 705422.3340
Technology     827455.8730
Name: Sales, dtype: float64
```

## Módulo 9 – Respondendo qual foi a categoria mais vendida

Então alterando o nosso group by para:

```
categoria = base.groupby(["Category","Ano"])["Sales"].sum()
```

categoria

```
categoria = base.groupby(["Category","Ano"])["Sales"].sum()
categoria
```

Category	Ano	Sales
Furniture	2015	156477.8811
	2016	164053.8674
	2017	195813.0400
	2018	212313.7872
Office Supplies	2015	149512.8200
	2016	133124.4070
	2017	182417.5660
	2018	240367.5410
Technology	2015	173865.5070
	2016	162257.7310
	2017	221961.9440
	2018	269370.6910

Name: Sales, dtype: float64

Vamos ter que  
transformar em Data  
Frame

## Módulo 9 – Respondendo qual foi a categoria mais vendida

Transformando em Data Frame utilizando o reset\_index:

```
categoria = categoria.reset_index()
```

```
categoria
```

index	Category	Ano	Sales
0	0	Furniture	2015 156477.8811
1	1	Furniture	2016 164053.8674
2	2	Furniture	2017 195813.0400
3	3	Furniture	2018 212313.7872
4	4	Office Supplies	2015 149512.8200
5	5	Office Supplies	2016 133124.4070
6	6	Office Supplies	2017 182417.5660
7	7	Office Supplies	2018 240367.5410
8	8	Technology	2015 173865.5070
9	9	Technology	2016 162257.7310
10	10	Technology	2017 221961.9440
11	11	Technology	2018 269370.6910

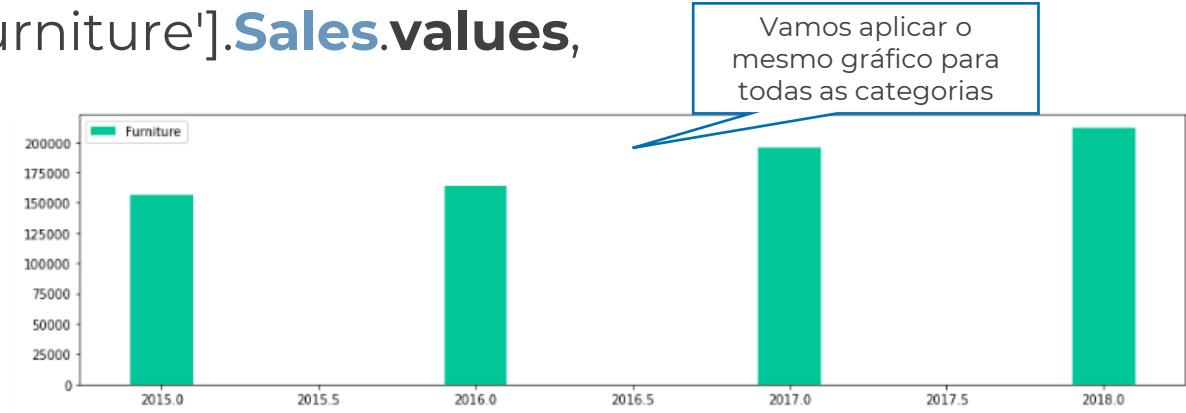
## Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos copiar exatamente o código anterior para editar e criar o nosso gráfico de vendas. Vamos ajustar os valores de x e y:

```
fig,ax = plt.subplots(figsize = (16,4))  
wid = 0.2
```

```
ax.bar(categoria[categoria.Category == 'Furniture'].Ano.values,  
       categoria[categoria.Category == 'Furniture'].Sales.values,  
       color="#00c698",  
       label="Furniture",  
       width=wid)
```

```
ax.legend()  
plt.show()
```



## Módulo 9 – Respondendo qual foi a categoria mais vendida

```
fig,ax = plt.subplots(figsize = (16,4))
ax.bar(categoria[categoria.Category == 'Furniture'].Ano.values,
       categoria[categoria.Category == 'Furniture'].Sales.values,
       color="#00c698", label="Furniture", width=wid)

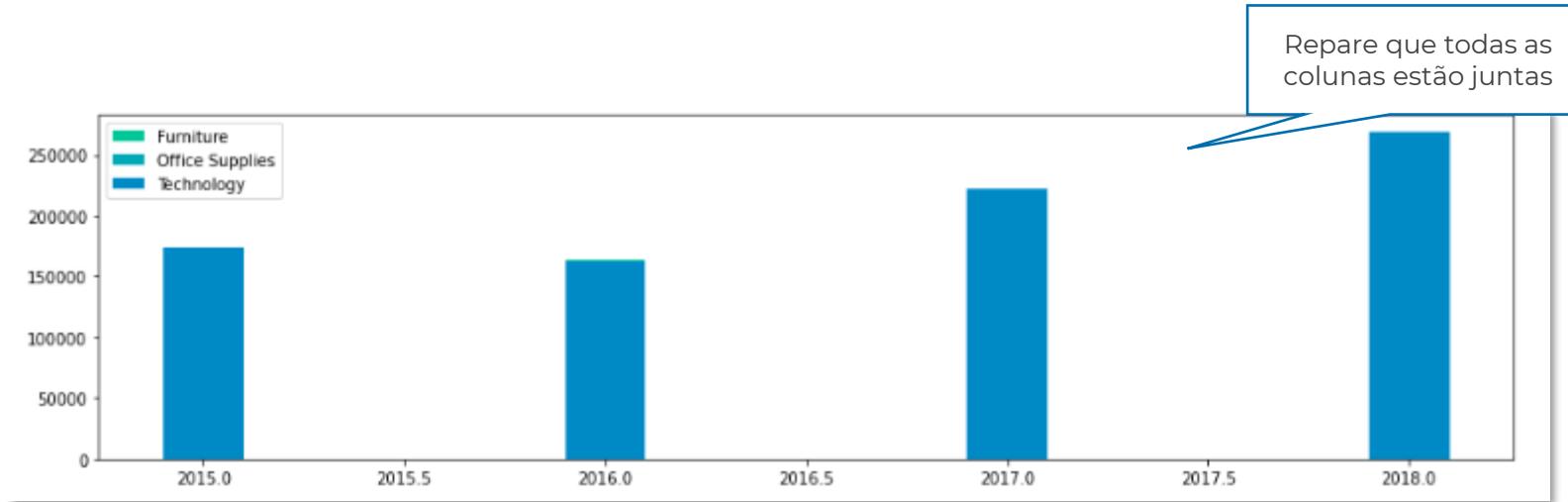
ax.bar(categoria[categoria.Category == 'Office Supplies'].Ano.values,
       categoria[categoria.Category == 'Office Supplies'].Sales.values,
       color="#00a9b5", label="Office Supplies", width=wid)

ax.bar(categoria[categoria.Category == 'Technology'].Ano.values,
       categoria[categoria.Category == 'Technology'].Sales.values,
       color="#008ac5", label="Technology", width=wid)
ax.legend()
plt.show()
```

## Módulo 9 – Respondendo qual foi a categoria mais vendida

Como resultado do código anterior temos.

Agora vamos precisar deslocar uma coluna para direita e outra para a esquerda



## Módulo 9 – Respondendo qual foi a categoria mais vendida

```
fig,ax = plt.subplots(figsize = (16,4))
wid = 0.2
ax.bar(categoria[categoria.Category == 'Furniture'].Ano.values-wid,
        categoria[categoria.Category == 'Furniture'].Sales.values,
        color="#00c698", label="Furniture", width=wid)
```

Deslocar coluna para a esquerda

```
ax.bar(categoria[categoria.Category == 'Office Supplies'].Ano.values,
        categoria[categoria.Category == 'Office Supplies'].Sales.values,
        color="#00a9b5", label="Office Supplies", width=wid)
```

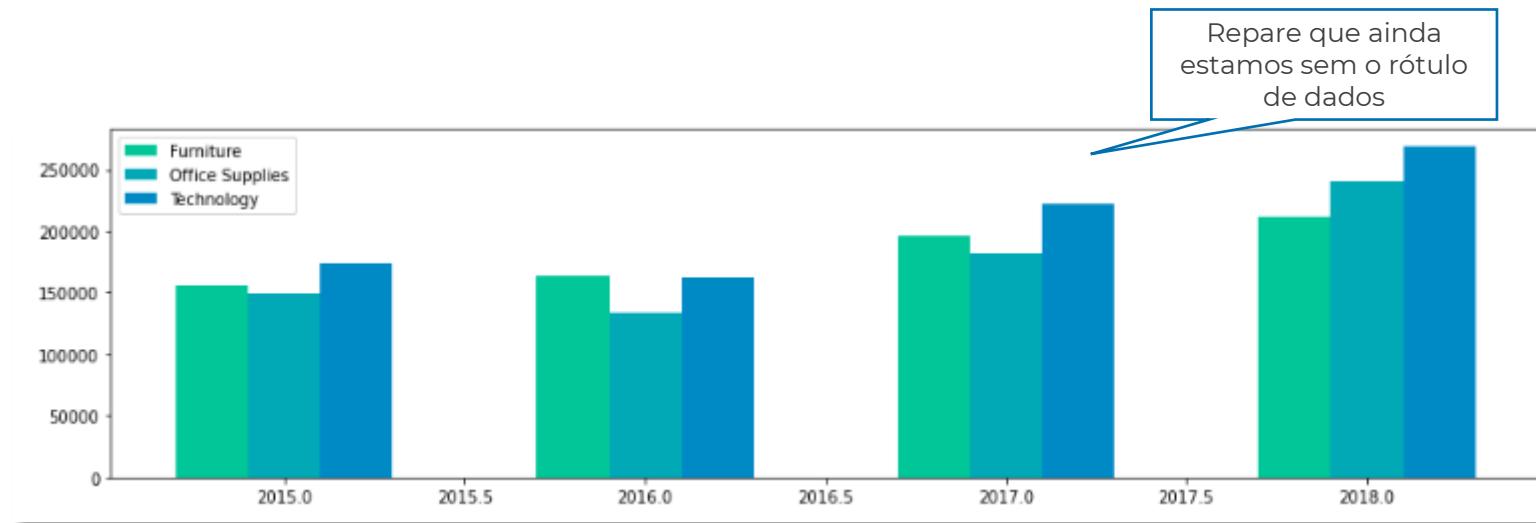
Deslocar coluna para a direita

```
ax.bar(categoria[categoria.Category == 'Technology'].Ano.values+wid,
        categoria[categoria.Category == 'Technology'].Sales.values,
        color="#008ac5", label="Technology", width=wid)
ax.legend()
plt.show()
```

## Módulo 9 – Respondendo qual foi a categoria mais vendida

Então, ficamos com o gráfico abaixo como resultado.

O próximo passo vai ser utilizando o **annotate** do outro gráfico para colocar os rótulos de dados e alterando os valores de x e y:

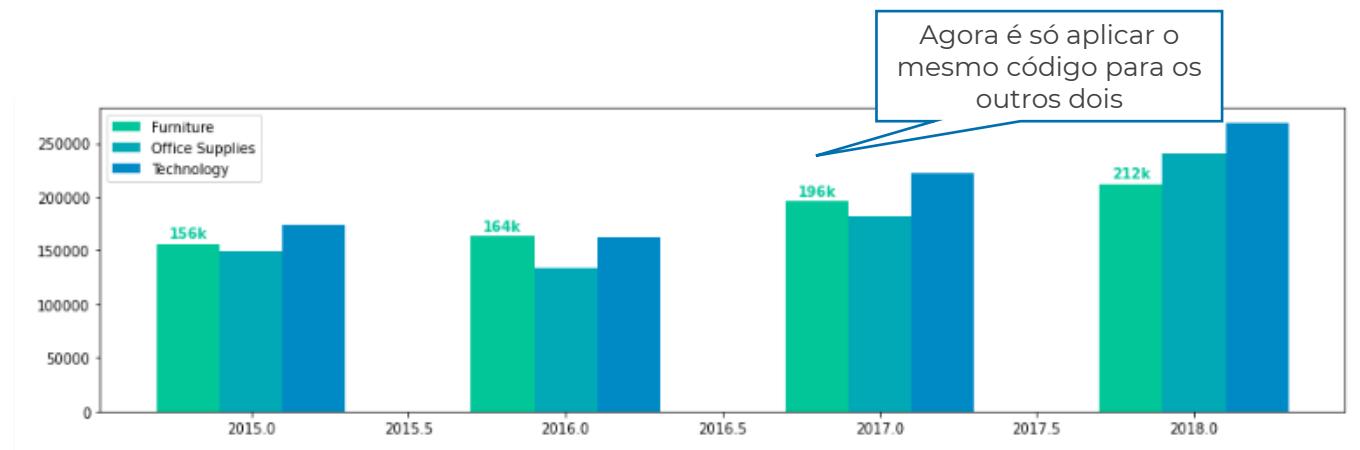


## Módulo 9 – Respondendo qual foi a categoria mais vendida

```
for i in np.arange(0,4):
```

```
    ax.annotate('{:.0f}k'.format(categoria[categoria.Category=='Furniture'].Sales.values[i]/1000),
```

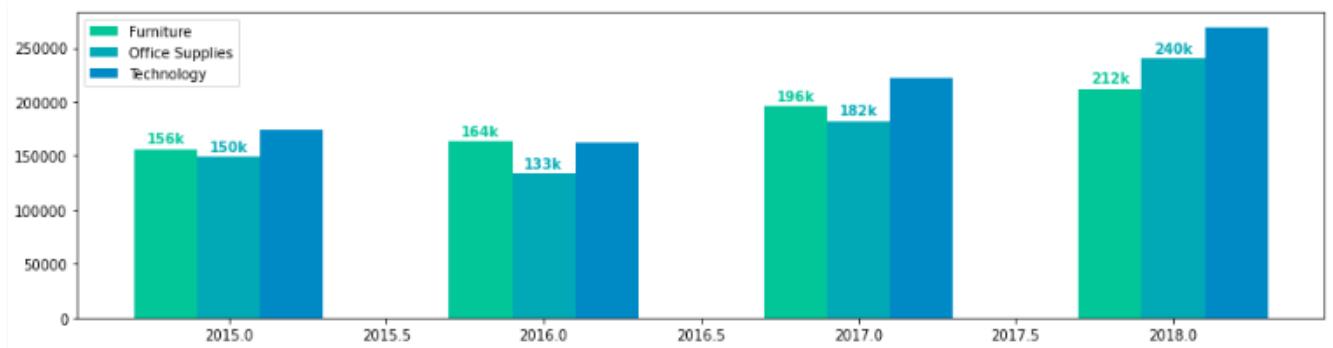
```
        (categoria[categoria.Category == 'Furniture'].Ano.values[i]-wid,
         categoria[categoria.Category == 'Furniture'].Sales.values[i]),
        ha="center",
        va="top",
        xytext=(0,12),
        textcoords="offset points",
        fontsize=10,
        fontweight='bold',
        color="#00c698")
```



## Módulo 9 – Respondendo qual foi a categoria mais vendida

```
for i in np.arange(0,4):
```

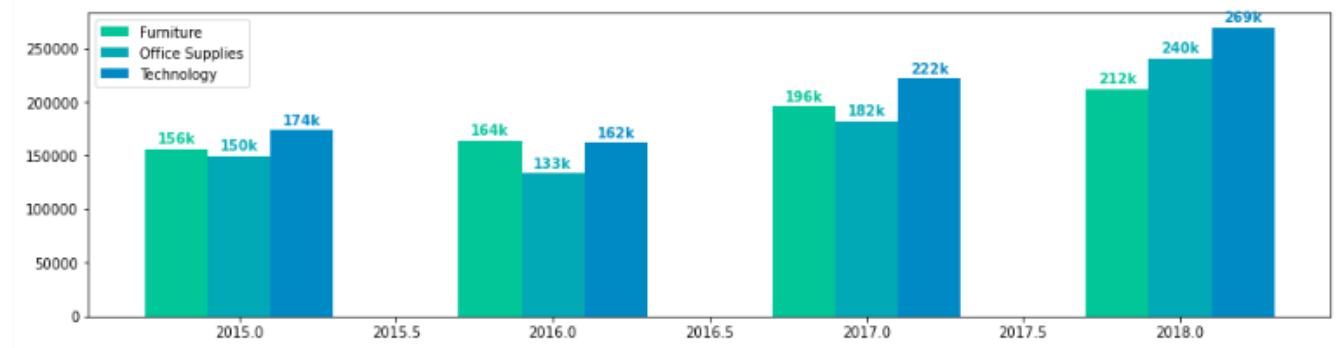
```
    ax.annotate('{:.0f}k'.format(categoria[categoria.Category=='Office Supplies'].Sales.values[i]/1000),  
              (categoria[categoria.Category == 'Office Supplies'].Ano.values[i]-wid,  
               categoria[categoria.Category == 'Office Supplies'].Sales.values[i]),  
              ha="center",  
              va="top",  
              xytext=(0,12),  
              textcoords="offset points",  
              fontsize=10,  
              fontweight='bold',  
              color="#00a9b5")
```



## Módulo 9 – Respondendo qual foi a categoria mais vendida

```
for i in np.arange(0,4):
```

```
    ax.annotate('{:.0f}k'.format(categoria[categoria.Category=='Technology'].Sales.values[i]/1000),  
              (categoria[categoria.Category == 'Technology'].Ano.values[i]-wid,  
               categoria[categoria.Category == 'Technology'].Sales.values[i]),  
              ha="center",  
              va="top",  
              xytext=(0,12),  
              textcoords="offset points",  
              fontsize=10,  
              fontweight='bold',  
              color="#008ac5")
```

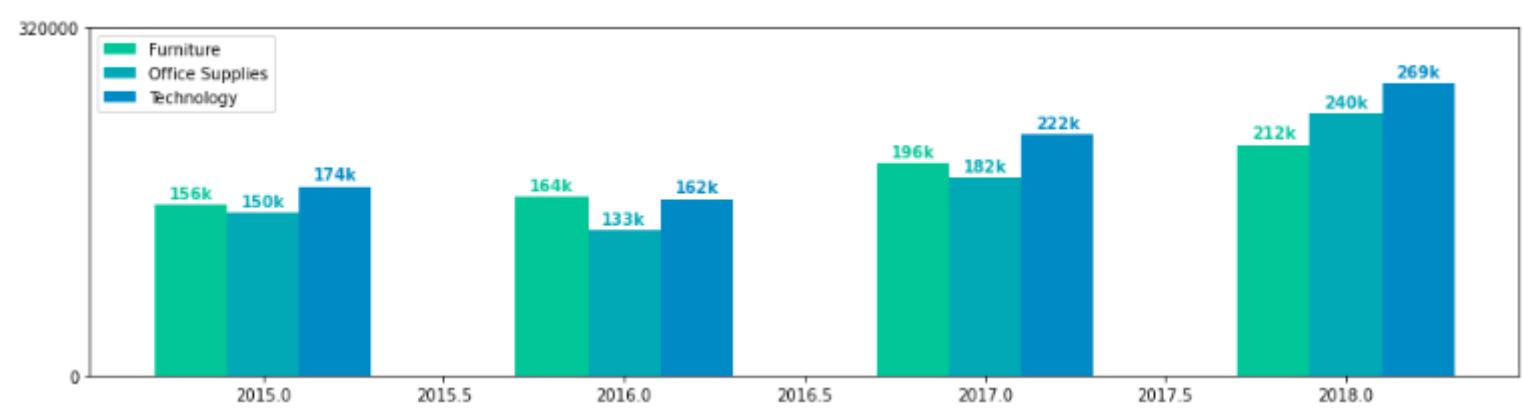


## Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos melhorar os valores do eixo y:

- Diminuindo o tamanho das barras para não cortar o rótulo

```
ax.set_yticks(np.array([0,320000]))
```



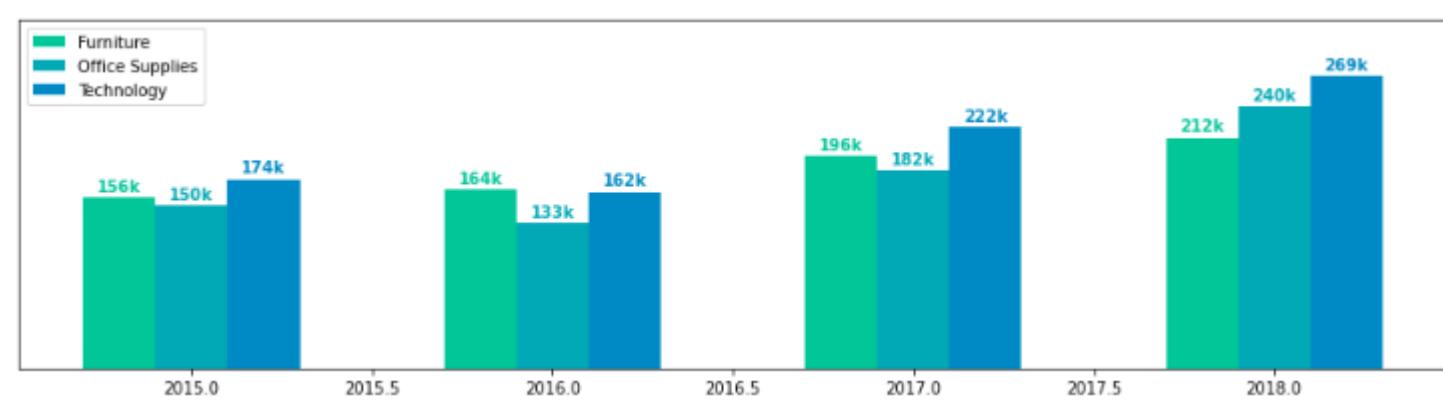
## Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos melhorar os valores do eixo y:

- Retirando os valores do eixo y

```
ax.set_yticks(np.array([0,320000]))
```

```
ax.yaxis.set_visible(False)
```



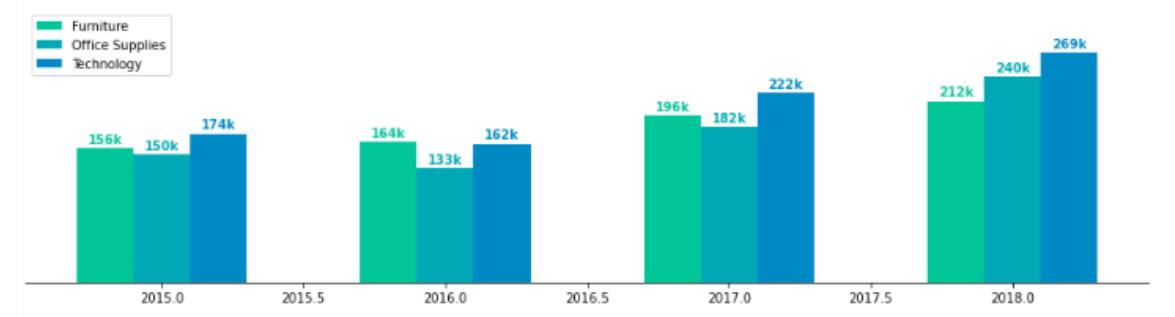
## Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos retirar as bordas do gráfico:

```
ax.spines['top'].set_visible(False)
```

```
ax.spines['left'].set_visible(False)
```

```
ax.spines['right'].set_visible(False)
```

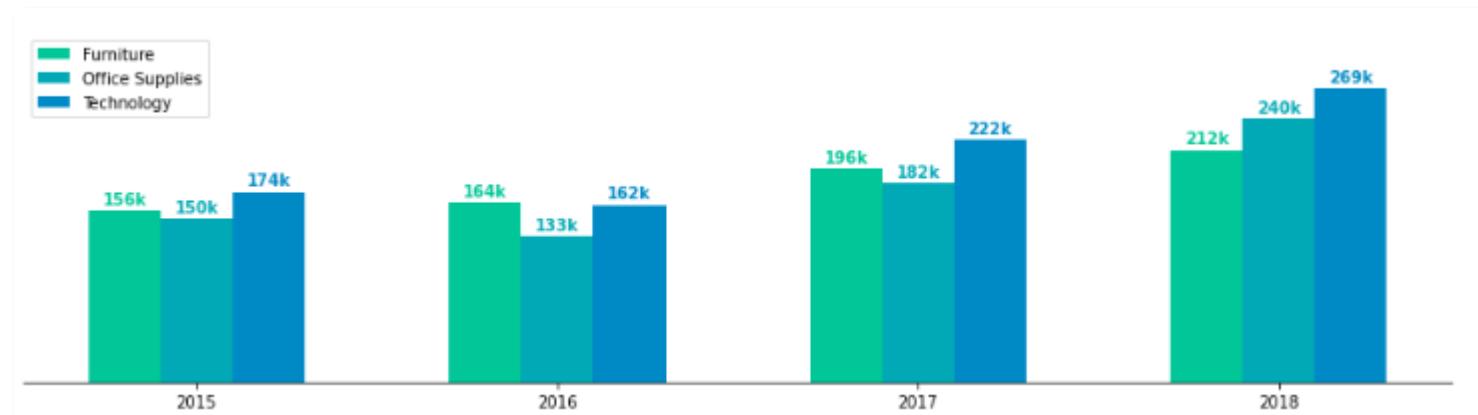


## Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos melhorar os valores do eixo x:

- Ajustando os valores de x

```
ax.xaxis.set_ticks([2015,2016,2017,2018])
```

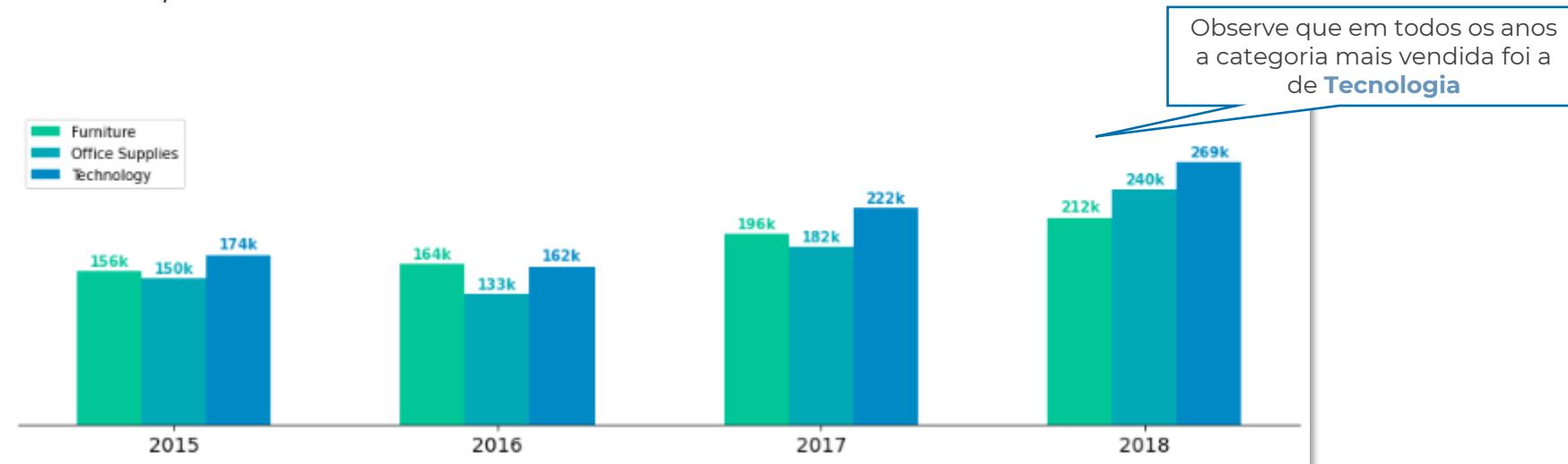


## Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos melhorar os valores do eixo x:

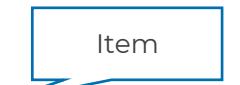
- Melhorando o visual do eixo x

`ax.tick_params(axis='x',labelsize=14)`



## Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

O primeiro passo para responder qual foi o item mais vendido é verificar na nossa base qual a coluna nós vamos utilizar.



Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales	Ano	Mes
Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	261.96	2017	8
Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs....	731.94	2017	8

## Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Podemos fazer um group by para os itens, utilizando a mesma lógica:

```
itens = base.groupby("Product Name")["Sales"].sum()
```

```
itens = itens.reset_index()
```

```
itens.head(10)
```

Repare que os itens não estão ordenados

	Product Name	Sales
0	"While you Were Out" Message Book, One Form per page	25.228
1	#10 Gummed Flap White Envelopes, 100/Box	41.300
2	#10 Self-Seal White Envelopes	108.682
3	#10 White Business Envelopes, 4 1/8" x 9 1/2"	379.214
4	#10- 4 1/8" x 9 1/2" Recycled Envelopes	286.672
5	#10- 4 1/8" x 9 1/2" Security-Tint Envelopes	146.688
6	#10-4 1/8" x 9 1/2" Premium Diagonal Seam Envelopes	176.288
7	#6 3/4 Gummed Flap White Envelopes	71.280
8	1.7 Cubic Foot Compact "Cube" Office Refrigerator	2706.080
9	1/4 Fold Party Design Invitations & White Envelopes	49.980

## Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Vamos colocar em ordem decrescente utilizando o **sort\_values**.

```
itens = base.groupby("Product Name")["Sales"].sum()
```

```
itens = itens.reset_index()
```

```
itens = itens.sort_values("Sales", ascending=False)
```

```
itens.head(10)
```



	Product Name	Sales
404	Canon imageCLASS 2200 Advanced Copier	61599.824
649	Fellowes PB600 Electric Punch Plastic Comb Bin...	27453.384
444	Cisco TelePresence System EX90 Videoconferenci...	22638.480
785	HON 5400 Series Task Chairs for Big and Tall	21870.576
685	GBC DocuBind TL300 Electric Binding System	19823.479
687	GBC Ibimaster 500 Manual ProClick Binding System	19024.500
804	Hewlett Packard LaserJet 3310 Copier	18839.686
786	HP Designjet T520 Inkjet Large Format Printer ...	18374.895
682	GBC DocuBind P400 Electric Binding System	17965.068
812	High Speed Automatic Electric Letter Opener	17030.312

## Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Uma forma de resolver a quantidade de itens seria fazer apenas para os top 10 itens

```
top_20_itens = itens.head(20)
```

```
top_20_itens
```

	Product Name	Sales
404	Canon imageCLASS 2200 Advanced Copier	61599.8240
649	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.3840
444	Cisco TelePresence System EX90 Videoconferenci...	22638.4800
785	HON 5400 Series Task Chairs for Big and Tall	21870.5760
685	GBC DocuBind TL300 Electric Binding System	19823.4790
687	GBC Ibimaster 500 Manual ProClick Binding System	19024.5000
804	Hewlett Packard LaserJet 3310 Copier	18839.6860
786	HP Designjet T520 Inkjet Large Format Printer ...	18374.8950
682	GBC DocuBind P400 Electric Binding System	17985.0680
812	High Speed Automatic Electric Letter Opener	17030.3120
984	Lexmark MX611dhe Monochrome Laser Printer	16829.9010
1042	Martin Yale Chainless Opener Electric Letter Op...	16656.2000
894	Ibico EPK-21 Electric Binding System	15875.9160
1350	Riverside Palais Royal Lawyers Bookcase, Royal...	15610.9656
19	3D Systems Cube Printer, 2nd Generation, Magenta	14299.8900
1387	Samsung Galaxy Mega 6.3	13943.6680
155	Apple iPhone 5	12996.6000
368	Bretford Rectangular Conference Table Tops	12995.2915
768	Global Troy Executive Leather Low-Back Tilter	12975.3820
1362	SAFCO Arco Folding Chair	11572.7800

## Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

A cada mudança do número de itens seria necessário alterar a variável, para isso não termos esse problema vamos utilizar o top N itens:

**n** = 10

**top\_n\_itens = itens.head(n)**

**top\_n\_itens**

	Product Name	Sales
404	Canon imageCLASS 2200 Advanced Copier	61599.824
649	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.384
444	Cisco TelePresence System EX90 Videoconferenci...	22638.480
785	HON 5400 Series Task Chairs for Big and Tall	21870.576
685	GBC DocuBind TL300 Electric Binding System	19823.479
687	GBC Ibimaster 500 Manual ProClick Binding System	19024.500
804	Hewlett Packard LaserJet 3310 Copier	18839.686
786	HP Designjet T520 Inkjet Large Format Printer ...	18374.895
682	GBC DocuBind P400 Electric Binding System	17965.068
812	High Speed Automatic Electric Letter Opener	17030.312

## Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Vamos traçar um gráfico de barras horizontais para mostrar o top 10 itens. Vamos utilizar como base para o nosso gráfico a [documentação](#).

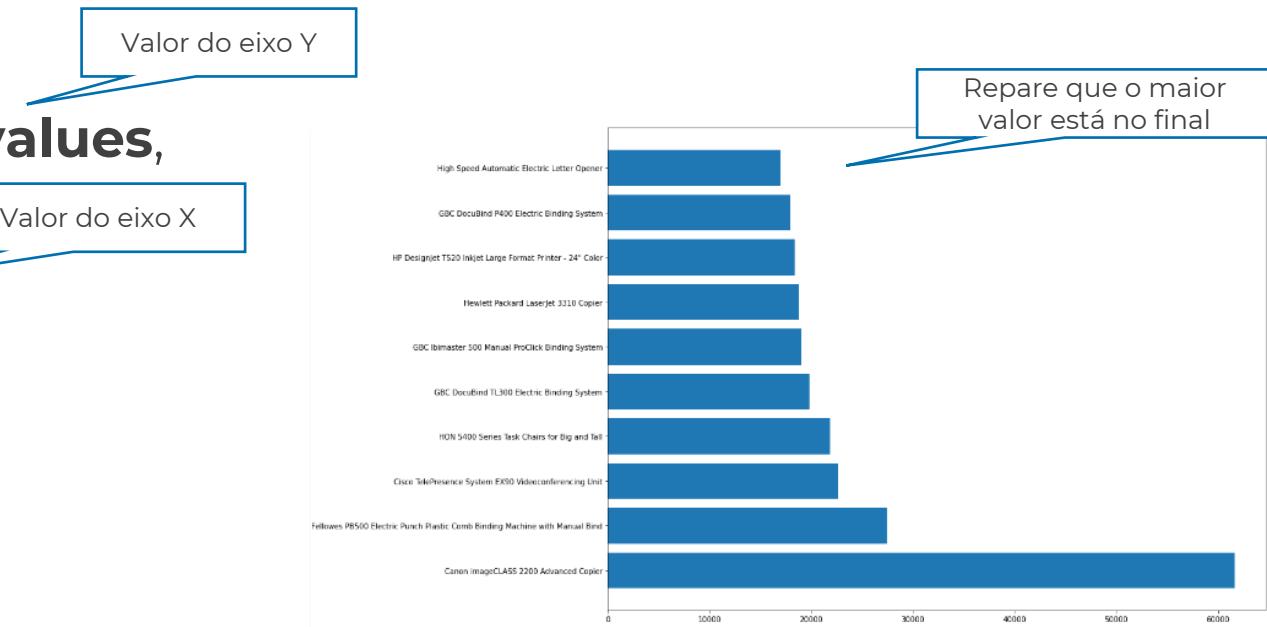
```
fig,ax = plt.subplots(figsize = (16,12))
```

```
ax.barh(top_n_itens['Product Name'].values,
```

```
top_n_itens['Sales'].values,
```

```
align='center')
```

```
plt.show()
```



## Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Vamos traçar um gráfico de barras horizontais para mostrar o top 10 itens. Vamos utilizar como base para o nosso gráfico a [documentação](#).

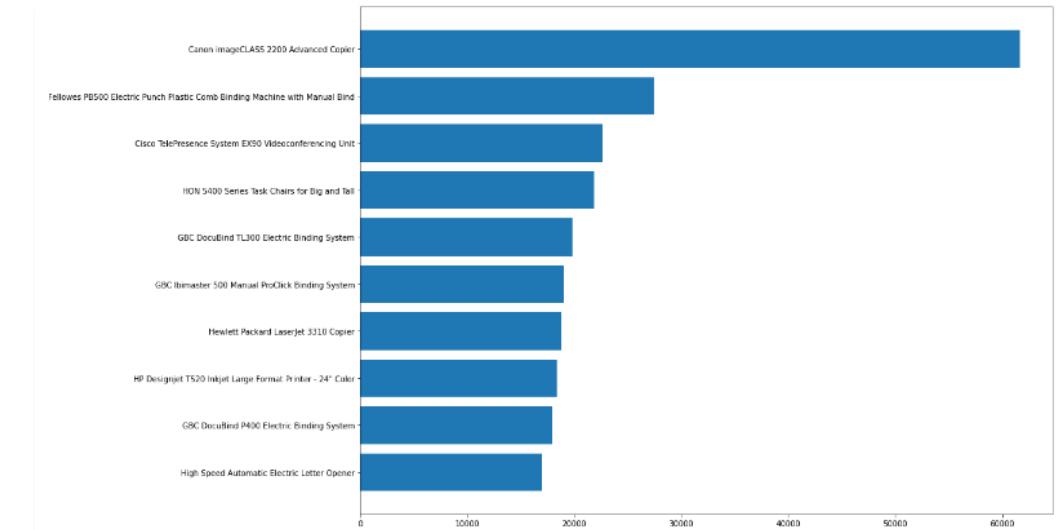
```
fig,ax = plt.subplots(figsize = (16,12))
```

```
ax.barh(top_n_itens['Product Name'].values,  
        top_n_itens['Sales'].values,  
        align='center')
```

Intervertendo o eixo Y

```
ax.invert_yaxis()
```

```
plt.show()
```



## Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

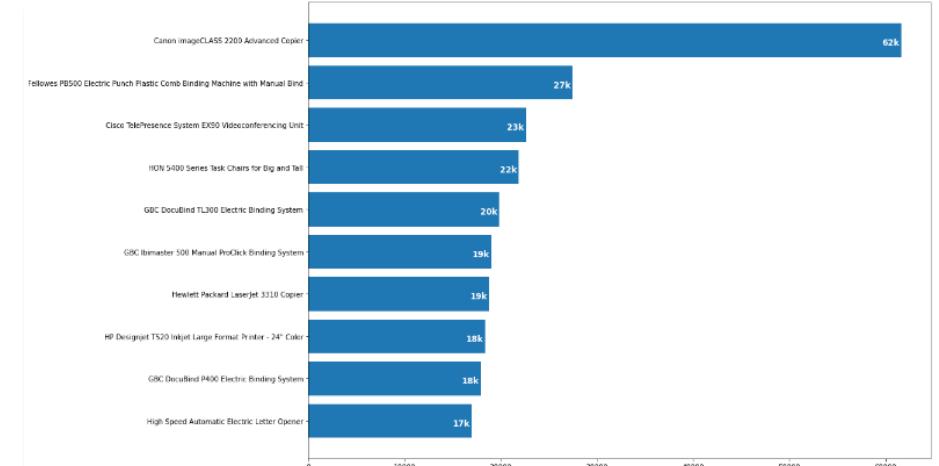
Para melhorar a visualização do gráfico vamos fazer o **annotate**:

```
for i in np.arange(0,n):
```

Annotate vai depender de **n**

```
    ax.annotate('{:.0f}k'.format(top_n_itens['Sales'].values[i]/1000),  
               (top_n_itens['Sales'].values[i],i),  
               ha="center",  
               va="top",  
               xytext=(-15,3),  
               textcoords="offset points",  
               fontsize=12,  
               fontweight='bold',  
               color="white")
```

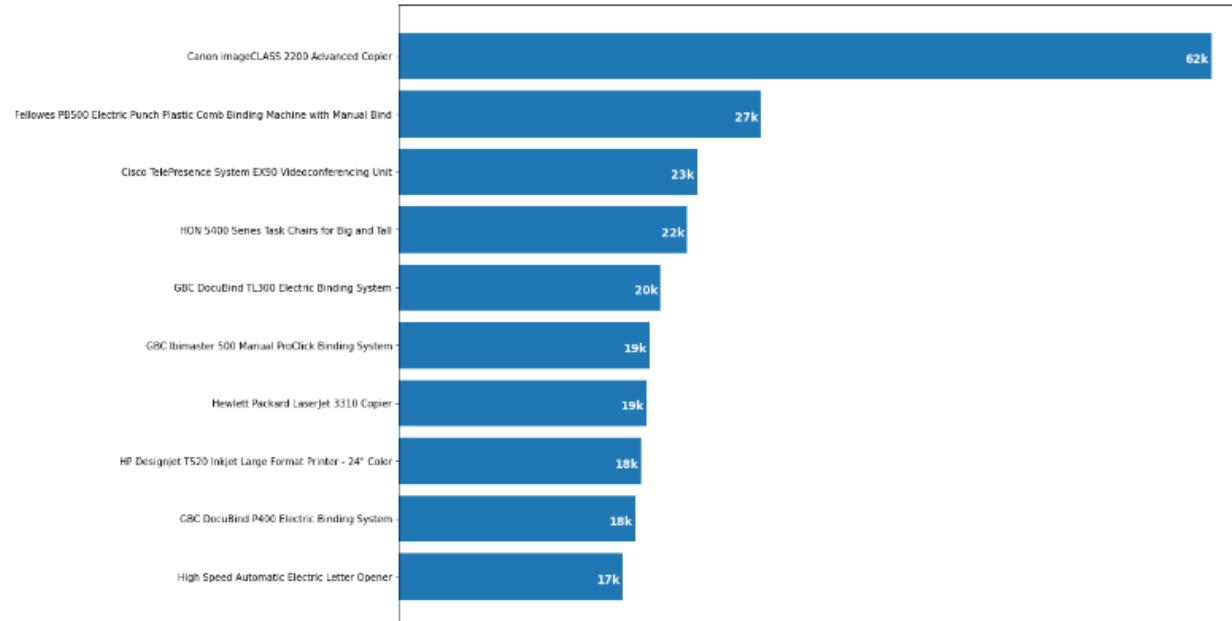
Não pode passar o valor  
texto, por isso utilizamos i



## Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Vamos retirar os valores do eixo X:

```
ax.xaxis.set_visible(False)
```



# Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

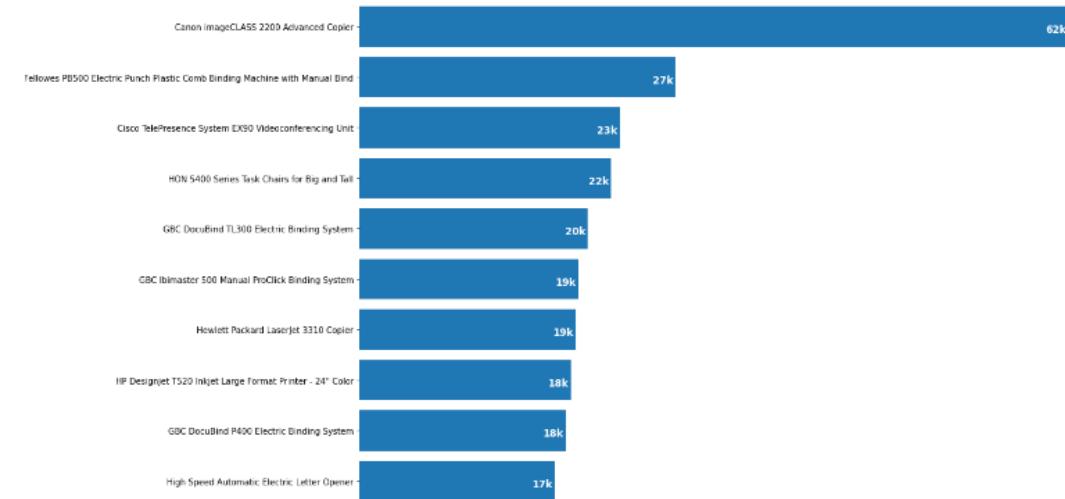
Vamos retirar as bordas:

```
ax.spines['top'].set_visible(False)
```

```
ax.spines['left'].set_visible(False)
```

```
ax.spines['right'].set_visible(False)
```

```
ax.spines['bottom'].set_visible(False)
```



## Módulo 9 – Usando o merge para unir 2 bases no pandas

Na nossa apresentação ainda não conseguimos responder o item mais vendido em um determinado ano.

Então para responder essa pergunta primeiro vamos fazer um **groupby** de item e ano

```
itens_ano = base.groupby(["Product Name","Ano"])["Sales"].sum()
```

```
itens_ano = itens_ano.reset_index()
```

```
itens_ano
```

	Product Name	Ano	Sales
0	"While you Were Out" Message Book, One Form pe...	2018	25.228
1	#10 Gummed Flap White Envelopes, 100/Box	2016	6.608
2	#10 Gummed Flap White Envelopes, 100/Box	2017	24.780
3	#10 Gummed Flap White Envelopes, 100/Box	2018	9.912
4	#10 Self-Seal White Envelopes	2017	88.502
...	...	...	...
5269	invisibleSHIELD by ZAGG Smudge-Free Screen Pro...	2018	205.088
5270	netTALK DUO VoIP Telephone Service	2015	335.936
5271	netTALK DUO VoIP Telephone Service	2016	230.956
5272	netTALK DUO VoIP Telephone Service	2017	377.928
5273	netTALK DUO VoIP Telephone Service	2018	167.968

5274 rows × 3 columns

## Módulo 9 – Usando o merge para unir 2 bases no pandas

Agora vamos supor que queremos visualizar alguns dos itens do TOP N.

`itens_ano[itens_ano["Product Name"] == top_itens[[2,0]]]`



```
itens_ano[itens_ano["Product Name"] == top_n_itens.iloc[2,0]]
```

	Product Name	Ano	Sales
1270	Cisco TelePresence System EX90 Videoconferenci...	2015	22638.48

Foi vendido em  
apenas um ano

## Módulo 9 – Usando o merge para unir 2 bases no pandas

Agora vamos criar um gráfico que mostre como foi a nossa venda anual utilizando o **merge** que irá juntar duas bases. Para isso, devemos passar:

- base 1
- base 2
- **how**: forma que iremos fazer essa junção das bases
  - inner: o que tiver em comum entre as 2 bases (base 1 E base 2)
  - outer: tudo o que tiver nas 2 bases (base 1 OU base 2)
  - left: tudo o que tem na **PRIMEIRA** base, juntando com o que tiver na segunda
  - right: tudo o que tem na **SEGUNDA** base, juntando com o que tiver na primeira
- **on**: colunas que vamos usar para fazer a junção da base

## Módulo 9 – Usando o merge para unir 2 bases no pandas

Para ficar mais claro vamos fazer um exemplo e criar 2 dataframes:

```
dic1 = {  
    "nomes": ['Nome1','Nome2','Nome3'],  
    "valores": [1,2,3]  
}
```

```
base_dic1 = pd.DataFrame(dic1)
```

```
dic2 = {  
    "nomes": ['Nome1','Nome2','Nome4'],  
    "valores": [9,8,7]  
}
```

```
base_dic2 = pd.DataFrame(dic2)
```

## Módulo 9 – Usando o merge para unir 2 bases no pandas

Para ficar mais claro vamos fazer um exemplo e criar 2 dataframes:

```
base_merge = pd.merge(
```

base\_dic1, Primeira base

base\_dic2, Segunda base

how='outer', Tipo de junção que  
vamos fazer

on="nomes" das bases

) coluna que vamos usar para  
fazer essa junção das bases

```
display(base_merge)
```

	nomes	valores_x	valores_y
0	Nome1	1.0	9.0
1	Nome2	2.0	8.0
2	Nome3	3.0	NaN
3	Nome4	NaN	7.0

## Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Agora vamos unir a nossa base de TOP N com uma base de datas utilizando o **merge**.

Como podemos cruzar a base com todos os anos de 2015 a 2018?

Basta transformar as datas em um **DataFrame** e usar o **merge** para fazer essa união.

```
df_datas = pd.DataFrame([2018,2017,2016,2015])
```

```
df_datas.columns = ['Ano']
```

```
df_datas
```

	Ano
0	2018
1	2017
2	2016
3	2015

## Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Só que pra conseguir unir as bases, precisamos ter uma coluna em comum entre elas

Então podemos, nas duas bases, criar uma coluna chamada `uniao` (ou qualquer outro nome) com o mesmo valor.

`top_n_itens['uniao'] = 'unir'`

`df_datas['uniao'] = 'unir'`

Todos os valores vão ser  
'unir' para conseguir juntar  
as duas colunas

Deu um erro porque  
tentamos adicionar uma  
coluna no TOP N

```
top_n_itens['uniao'] = 'unir'  
df_datas['uniao'] = 'unir'
```

```
C:\Users\Pichau\AppData\Local\Temp\ipykernel_16868\2128599823.py:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
top_n_itens['uniao'] = 'unir'
```

## Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Agora que temos uma coluna que conseguimos relacionar, podemos fazer o merge das bases.

```
produto_ano = pd.merge(
```

```
    top_n_itens,
```

Primeira base

```
    df_datas,
```

Segunda base

```
    how='outer'
```

Tipo de junção que  
vamos fazer

```
    on="uniao"
```

)  
coluna que vamos usar para  
fazer essa junção das bases

```
produto_ano.head()
```

	Product Name	Sales	uniao	Ano
0	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2018
1	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2017
2	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2016
3	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2015
4	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.384	unir	2018

## Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Agora que temos uma coluna que conseguimos relacionar, podemos fazer o merge das bases.

Para fazer isso podemos utilizar o **itens\_ano**

display(**itens\_ano**)

	Product Name	Ano	Sales
0	"While you Were Out" Message Book, One Form per page	2018	25.228
1	#10 Gummed Flap White Envelopes, 100/Box	2016	6.608
2	#10 Gummed Flap White Envelopes, 100/Box	2017	24.780
3	#10 Gummed Flap White Envelopes, 100/Box	2018	9.912
4	#10 Self-Seal White Envelopes	2017	86.502
...	...	...	...
5269	invisibleSHIELD by ZAGG Smudge-Free Screen Protector	2018	205.086
5270	netTALK DUO VoIP Telephone Service	2015	335.936
5271	netTALK DUO VoIP Telephone Service	2016	230.956
5272	netTALK DUO VoIP Telephone Service	2017	377.928
5273	netTALK DUO VoIP Telephone Service	2018	167.968

5274 rows × 3 columns

## Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Então, vamos fazer novamente o merge para colocar a coluna de vendas

```
itens_ano_grafico = pd.merge(
```

```
    produto_ano,
```

Primeira base

```
    itens_ano,
```

Segunda base

```
    how='left',
```

Para manter a  
primeira base

```
    on=['Product Name','Ano']
```

```
)
```

colunas que vamos usar para  
fazer essa junção das bases

```
itens_ano_grafico.head()
```

	Product Name	Sales_x	uniao	Ano	Sales_y
0	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2018	35699.898
1	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2017	25899.926
2	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2016	NaN
3	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2015	NaN
4	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.384	unir	2018	7371.742

## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos começar a criar o gráfico baseado n e DataFrame que acabamos de criar:

Os valores de y nós vamos utilizar:

**itens\_ano\_grafico[itens\_ano\_grafico.Ano == 2018]['Product Name'].values**

```
# Vamos criar o gráfico baseado nesse DataFrame que acabamos de criar
itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Product Name'].values

array(['Canon imageCLASS 2200 Advanced Copier',
       'Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind',
       'Cisco TelePresence System EX90 Videoconferencing Unit',
       'HON 5400 Series Task Chairs for Big and Tall',
       'GBC DocuBind TL300 Electric Binding System',
       'GBC Ibimaster 500 Manual ProClick Binding System',
       'Hewlett Packard LaserJet 3310 Copier',
       'HP Designjet T520 Inkjet Large Format Printer - 24" Color',
       'GBC DocuBind P400 Electric Binding System',
       'High Speed Automatic Electric Letter Opener'], dtype=object)
```

## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

E para os valores de y nós não podemos utilizar a coluna Sales\_x porque esses valores são as somas da venda, para o nosso gráfico vamos utilizar o Sales\_y que é a venda ano a ano.

`itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Sales_y'].values`

	Product Name	Sales_x	uniao	Ano	Sales_y
0	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2018	35699.898
1	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2017	25699.926
2	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2018	NaN
3	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2015	NaN
4	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.384	unir	2018	7371.742

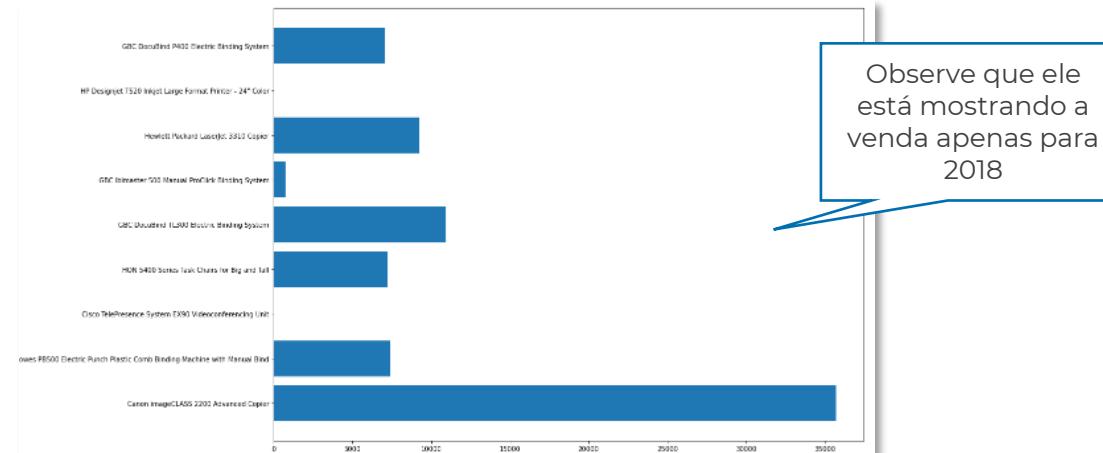
## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Agora colocando essas informações no código do gráfico:

```
fig,ax = plt.subplots(figsize = (16,12))
```

```
ax.barh(itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Product Name'].values,  
       itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Sales_y'].values,  
       align='center'  
)
```

```
plt.show()
```



## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

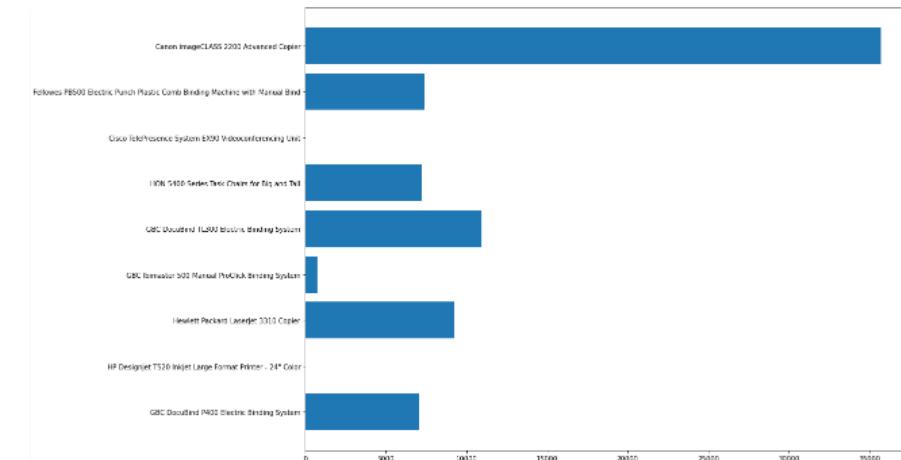
Vamos fazer a inversão do gráfico que fizemos anteriormente:

```
fig,ax = plt.subplots(figsize = (16,12))
```

```
ax.barh(itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Product Name'].values,  
       itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Sales_y'].values,  
       align='center'  
)
```

```
ax.invert_yaxis()
```

```
plt.show()
```



## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos aplicar o mesmo código para 2017:

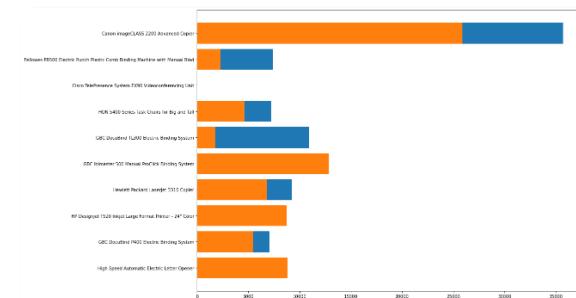
```
fig,ax = plt.subplots(figsize = (16,12))
```

```
ax.barh(itens_ano_grafico[itens_ano_grafico.Ano == 2017]['Product Name'].values,  
       itens_ano_grafico[itens_ano_grafico.Ano == 2017]['Sales_y'].values,  
       align='center')
```

```
ax.barh(itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Product Name'].values,  
       itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Sales_y'].values, align='center')
```

```
ax.invert_yaxis()
```

```
plt.show()
```



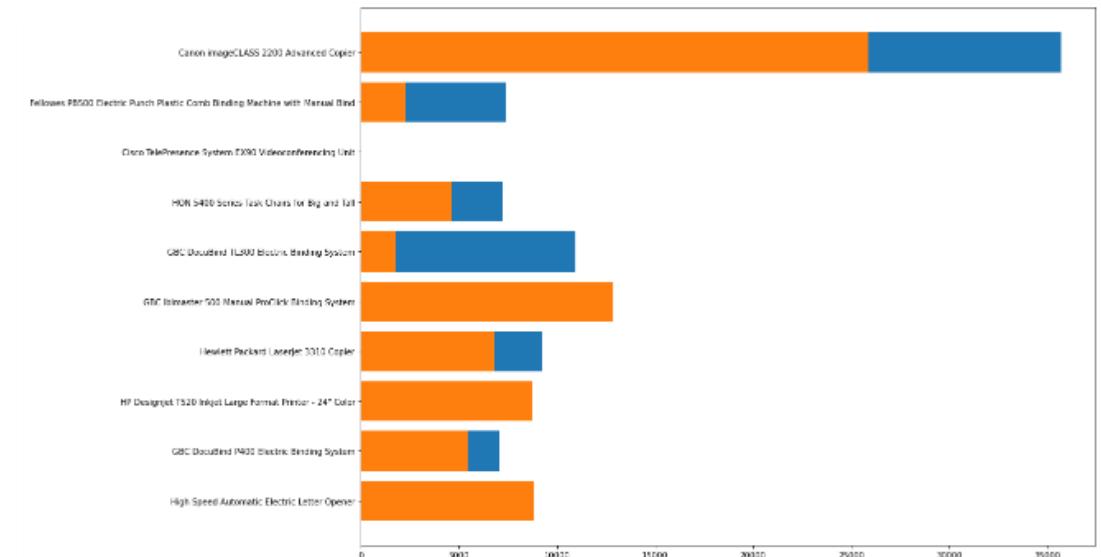
## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Agora vamos deslocar a coluna igual fizemos para o gráfico de barras, só que agora vamos utilizar a variável com o comprimento.

Vamos utilizar:

**hgt** = 0.2

Atenção: como estamos trabalhando com o texto é necessário transformar ele em uma lista de valores.



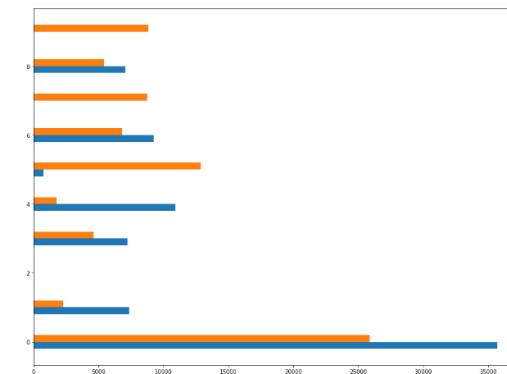
## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Utilizando o comprimento para ajustar:

```
fig,ax = plt.subplots(figsize = (16,12))
```

```
hgt = 0.2
```

```
ax.barh(np.arange(0,qtd_itens+hgt/2,  
itens_ano_grafico[itens_ano_grafico.Ano==2017]['Sales_y'].values, align='center')  
ax.barh(np.arange(0,qtd_itens)-  
hgt/2,itens_ano_grafico[itens_ano_grafico.Ano==2018]['Sales_y'].values,  
align='center')  
  
ax.invert_yaxis()  
plt.show()
```



## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

```
fig,ax = plt.subplots(figsize = (16,12))
```

```
hgt = 0.2
```

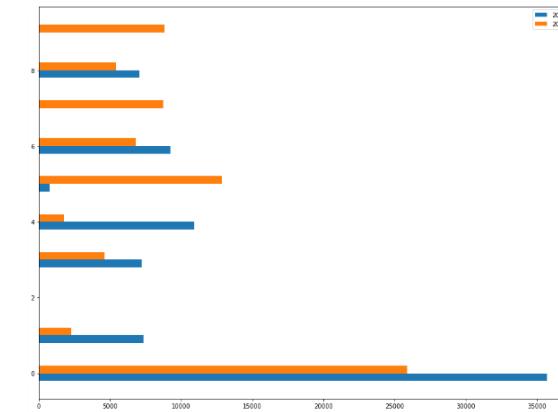
```
ax.barh(np.arange(0,qtd_itens)+hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2017]['Sales_y'].values, align='center', label='2017')
```

```
ax.barh(np.arange(0,qtd_itens)-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Sales_y'].values, align='center', label='2018'))
```

```
ax.invert_yaxis()
```

```
ax.legend()
```

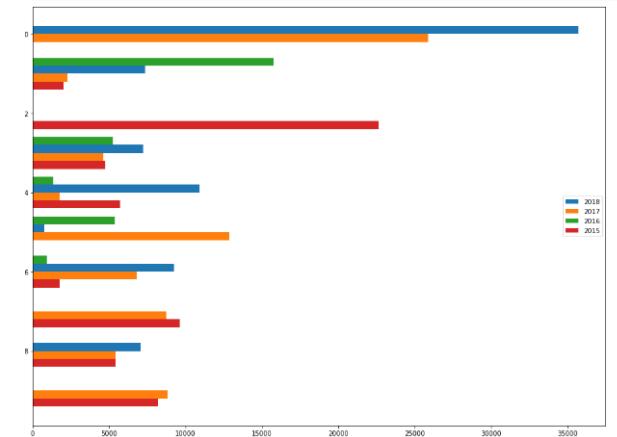
```
plt.show()
```



## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos incluir no código anterior as informações de 2016 e 2015:

```
ax.barh(np.arange(0,qtd_itens)-hgt-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano  
== 2016]['Sales_y'].values, align='center', label='2016')  
  
ax.barh(np.arange(0,qtd_itens)-hgt-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano  
== 2015]['Sales_y'].values, align='center', label='2015'))  
  
ax.invert_yaxis()  
  
ax.legend()  
  
plt.show()
```



## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos padronizar as cores:

```
ax.barh(np.arange(0,qtd_itens)-hgt-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2018].Sales_y.values, align='center', height=hgt, label=2018,color="#0069c0")
```

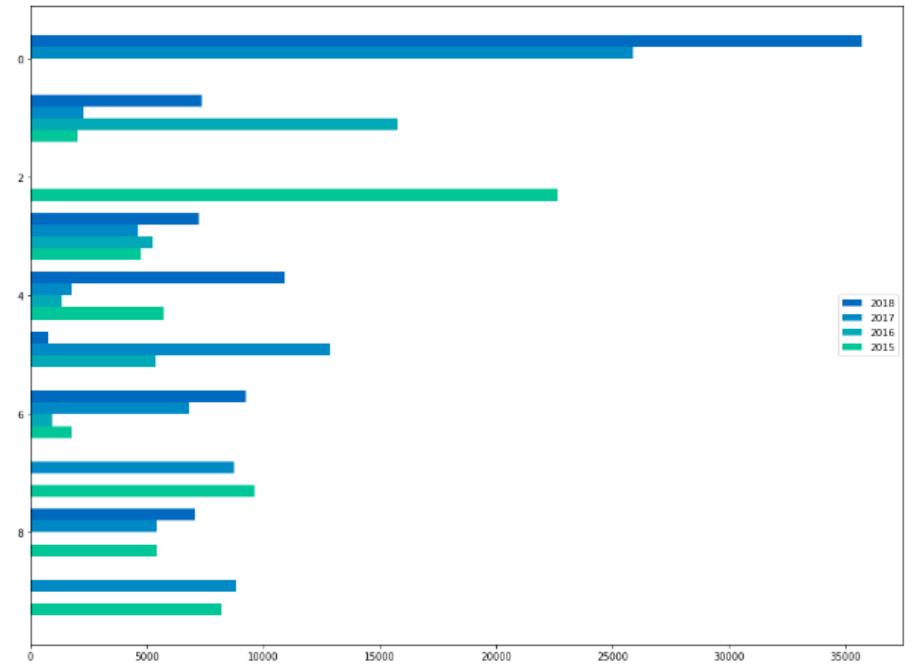
```
ax.barh(np.arange(0,qtd_itens)-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2017].Sales_y.values, align='center',height=hgt,label=2017,color="#008ac5")
```

```
ax.barh(np.arange(0,qtd_itens)+hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2016].Sales_y.values, align='center',height=hgt,label=2016,color="#00a9b5")
```

```
ax.barh(np.arange(0,qtd_itens)+hgt+hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2015].Sales_y.values, align='center',height=hgt,label=2015,color="#00c698")
```

## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

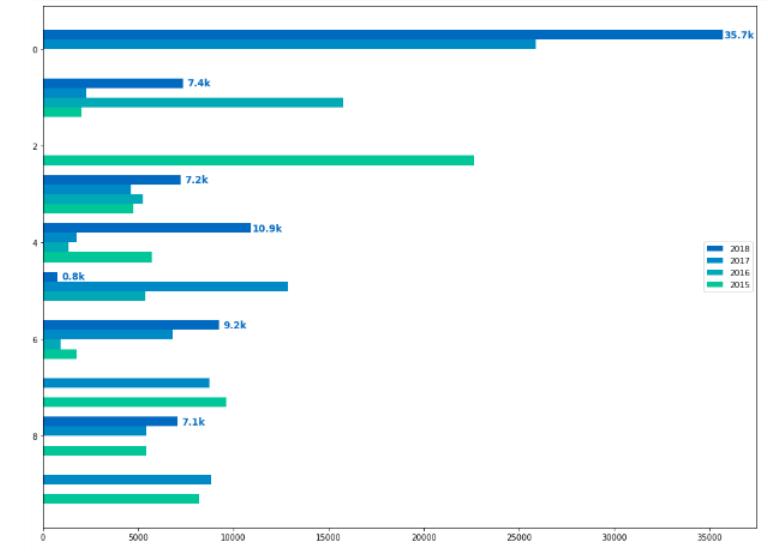
Padronizando as cores temos o gráfico abaixo:



## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos aplicar o `annotate` para 2018 no nosso código:

```
for i in np.arange(0,qtd_itens):  
  
    ax.annotate('{:.1f}k'.format(itens_ano_grafico[itens  
_ano_grafico.Ano==2018].Sales_y.values[i]/1000),  
    (itens_ano_grafico[itens_ano_grafico.Ano==2018].  
Sales_y.values[i],i-hgt-hgt/2),  
    ha="center", va="top", xytext=(20,6),  
    textcoords="offset points", fontsize=12, fontweight='bold',  
    color="#0069c0")
```



## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos com os valores do nosso eixo x no código:

```
ax.yaxis.set_ticks(np.arange(0,qtd_itens))
```

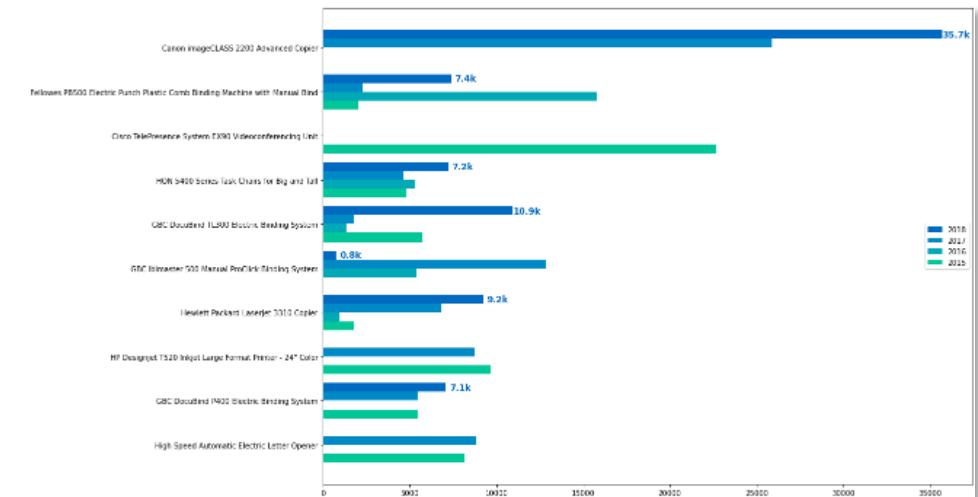
Ajustando os  
valores de x

```
ax.set_yticklabels(top_n_itens["Product Name"])
```

Colocando o rótulo  
no eixo

```
ax.tick_params(axis='y',labelsize=10)
```

Melhorando o  
visual do eixo x



## Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

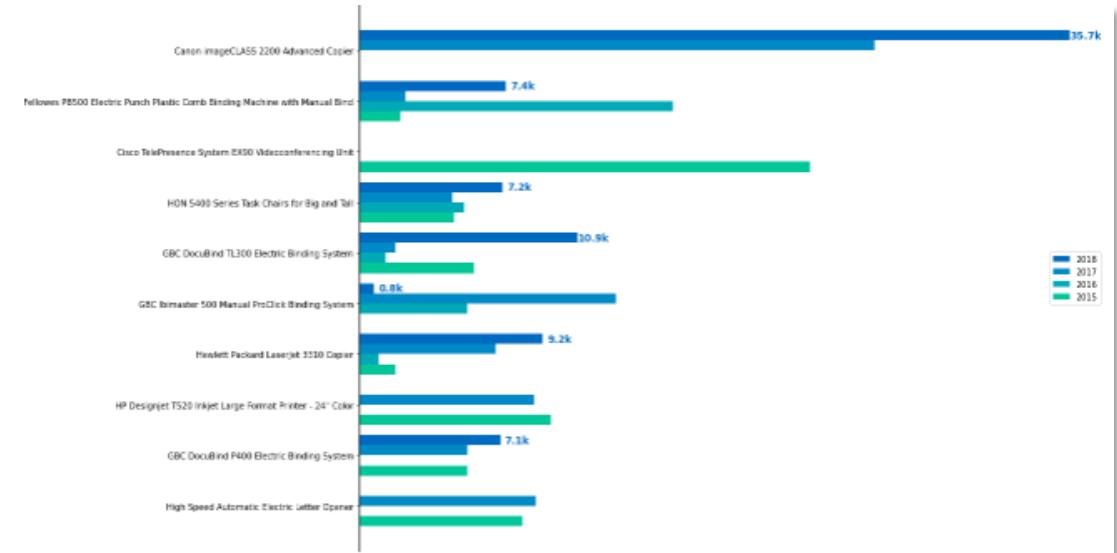
Para finalizar vamos retirar as bordas:

```
ax.xaxis.set_visible(False)
```

```
ax.spines['top'].set_visible(False)
```

```
ax.spines['right'].set_visible(False)
```

```
ax.spines['bottom'].set_visible(False)
```



## Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

Para fechar esse projeto vamos colocar todos os gráficos um embaixo do outro para ficar mais resumido e ficar mais fácil de achar os gráficos.

Então apenas vamos copiar os códigos dos gráficos:

- Vendas por ano
- Vendas por mês
- Vendas por categoria



# Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

- Vendas por ano

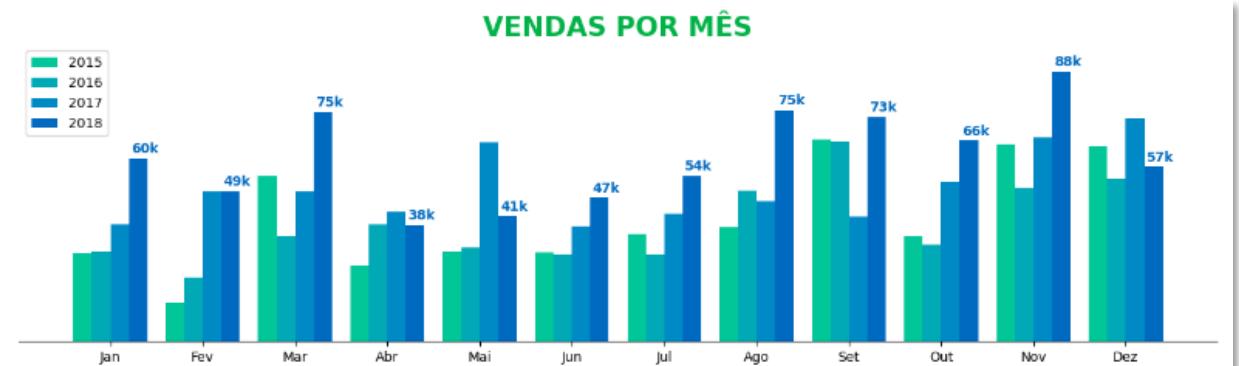
```
fig,ax = plt.subplots(  
    figsize = (9,4)  
)  
  
soma_ano = base.groupby("Ano")["Sales"].sum()  
  
ax.bar(soma_ano.index,soma_ano.values,  
       color="#84ba5b"  
)  
  
ax.set_title("VENDAS POR ANO",  
            fontsize=20,  
            fontweight='bold',  
            color="#00b247"  
)  
  
ax.xaxis.set_ticks([2015,2016,2017,2018])  
ax.tick_params(axis='x',labelsize=14)  
  
for i in np.arange(0,4):  
    ax.annotate('{:.0f}'.format(soma_ano.values[i]).replace(',','.'),  
                (soma_ano.index[i],soma_ano.values[i]),  
                ha="center",  
                va="top",  
                xytext=(0,+15),  
                textcoords="offset points",  
                fontsize=14,  
                fontweight='bold',  
                color="green"  
)  
  
ax.set_yticks(np.array([0,790000]))  
ax.yaxis.set_visible(False)  
  
ax.spines['top'].set_visible(False)  
ax.spines['left'].set_visible(False)  
ax.spines['right'].set_visible(False)  
  
plt.show()
```



# Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

- Vendas por mês

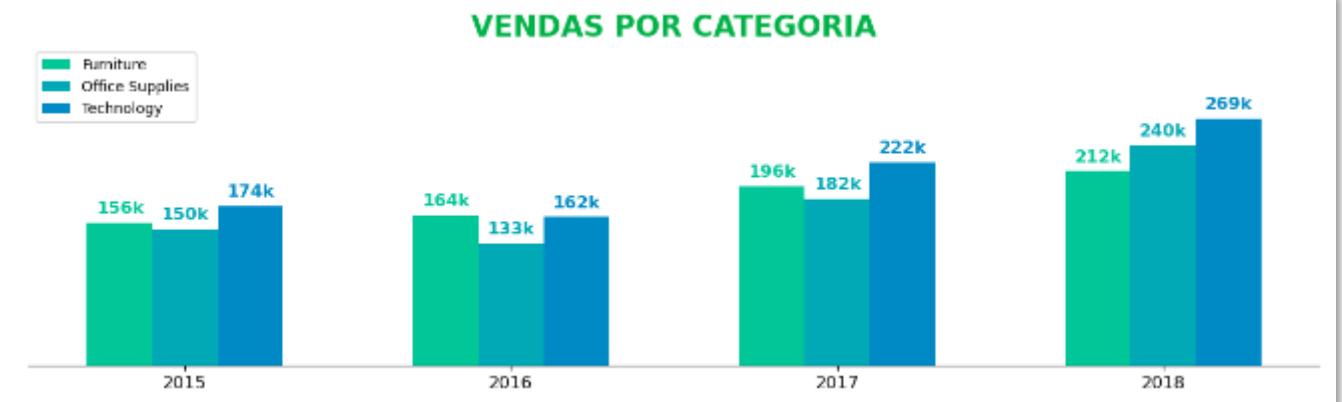
```
: fig,ax = plt.subplots(  
    figsize = (16,4)  
)  
  
ax.set_title("VENDAS POR MÊS",  
            fontsize=20,  
            fontweight='bold',  
            color="#000024")  
  
wid = 0.2 # Tamanho das barras  
  
ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values-wid-wid/2,  
       soma_mes[soma_mes.Ano == 2015].Sales.values,  
       color="#000069", # Cor das barras  
       label=2015, # Label dos dados  
       width=wid # Largura das barras  
       )  
ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values-wid/2,  
       soma_mes[soma_mes.Ano == 2016].Sales.values,  
       color="#008090",  
       label=2016,  
       width=wid  
       )  
  
# Adicionando o restante das barras  
ax.bar(soma_mes[soma_mes.Ano == 2017].Mes.values+wid/2,  
       soma_mes[soma_mes.Ano == 2017].Sales.values,  
       color="#0080ac",  
       label=2017,  
       width=wid  
       )  
ax.bar(soma_mes[soma_mes.Ano == 2018].Mes.values+wid+wid/2,  
       soma_mes[soma_mes.Ano == 2018].Sales.values,  
       color="#000090",  
       label=2018,  
       width=wid  
       )  
  
ax.legend()  
  
# Adicionando o rotulo dos dados para 2018  
for i in np.arange(0,12):  
    ax.annotate('%.0f k' %format(soma_mes[soma_mes.Ano == 2018].Sales.values[i]/1000),  
               (soma_mes[soma_mes.Ano == 2018].Mes.values[i]+wid+wid/2,soma_mes[soma_mes.Ano == 2018].Sales.values[i]),  
               ha='center',  
               va='top',  
               xytext=(5,12),  
               textcoords="offset points",  
               fontsize=10,  
               fontweight='bold',  
               color="#000069")  
  
ax.set_yticks(np.array([0,97000]))  
ax.yaxis.set_visible(False)  
  
ax.spines['top'].set_visible(False)  
ax.spines['left'].set_visible(False)  
ax.spines['right'].set_visible(False)  
  
ax.xaxis.set_ticks(np.arange(1,13))  
ax.set_xticklabels(['Jan','Fev','Mar','Abr','Mai','Jun','Jul','Ago','Set','Out','Nov','Dez'])  
ax.tick_params(axis='x',labelsize=10)  
  
plt.show()
```



# Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

- Vendas por categoria

```
fig,ax = plt.subplots(  
    figsize = (16,4)  
)  
  
ax.set_title("VENDAS POR CATEGORIA",  
            fontsize=20,  
            fontweight="bold",  
            color="#00b247")  
  
wid = 0.2  
  
ax.bar(categoria[categoria.Category == 'Furniture'].Ano.values-wid,  
       categoria[categoria.Category == 'Furniture'].Sales.values,  
       color="#00c698",  
       label="Furniture",  
       width=wid)  
  
ax.bar(categoria[categoria.Category == 'Office Supplies'].Ano.values,  
       categoria[categoria.Category == 'Office Supplies'].Sales.values,  
       color="#0099b5",  
       label="Office Supplies",  
       width=wid)  
  
ax.bar(categoria[categoria.Category == 'Technology'].Ano.values+wid,  
       categoria[categoria.Category == 'Technology'].Sales.values,  
       color="#0088ac",  
       label="Technology",  
       width=wid)  
  
ax.legend()  
  
for i in np.arange(0,4):  
    ax.annotate('{:.0f}k'.format(categoria[categoria.Category == 'Furniture'].Sales.values[i]/1000),  
               (categoria[categoria.Category == 'Furniture'].Ano.values[i]-wid,categoria[categoria.Category == 'Furniture'].Sal  
               ha="center", va="top", xytext=(0,15), textcoords="offset points",  
               fontsize=12, fontweight="bold", color="#00c698")  
    ax.annotate('{:.0f}k'.format(categoria[categoria.Category == 'Office Supplies'].Sales.values[i]/1000),  
               (categoria[categoria.Category == 'Office Supplies'].Ano.values[i]),categoria[categoria.Category == 'Office Suppli  
               ha="center", va="top", xytext=(0,15), textcoords="offset points",  
               fontsize=12, fontweight="bold", color="#0099b5")  
    ax.annotate('{:.0f}k'.format(categoria[categoria.Category == 'Technology'].Sales.values[i]/1000),  
               (categoria[categoria.Category == 'Technology'].Ano.values[i]+wid,categoria[categoria.Category == 'Technology'].S  
               ha="center", va="top", xytext=(0,15), textcoords="offset points",  
               fontsize=12, fontweight="bold", color="#0088ac")  
  
ax.set_yticks(np.array([0,350000]))  
ax.yaxis.set_visible(False)  
  
ax.spines['top'].set_visible(False)  
ax.spines['left'].set_visible(False)  
ax.spines['right'].set_visible(False)  
  
ax.xaxis.set_ticks(np.arange(2015,2019))  
ax.tick_params(axis='x',labelsize=12)  
plt.show()  
4
```



# Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

- TOP itens

```
# Vamos traçar um gráfico de barras horizontais para mostrar o top itens
fig,ax = plt.subplots(
    figsize = (16,12)
)

ax.barh(top_n_itens['Product Name'].values,
        top_n_itens['Sales'].values,
        align='center')

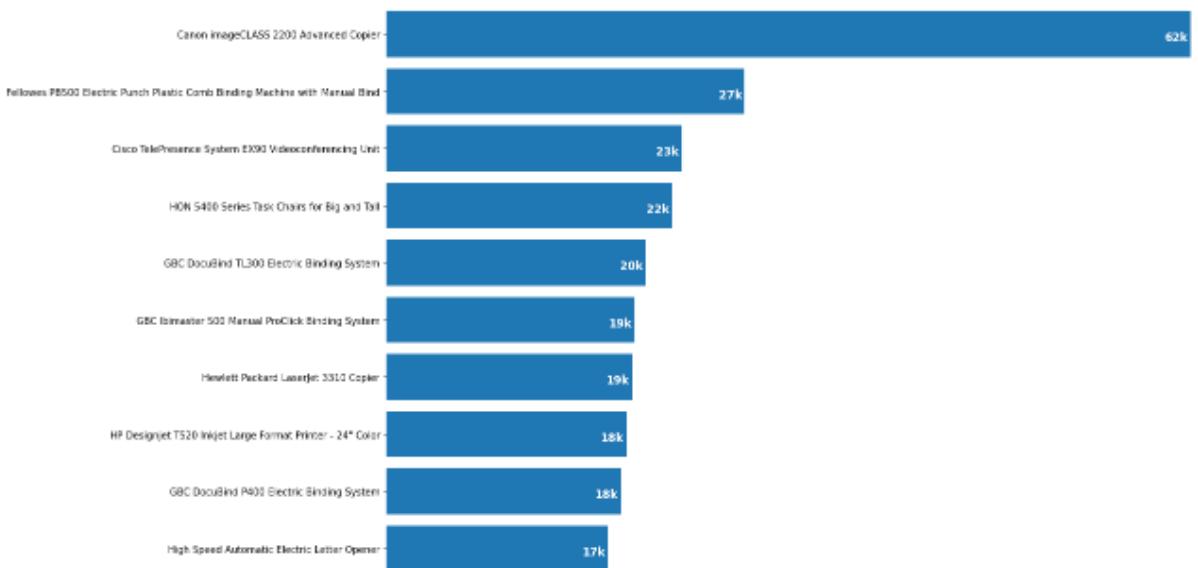
ax.invert_yaxis()

for i in np.arange(0,n):
    ax.annotate('{:.0f}k'.format(top_n_itens['Sales'].values[i]/1000),
                (top_n_itens['Sales'].values[i],i),
                ha="center", va="top", xytext=(-15,3), textcoords="offset points",
                fontsize=12, fontweight="bold", color="white")

ax.xaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)

plt.show()
```



# Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

- Itens x anos

```
# Vamos criar o gráfico baseado nesse DataFrame que acabamos de criar
fig,ax = plt.subplots(
    figsize = (16,12)
)

hgt = 0.2

qtd_itens = len(top_n_itens)

ax.barh(np.arange(0,qtd_itens)-hgt-hgt/2,
       itens_ano_grafico[itens_ano_grafico.Ano == 2018].Sales_y.values,
       align='center',
       height=hgt,
       label=2018,
       color="#0069c0"
)
ax.barh(np.arange(0,qtd_itens)-hgt/2, #Valores de y
       itens_ano_grafico[itens_ano_grafico.Ano == 2017].Sales_y.values,
       align='center',height=hgt,label=2017,color="#008ac5")
ax.barh(np.arange(0,qtd_itens)+hgt/2, #Valores de y
       itens_ano_grafico[itens_ano_grafico.Ano == 2016].Sales_y.values,
       align='center',height=hgt,label=2016,color="#00a9b5")
ax.barh(np.arange(0,qtd_itens)+hgt+hgt/2, #Valores de y
       itens_ano_grafico[itens_ano_grafico.Ano == 2015].Sales_y.values,
       align='center',height=hgt,label=2015,color="#00c698")

ax.invert_yaxis()
ax.legend()

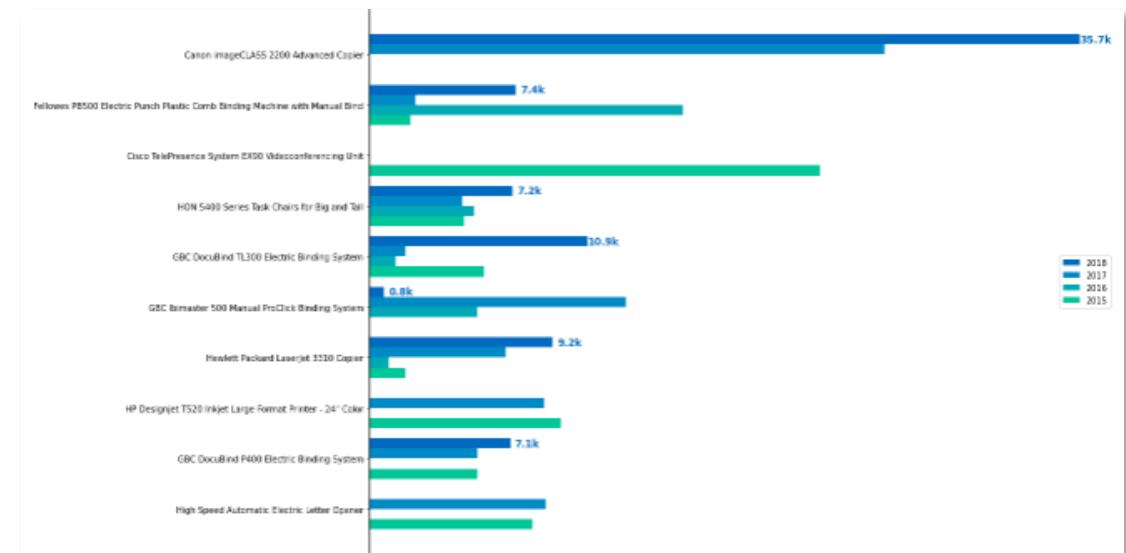
for i in np.arange(0,qtd_itens):
    ax.annotate('{:,1f}'.format(itens_ano_grafico[itens_ano_grafico.Ano == 2018].Sales_y.values[i]/1000),
               (itens_ano_grafico[itens_ano_grafico.Ano == 2018].Sales_y.values[i],i-hgt-hgt/2),
               ha="center", va="top", xytext=(20,6), textcoords="offset points",
               fontsize=12, fontweight='bold', color="#0069c0")

ax.yaxis.set_ticks(np.arange(0,qtd_itens))
ax.set_yticklabels(top_n_itens['Product Name'])
ax.tick_params(axis='y',labelsize=10)

ax.xaxis.set_visible(False)

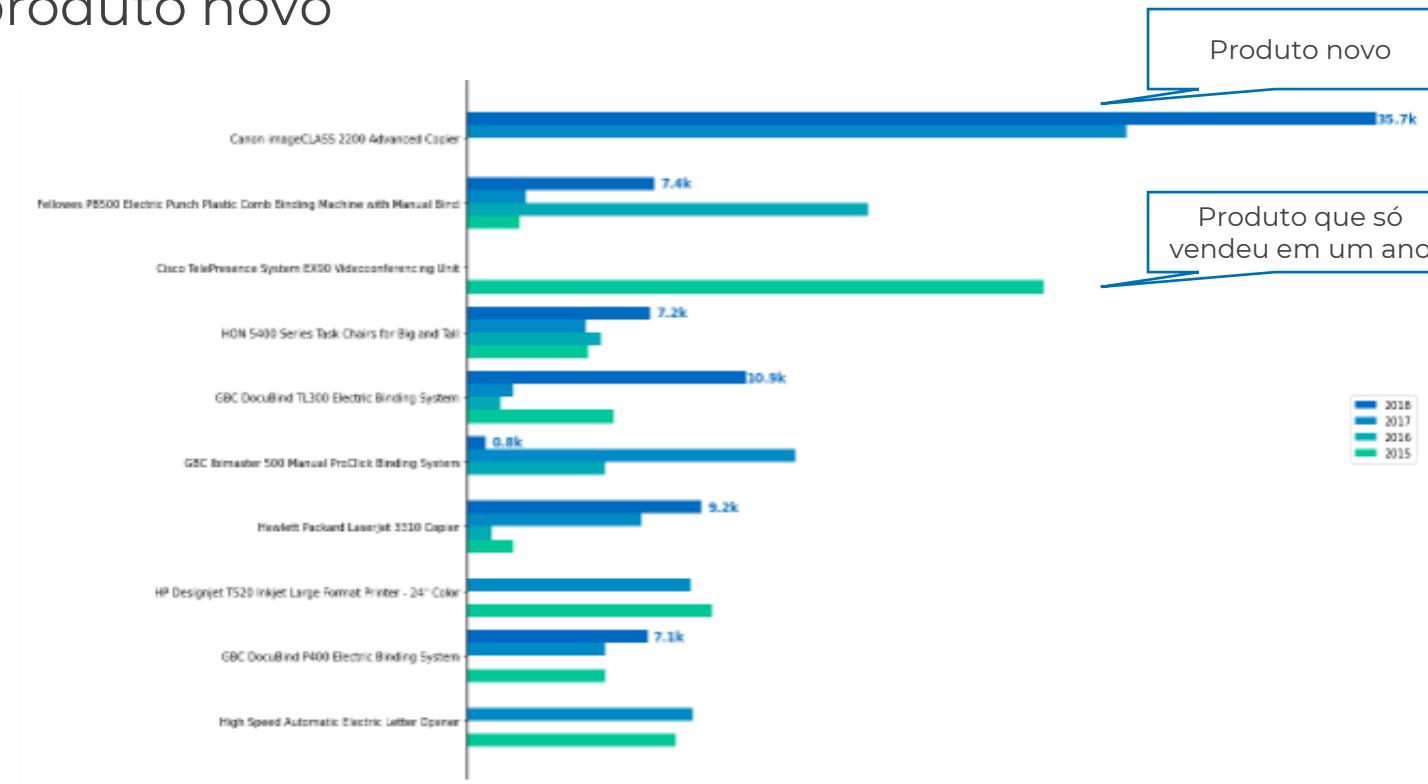
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)

plt.show()
```



## Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

- Podemos verificar que um produto só vendeu em um ano, também é possível observar um produto novo



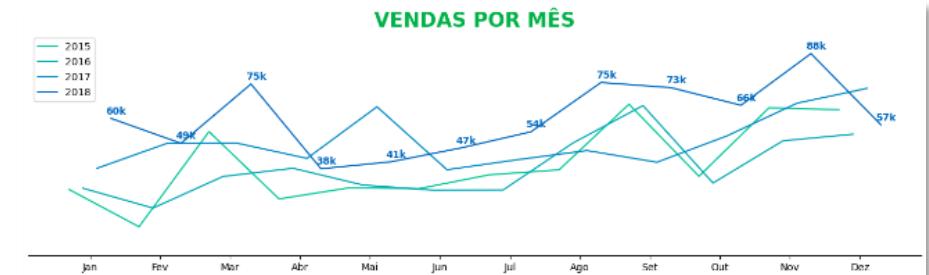
## Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

Para trocar um gráfico de barra para um linha, basta substituir o termo **bar** do código por **plot**.

Vamos **excluir a largura do gráfico**, já que o gráfico de linha não possui largura.

Aplicando essas mudanças em uma parte do nosso código:

```
ax.plot(soma_mes[soma_mes.Ano == 2018].Mes.values,  
        soma_mes[soma_mes.Ano == 2018].Sales.values,  
        color="#0069c0",  
        label=2018)
```



## Módulo 9 – Apresentando as informações em um PowerPoint

Nessa aula vamos aprender como colocar a imagem no PowerPoint.

Vamos utilizar o modelo de PowerPoint da Hashtag que está na descrição do vídeo.

Para salvar um gráfico como imagem temos que colocar no código a função:

```
plt.savefig('Título')
```

O arquivo ficará salvo na pasta em que o seu arquivo do Jupyter está armazenado.



## Módulo 9 – Apresentando as informações em um PowerPoint

Depois de salvar a imagem e importá-la para o PowerPoint o fundo branco do gráfico não faz sentido, já que o nosso fundo é cinza.

Para resolver esse problema vamos alterar o código para:

```
plt.savefig('Vendas por ano', transparent = True)
```

O arquivo antigo será substituído pelo atual após rodar o comando.



# Módulo 9 – Apresentando as informações em um PowerPoint

Repare como melhorou a visualização da imagem



## Módulo 9 – Apresentando as informações em um PowerPoint

Repare como melhorou a visualização da imagem, vamos fazer esse mesmo processo para todas as imagens.



## Módulo 9 – Apresentando as informações em um PowerPoint

Quando vamos incluir o gráfico de top itens na apresentação ele vai cortar, mas para resolver isso basta inserir no código o comando **plt.tight\_layout** ele vai fazer com que o gráfico se ajuste, por exemplo :

**plt.tight\_layout()**

**plt.savefig('TOP ITENS')**



## Módulo 9 – Corrigindo o erro na transformação da data

Durante o nosso projeto quando falamos de transformação de datas, o Python considerou em algumas datas o mês e o ano de forma correta, enquanto em outras ele considerou de forma invertida.

**Repare que na nossa base o formato da data é DD/MM/AAAA**

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture

## Módulo 9 – Corrigindo o erro na transformação da data

Vamos fazer a transformação das datas criando uma nova coluna.

`base["Order Date 2"] = pd.to_datetime(base["Order Date"])`

`base[["Order Date", "Order Date 2"]]`

Aqui o Python entende como:  
mês/dia/ano

```
base["Order Date 2"] = pd.to_datetime(base["Order Date"])
base[["Order Date", "Order Date 2"]]
```

	Order Date	Order Date 2
0	2017-08-11	2017-08-11
1	2017-08-11	2017-08-11
2	2017-12-06	2017-12-06
3	2016-11-10	2016-11-10
4	2016-11-10	2016-11-10
...	...	...
5795	2017-05-21	2017-05-21
5796	2016-12-01	2016-12-01
5797	2016-12-01	2016-12-01
5798	2016-12-01	2016-12-01
5799	2016-12-01	2016-12-01

9800 rows × 2 columns

Aqui o Python entende como:  
dia/ mês/ano

## Módulo 9 – Corrigindo o erro na transformação da data

Para confirmar esse erro podemos criar uma coluna com dia e outra com o mês.

**base['Dia']=base['Order Date 2'].dt.day**

**base['Mes']=base['Order Date 2'].dt.month**

**base[["Order Date","Order Date 2","Dia","Mes"]]**

```
base['Dia']=base['Order Date 2'].dt.day  
base['Mes']=base['Order Date 2'].dt.month  
base[["Order Date","Order Date 2","Dia","Mes"]]
```

	Order Date	Order Date 2	Dia	Mes
0	2017-08-11	2017-08-11	11	8
1	2017-08-11	2017-08-11	11	8
2	2017-12-06	2017-12-06	6	12
3	2016-11-10	2016-11-10	10	11
4	2016-11-10	2016-11-10	10	11
...	...	...	...	...
9795	2017-05-21	2017-05-21	21	5
9796	2016-12-01	2016-12-01	1	12
9797	2016-12-01	2016-12-01	1	12
9798	2016-12-01	2016-12-01	1	12
9799	2016-12-01	2016-12-01	1	12

9800 rows × 4 columns

O 8 foi considerado como mês

O 21 foi considerado como dia

## Módulo 9 – Corrigindo o erro na transformação da data

Para resolver esse problema podemos utilizar o **format** e dizer o formato da nossa base.

**base["Order Date 2"] = pd.to\_datetime(base["Order Date"],format ='\$d/\$m/\$Y')**

**base['Dia']=base['Order Date 2'].dt.day**

**base['Mes']=base['Order Date 2'].dt.month**

**base[["Order Date","Order Date 2","Dia","Mes"]]**

```
base["Order Date 2"] = pd.to_datetime(base["Order Date"],format="$d/$m/$Y")
base['Dia']=base['Order Date 2'].dt.day
base['Mes']=base['Order Date 2'].dt.month
base[["Order Date","Order Date 2","Dia","Mes"]]
```

	Order Date	Order Date 2	Dia	Mes
0	2017-08-11	2017-08-11	11	8
1	2017-08-11	2017-08-11	11	8
2	2017-12-06	2017-12-06	6	12
3	2016-11-10	2016-11-10	10	11
4	2016-11-10	2016-11-10	10	11
...	...	...	...	...
5795	2017-05-21	2017-05-21	21	5
5796	2016-12-01	2016-12-01	1	12
5797	2016-12-01	2016-12-01	1	12
5798	2016-12-01	2016-12-01	1	12
5799	2016-12-01	2016-12-01	1	12

9800 rows × 4 columns

O 11 foi considerado  
como dia

O 21 foi considerado  
como dia

## MÓDULO 10

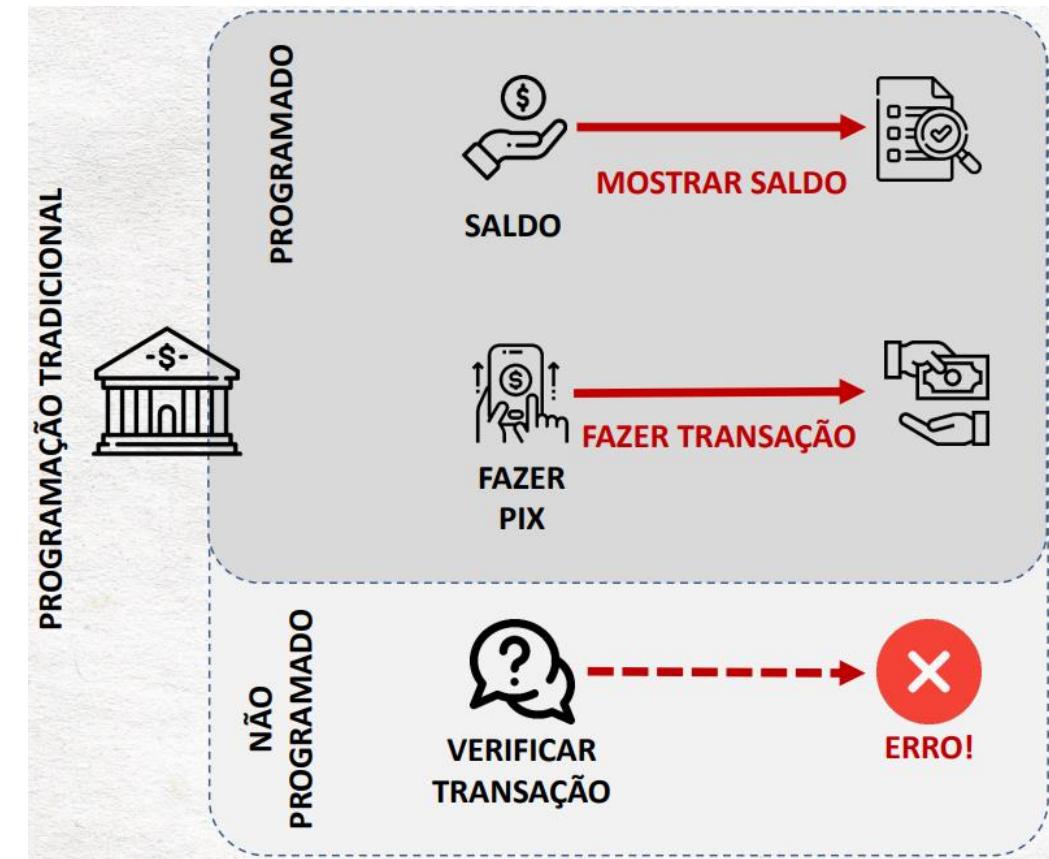
# Introdução ao aprendizado de máquinas

## Módulo 10 – O que é aprendizado de máquinas (Machine Learning)?

É o campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados.

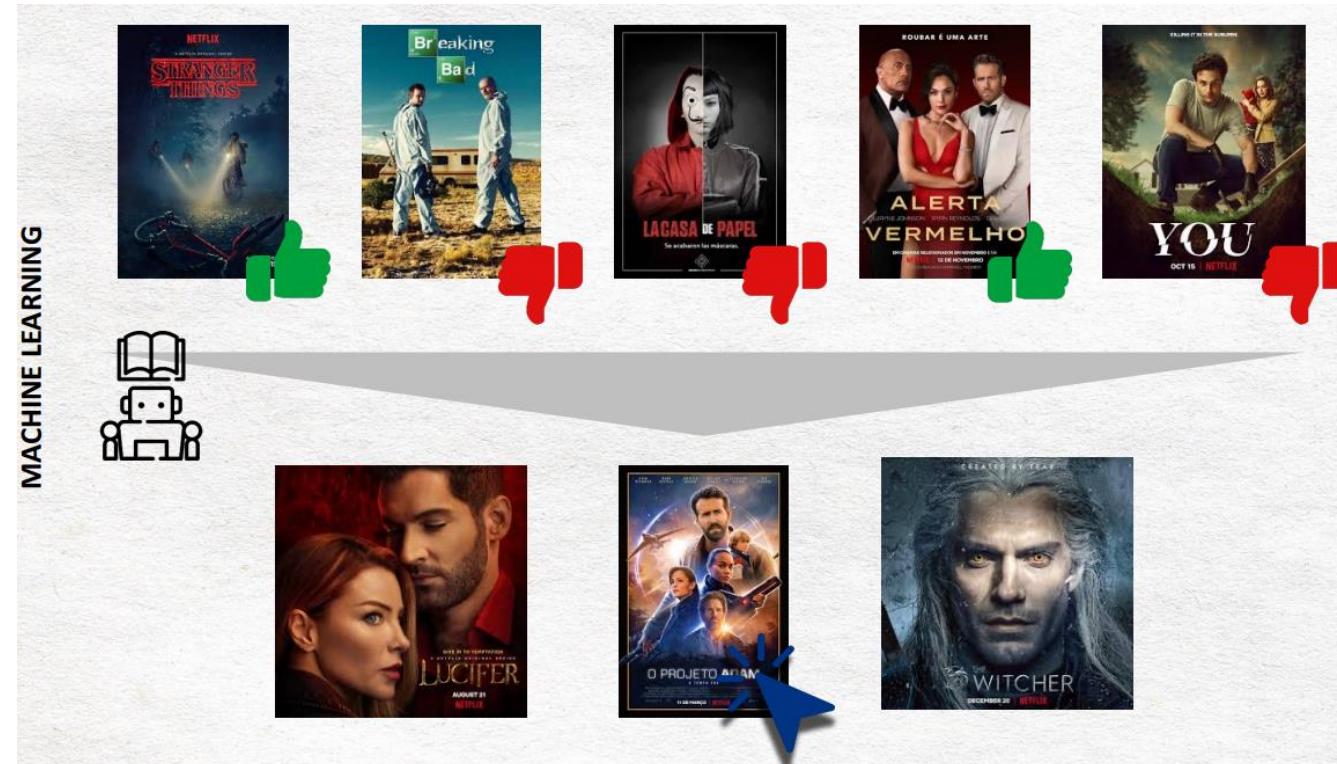
Na programação tradicional, o programa fará apenas aquilo para o qual foi estritamente programado.

Para uma dada entrada, é possível saber exatamente qual será a saída.



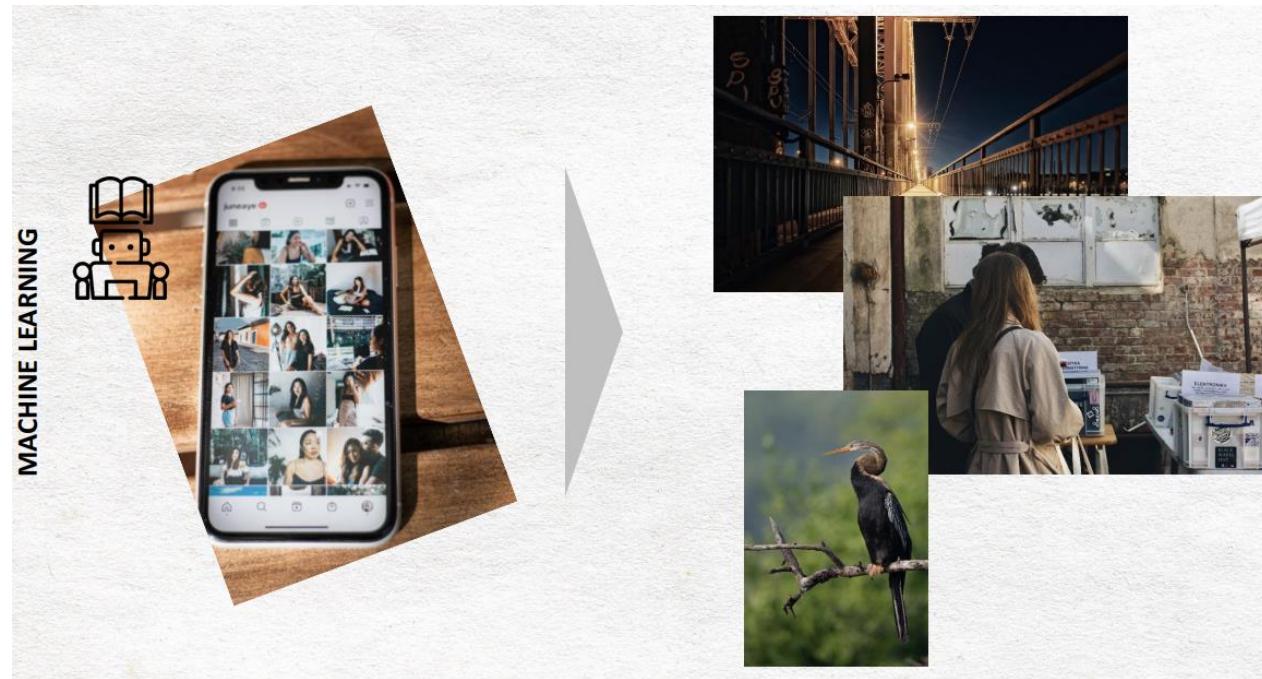
## Módulo 10 – O que é aprendizado de máquinas (Machine Learning)?

Com Machine Learning, o programa vai aprendendo a cada interação. Por exemplo, as recomendações de filmes na Netflix dependem de como você interage e avalia outros filmes.



## Módulo 10 – O que é aprendizado de máquinas (Machine Learning)?

Da mesma forma, o feed do Instagram de cada pessoa depende da forma que ela usa o aplicativo. Se ela curte mais fotos ou vídeos; se interage mais com amigos ou com perfis diversos; se costuma comentar, dentre diversos outros critérios.

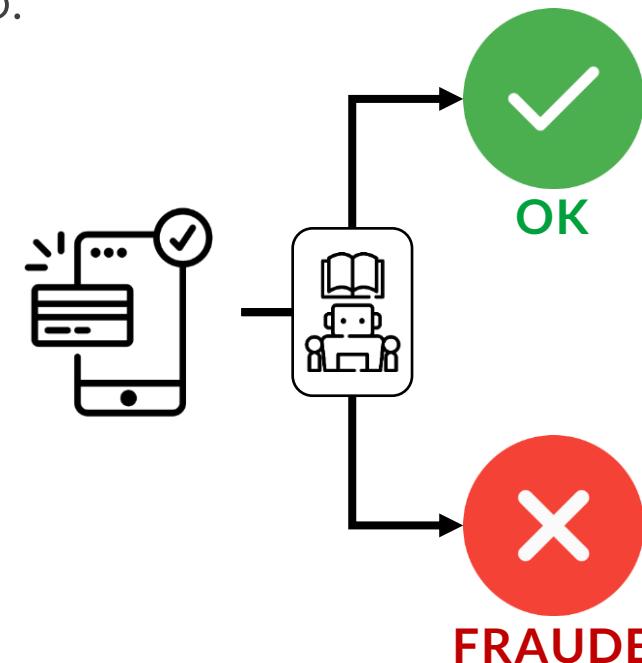


## Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

Primeiro, precisamos lembrar que, para ter um projeto de ciência de dados, precisamos de... **DADOS!**

Com os dados, conseguimos encontrar padrões que antes seriam imperceptíveis, e usar esses padrões para gerar diferenciais de negócio.

Para entender, vamos pegar um exemplo clássico de aprendizado de máquinas: a identificação de **fraudes** no setor bancário.



# Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

Como precisamos de dados, vamos fazer um levantamento do que é possível obter da base de dados de um sistema bancário:



**INFORMAÇÕES  
HISTÓRICAS**



- Valor médio das transações anteriores
- Valor máximo entre os últimos pagamentos
- Perfil de compra
- Lojas que o cliente costuma comprar
- ...



- Já ocorreu algum problema nessa mesma loja?
- A loja vende produtos confiáveis?
- Outros clientes tiveram problema no estabelecimento?
- ...



- Hora das transações anteriores
- Preferência por transações presenciais ou online
- Histórico de parcelamento
- Frequência de compras
- ...



- Lojas que são facilmente fraudadas
- Perfil dos clientes que já sofreram fraudes
- Regiões / sites mais propensos a fraudes
- ...

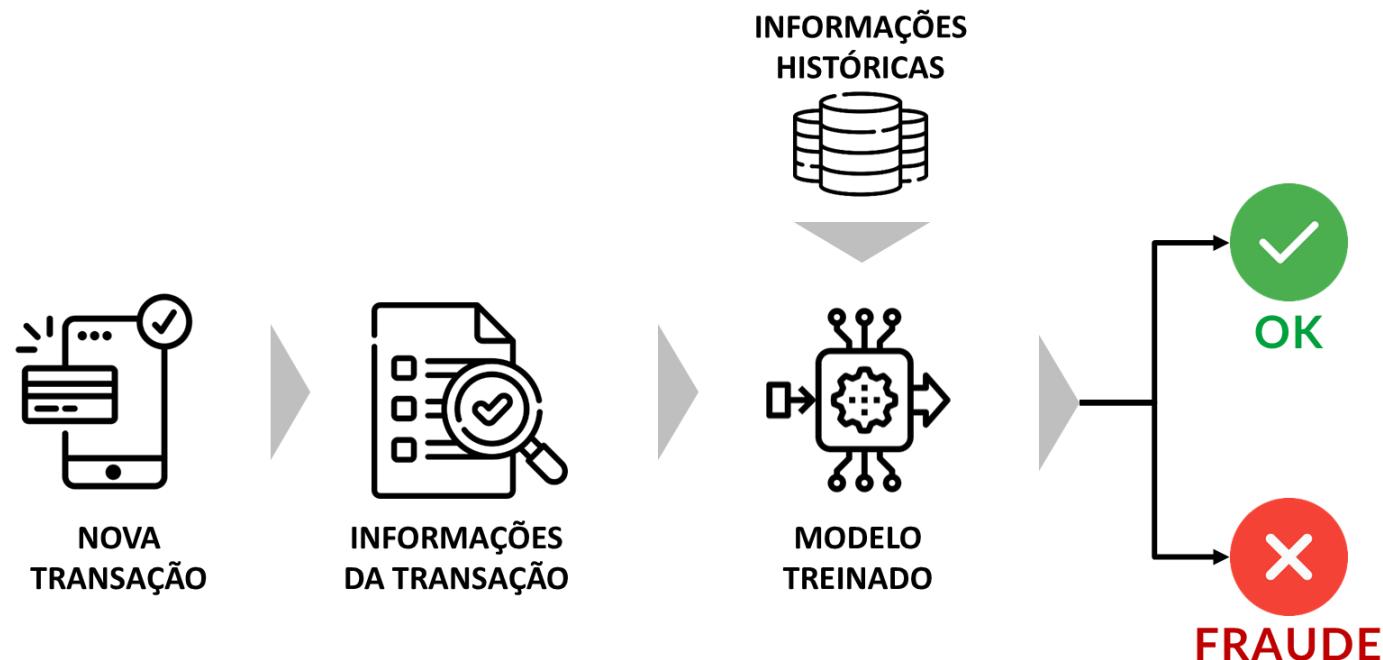
# Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

Com os dados históricos, um modelo de aprendizado de máquinas pode detectar padrões e identificar quais dos dados são mais relevantes, ranqueando-os por relevância. Na imagem do nosso exemplo abaixo, uma cor verde mais forte indica maior relevância. Um risco indica dados que o modelo considera irrelevante.



## Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

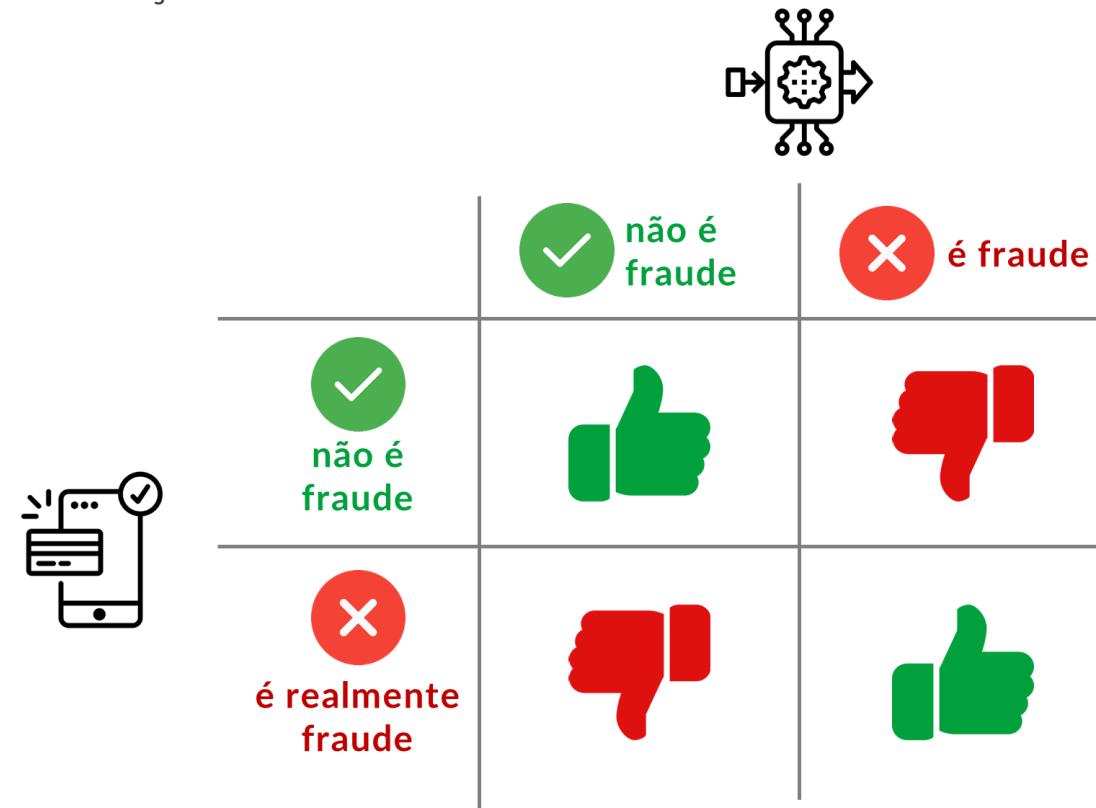
Agora que nosso modelo já foi treinado com dados históricos da base do banco, ele pode analisar novas transações e classificá-las como fraudulentas ou não:



Esse modelo não vai ser 100% preciso, queremos que ele seja **o mais preciso possível!**

## Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

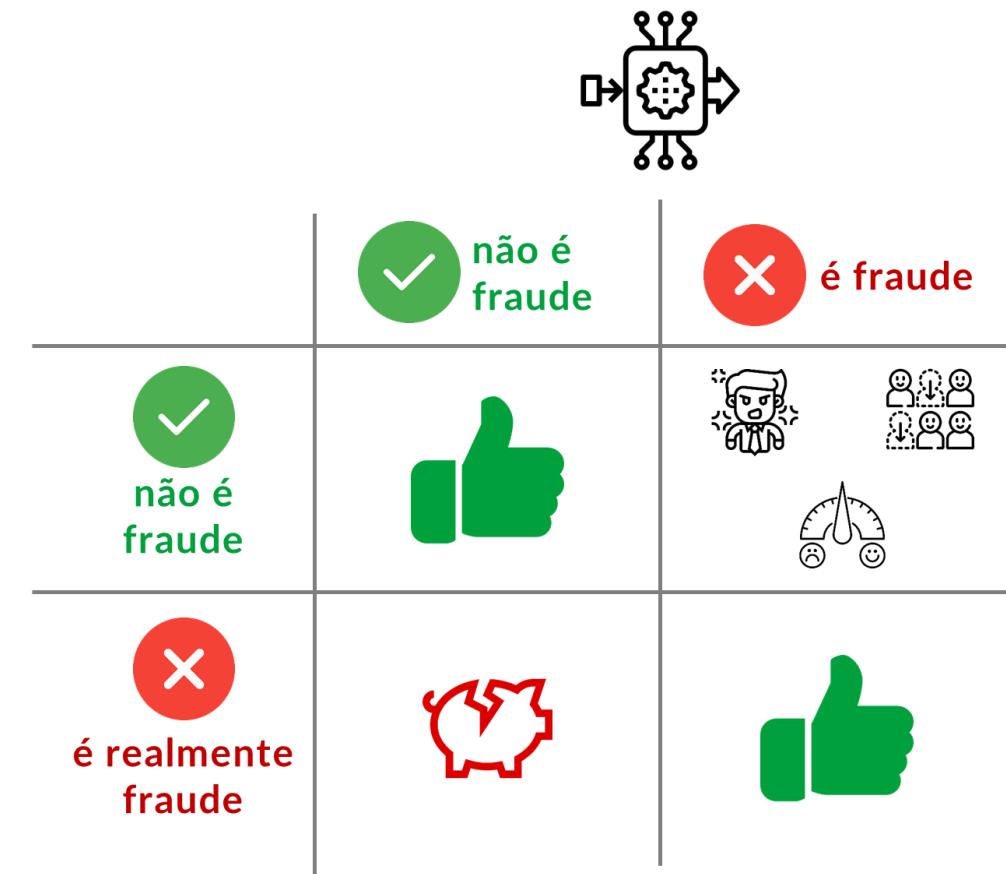
Na vida real, é muito difícil que um modelo acerte todas as previsões. Assim, teremos os chamados falsos positivos (transações não fraudulentas erroneamente classificadas como fraude) e os falsos negativos (transações fraudulentas erroneamente classificadas como legítimas):



## Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

É importante entender a consequência desses erros:

- Ao classificar erroneamente uma transação fraudulenta como legítima, um usuário teve seu dinheiro roubado via, por exemplo, clonagem do cartão. A companhia do cartão terá que reembolsar o usuário.
- Por outro lado, uma transação legítima ser considerada fraudulenta significa que o usuário terá seu cartão bloqueado indevidamente, causando conflito e desconforto no cliente.



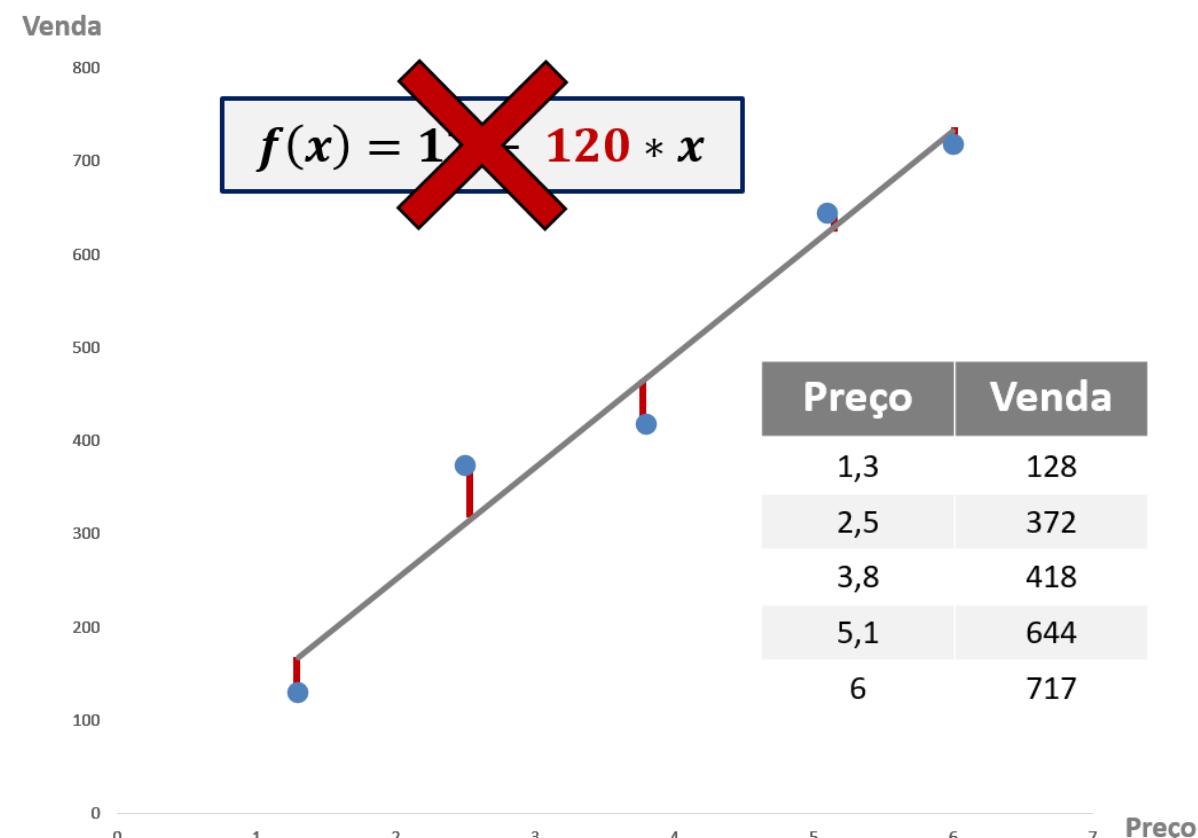
A empresa precisa definir os parâmetros do modelo de forma a tentar minimizar os custos decorrentes de ambos os erros.

## Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

Tudo que vimos conceitualmente para um problema de classificação como o de fraudes também vale para um problema de regressão.

Considere um exemplo simples, onde se busca entender se a relação entre preço e venda é linear, como na figura.

Muito dificilmente se encontrará uma reta que passa por todos os pontos. O que se busca é uma reta que represente bem a tendência. Observe na figura que se marcou em vermelho a distância de cada ponto à reta. Assim, pode-se pensar que uma boa reta seria aquela que minimiza esta distância, que pode ser entendida como o erro da regressão.



## Módulo 10 – O aprendizado de máquinas no Python

Agora que já temos um bom embasamento teórico, vamos aplicar.

Comecemos com um exemplo simples, de dados que queremos verificar se possuem uma tendência linear.

Vamos começar importando as bibliotecas que precisaremos para criar uma base com esses dados e gráficos: NumPy, Pandas e Matplotlib.

```
In [1]: import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np
```

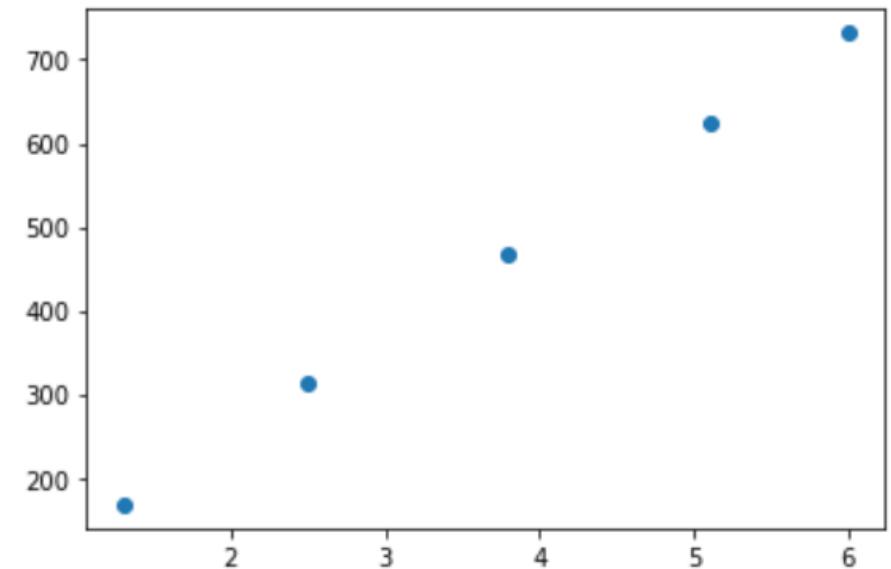
```
In [2]: dados = {  
    'X': [1.3, 2.5, 3.8, 5.1, 6],  
    'Y': [168, 312, 468, 624, 732]  
}  
  
dados = pd.DataFrame(dados)
```

## Módulo 10 – O aprendizado de máquinas no Python

Prosseguimos fazendo um gráfico de pontos com estes dados.

Veja que, por simples observação, já percebemos um comportamento linear.

```
In [3]: fig,ax = plt.subplots()  
ax.scatter(dados.X,dados.Y)  
plt.show()
```



## Módulo 10 – O aprendizado de máquinas no Python

Podemos pegar as coordenadas dos pontos inicial (`dados.X[0]`, `dados.Y[0]`) e final (`dados.X[4]`, `dados.Y[4]`) e plotar uma reta.

A equação dessa reta é:

$$f(x) = 12 + 120x$$

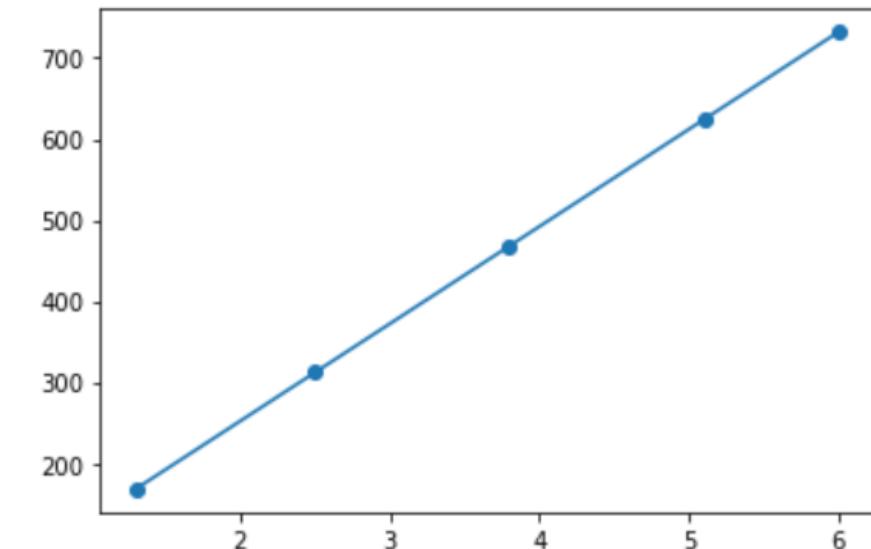
Se quisermos estimar o valor para, por exemplo,  $x= 4,5$ , basta substituir o valor na equação:

$$f(4,5) = 12 + 120 \cdot (4,5) = 552$$

In [4]: `fig,ax = plt.subplots()`

```
ax.scatter(dados.X,dados.Y)
x = [dados.X[0],dados.X[4]]
y = [dados.Y[0],dados.Y[4]]
ax.plot(x,y)
```

`plt.show()`



## Módulo 10 – O aprendizado de máquinas no Python

Agora, não faz muito sentido fazermos a equação e a substituição manualmente se temos o Python.

Podemos importar do Scikit-Learn o modelo linear LinearRegression.

Passamos, então, para o método fit, o valores de X e Y:

```
In [5]: from sklearn.linear_model import LinearRegression  
modelo = LinearRegression()  
modelo.fit(dados.X.values.reshape(-1, 1), dados.Y)  
  
Out[5]: LinearRegression()
```

## Módulo 10 – O aprendizado de máquinas no Python

Agora, o modelo pode nos retornar os coeficientes linear e angular:

**O coeficiente linear desse modelo é dado por:**

```
In [6]: # Coeficiente linear  
modelo.intercept_
```

Out[6]: 12.0

**E o coeficiente angular:**

```
In [7]: # Coeficiente angular  
modelo.coef_
```

Out[7]: array([120.])

## Módulo 10 – O aprendizado de máquinas no Python

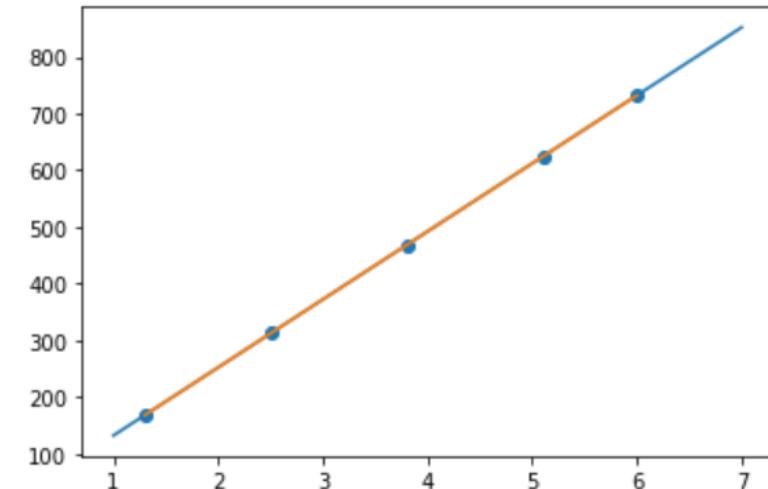
Lembre-se que a equação básica de uma reta é  $y = ax + b$ , sendo  $a$  o coeficiente angular, e  $b$  o coeficiente linear. Assim, podemos criar um conjunto de valores  $x$  com o método `arange` do NumPy e seus respectivos valores de  $y$ .

Com isso, podemos pegar nossos pontos originais e plotar a reta do nosso modelo.

Na figura, em laranja está a reta que havíamos criado anteriormente com os pontos inicial e final; e, em azul, a reta do nosso modelo.

```
In [9]: x = np.arange(1,8)  
y = modelo.intercept_ + x*modelo.coef_[0]
```

```
In [10]: fig,ax = plt.subplots()  
  
ax.scatter(dados.X,dados.Y)  
ax.plot(x,y)  
x2 = [dados.X[0],dados.X[4]]  
y2 = [dados.Y[0],dados.Y[4]]  
ax.plot(x2,y2)  
  
plt.show()
```



## Módulo 10 – O aprendizado de máquinas no Python

Por fim, com o método `predict`, podemos estimar o valor de  $y$  para um dado valor de  $x$ :

```
In [11]: # Prevendo algum valor usando o modelo que acabamos de criar  
modelo.predict(pd.DataFrame([4.5]))  
  
Out[11]: array([552.])
```

Dessa forma, vimos como reproduzir com Python tudo que fizemos anteriormente de forma manual.

## Módulo 10 – O aprendizado de máquinas no Python

Vimos na parte teórica que, na vida real, usualmente não temos pontos tão perfeitamente lineares. Vamos gerar um conjunto de dados que seja mais próximo do que efetivamente encontramos na realidade:

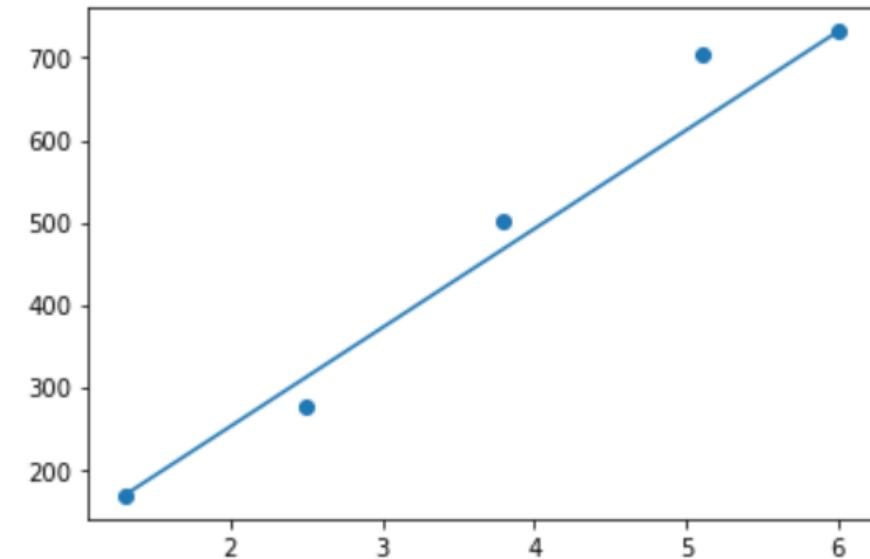
```
In [12]: dados2 = {  
    'X': [1.3,2.5,3.8,5.1,6],  
    'Y': [168,275,501,705,732]  
}  
  
dados2 = pd.DataFrame(dados2)
```

## Módulo 10 – O aprendizado de máquinas no Python

Veja o perfil dos pontos graficamente. A reta azul liga o ponto inicial ao ponto final.

Mas será essa a melhor reta que descreve estes pontos?

```
In [13]: fig,ax = plt.subplots()  
ax.scatter(dados2.X,dados2.Y)  
x = [dados.X[0],dados.X[4]]  
y = [dados.Y[0],dados.Y[4]]  
ax.plot(x,y)  
plt.show()
```



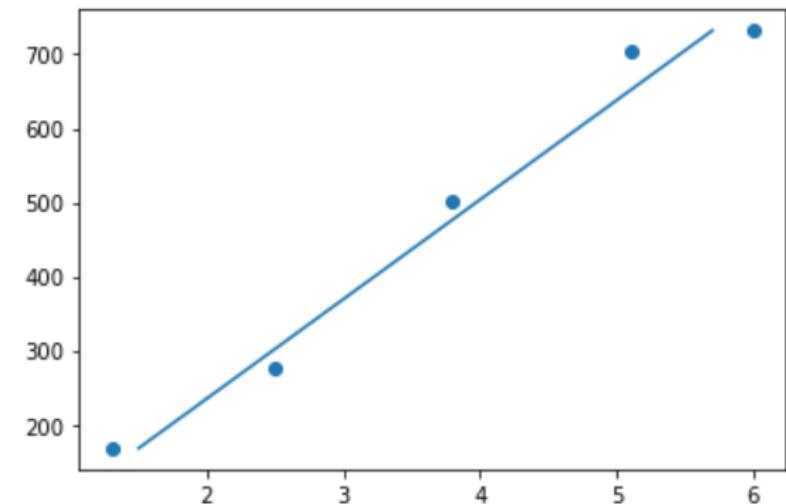
## Módulo 10 – O aprendizado de máquinas no Python

Podemos, por exemplo, modificar levemente essa reta deslocando-a. E agora? Essa reta é melhor que a anterior?

Como vimos na discussão inicial, podemos considerar que a melhor reta será aquela que minimiza a distância dos pontos à reta.

O Scikit-Learn nos permite encontrar esta reta com mínimo esforço.

```
In [14]: fig,ax = plt.subplots()  
ax.scatter(dados2.X,dados2.Y)  
x = [dados.X[0]+0.2,dados.X[4]-0.3]  
y = [dados.Y[0],dados.Y[4]]  
ax.plot(x,y)  
plt.show()
```



## Módulo 10 – O aprendizado de máquinas no Python

Primeiro, importamos o modelo `LinearRegression`. Então, passamos nossos valores de X e Y para o método `fit` do modelo:

```
In [15]: from sklearn.linear_model import LinearRegression  
modelo = LinearRegression()  
modelo.fit(dados2.X.values.reshape(-1, 1), dados2.Y)
```

```
Out[15]: LinearRegression()
```

Agora conseguimos obter o intercepto e o coeficiente angular de nossa reta:

```
In [16]: # Coeficiente linear  
print(modelo.intercept_)  
# Coeficiente angular  
print(modelo.coef_[0])
```

```
-13.493634099086592  
130.93412676446167
```

## Módulo 10 – O aprendizado de máquinas no Python

Veja que a reta gerada pelo modelo, em vermelho, nem é a união dos pontos inicial e final, nem coincide com a reta que fizemos manualmente.

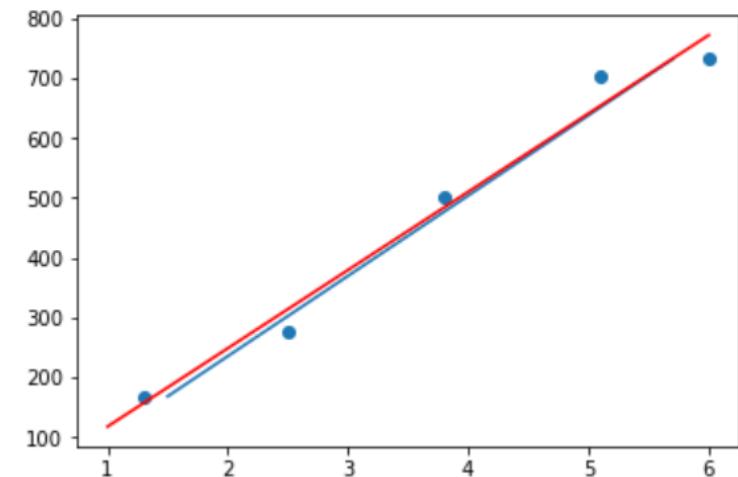
O algoritmo do modelo irá obter a reta que melhor descreve os pontos.

```
In [17]: x_ML = np.arange(1,7)
y_ML = modelo.intercept_ + x_ML*modelo.coef_[0]
```

```
In [18]: fig,ax = plt.subplots()

ax.scatter(dados2.X,dados2.Y)
x = [dados.X[0]+0.2,dados.X[4]-0.3]
y = [dados.Y[0],dados.Y[4]]
ax.plot(x,y)
ax.plot(x_ML,y_ML,c="red")

plt.show()
```



## Módulo 10 – O aprendizado de máquinas no Python

Tudo que foi feito para pontos em duas dimensões também funciona em 3 dimensões.

Vamos gerar um conjunto de dados:

```
In [26]: dados4 = {  
    'X': [1.3, 2.5, 3.8, 5.1, 6],  
    'Z': [0.2, 0.1, 0.3, 0.2, 0.1],  
    'Y': [158, 315, 450, 630, 700]  
}  
  
dados4 = pd.DataFrame(dados4)
```

E, da mesma forma que antes, passar para o método `fit`:

```
In [28]: from sklearn.linear_model import LinearRegression  
modelo = LinearRegression()  
modelo.fit(dados4.drop("Y", axis=1), dados4.Y)
```

```
Out[28]: LinearRegression()
```

## Módulo 10 – O aprendizado de máquinas no Python

E, assim como visto quando tínhamos duas dimensões, podemos obter nossos coeficientes. Só que agora será uma reta em três dimensões:

```
In [29]: # Coeficiente linear  
print(modelo.intercept_)  
# Coeficiente angular  
print(modelo.coef_)
```

```
14.948614201901592  
[116.86425248 -7.8939915 ]
```

A equação dessa reta é, portanto, aproximadamente:

$$y = 14.9486 + 116.8642x - 7.89399z$$

## Módulo 10 – O aprendizado de máquinas no Python

Utilizando o método `predict`, podemos ver como nosso modelo prevê valores de Y e comparar com os valores iniciais:

```
In [30]: dados4['Y_predicao'] = modelo.predict(dados4.drop("Y",axis=1))
```

```
In [31]: display(dados4)
```

	X	Z	Y	Y_predicao
0	1.3	0.2	158	165.293344
1	2.5	0.1	315	306.319846
2	3.8	0.3	450	456.664576
3	5.1	0.2	630	609.377504
4	6.0	0.1	700	715.344730

## Módulo 10 – Regressão linear no Scikit-Learn

Agora que temos um embasamento de como funciona com pontos fictícios, vamos pegar uma base mais próxima do real, com dados de vendas de uma loja. Nosso objetivo é estimar o estoque ideal para a loja.

Primeiro, vamos começar importando o Pandas, que será a biblioteca que utilizaremos para criar nossa base de dados a partir de uma planilha Excel.

Lembrando, o método `head` nos permite visualizar as primeiras entradas da base.

```
# Importando a base de dados
# - Base de dados: "dadosVenda.xlsx"
import pandas as pd
dados = pd.read_excel("dadosVenda.xlsx")
```

```
# Visualizando os dados
dados.head()
```

IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
0	1	21.85	23	1.15
1	2	4.30	5	0.70
2	3	13.65	15	1.35
3	4	4.97	7	2.03
4	5	9.60	10	0.40

## Módulo 10 – Regressão linear no Scikit-Learn

O método `info` nos fornece informações básicas para cada coluna do dataframe. Por exemplo, vemos que há 1634 linhas. No entanto, na coluna `Desconto` há apenas 1606 valores. Logo, há valores vazios nessa coluna.

É muito importante fazer essa checagem sempre, logo no início. Para definir qual a estratégia que será utilizada para tratar estas situações.

*# Verificando as informações da base  
dados.info()*

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1634 entries, 0 to 1633
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   IDRegistro      1634 non-null    int64  
 1   PrecoVenda      1634 non-null    float64 
 2   PrecoOriginal   1634 non-null    int64  
 3   Desconto        1606 non-null    float64 
 4   VendaQtd        1634 non-null    int64  
dtypes: float64(2), int64(3)
memory usage: 64.0 KB
```

## Módulo 10 – Regressão linear no Scikit-Learn

O método `describe` também é muito útil, dando um resumo estatístico de cada coluna. Aqui vemos também que há valores ausentes na coluna `Desconto`.

Em breve, ainda nesse módulo, entenderemos melhor o significado de cada descrição estatística dessa tabela.

`dados.describe()`

	IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
<b>count</b>	1634.000000	1634.000000	1634.000000	1606.000000	1634.000000
<b>mean</b>	817.500000	11.152889	13.913097	2.808331	1793.477356
<b>std</b>	471.839485	4.675302	5.422824	2.055469	960.099218
<b>min</b>	1.000000	2.820000	3.000000	0.050000	45.000000
<b>25%</b>	409.250000	7.200000	9.000000	1.140000	995.000000
<b>50%</b>	817.500000	10.920000	14.000000	2.250000	1753.000000
<b>75%</b>	1225.750000	14.520000	19.000000	4.200000	2646.000000
<b>max</b>	1634.000000	23.000000	23.000000	9.200000	3711.000000

## Módulo 10 – Regressão linear no Scikit-Learn

No nosso caso, os valores vazios simplesmente significam situações onde não houve desconto. Logo, podemos com o método `loc` substituir essas situações pelo valor zero.

Após a substituição, vemos com o método `info` que há 1634 valores na coluna `Desconto`:

```
# Ajustando os dados com valores de desconto vazio  
dados.loc[dados.Desconto.isnull(),"Desconto"] = 0
```

```
# Verificando novamente as informações da base  
dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1634 entries, 0 to 1633  
Data columns (total 5 columns):  
 #   Column           Non-Null Count  Dtype     
 ---  -----           -----            
 0   IDRegistro       1634 non-null   int64    
 1   PrecoVenda       1634 non-null   float64  
 2   PrecoOriginal    1634 non-null   int64    
 3   Desconto         1634 non-null   float64  
 4   VendaQtd         1634 non-null   int64    
 dtypes: float64(2), int64(3)  
 memory usage: 64.0 KB
```

## Módulo 10 – Regressão linear no Scikit-Learn

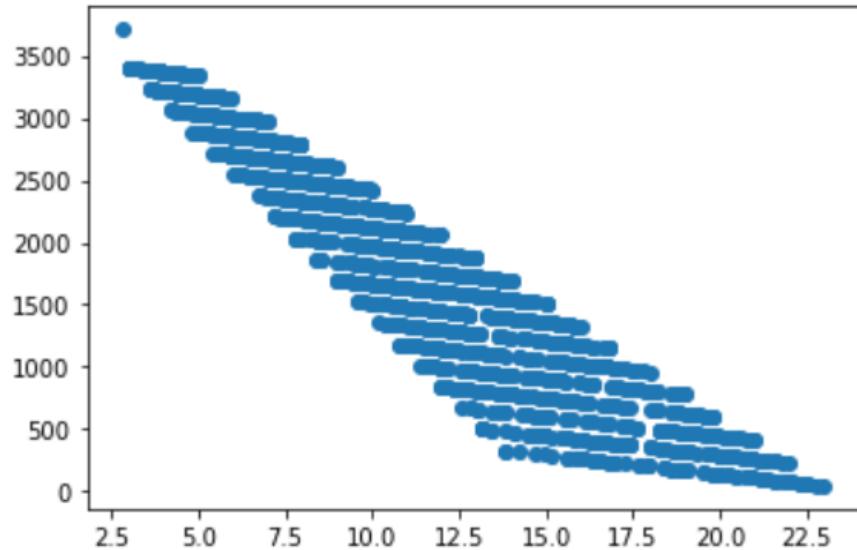
Uma boa forma de entender os dados, é fazer um gráfico de dispersão, *scatter plot* em inglês. Para isso, importamos o Matplotlib e utilizamos o método `scatter`, passando nosso X e Y desejado. No caso, preço de vendas e quantidade de vendas.

```
# Podemos fazer o scatter plot
import matplotlib.pyplot as plt

# plot
fig, ax = plt.subplots()

ax.scatter(dados.PrecoVenda, dados.VendaQtd)

plt.show()
```



## Módulo 10 – Regressão linear no Scikit-Learn

Após ajustar os dados e visualizá-los, podemos partir para nosso modelo de machine learning. Assim como já fizemos antes, importaremos o `LinearRegression` do Scikit-Learn e passaremos nosso X e nosso y desejados:

```
# Importando o LinearRegression
from sklearn.linear_model import LinearRegression
```

```
# Criando os dados
X = dados.PrecoVenda
y = dados.VendaQtd
```

```
# Fazendo o fit do modelo
reg = LinearRegression().fit(X.values.reshape(-1, 1), y)
```

## Módulo 10 – Regressão linear no Scikit-Learn

Para verificar o quanto bom um modelo linear descreve nossos dados, usamos o método `score`. Quanto mais próximo de 1, melhor o modelo linear descreve nossos dados:

```
reg.score(X.values.reshape(-1, 1), y)  
0.8956436940447059
```

## Módulo 10 – Regressão linear no Scikit-Learn

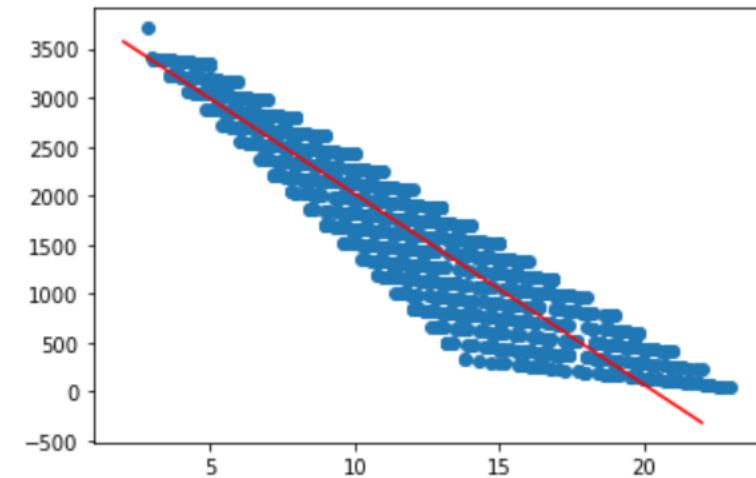
Vimos, também, que podemos obter o coeficiente angular da reta e o intercepto com `coef_` e `Intercept_`. Assim, podemos criar nossa reta graficamente:

```
# Podemos fazer o scatter plot
import matplotlib.pyplot as plt
import numpy as np

# plot
fig, ax = plt.subplots()

ax.scatter(dados.PrecoVenda, dados.VendaQtd)
x_plot = np.arange(2,23)
y_plot = reg.coef_[0]*x_plot + reg.intercept_
ax.plot(x_plot,y_plot,c='r')

plt.show()
```



## Módulo 10 – Regressão linear no Scikit-Learn

Vamos criar um conjunto de valores para os quais queremos fazer nossa previsão:

```
# Podemos definir uma lista de valores para fazermos a previsão
valores = {
    'valores': [17.50, 17.40, 17.30, 17.20, 17.10]
}

valores = pd.DataFrame(valores)
```

Com o método `predict`, podemos verificar qual a previsão de vendas para cada preço:

```
# Fazendo o predict
reg.predict(valores)

array([559.94606196, 579.3805924 , 598.81512285, 618.2496533 ,
       637.68418374])
```

## Módulo 10 – Regressão linear no Scikit-Learn

Será que essa previsão faz sentido? Veja abaixo que, pegando apenas o valor de 17.10, o valor previsto (~637) é bem distante do valor que realmente foi vendido:

```
# Verificando os dados para esse mesmo preço de venda  
dados[dados.PrecoVenda == 17.10]
```

IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
397	398	17.1	19	1.9
492	493	17.1	19	1.9
1414	1415	17.1	18	0.9

## Módulo 10 – Regressão linear no Scikit-Learn

Podemos olhar com mais cuidado a faixa entre 17 e 18 de preço original, como abaixo. Os valores de venda são bem distintos do previsto. Ou seja, erramos no nosso modelo. Afinal, fizemos apenas com o preço de venda, mas talvez seja interessante considerar o preço original e o desconto.

```
# Verificando os dados para um range de preços e descontos  
dados[(dados.PrecoOriginal >= 17) & (dados.PrecoOriginal <= 18) & (dados.Desconto >= 0) & (dados.Desconto <= 0.4)]
```

IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
108	109	16.83	17	0.17
284	285	16.66	17	0.34
452	453	18.00	18	0.00
989	990	16.83	17	0.17
998	999	16.83	17	0.17
1115	1116	17.82	18	0.18
1220	1221	17.64	18	0.36
1539	1540	16.66	17	0.34
1548	1549	16.66	17	0.34
1576	1577	16.83	17	0.17

## Módulo 10 – Regressão linear no Scikit-Learn

Vamos então, recriar nosso modelo colocando como X as colunas **PrecoOriginal** e **Desconto**. Veja que é basicamente a mesma estrutura de código anterior.

Perceba como o valor do **score** melhorou bastante, mostrando que o novo modelo descreve muito melhor este conjunto de dados.

```
# Importar o algoritmo
from sklearn.linear_model import LinearRegression
```

```
# Dados
```

```
X2 = dados[["PrecoOriginal", "Desconto"]]
y2 = dados.VendaQtd
```

```
# Fazendo o fit
```

```
reg2 = LinearRegression().fit(X2, y2)
```

```
reg2.score(X2, y2)
```

```
0.9999999083594651
```

```
reg.score(X.values.reshape(-1, 1), y)
```

```
0.8956436940447059
```

```
# Coeficiente angular
```

```
reg2.coef_
```

```
array([-182.99974391,  31.00137454])
```

```
# Coeficiente linear
```

```
reg2.intercept_
```

```
4254.00024394365
```

## Módulo 10 – Regressão linear no Scikit-Learn

Veja que, com esse novo modelo, passando valores para previsão, temos valores mais próximos aos que efetivamente ocorreram:

```
valores = {  
    'valores': [17.50, 17.40, 17.30, 17.20, 17.10],  
    'precos': [17.50, 17.50, 17.50, 17.50, 17.50],  
    'descontos': [0, 0.1, 0.2, 0.3, 0.4]  
}  
  
valores = pd.DataFrame(valores)  
  
# Fazendo a predição  
reg2.predict(valores[["precos", "descontos"]])  
  
array([1051.50472554, 1054.604863 , 1057.70500045, 1060.8051379 ,  
       1063.90527536])
```

## Módulo 10 – Regressão linear no Scikit-Learn

Compare a última coluna com as previsões anteriores:

```
# Verificando novamente os dados para um range de preços e descontos  
dados[(dados.PrecoOriginal >= 17) & (dados.PrecoOriginal <= 18) & (dados.Desconto >= 0) & (dados.Desconto <= 0.4)]
```

IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
108	109	16.83	17	0.17
284	285	16.66	17	0.34
452	453	18.00	18	0.00
989	990	16.83	17	0.17
998	999	16.83	17	0.17
1115	1116	17.82	18	0.18
1220	1221	17.64	18	0.36
1539	1540	16.66	17	0.34
1548	1549	16.66	17	0.34
1576	1577	16.83	17	0.17

## Módulo 10 – Regressão linear no Scikit-Learn

Vamos comparar as previsões dos dois modelos lado a lado, criando colunas no nosso dataframe.

Novamente, veja como o segundo modelo apresenta valores muito mais próximos para a quantidade de vendas real.

```
dados['y_model1'] = reg.predict(dados.PrecoVenda.values.reshape(-1, 1))
dados['y_model2'] = reg2.predict(dados[['PrecoOriginal', 'Desconto']])
```

```
dados.head()
```

IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd	y_model1	y_model2
0	1	21.85	23	1.15	81	-285.456012
1	2	4.30	5	0.70	3361	3125.304081
2	3	13.65	15	1.35	1551	1308.175484
3	4	4.97	7	2.03	3036	2995.092727
4	5	9.60	10	0.40	2436	2095.273967

```
dados.tail()
```

IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd	y_model1	y_model2
1629	1630	11.96	13	1.04	1907	1636.619049
1630	1631	7.40	10	2.60	2505	2522.833637
1631	1632	12.15	15	2.85	1597	1599.693441
1632	1633	13.44	16	2.56	1405	1348.987998
1633	1634	2.82	3	0.18	3711	3412.935132

## Módulo 10 – Regressão linear no Scikit-Learn

No Scikit-Learn, estão disponíveis muitas métricas para calcular erros de modelos. No momento, não nos aprofundaremos no significado de cada uma delas. Mais veja como, nas duas que importamos, o segundo modelo tem valores bem menores que o primeiro modelo.

À medida que formos avançando no curso, nos aprofundaremos no significado de cada uma dessas métricas de erro. Nesse caso, podemos interpretar como sendo a distância dos pontos à reta calculada. O segundo modelo tem uma reta que descreve melhor nossos dados, tendo estes uma menor distância à reta.

Para começar a ter o embasamento necessário para o entendimento dessas métricas, vejamos um pouco de estatística, aproveitando para aprender mais sobre Pandas, até para entender também o método `describe` visto anteriormente.

```
from sklearn.metrics import mean_absolute_error  
  
# para o modelo 1  
mean_absolute_error(dados.VendaQtd, dados.y_model1)  
254.3109951899015
```

```
# para o modelo 2  
mean_absolute_error(dados.VendaQtd, dados.y_model2)  
0.24931777290882204
```

```
from sklearn.metrics import mean_absolute_percentage_error  
  
# para o modelo 1  
mean_absolute_percentage_error(dados.VendaQtd, dados.y_model1)*100  
33.713612693046244
```

```
# para o modelo 2  
mean_absolute_percentage_error(dados.VendaQtd, dados.y_model2)*100  
0.027306697265634318
```

## Módulo 10 – Descrição estatística com Pandas

Para começar nosso estudo, vamos considerar o seguinte exemplo, de notas de usuários quanto ao atendimento de problemas de duas lojas de carros.

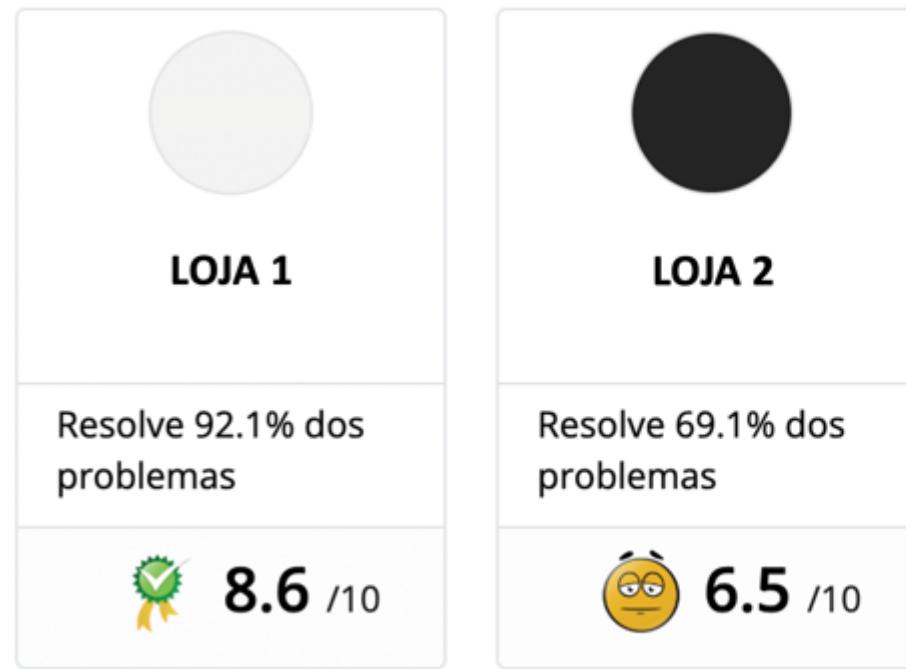
Qual das duas empresas abaixo você escolheria pra comprar seu carro? Sabendo que a nota apresentada é uma média das notas dos usuários.



## Módulo 10 – Descrição estatística com Pandas

A média é insuficiente para fazermos uma boa escolha! Nossa objetivo é entender o porquê.

Temos diversas medidas estatísticas que nos permitem uma análise mais completa de dados:



- Média
- Mediana
- Moda

- Medidas de Dispersão**

**Medidas de Tendência Central**

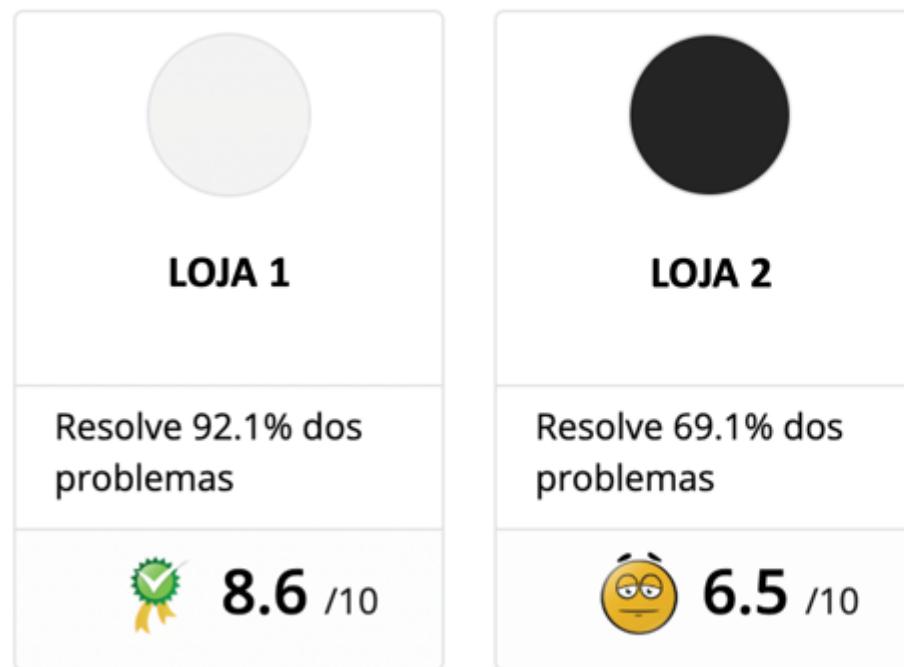
- Range
- Variância
- Desvio Padrão

- Medidas de Tendência Central**

- Quartis

## Módulo 10 – Descrição estatística com Pandas

Não é necessário pegar cada valor individualmente, embora seja possível com métodos específicos:



```
base.empresal.mean()
```

```
3.0
```

```
base.empresal.median()
```

```
3.0
```

```
base.empresal.std()
```

```
2.1908902300206643
```

```
base.empresal.max()
```

```
5
```

```
base.empresal.min()
```

```
1
```

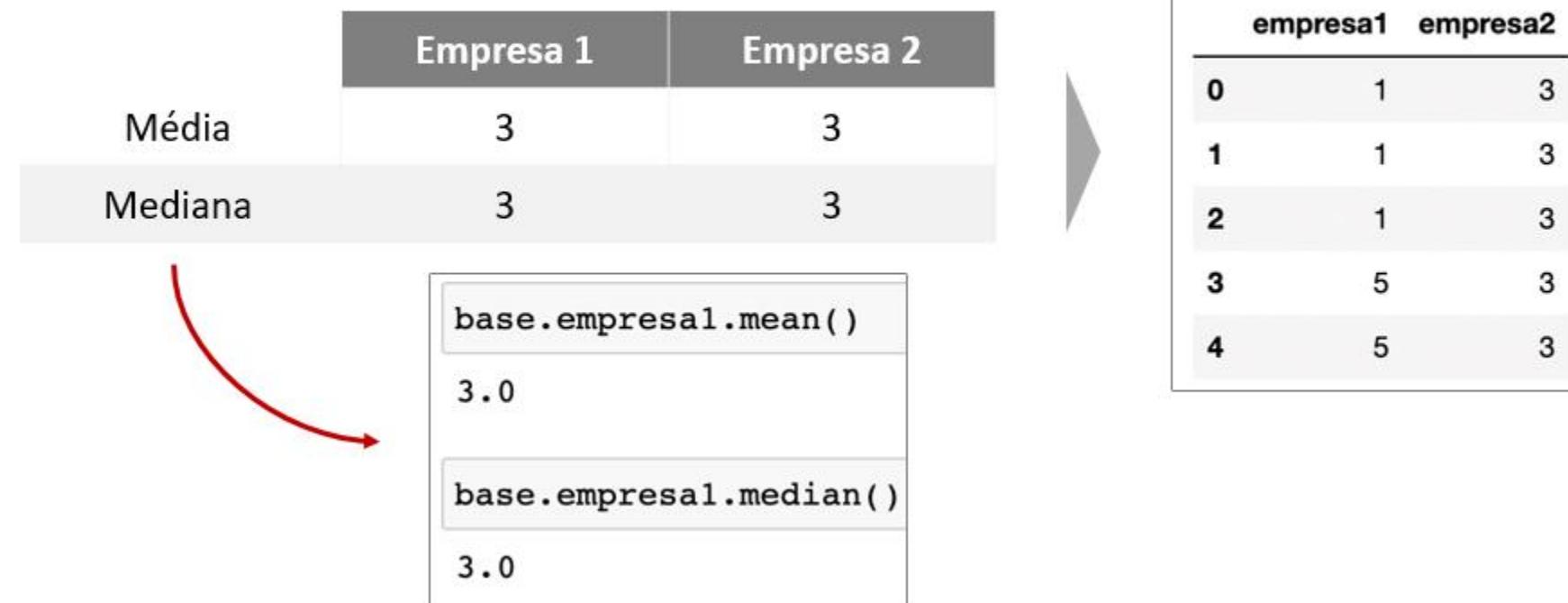
## Módulo 10 – Descrição estatística com Pandas

Como vimos anteriormente, o método `describe` do Pandas já apresenta as principais métricas todas resumidas em um dataframe fácil de compreender. Veja que, no nosso exemplo, há 6 avaliações para cada empresa. A **Empresa 1** apresenta valor de nota mínima 1 e máxima 5, enquanto que a **Empresa 2** apresenta todas as notas iguais a 3.



## Módulo 10 – Descrição estatística com Pandas

Uma curiosidade nesse caso, é que ambas as empresas possuem mesma média e mesma mediana:

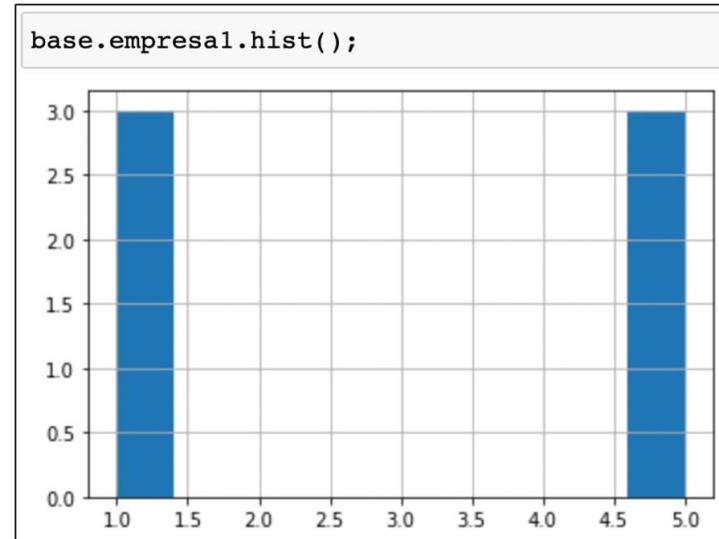


## Módulo 10 – Descrição estatística com Pandas

Ora, então o que as difere? A **dispersão** dos dados. Podemos construir histogramas para as duas empresas, seja pelo método `hist` do Pandas, seja pelo `bar` do Matplotlib. Abaixo, as duas formas para a **Empresa 1**.

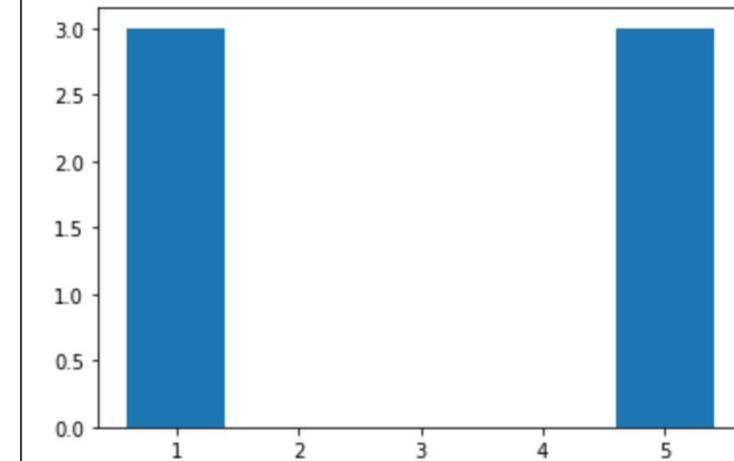
```
display(base)
```

	empresa1	empresa2
0	1	3
1	1	3
2	1	3
3	5	3
4	5	3
5	5	3



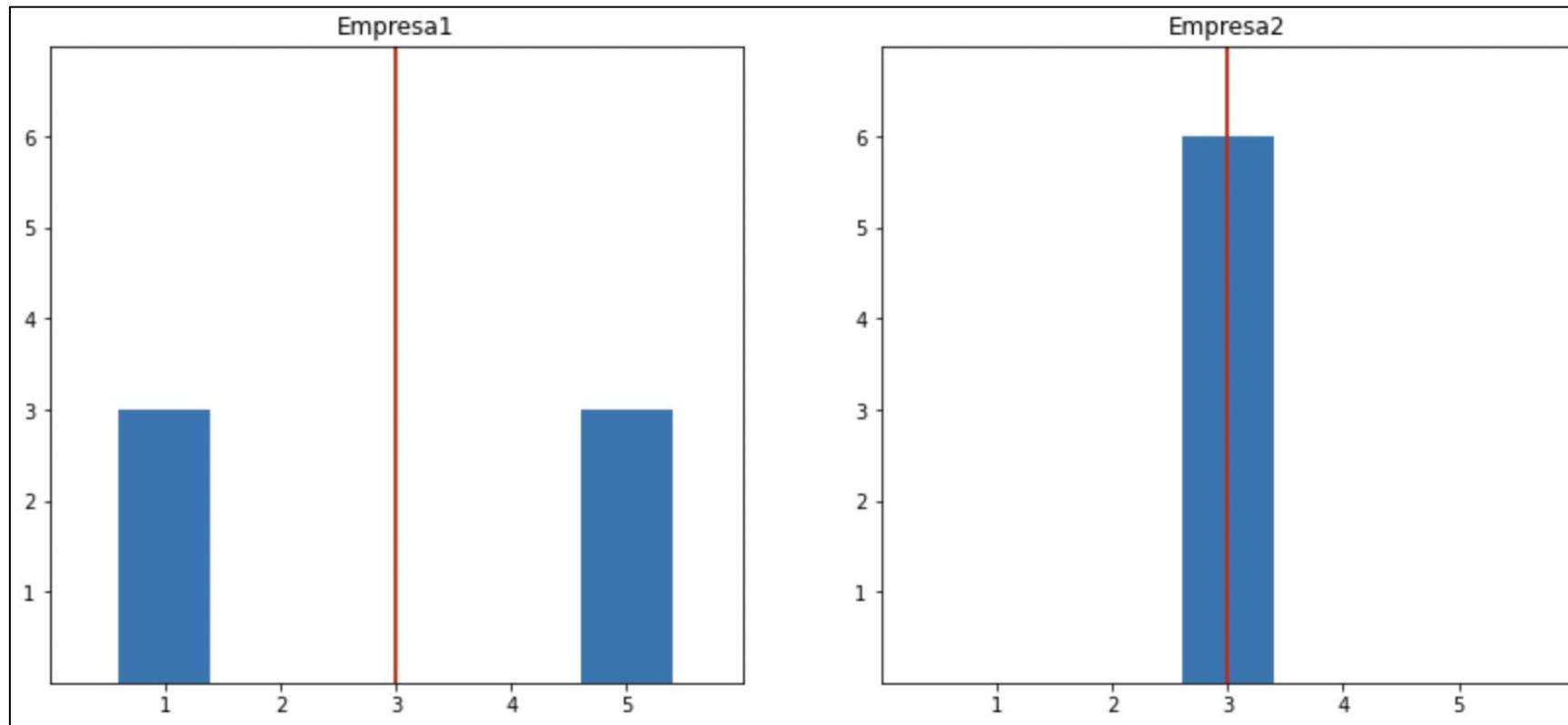
```
x = base.empresal.value_counts().index  
y = base.empresal.value_counts().values
```

```
import matplotlib.pyplot as plt  
fig, ax = plt.subplots()  
ax.bar(x, y)  
plt.show()
```



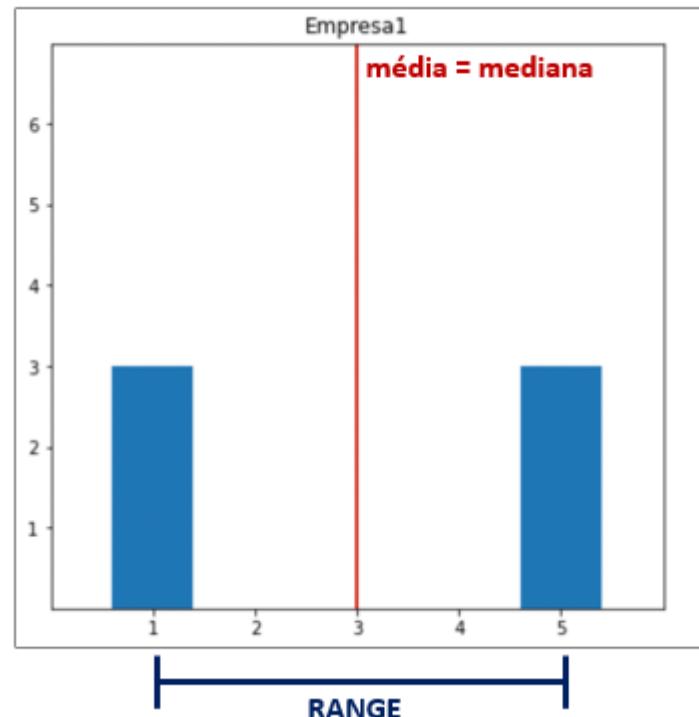
## Módulo 10 – Descrição estatística com Pandas

Observe que a **Empresa 2** é consistente nas avaliações, enquanto que a **Empresa 1** possui avaliações muito ruins e muito boas. Logo, o que você prefere? Consistência de um atendimento regular, ou a chance de ter um atendimento muito bom ou muito ruim?



## Módulo 10 – Descrição estatística com Pandas

Vemos na **Empresa 1**, que há uma diferença entre os valores máximo e mínimo, que chamamos de *range*, ou amplitude. A amplitude é muito sensível a outliers e, consequentemente, a média também.



RANGE (ou amplitude):

$$R = \max - \min$$

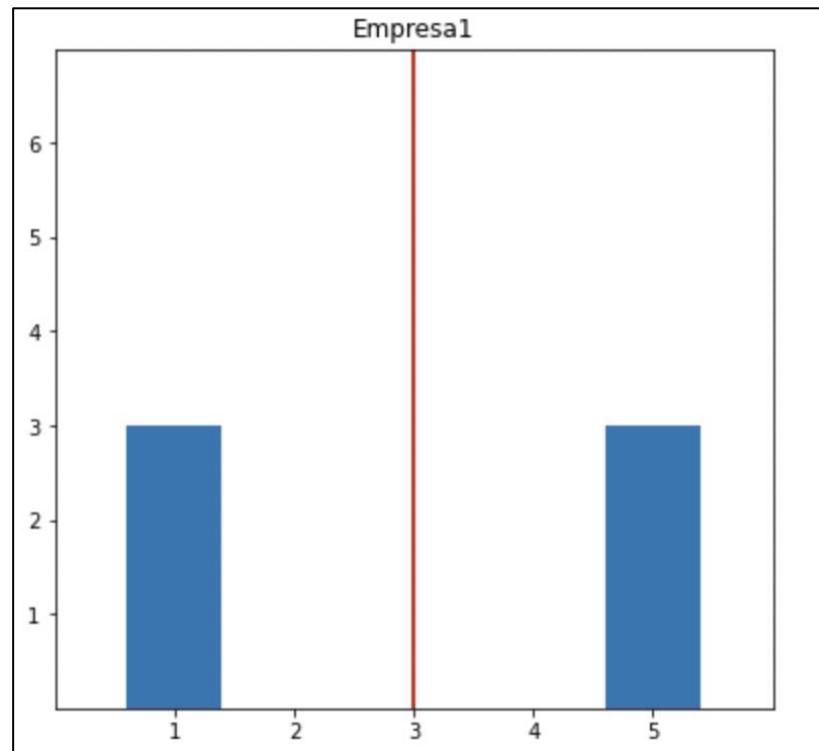
```
max1 = base.empresal.max()  
min1 = base.empresal.min()
```

```
R = max1 - min1  
print(R)
```

4

## Módulo 10 – Descrição estatística com Pandas

Uma forma de detectar a dispersão de dados, é utilizar a distância de cada ponto a média. Como há distâncias positivas e negativas, elevamos ao quadrado os valores. O nome dessa métrica é **variância amostral**:



$x$	$x - \text{média}$	$(x - \text{média})^2$
1	$1 - 3 = -2$	4
1	$1 - 3 = -2$	4
1	$1 - 3 = -2$	4
5	$5 - 3 = +2$	4
5	$5 - 3 = +2$	4
5	$5 - 3 = +2$	4

**VARIÂNCIA AMOSTRAL:**

$$s^2 = \frac{\sum(x - \bar{x})^2}{n - 1}$$

$$s^2 = 4,8$$

## Módulo 10 – Descrição estatística com Pandas

O **desvio padrão** é simplesmente a raiz quadrada da variância amostral. É o **std** que aparece no `describe` do Pandas:

### DESVIO PADRÃO AMOSTRAL:

$$s = \sqrt{S^2}$$

$$s = \sqrt{4,8}$$

$$s \approx 2,19$$



```
desvio_padrao = base.empresal.std()  
print(desvio_padrao)
```

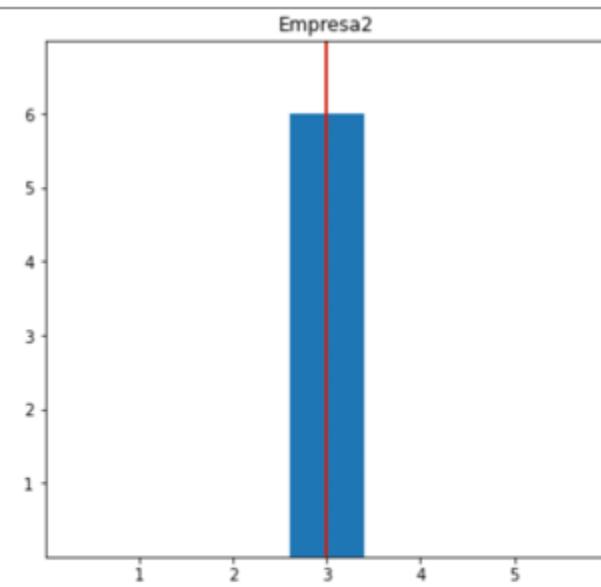
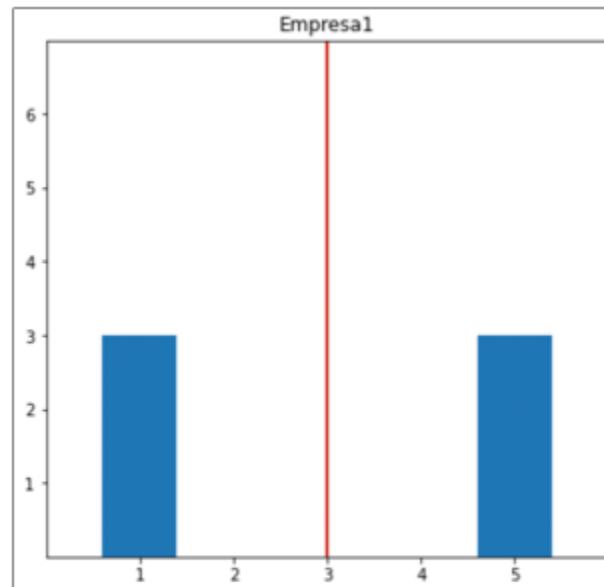
```
2.1908902300206643
```

```
variancia = (desvio_padrao)**2  
print(variancia)
```

```
4.799999999999999
```

## Módulo 10 – Descrição estatística com Pandas

Veja como o desvio padrão da **Empresa 1** é maior que o da **Empresa 2**, mostrando uma maior dispersão das notas para a primeira empresa.



```
desvio_padrao = base.empresal.std()  
print(desvio_padrao)
```

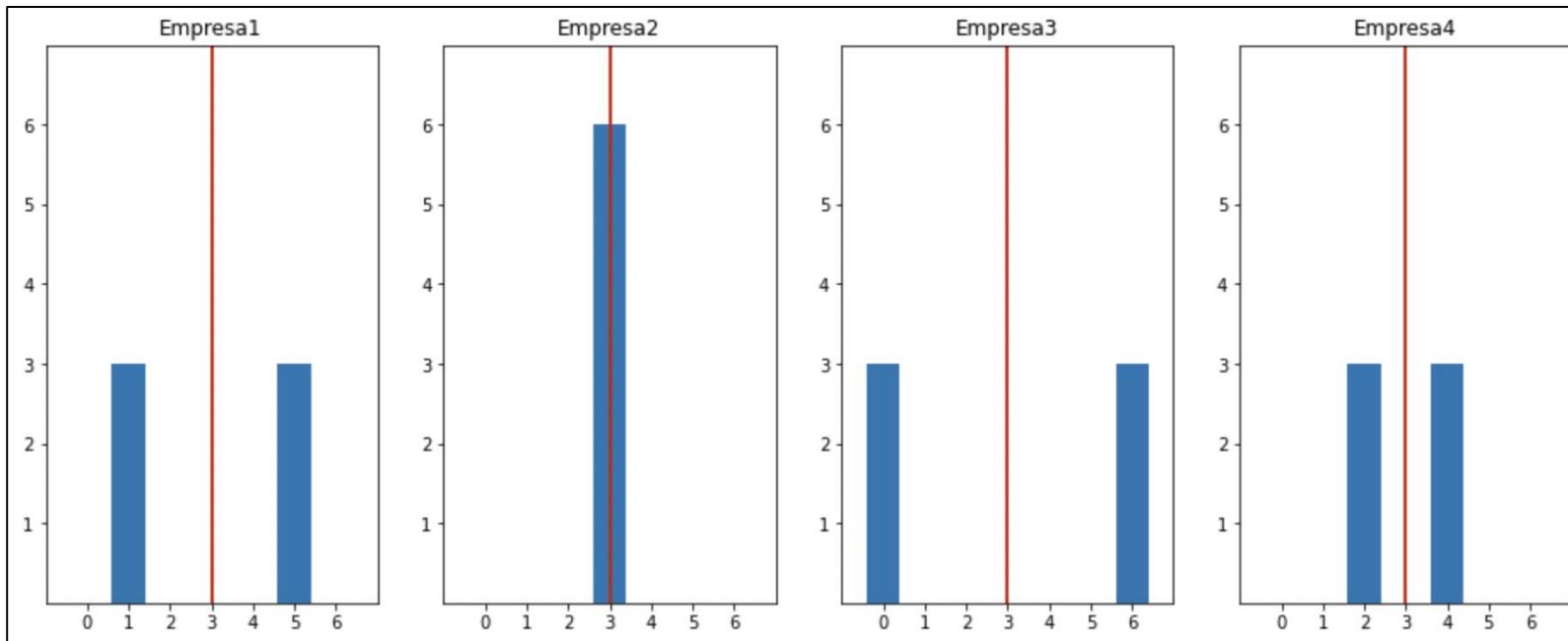
2.1908902300206643

```
desvio_padrao = base.empresa2.std()  
print(desvio_padrao)
```

0.0

## Módulo 10 – Descrição estatística com Pandas

Apenas para reforçar o conceito, vamos adicionar mais duas empresas que, também, possuem média de notas igual a 3. Observe como a **Empresa 3** possui uma maior dispersão que as demais.



```
base.describe()
```

	empresa1	empresa2	empresa3	empresa4
count	6.00000	6.0	6.000000	6.000000
mean	3.00000	3.0	3.000000	3.000000
std	2.19089	0.0	3.286335	1.095445

## Módulo 10 – Descrição estatística com Pandas

Além das dispersão de valores, o desvio padrão também pode me dar uma ideia de proporção. É sobre isso que fala o **Teorema de Chebyshev**.

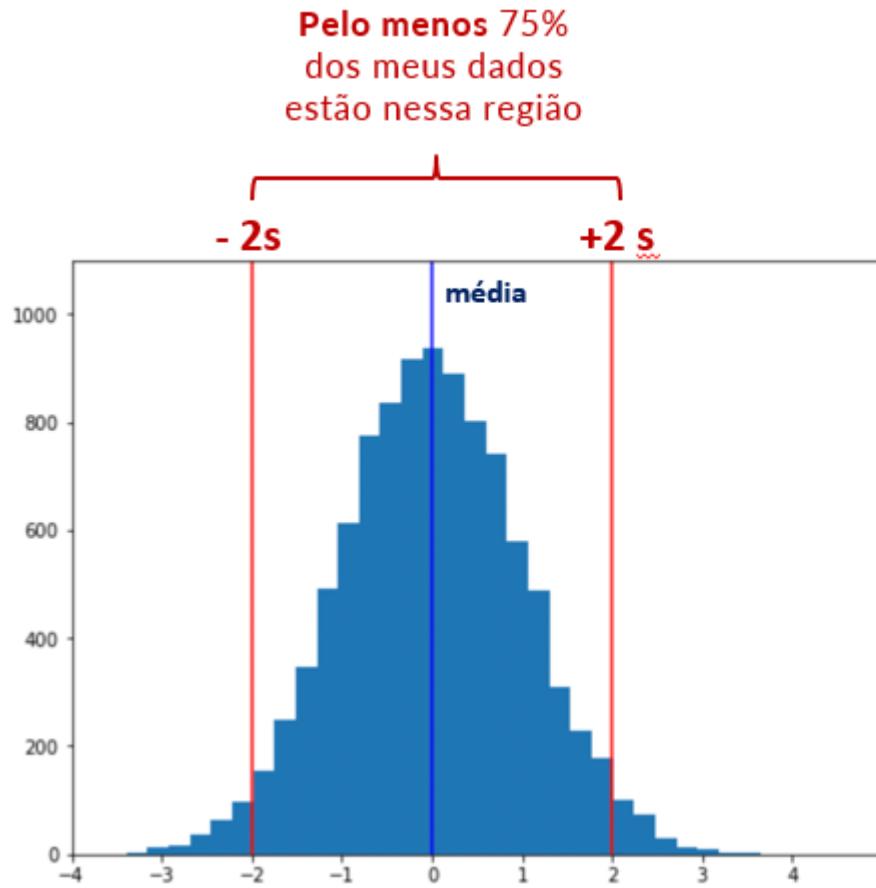
Teorema de Chebyshev: para qualquer distribuição de dados, a *proporção mínima* de dados que está a menos de  $k$  desvios padrões é dada por

$$\% = 1 - \frac{1}{k^2}$$

Vamos ter uma ideia visual deste teorema.

## Módulo 10 – Descrição estatística com Pandas

Com base no teorema, temos **pelo menos**, 75 % dos nossos dados entre -2 e 2 desvios padrão:



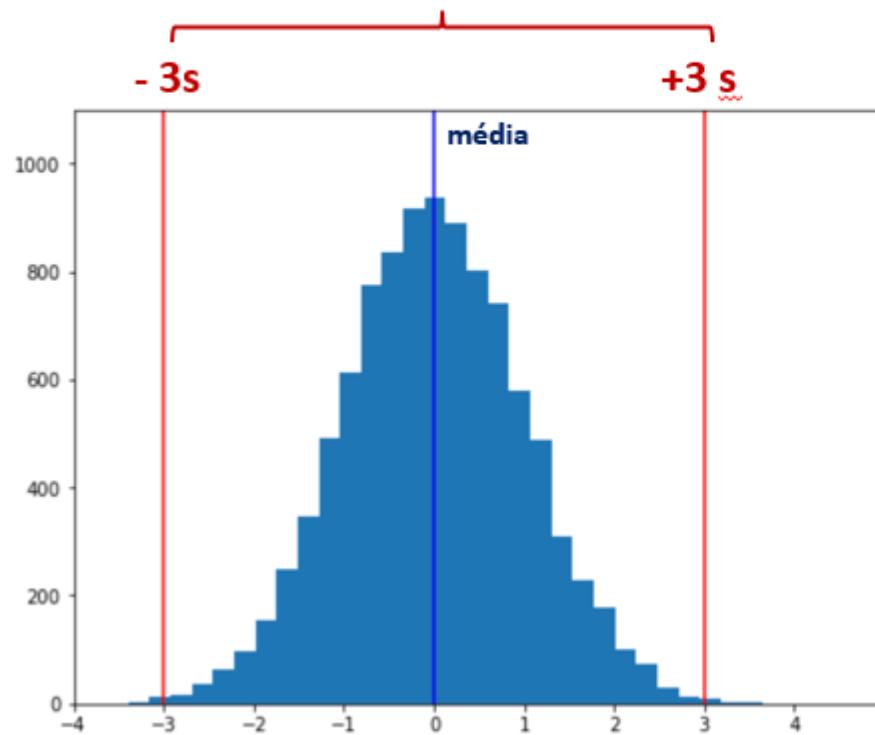
$$k = 2$$

$$\% = 1 - \frac{1}{2^2}$$
$$\% = 75\%$$

# Módulo 10 – Descrição estatística com Pandas

Com base no teorema, temos **pelo menos**, 89 % dos nossos dados entre -3 e 3 desvios padrão:

**Pelo menos 89%  
dos meus dados  
estão nessa região**



$$k = 3$$

$$\% = 1 - \frac{1}{3^2}$$

% = 89%

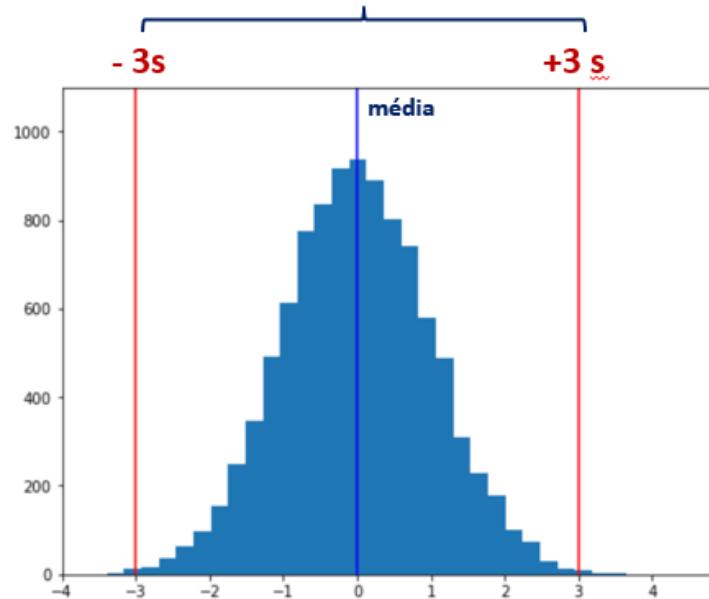
## Módulo 10 – Descrição estatística com Pandas

Observe que a definição cita “proporção mínima”. E, nas imagens, colocamos “pelo menos”. Isto é, para uma dada distribuição, no mínimo aquela porcentagem será encontrada.

Veremos com frequência distribuições normais. Assim, vejamos os valores realmente encontrados em distribuições normais:

$k$	% mínima dos dados	% dos dados (normal)
1	-	68,3%
2	75%	95,5%
3	~ 88,89%	<b>99,7%</b>
4	93,75%	
5	96%	
...	...	

Como é uma normal, 99,7% dos meus dados estão nessa região



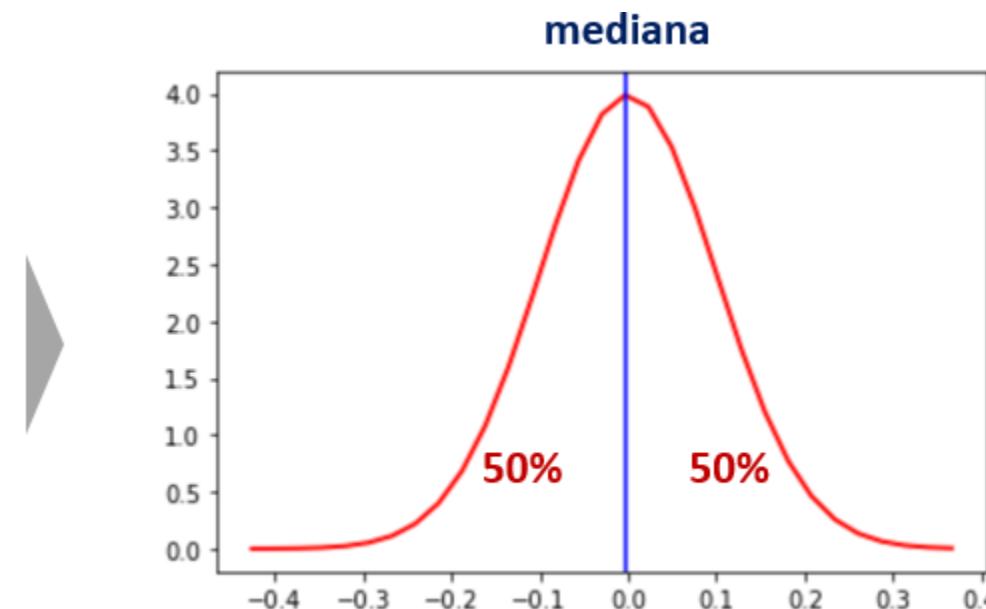
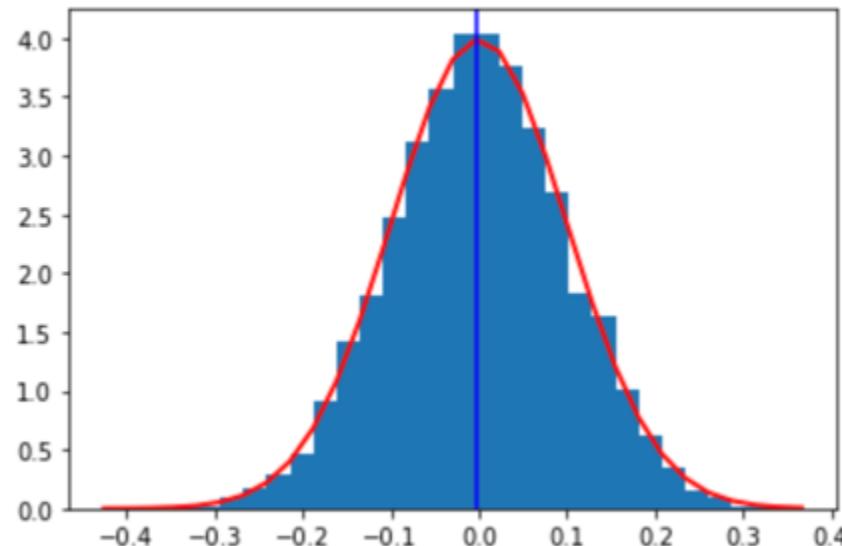
$$k = 3$$

$$\% = 1 - \frac{1}{3^2}$$
$$\% = 89\%$$

## Módulo 10 – Descrição estatística com Pandas

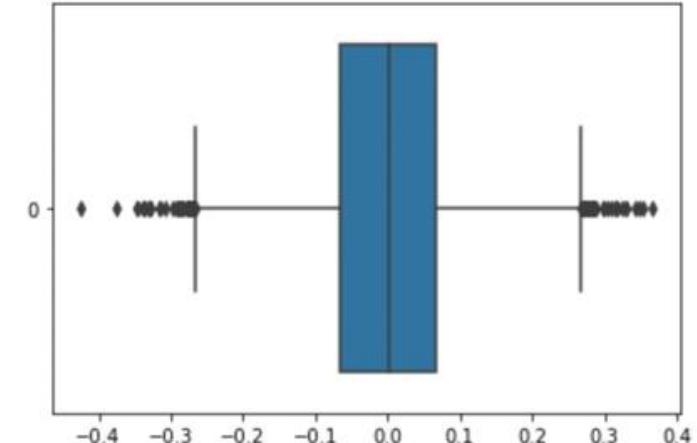
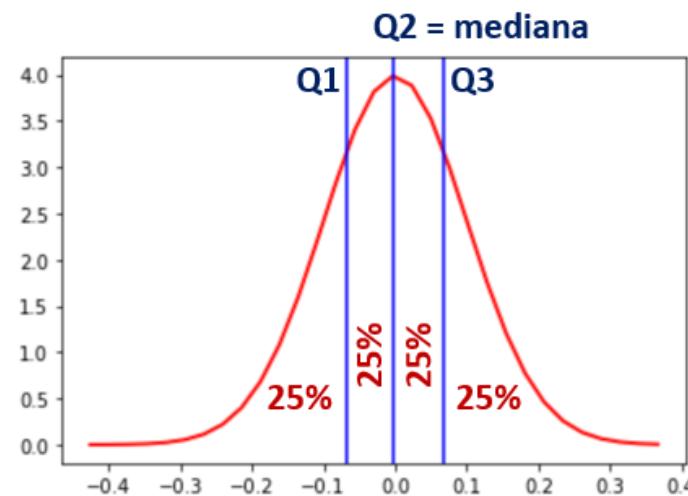
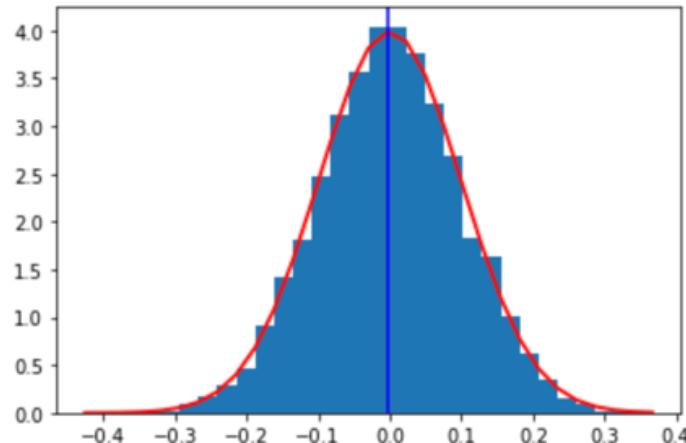
**Separatrizes** são pontos que dividem o conjunto de dados em partes iguais.

A **mediana** é uma separatrix, que divide o conjunto pela metade:



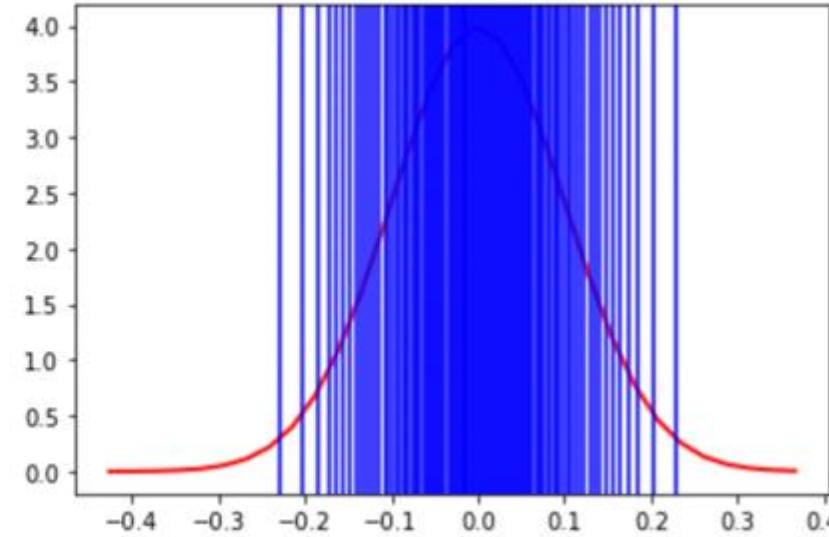
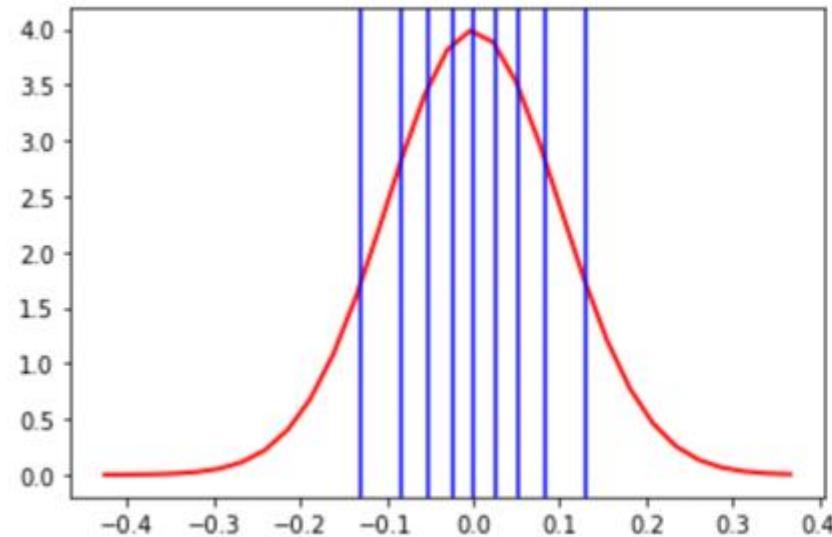
## Módulo 10 – Descrição estatística com Pandas

**Quartis** dividem a base em 4 partes iguais. É um conceito que aparecerá mais vezes por estar intimamente relacionado com os gráficos de boxplot:



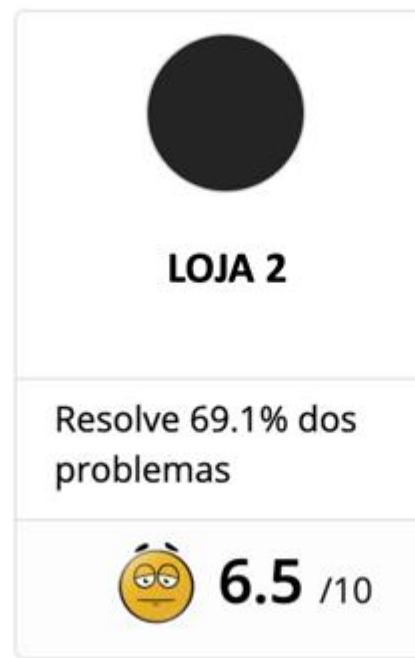
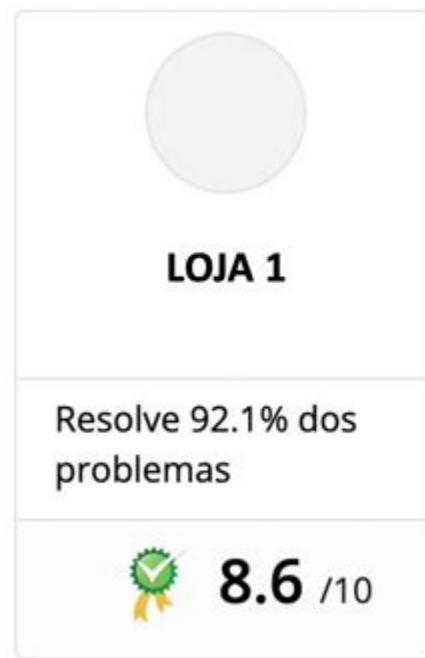
## Módulo 10 – Descrição estatística com Pandas

**Decis** dividem a base em 10, **centis** em 100, ...



## Módulo 10 – Descrição estatística com Pandas

E agora? Qual empresa? Já vimos o significado de cada entrada que aparece no retorno no `describe`. Com isto, você pode escolher a empresa com base no que prefere: consistência no atendimento, ou a chance de ter um atendimento muito bom ou muito ruim.



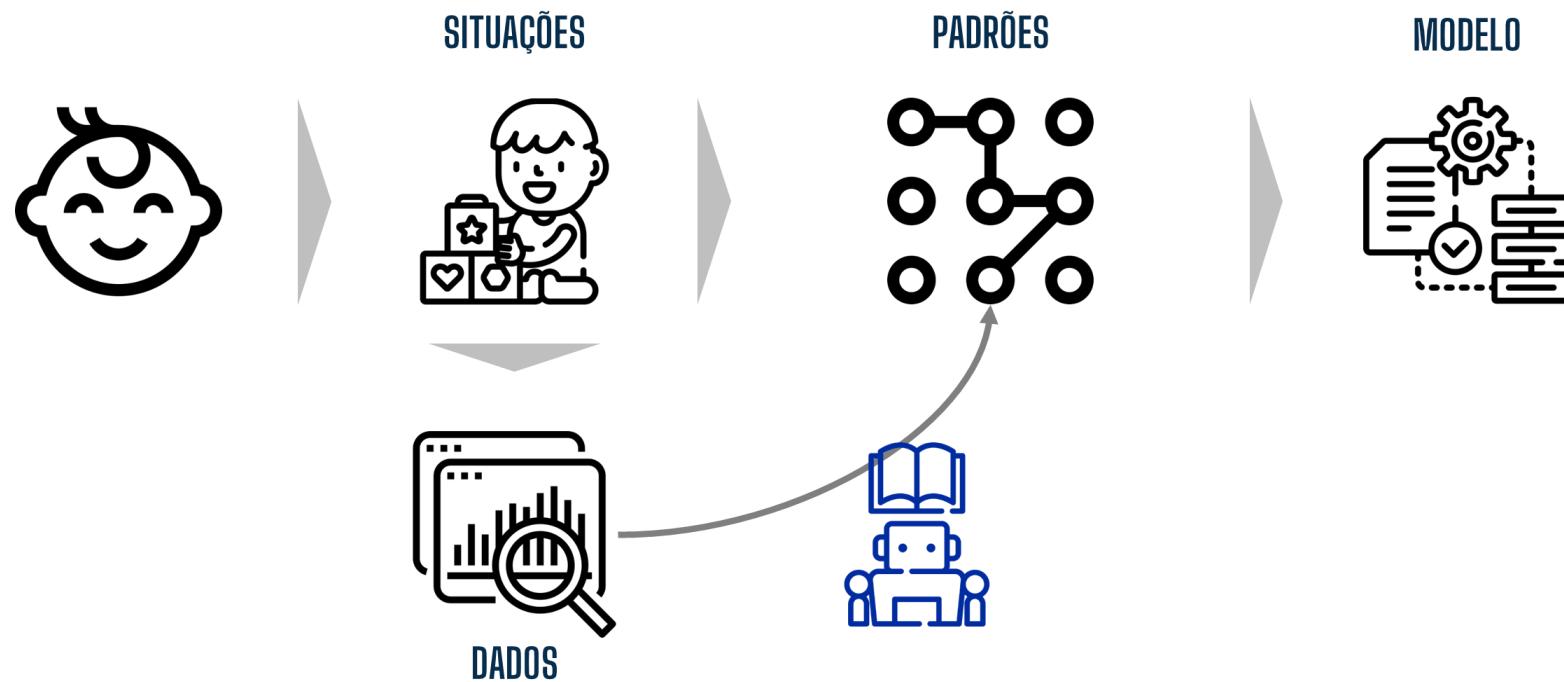
	empresa1	empresa2
count	6.00000	6.0
mean	3.00000	3.0
std	2.19089	0.0
min	1.00000	3.0
25%	1.00000	3.0
50%	3.00000	3.0
75%	5.00000	3.0
max	5.00000	3.0

## MÓDULO 11

# Como as máquinas aprendem?

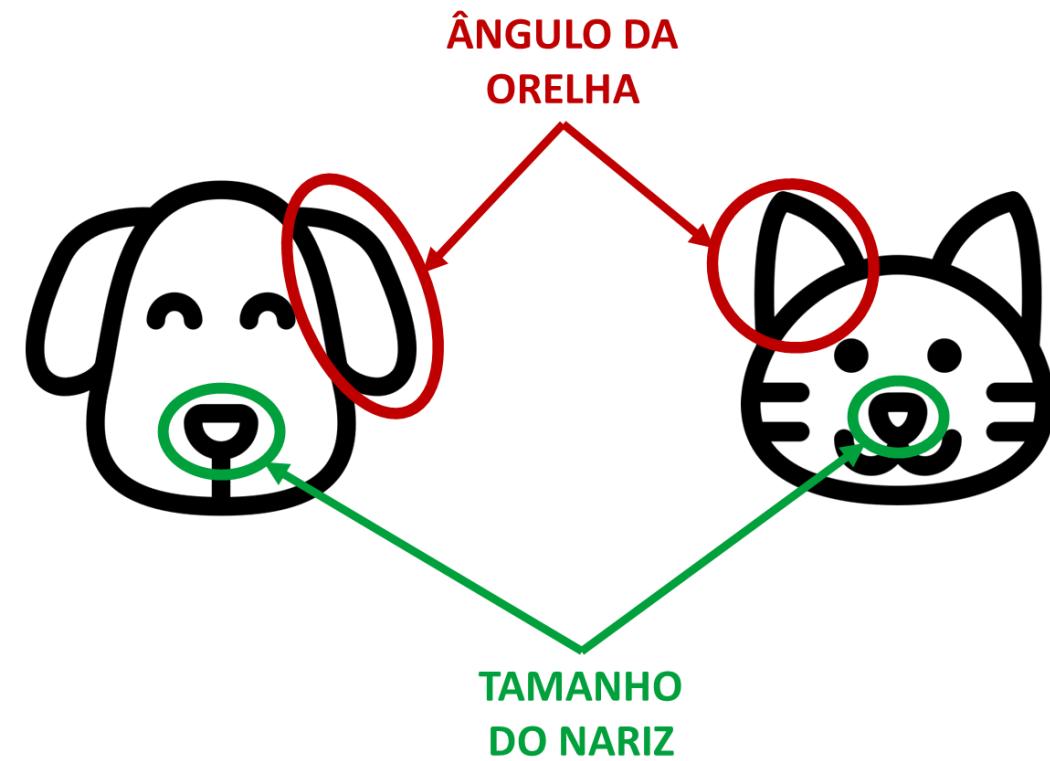
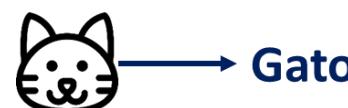
## Módulo 11 – Tipos de aprendizado

Como já vimos no módulo anterior, nosso modelo precisa de dados para aprender reconhecendo padrões. Muito similar a um aprendizado de uma criança.



## Módulo 11 – Tipos de aprendizado – Aprendizado supervisionado

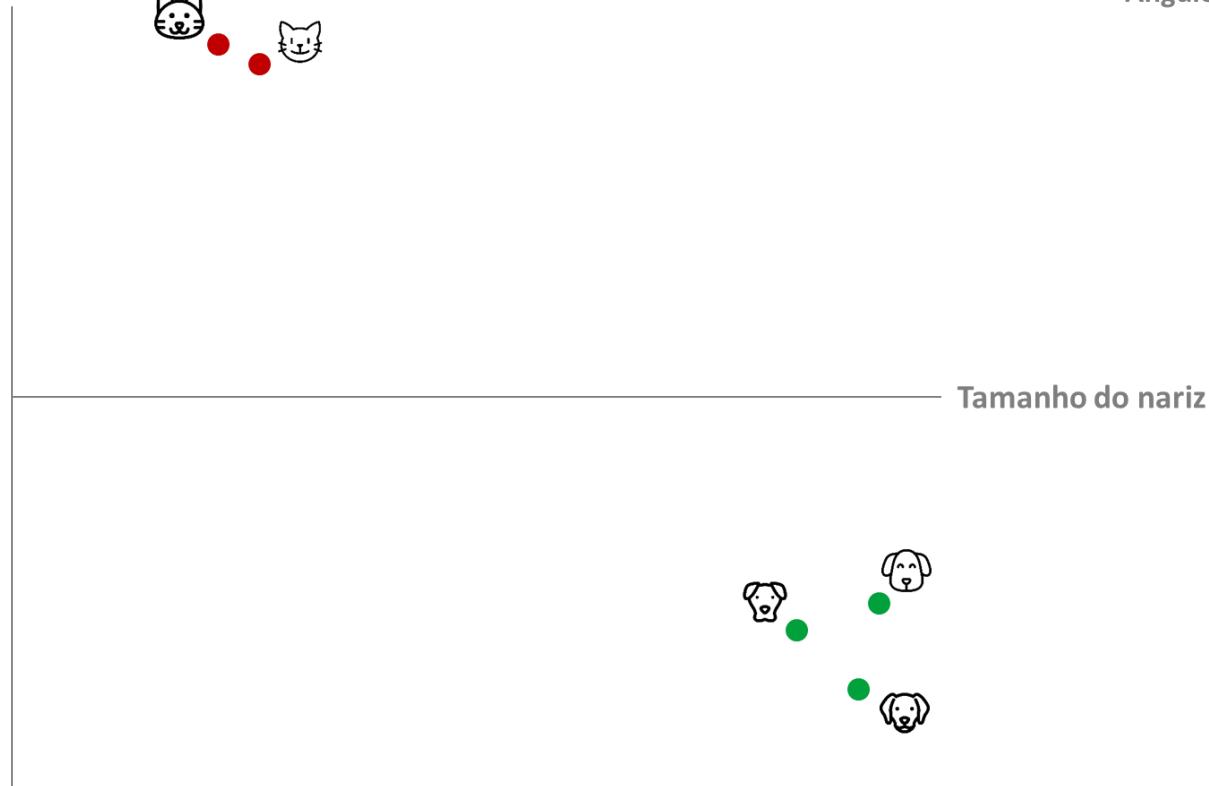
Considere um exemplo onde queremos criar um modelo que classifique corretamente imagens de animais como sendo cachorro ou gato. O modelo pode encontrar padrões nas imagens como, por exemplo, o formato da orelha e o tamanho do nariz:



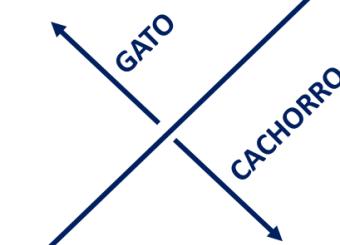
## Módulo 11 – Tipos de aprendizado – Aprendizado supervisionado

Com base nestes padrões, o modelo pode definir um limite, visualizado como uma reta abaixo, separando o que ele classificará como cachorro e o que será classificado como gato.

Ângulo da orelha



Ângulo da orelha

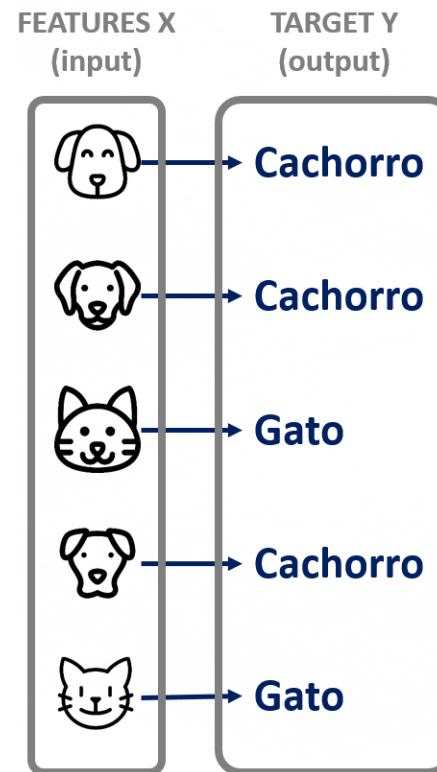


Tamanho do nariz

Tamanho do nariz

## Módulo 11 – Tipos de aprendizado – Aprendizado supervisionado

Repare, nesse nosso exemplo, que os dados de partida são rotulados. Desde o início nosso modelo já sabe que há duas classificações possíveis e fornecemos exemplos de cada uma das classificações ao modelo. Esse tipo de aprendizado é chamado de **supervisionado**.

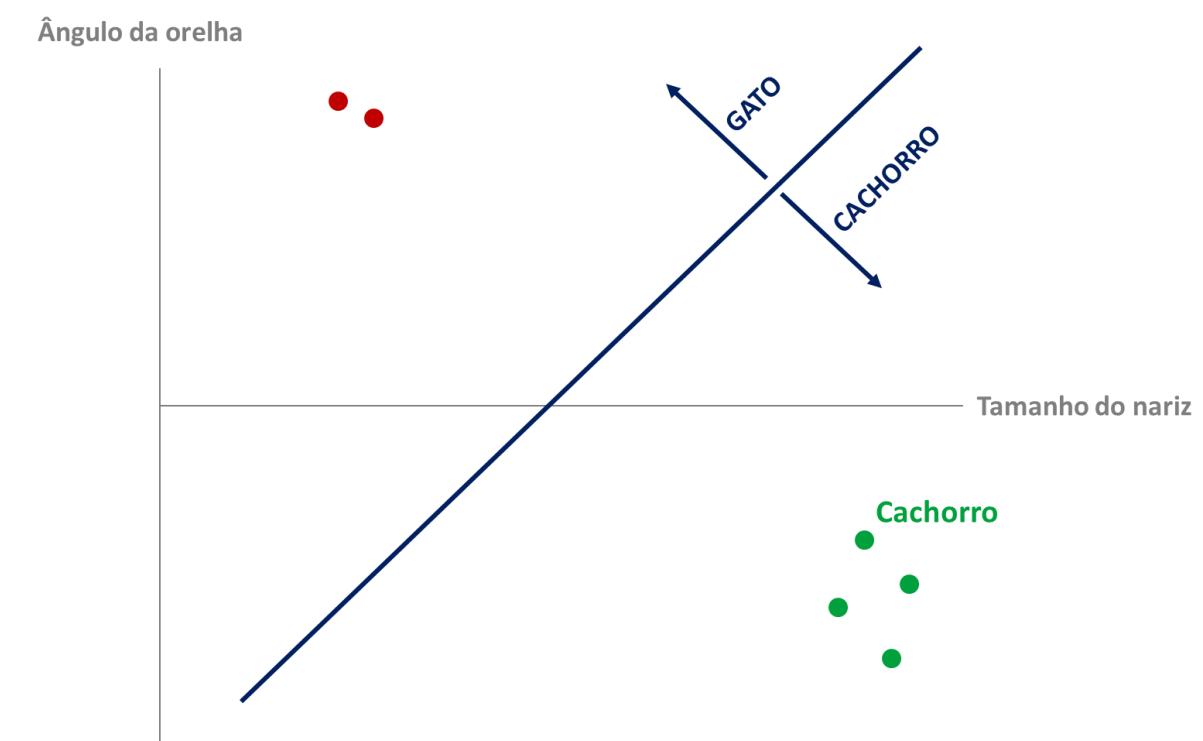
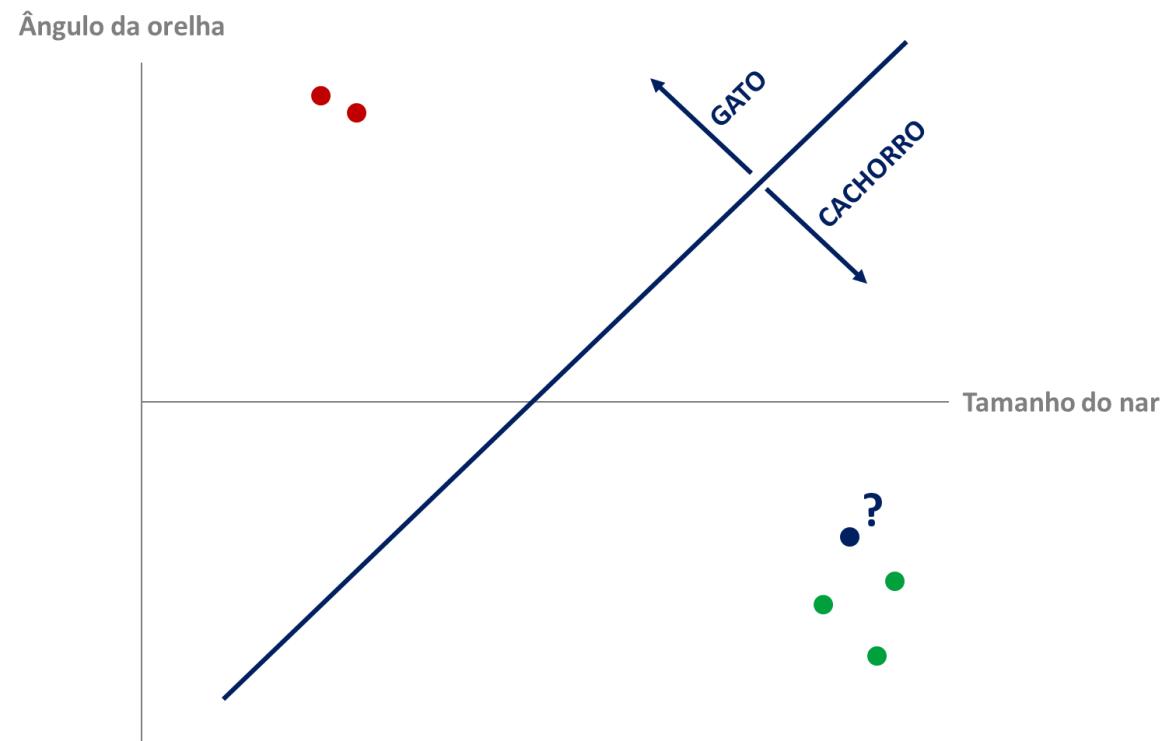


OS DADOS JÁ ESTÃO ROTULADOS!  
(eu já digo o que é gato e o que é cachorro)

APRENDIZADO  
SUPERVISIONADO

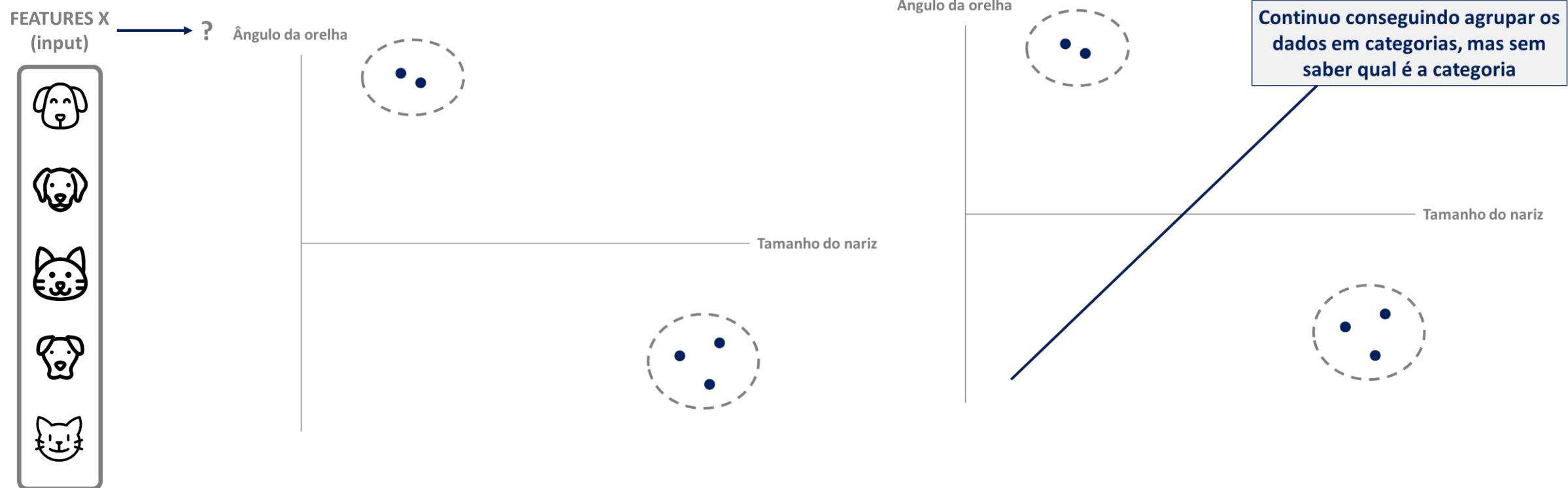
## Módulo 11 – Tipos de aprendizado – Aprendizado supervisionado

Neste caso, ao receber uma nova imagem, que não estava dentre as que foram usadas para treinar o modelo, o modelo reconhecerá os padrões na imagem e já será capaz de fazer a classificação:



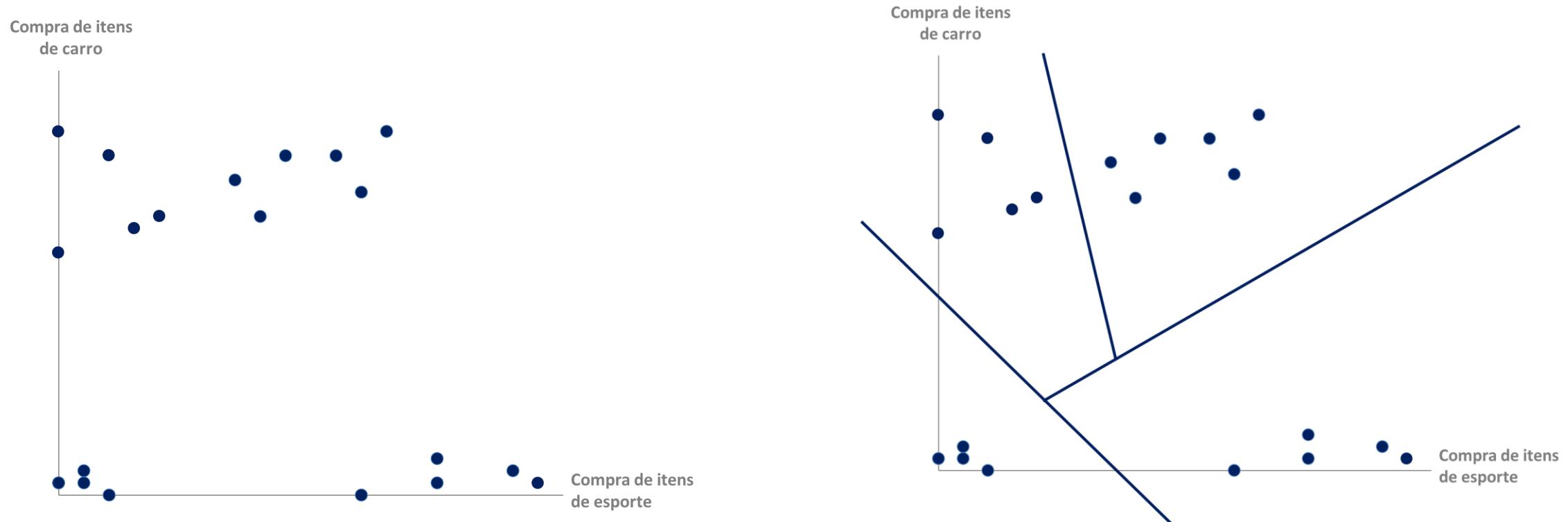
## Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Agora, vamos supor uma situação onde não passamos ao nosso modelo as classificações possíveis. O modelo ainda será capaz de reconhecer padrões e agrupar os dados. Mas não saberá o nome dos grupos.



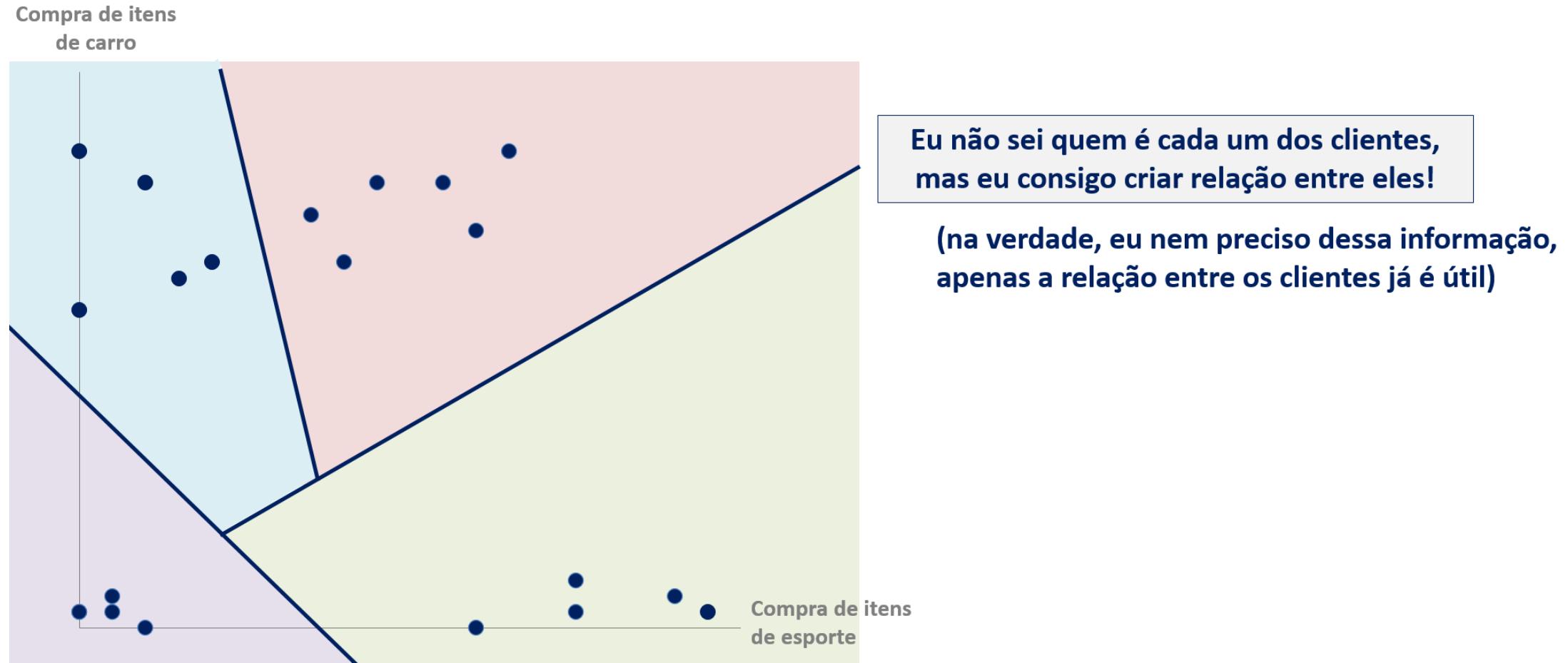
## Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Na vida real, não saber exatamente uma classificação é comum. Por exemplo, avaliando o perfil de compra de diversos clientes, em um primeiro momento podemos não saber como rotulá-los. E, na prática, o importante é saber reconhecer grupos de clientes para entender seus perfis e o que pode ser oferecido a cada grupo. Na figura abaixo, o modelo inicialmente dividiu em 4 grupos:



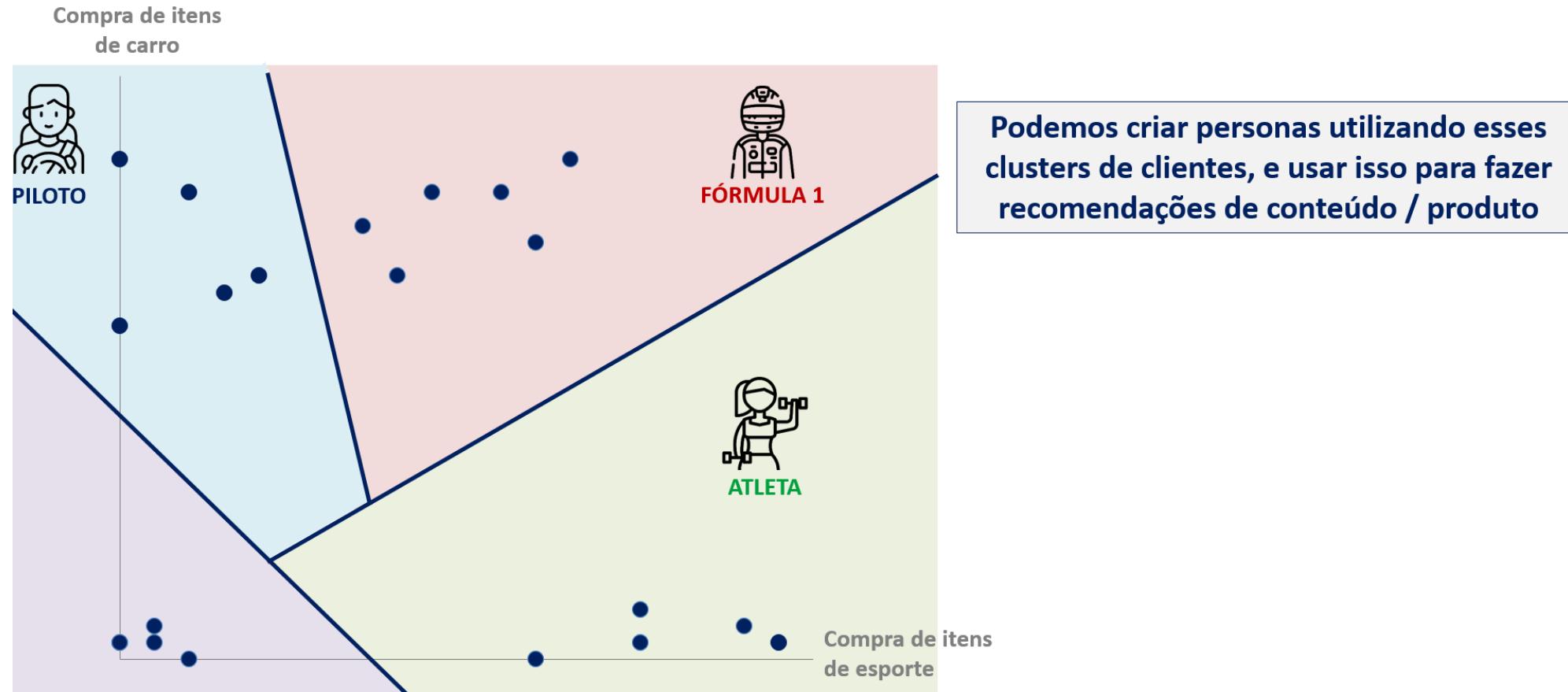
## Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Com esses grupos criados pelo modelo, podemos tentar entender as relações dentro de cada grupo



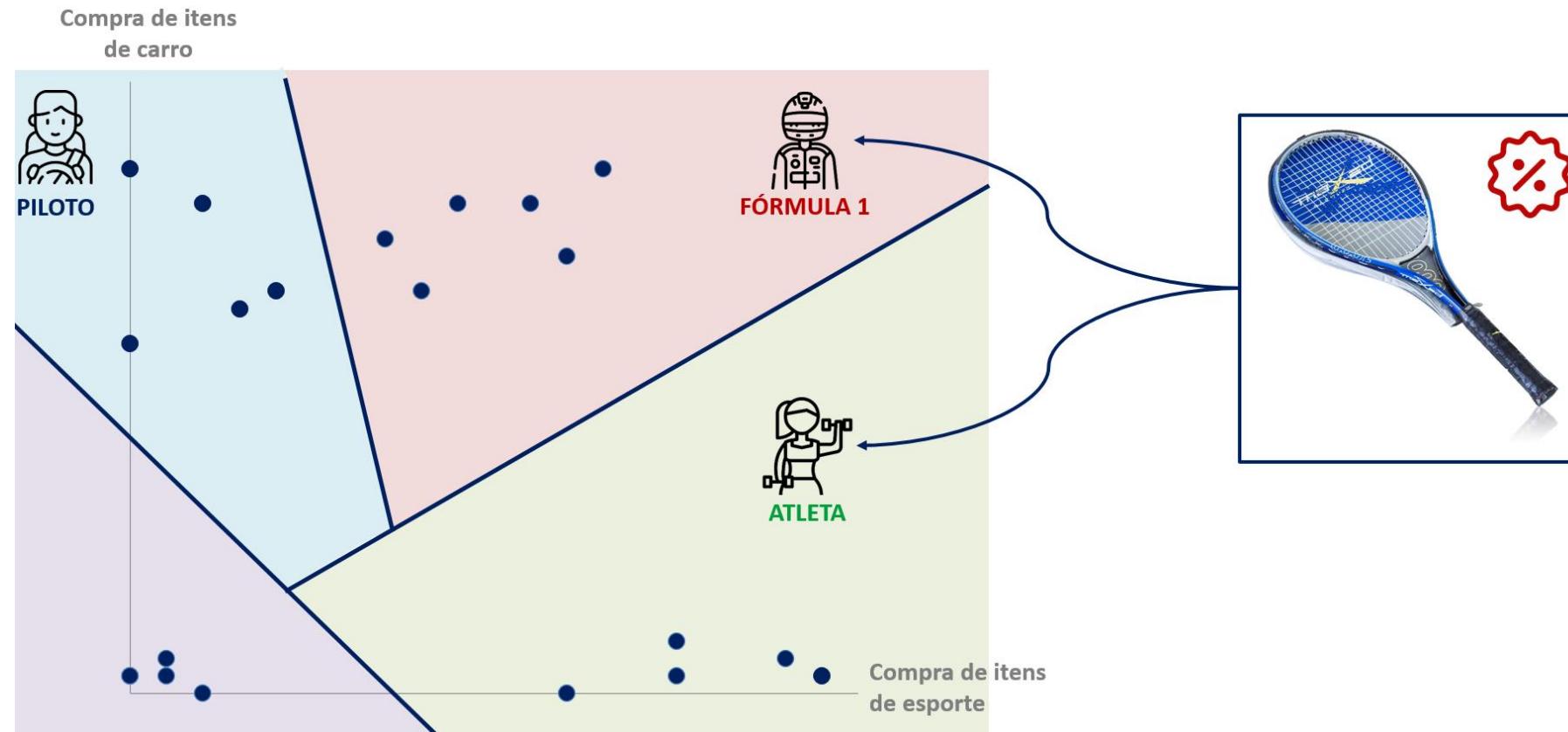
## Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Pode ser interessante associar pessoas a cada grupo, com base nas relações que vão sendo percebidas após o modelo ter feito o agrupamento:



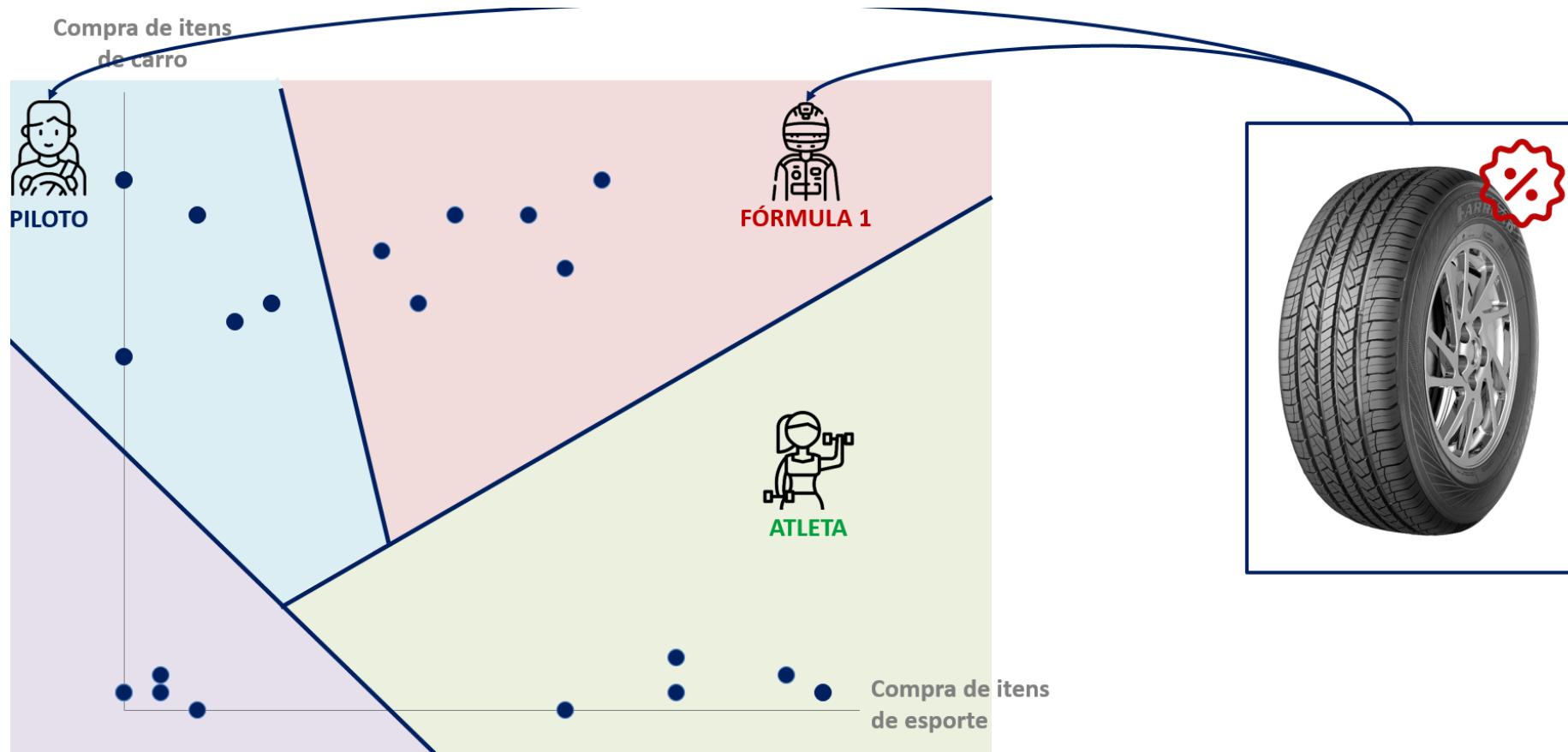
## Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Essas pessoas podem ajudar nas sugestões de produtos. Tanto um atleta quanto um piloto de Fórmula 1 podem se interessar por uma raquete, pelo perfil esportivo de ambos:



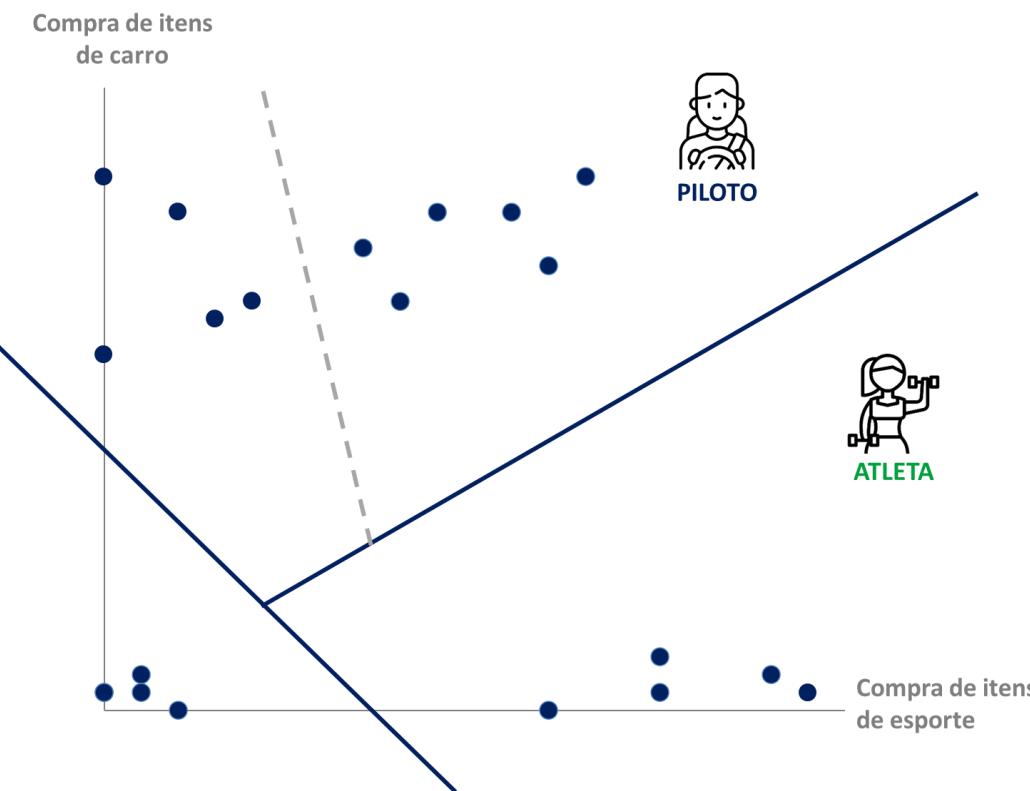
## Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Já um piloto de Fórmula 1 e um de outra categoria podem se interessar por pneus de alta performance:



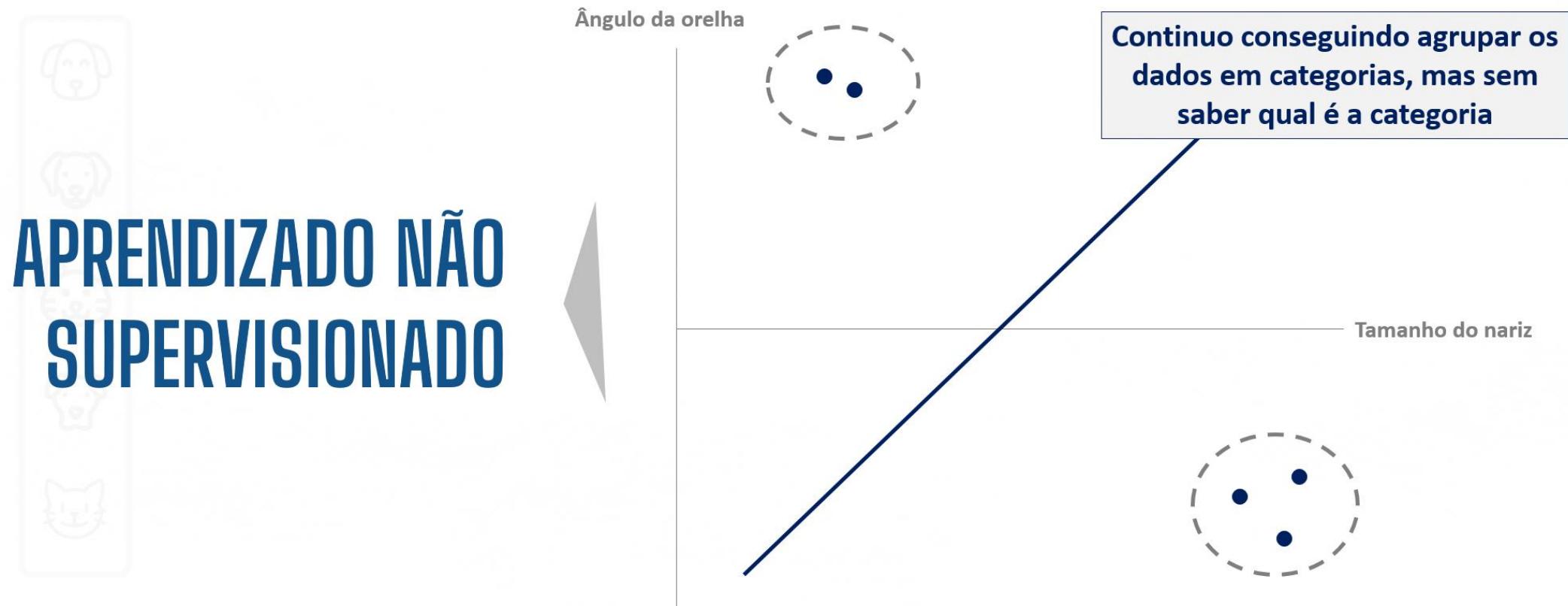
## Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Inclusive, podemos instruir nosso modelo a não separar estes dois últimos grupos, considerando apenas uma classificação única como “piloto”. Esta decisão pode ser tomada com base em testes, avaliando se há alteração na performance de indicações e vendas com base na modificação de grupos.



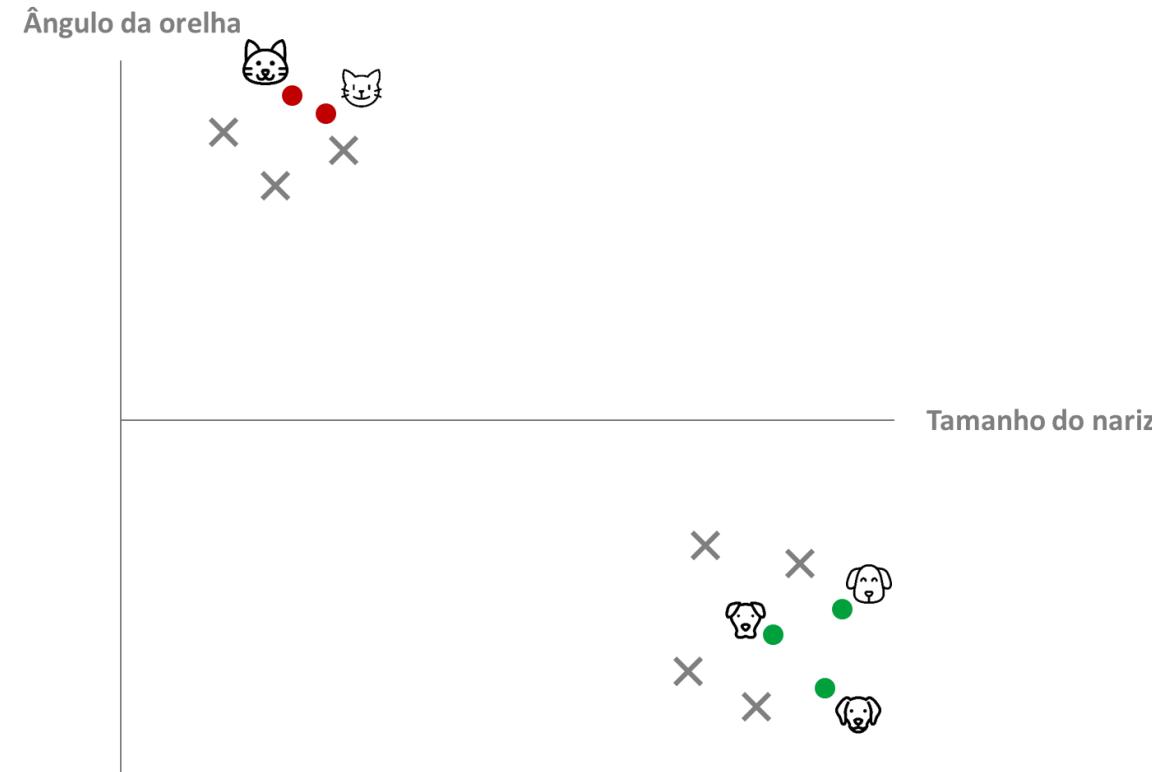
## Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Este exemplo é o que chamamos de **aprendizado não supervisionado**. Quando delegamos ao modelo a classificação, sem passar rótulos previamente.



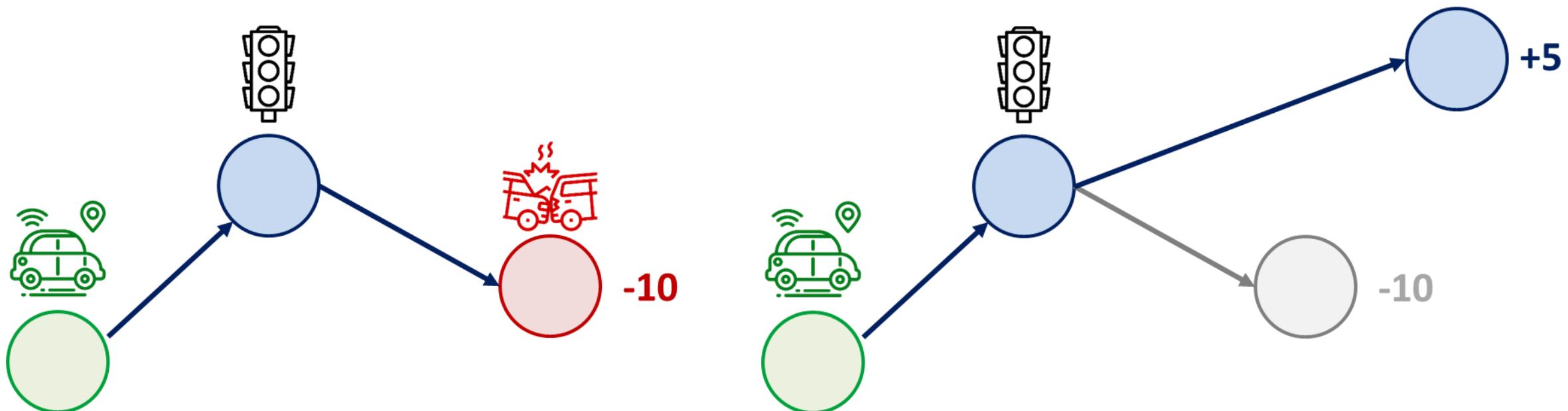
## Módulo 11 – Tipos de aprendizado – Aprendizado semi-supervisionado

No **aprendizado semi-supervisionado** temos alguns dados rotulados e outros não rotulados. Este tipo de aprendizado é comum em bases muito grandes. Por exemplo, de imagens, onde não se tem meios de rotular toda a base de treino. Rotula-se alguns e outros são deixados sem rótulo.



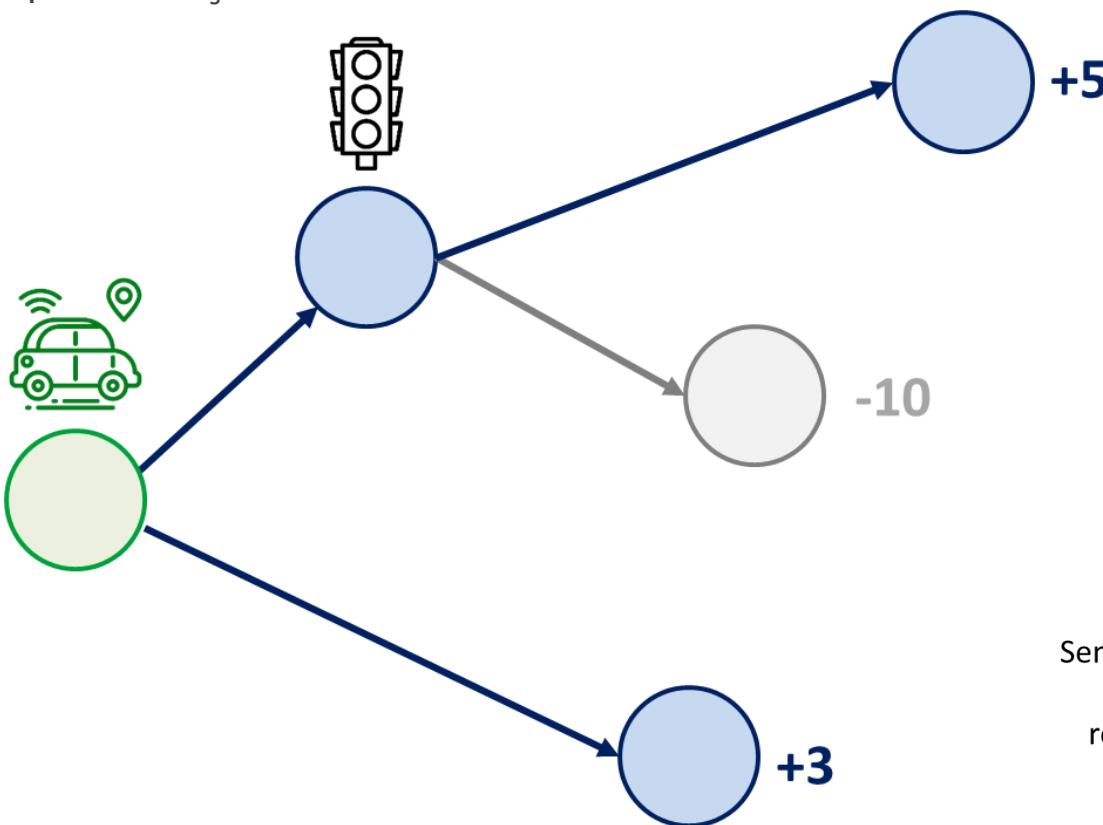
## Módulo 11 – Tipos de aprendizado – Aprendizado por reforço

O **aprendizado por reforço** é aquele que vai aprendendo por bonificação ou punição de acordo com escolhas. Por exemplo, um carro autônomo. Na figura, quando um carro atravessa o sinal vermelho, acaba batendo e isso é penalizado no modelo. Quando para no sinal e só avança no verde, sem acidentes, o modelo armazena como algo positivo.

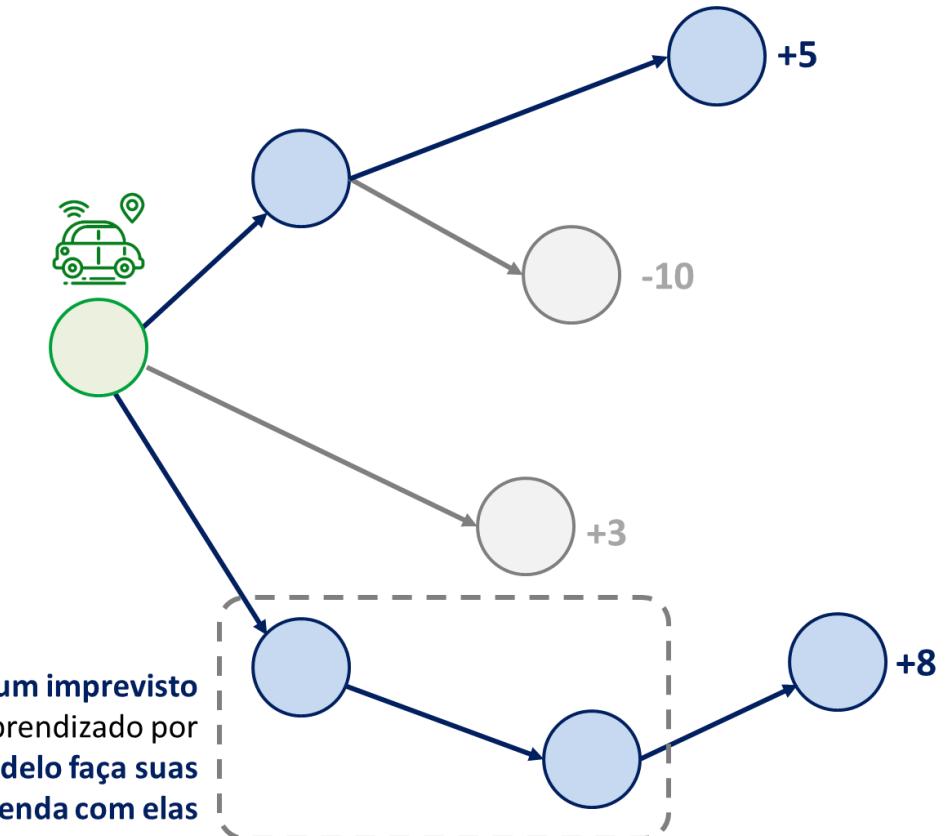


## Módulo 11 – Tipos de aprendizado – Aprendizado por reforço

Durante a fase de treinos, o modelo vai fazendo diversas escolhas, diversos caminhos, e sendo bonificado ou penalizado de acordo. O modelo treinado vai sempre buscar o caminho com maior pontuação.

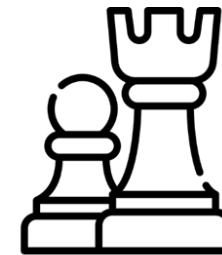


Sempre vai poder surgir **algum imprevisto** na estrada, então o aprendizado por reforço permite que **o modelo faça suas próprias escolhas e aprenda com elas**



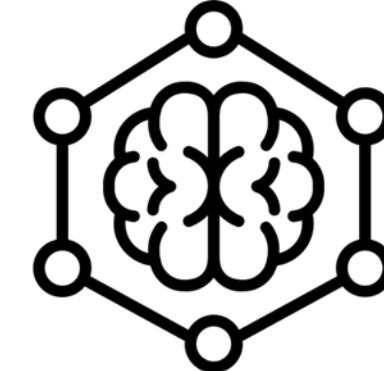
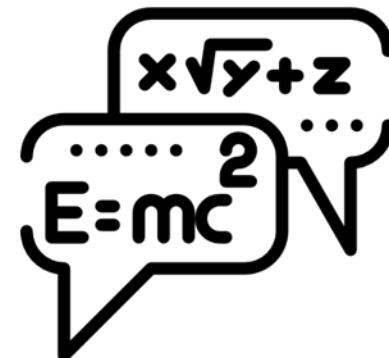
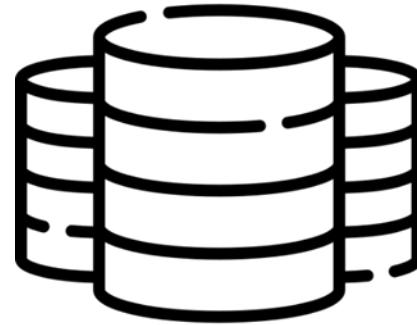
## Módulo 11 – Tipos de aprendizado – Aprendizado por reforço

Esse tipo de aprendizado é comum em jogos, como xadrez; em sistemas de determinação de rotas de trânsito; em sistemas de feed de notícias; e em programas de análise de mercados.



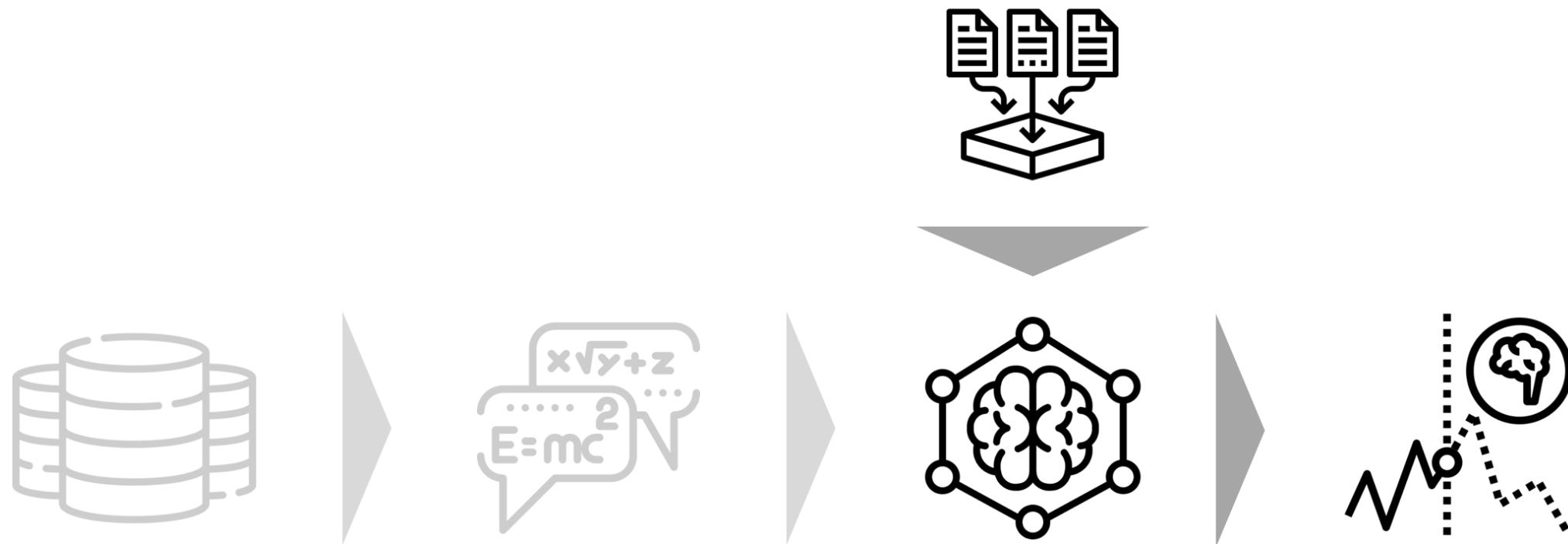
## Módulo 11 – Diferença entre aprender e decorar

Aprender é permitir que o nosso algoritmo **busque padrões nos dados** que poderão ser usados para **prever comportamentos futuros**.



## Módulo 11 – Diferença entre aprender e decorar

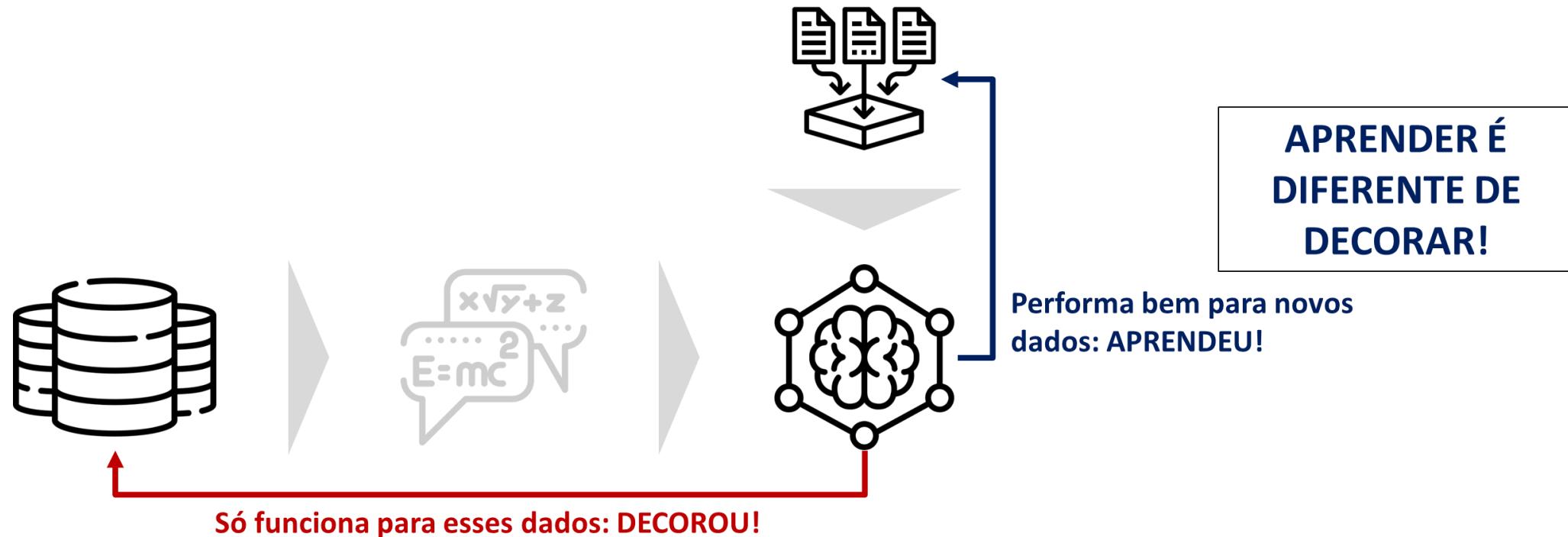
O que esperamos é que nosso modelo faça previsões corretas com novos dados. Ou seja, que os dados de treino tenham sido efetivos em criar um modelo que funcione bem com dados novos, que o modelo não tenha tido contato anteriormente.



## Módulo 11 – Diferença entre aprender e decorar

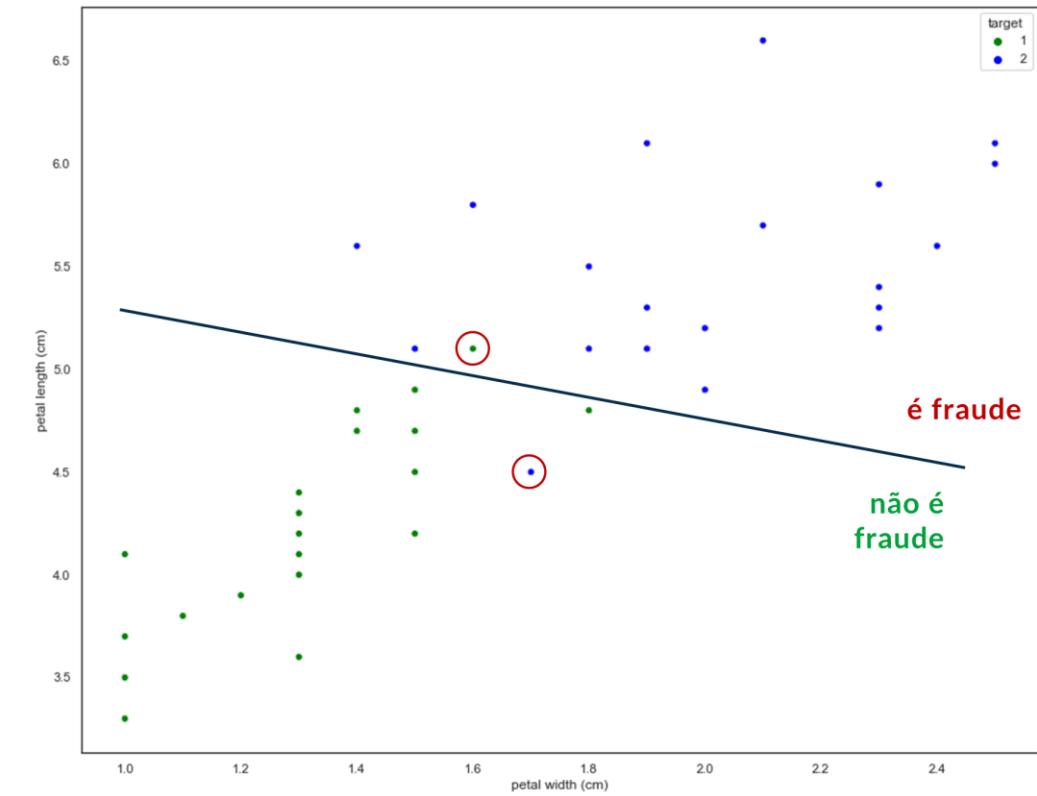
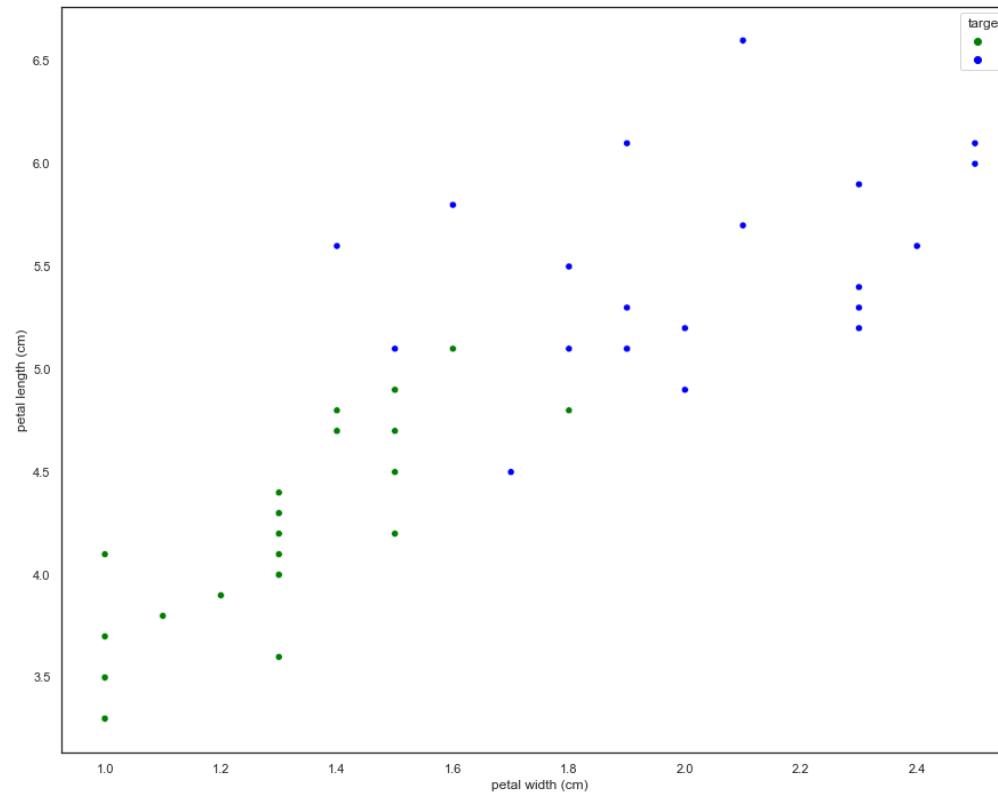
No entanto, pode ocorrer que o modelo performe bem na base de testes, mas não tenha boa performance com novos dados.

É como se o modelo tivesse “decorado” os dados de teste. Vamos aprender como isso pode ocorrer e como podemos evitar estas situações.



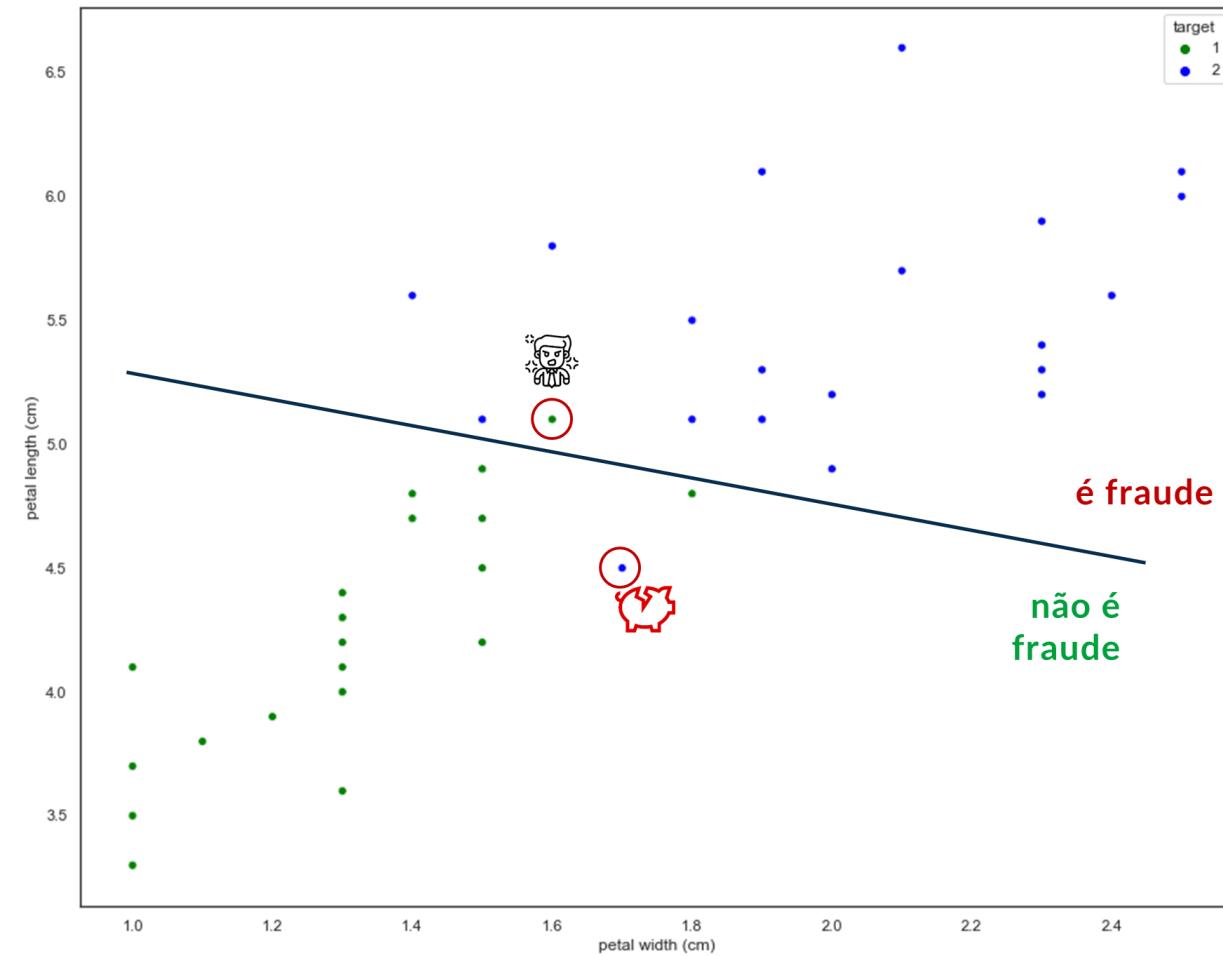
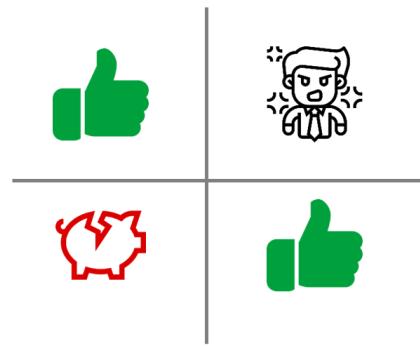
## Módulo 11 – Diferença entre aprender e decorar

Vamos nos lembrar do exemplo da fraude do módulo anterior. Suponha que um dado modelo tenha criado a seguinte reta de separação das classificações. Repare que há um caso de falso positivo e um de falso negativo.



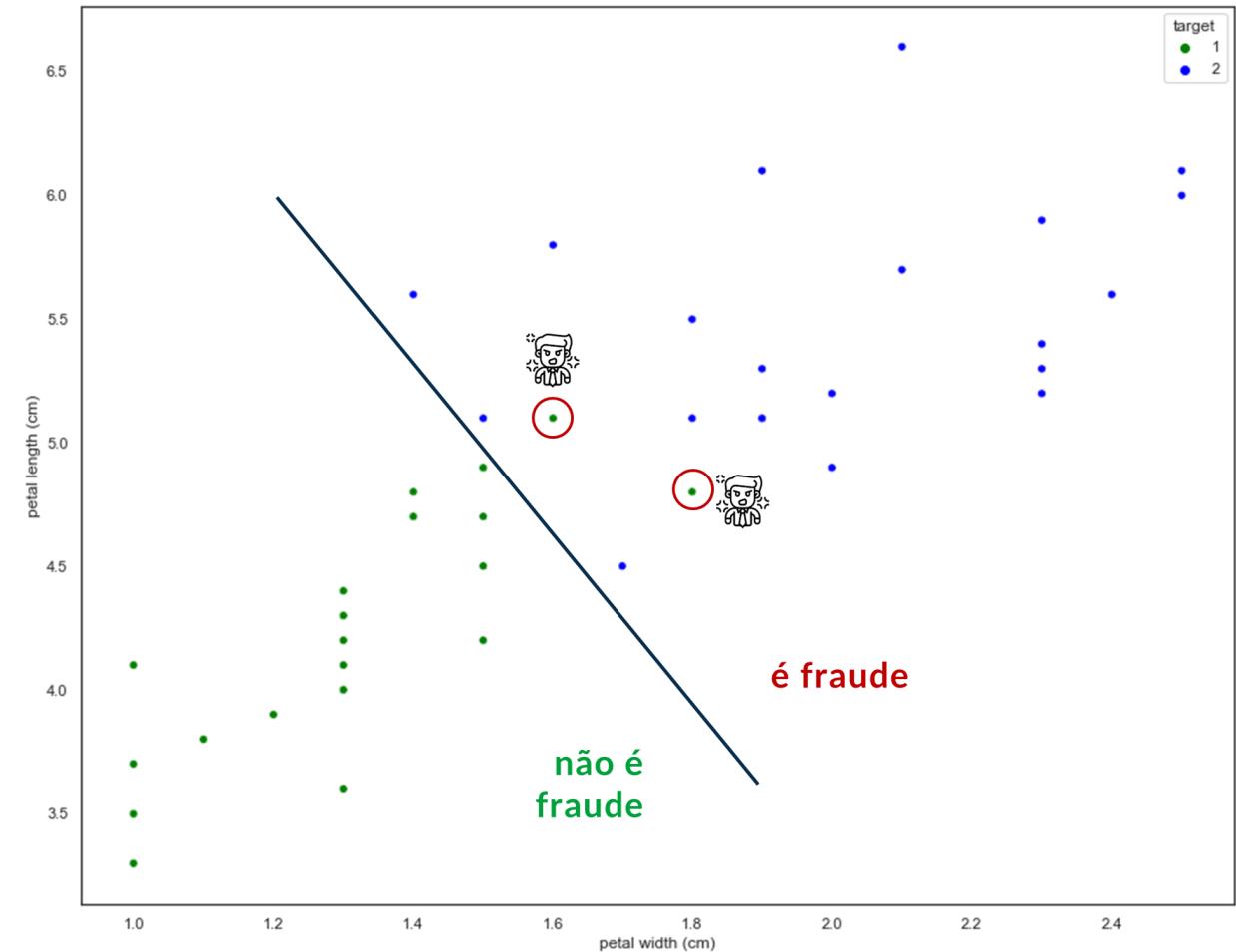
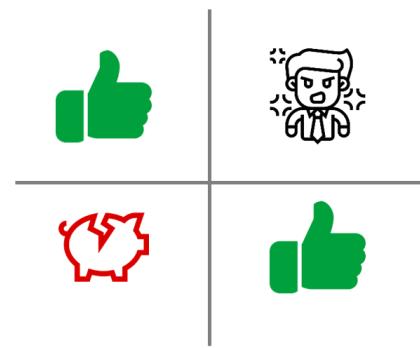
## Módulo 11 – Diferença entre aprender e decorar

Lembre-se que estas classificações erradas são prejudiciais, pois podem levar a clientes insatisfeitos e a perdas financeiras para a empresa, como já vimos no módulo anterior.



## Módulo 11 – Diferença entre aprender e decorar

Uma escolha que pode ser feita é minimizar algum dos erros. Por exemplo, a empresa pode achar mais interessante não ter perdas financeiras a custa de eventualmente ter mais problemas com clientes. Veja que esta escolha vai acabar mudando a reta no gráfico.

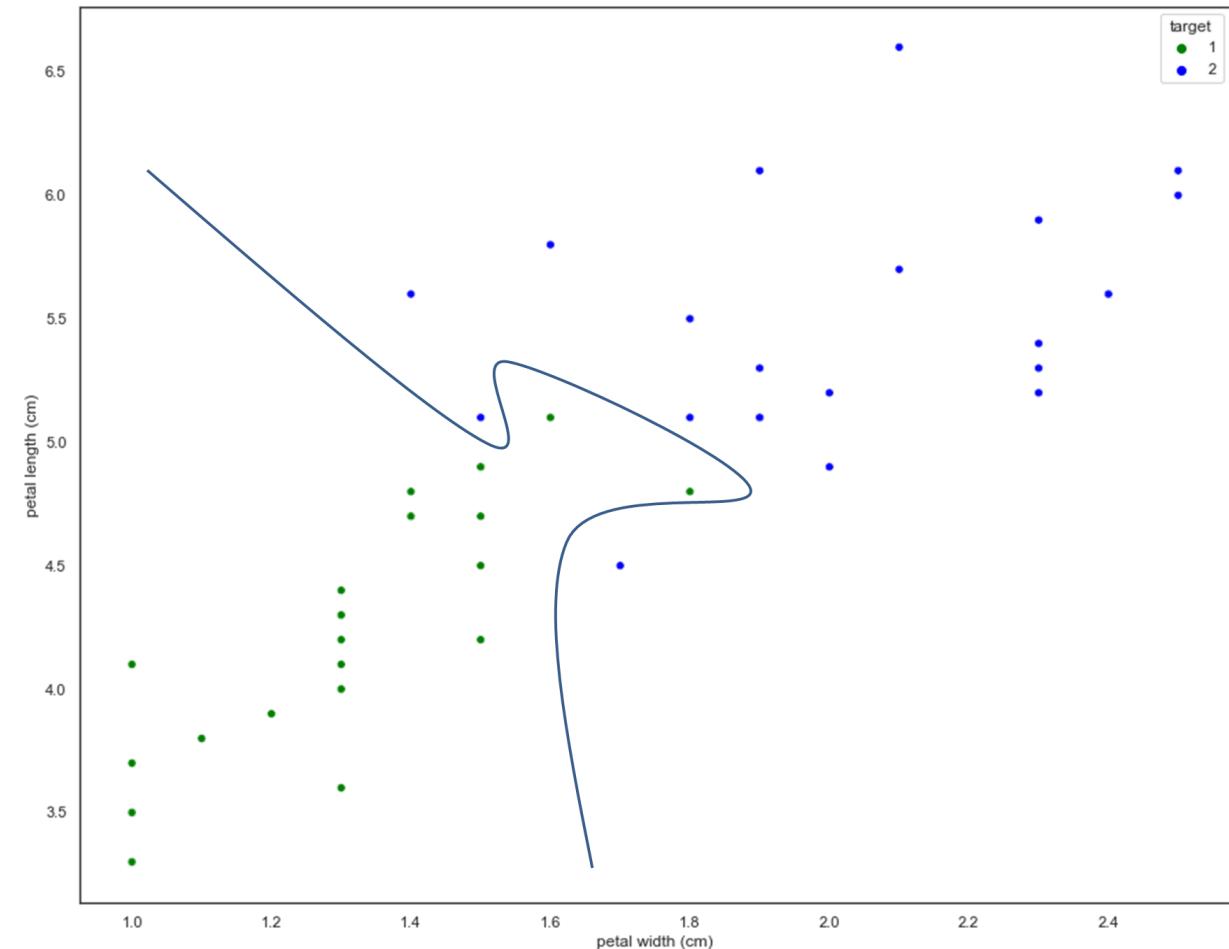


## Módulo 11 – Diferença entre aprender e decorar

Agora, suponha que se busque uma situação super ideal, onde há 100 % de acerto do modelo. Toda vez que ver 100 % de acerto, fique atento!

Veja que, teoricamente, nosso modelo é perfeito, criou uma curva que separa perfeitamente os casos de fraude dos que não são fraude. Mas, lembre-se, o modelo foi criado com os dados de treino, os dados prévios.

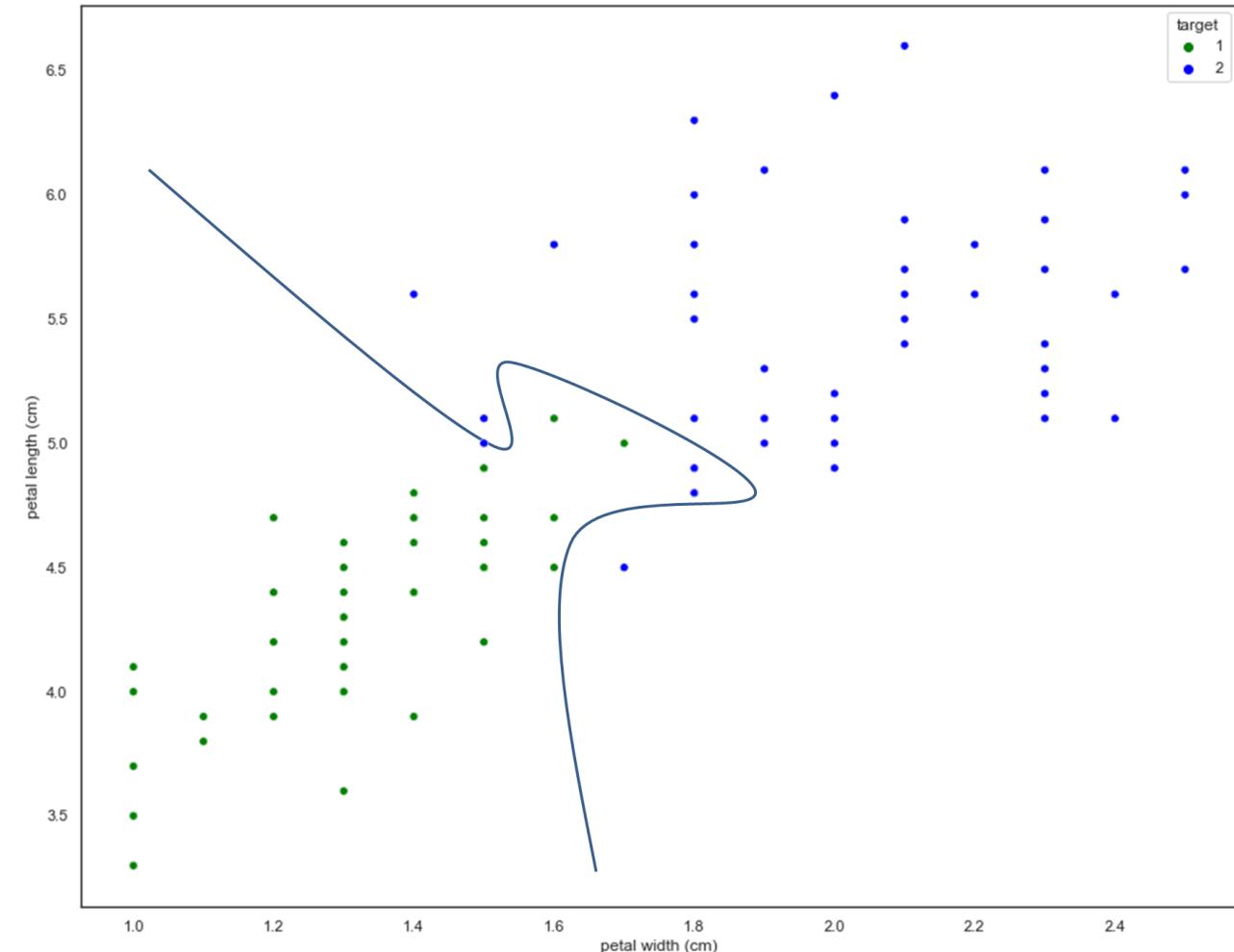
E o que pode ocorrer quando novos casos reais forem apresentados ao modelo?



## Módulo 11 – Diferença entre aprender e decorar

Como deixamos nosso modelo super otimizado para os dados de treino, com uma curva bem específica para aqueles dados, ao ser apresentado a novos dados ele acabou errando muito.

Perceba na imagem, pela coloração dos pontos, que temos muito mais erros agora do que com as duras retas que vimos anteriormente.

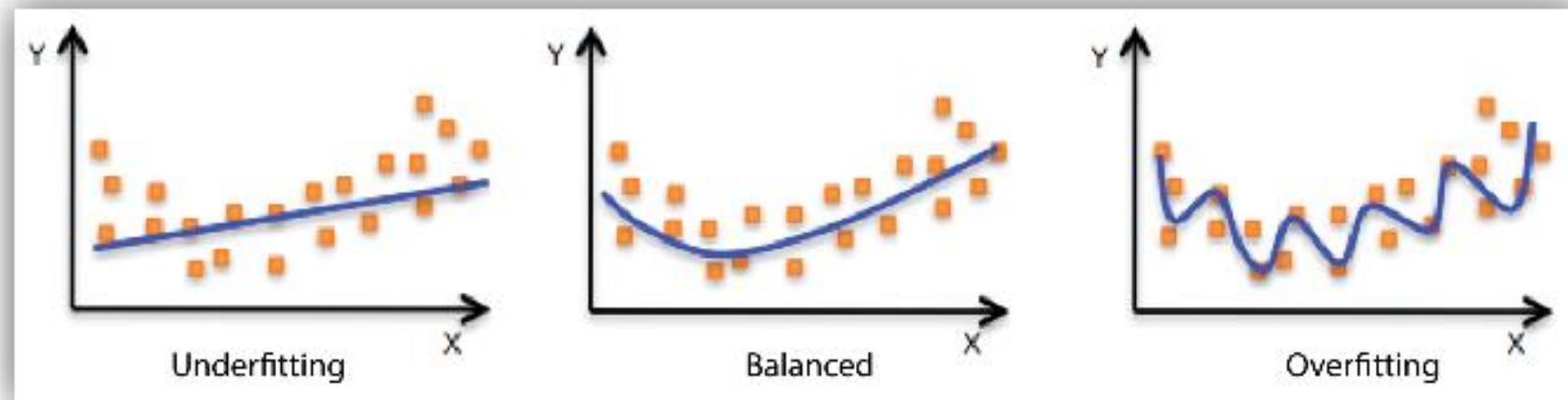


## Módulo 11 – Diferença entre aprender e decorar

Quando nosso modelo está muito mal ajustado aos dados, temos um caso de **underfitting**.

Um modelo que tenta passar por todos os pontos, tenta chegar em uma situação perfeita, caracteriza um caso de **overfitting**. O que ocorreu nas últimas páginas.

O ideal é que nosso modelo seja balanceado. Por mais que eventualmente erre, evita erros grosseiros dos outros extremos. E veremos durante o curso como podemos estimar uma margem de erro aceitável.



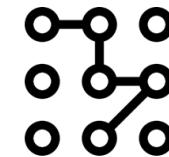
## Módulo 11 – Considerações importantes

Machine Learning sempre vai ser a melhor alternativa?

**NÃO.** Machine Learning será útil se:



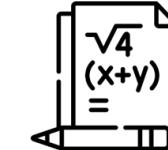
Temos **dados** disponíveis



Existe um **padrão** nos dados



O **problema a ser resolvido** está bem definido



**Não existe uma fórmula matemática** para determinar o resultado

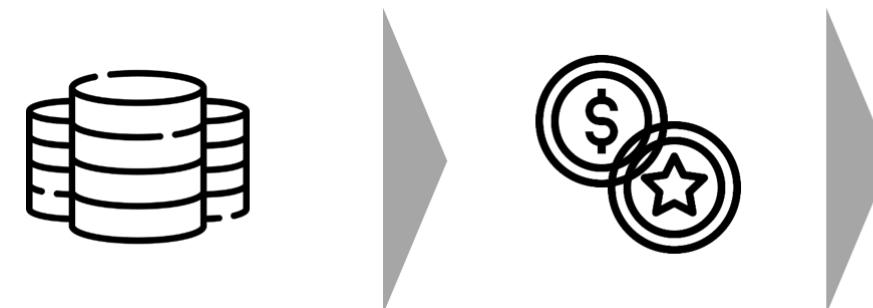
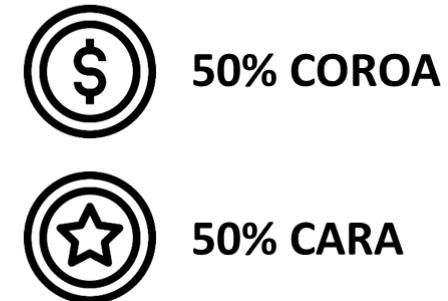
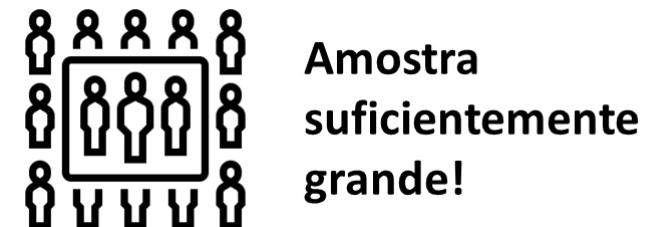
## Módulo 11 – Considerações importantes

Para entender a importância de dados disponíveis, vamos para um exemplo bem simples: lançamento de moeda.

Sabemos que a probabilidade é de 50 % para cara e para coroa. No entanto, pode ser que em 10 lançamentos, 7 sejam cara e 3 sejam coroa. Se nosso modelo for treinado com apenas esses 10 lançamentos, terá resultados completamente errados.

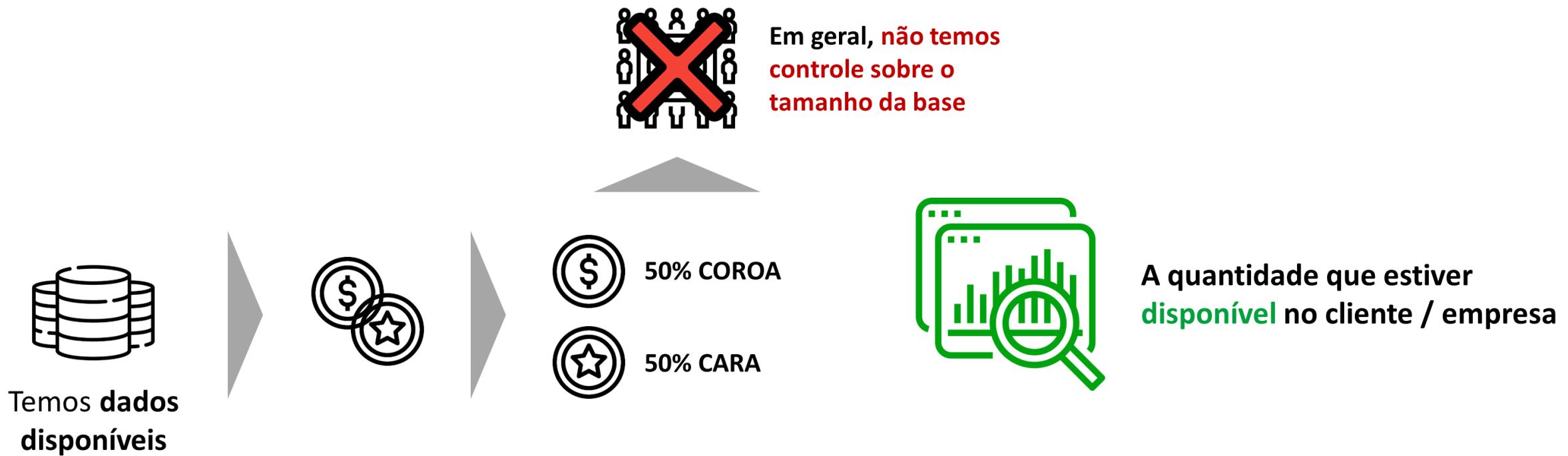
A tendência é que, com cada vez mais lançamentos, a proporção se aproxime da probabilidade esperada.

A mesma linha de raciocínio se aplica a dados reais. Precisamos de uma quantidade de dados significativa para que padrões sejam percebidos corretamente.



## Módulo 11 – Considerações importantes

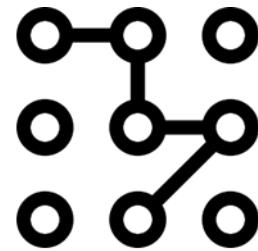
Agora, por vezes temos uma limitação no tamanho da base. Por exemplo, se estivermos avaliando o comportamento de vendas de uma empresa, não temos como aumentar a quantidade de vendas do passado. O máximo que podemos fazer é informar ao cliente que a quantidade pode não ser a ideal para uma boa previsão e estipular uma margem de erro ou mudar a forma de estudar o comportamento.



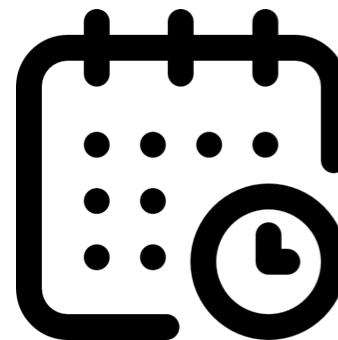
## Módulo 11 – Considerações importantes

Já vimos que é necessário haver um padrão para treinar modelos. Mas quando parar de melhorar o modelo?

**Depende!** Você, como cientista, precisa conseguir **ponderar o tempo gasto** para implementar uma melhoria no seu algoritmo com o **resultado** que ela vai trazer!

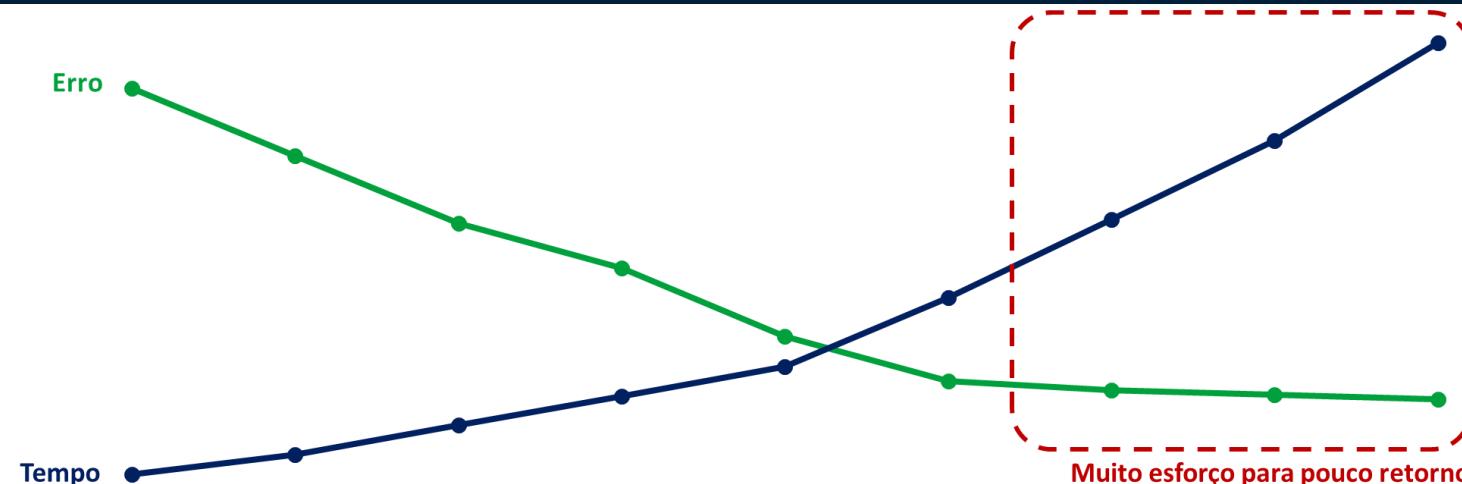


Existe um **padrão**  
nos dados



## Módulo 11 – Considerações importantes

Perceba, nesse exemplo ilustrativo, que diminuir o erro a partir de um determinado momento, passa a levar muito tempo e pode não ser viável economicamente.



	ALGORITMOS	1	1	1	2	2	3	4	5	6
	TRATAMENTO DE DADOS	N	S	S	S	S	S	S	S	S
	SELEÇÃO DE VARIÁVEIS	N	N	S	S	S	S	S	S	S
	AJUSTE DE PARÂMETROS	N	N	N	N	S	S	S	S	S

Meramente ilustrativo, apenas para fins didáticos

## MÓDULO 12

# Criando um modelo de classificação

## Módulo 12 – Entendendo o dataset iris

Neste módulo, utilizaremos a famosa base de dados Íris. Esta consiste de 50 amostras de cada espécie da planta Íris, com dados de comprimento e largura das pétalas e sépalas como mostrado na imagem:

### Base de dados das Flores de Íris

*Iris flower dataset*

Setosa



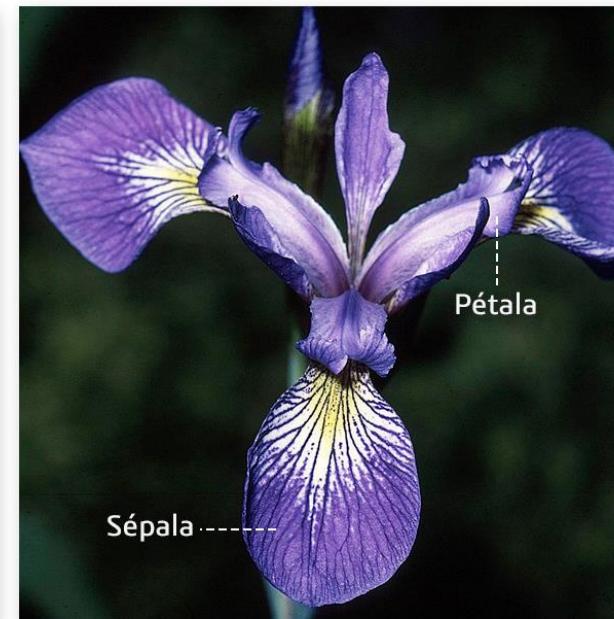
Tia Monto. CC BY-SA 4.0, via Wikimedia Commons

Versicolor



Charles de Mille-Isles from Mille-Isles, Canada. CC BY 2.0, via Wikimedia Commons

Virginica



Robert H. Mohlenbrock. Courtesy of USDA NRCS. Public domain, via Wikimedia Commons

https://pt.wikipedia.org/wiki/Iris



## Módulo 12 – Entendendo o dataset iris

A base de dados pode ser importada diretamente do Scikit-Learn, que possui um conjunto de bases via `sklearn.datasets`.

Vamos aproveitar e importar as demais bibliotecas que usaremos: Pandas, NumPy e Matplotlib.

E carregar a base de dados na variável `data`.

```
# Importando as bibliotecas (pandas, matplotlib, numpy)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# Importando o dataset iris
from sklearn.datasets import load_iris
```

```
# Obtendo os dados do dataset
data = load_iris()
```

# Módulo 12 – Entendendo o dataset iris

Por padrão, o Scikit-Learn armazena as bases de dados como arrays do NumPy. No caso em questão, um array para os atributos, que estamos chamando de `X` desde os últimos módulos; e um array para nossa variável alvo, o `y` dos últimos módulos. Podemos acessar com os atributos `data` e `target` de nossa variável:

# Valores de X  
data.data

# Valores de Y  
data.target

## Módulo 12 – Utilizando o Pandas

Seria mais conveniente utilizar objetos do Pandas. Assim, vamos utilizar o construtor `DataFrame` e instruir que as colunas tenham os nomes corretos, armazenados na variável `data` e que podem ser consultados com o atributo `feature_names`:

```
# Transformando em um DataFrame  
iris = pd.DataFrame(data.data)  
iris.head()
```

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
# Nome das colunas  
iris.columns = data.feature_names  
iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

## Módulo 12 – Utilizando o Pandas

A variável alvo também pode ser colocada no dataframe, na última coluna:

```
# Adicionando a coluna target ao DataFrame  
iris['Target'] = data.target  
iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

## Módulo 12 – Utilizando o Pandas

Como discutido no início do módulo, a base de dados apresenta 50 amostras de cada uma das espécies da planta. Podemos verificar isto com o método `value_counts()` na coluna da variável alvo `Target`.

Para facilitar a compreensão, vamos nos restringir a apenas duas das espécies, retirando a com classificação “2”.

```
# Contando a quantidade de cada um dos Targets  
iris['Target'].value_counts()
```

```
0    50  
1    50  
2    50  
Name: Target, dtype: int64
```

```
# Retirando Target = 2  
iris = iris[iris.Target != 2]
```

```
# Contando novamente os valores  
iris['Target'].value_counts()
```

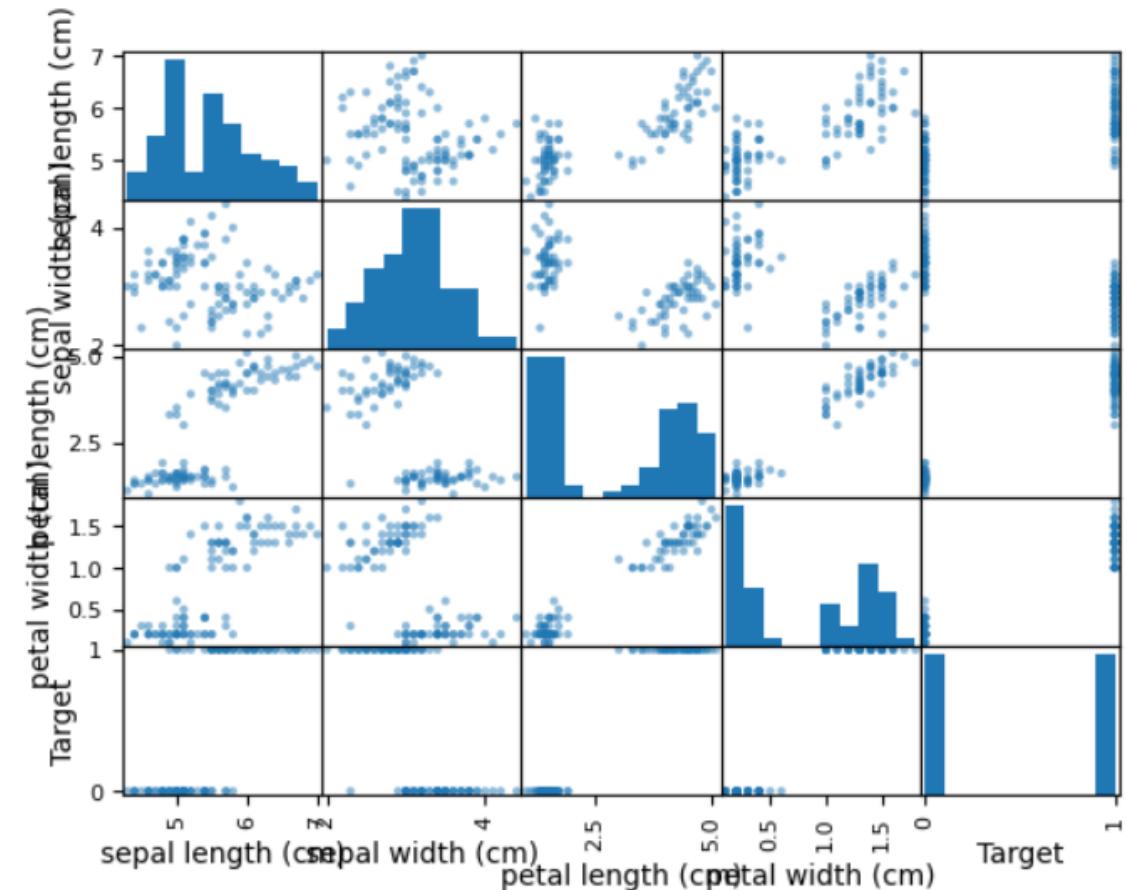
```
0    50  
1    50  
Name: Target, dtype: int64
```

## Módulo 12 – Visualizando os dados

Agora que já arrumamos os dados, vamos buscar entender como as variáveis se relacionam. Uma forma é através de matrizes de distribuição, como no Pandas `scatter_matrix`, que mostra a relação entre as variáveis numéricas.

Na diagonal da matriz, vemos a distribuição de valores da cada variável na forma de histograma. Perceba, por exemplo, que a última posição da matriz mostra que há apenas dois valores na coluna Target, com iguais quantidades.

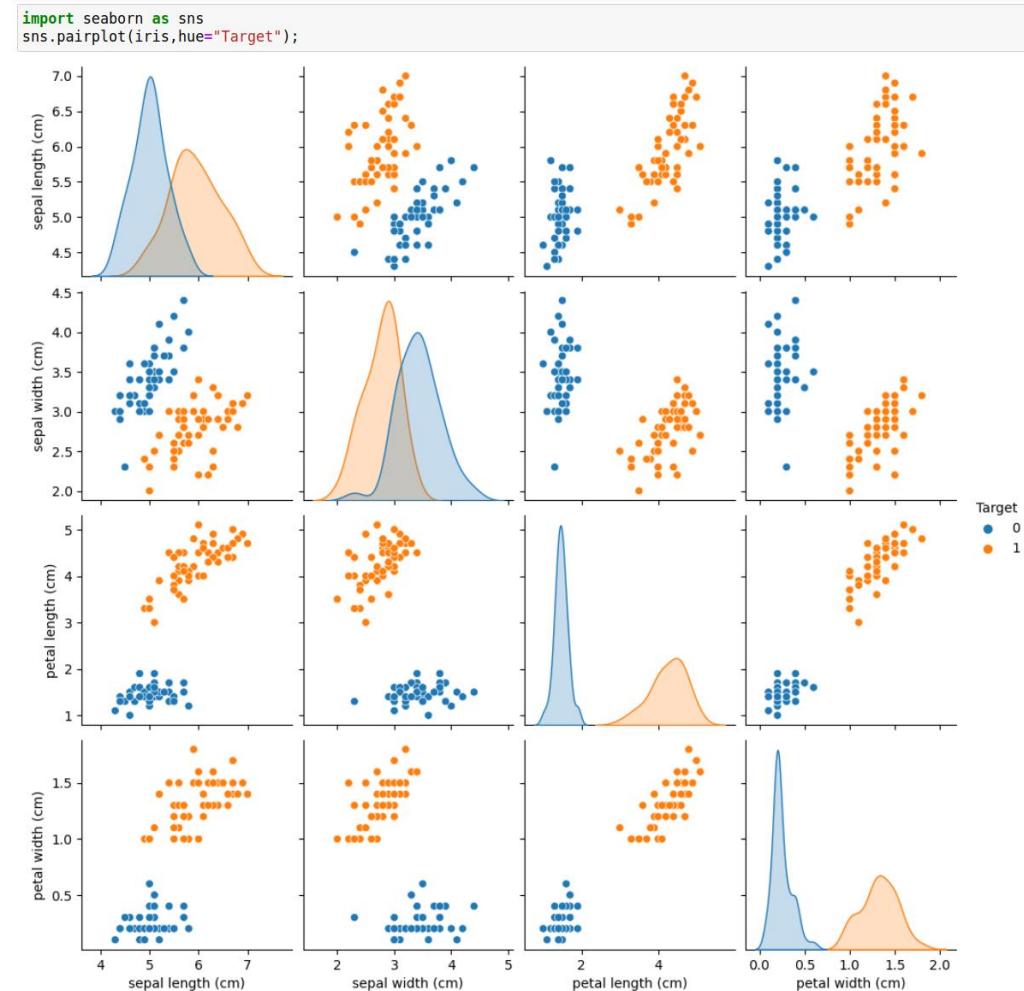
```
pd.plotting.scatter_matrix(iris);
```



## Módulo 12 – Visualizando os dados

O `scatter_matrix` do Pandas é muito útil, mas há uma ferramenta com visualizações ainda melhores e personalizáveis, o `pairplot` do Seaborn.

Em um problema de classificação, é útil verificar as distribuições comparando cada classe. Veja como se passou a coluna `Target` para o parâmetro `hue`. Com isto, conseguimos verificar como as variáveis se relacionam já destacando cada classificação. Perceba como os pontos de cada classe são bem distinguíveis.

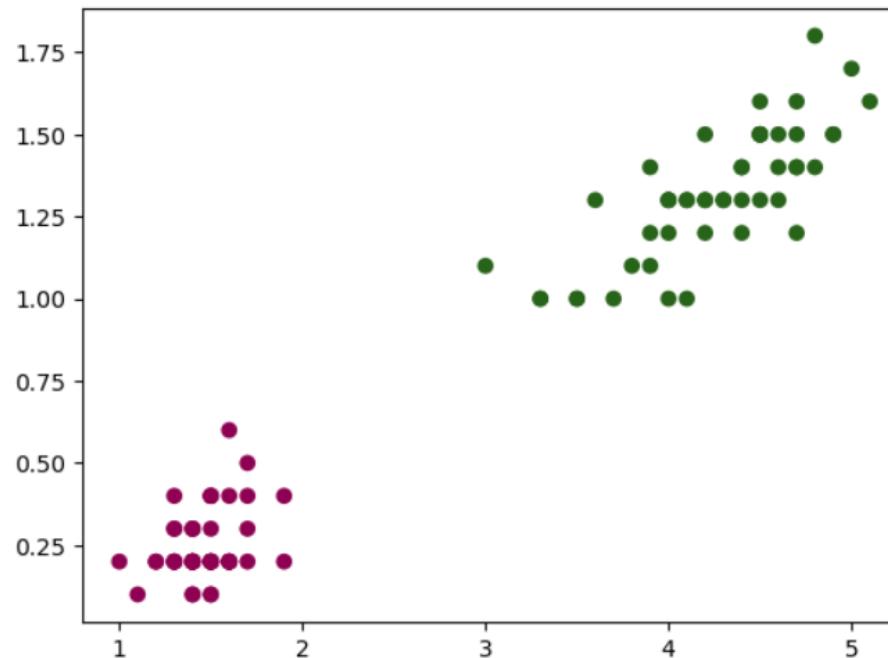


## Módulo 12 – Visualizando os dados

Vamos pegar um par de atributos, no caso comprimento e largura das pétalas, para olhar mais atentamente. Podemos utilizar o `scatter` do Matplotlib, passando a coluna `Target` para usar cores distintas para cada classificação. O `PiYG` é uma escala de cores, que estamos utilizando apenas para que tenham conhecimento que há escalas de cores pré-definidas no Matplotlib e que podem ser consultadas em:

<https://matplotlib.org/stable/tutorials/colors/colormaps.html>

```
# Traçando um scatter do matplotlib das colunas "petal length (cm)" e "petal width (cm)"
fig, ax = plt.subplots()
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')
plt.show()
```



## Módulo 12 – Criando uma reta capaz de separar os dados do modelo

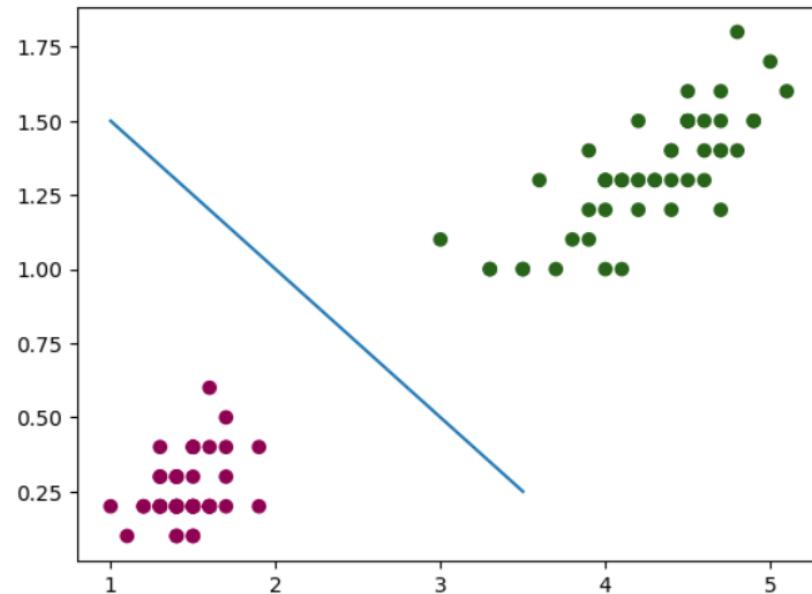
Assim como já vimos em módulos anteriores, podemos traçar uma reta que separa as duas classes. No código ao lado, criamos uma reta manualmente, com base em critérios visuais:

```
# Traçando o scatter plot e a reta
fig, ax = plt.subplots()

# Aqui estamos fazendo um gráfico de dispersão
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')

# Aqui estamos fazendo um gráfico de linha
x_reta = [1,3.5]
y_reta = [1.5,0.25]
ax.plot(x_reta,y_reta)

plt.show()
```



## Módulo 12 – Criando uma reta capaz de separar os dados do modelo

No entanto, várias retas poderiam ser utilizadas. Como vemos, qualquer uma das retas a seguir separa as duas classes:

```
# Buscando por melhores retas
fig, ax = plt.subplots()

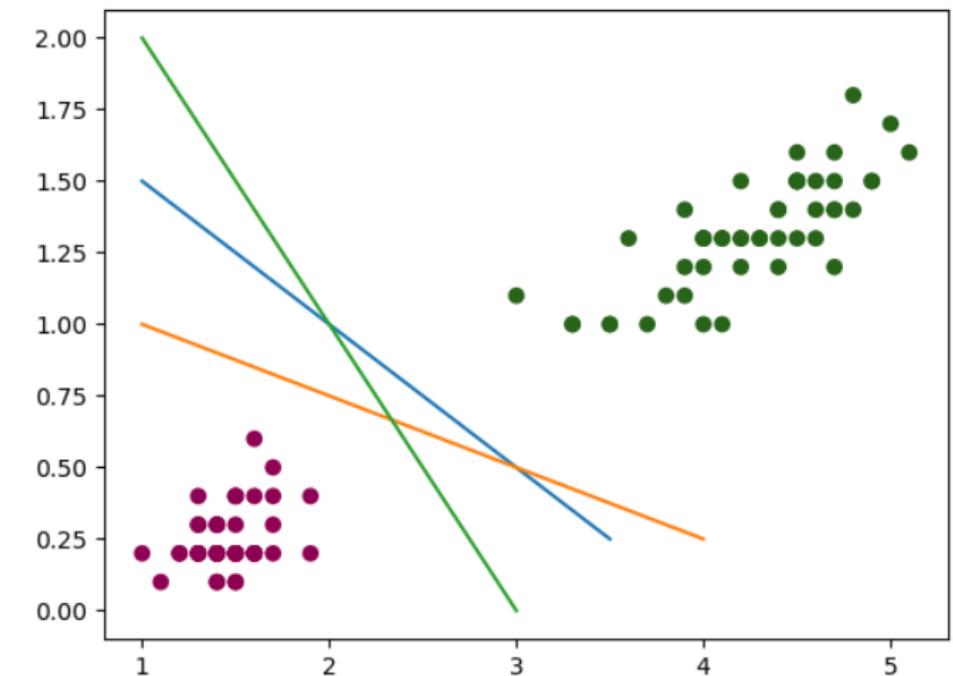
# Aqui estamos fazendo um gráfico de dispersão
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')

# Aqui estamos fazendo um gráfico de linha
x_reta = [1,3.5]
y_reta = [1.5,0.25]
ax.plot(x_reta,y_reta)

# Aqui estamos fazendo um gráfico de linha
x_reta2 = [1,4]
y_reta2 = [1,0.25]
ax.plot(x_reta2,y_reta2)

# Aqui estamos fazendo um gráfico de linha
x_reta3 = [1,3]
y_reta3 = [2,0]
ax.plot(x_reta3,y_reta3)

plt.show()
```



## Módulo 12 – Criando uma reta capaz de separar os dados do modelo

Vamos pegar uma dessas retas e alongá-la um pouco mais. E criar dois pontos, com cores vermelha e azul.

Perceba que o ponto vermelho está abaixo da nossa reta, mais próximo a um dos grupos de pontos, de cor roxa.

O ponto azul está levemente acima da reta. Como o objetivo da reta é separar as classes, esse ponto será classificado como pertencente ao grupo dos pontos verdes, mesmo estando relativamente distante destes. Daí a importância de se ter uma boa reta.

```
# Traçando novamente a reta e adicionando o ponto que acima  
fig, ax = plt.subplots()
```

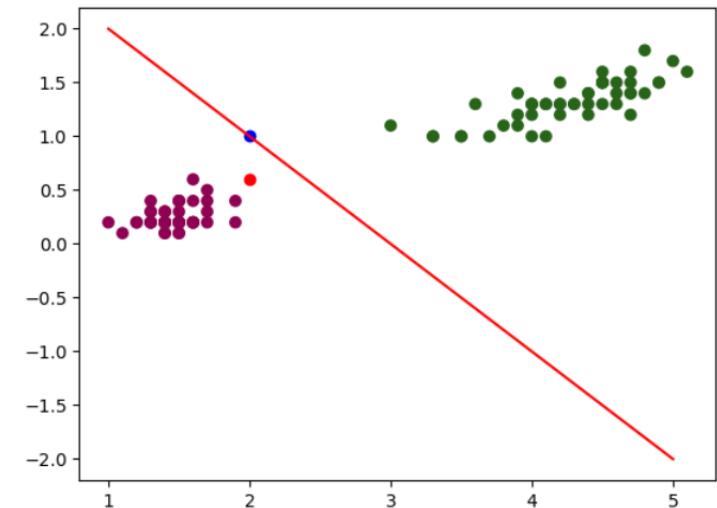
```
# Aqui estamos fazendo um gráfico de dispersão  
x = iris["petal length (cm)"]  
y = iris["petal width (cm)"]  
ax.scatter(x,y,c=iris.Target,cmap='PiYG')
```

```
# Aqui estamos fazendo um gráfico de linha  
x_reta3 = np.arange(1,6)  
y_reta3 = (-1)*x_reta3+3  
ax.plot(x_reta3,y_reta3,c='r')
```

```
# Fazendo o gráfico de dispersão somente desse ponto  
ax.scatter(2,+0.6,c='r')
```

```
# Fazendo o gráfico de dispersão somente desse ponto  
ax.scatter(2,+1,c='b')
```

```
plt.show()
```



## Módulo 12 – Criando uma função classificadora

O “acima” e “abaixo” da página anterior se refere ao valor da ordenada do ponto, o  $y$ , que pode ser maior ou menor que o  $y$  da reta para um determinado valor de  $x$ .

Assim, podemos criar uma função que classifica um par  $(x, y)$  verificando se tal ponto está acima ou abaixo de uma dada reta. A reta da página anterior tem equação  $y = -x + 3$ , daí o seu uso no corpo da função:

```
def classifica_modelo(x,y):
    y_equacao = -x + 3
    y_ponto = y

    if y_ponto > y_equacao:
        return 1
    elif y_ponto < y_equacao:
        return 0
    else:
        return -1
```

```
# (3.5,0.7)
classifica_modelo(3.5,0.7)
```

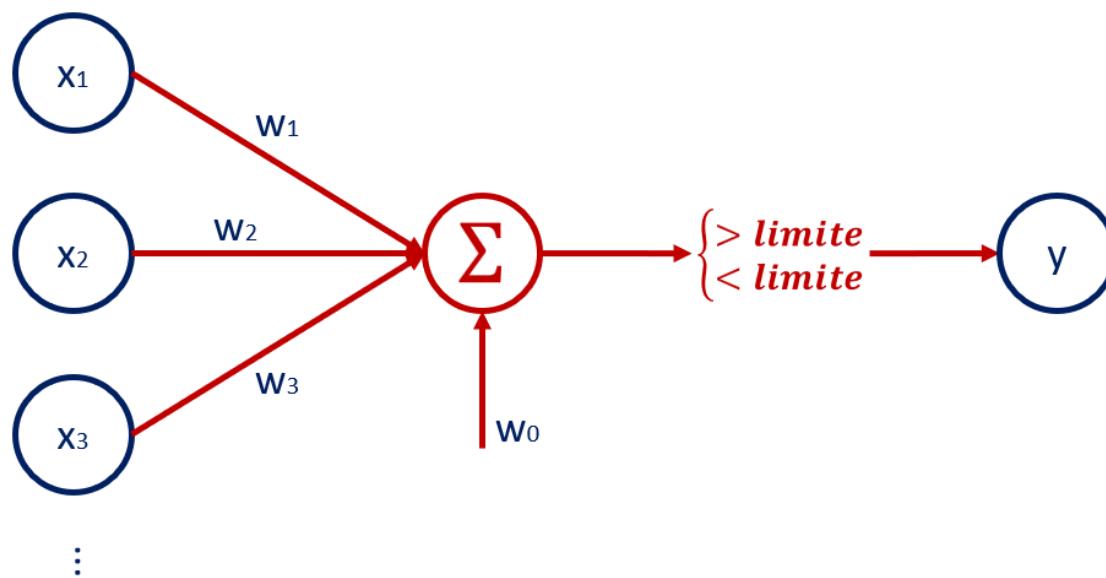
1

```
# (2,0.6)
classifica_modelo(2,0.6)
```

0

## Módulo 12 – Perceptron

Tudo que fizemos manualmente pode ser automatizado com modelos do Scikit-Learn. E não apenas para duas variáveis, podendo ter diversas variáveis como mostrado na figura a seguir. A lógica é a mesma vista, mas a vantagem é delegar todo o trabalho que tivemos manualmente para um modelo. No caso, o modelo Perceptron.



```
# Importando o perceptron
from sklearn.linear_model import Perceptron
clf = Perceptron(tol=1e-3, random_state=0)
```

## Módulo 12 – Perceptron

Agora, precisamos de nossos valores de X e de y. Passamos para o `fit` do modelo e podemos obter os coeficientes e o intercepto.

Ou seja, tudo o feito anteriormente, mas de forma bem mais rápida e reproduzível.

```
# Selecionando o X e o y
X = iris[['petal length (cm)', 'petal width (cm)']]
y = iris.Target
```

```
# Fazendo o fit com o modelo
clf.fit(X, y)
```

```
Perceptron()
```

```
# w1 e w2
clf.coef_
array([[0.9, 1.7]])
```

```
# w0
clf.intercept_
array([-3.])
```

```
clf.coef_[0][1]
```

1.7

## Módulo 12 – Perceptron

Com os coeficientes e o intercepto, podemos traçar graficamente a reta do modelo. Abaixo, a reta vermelha é que fizemos manualmente e a laranja a reta do modelo. Veja que há uma diferença significativa entre as duas.

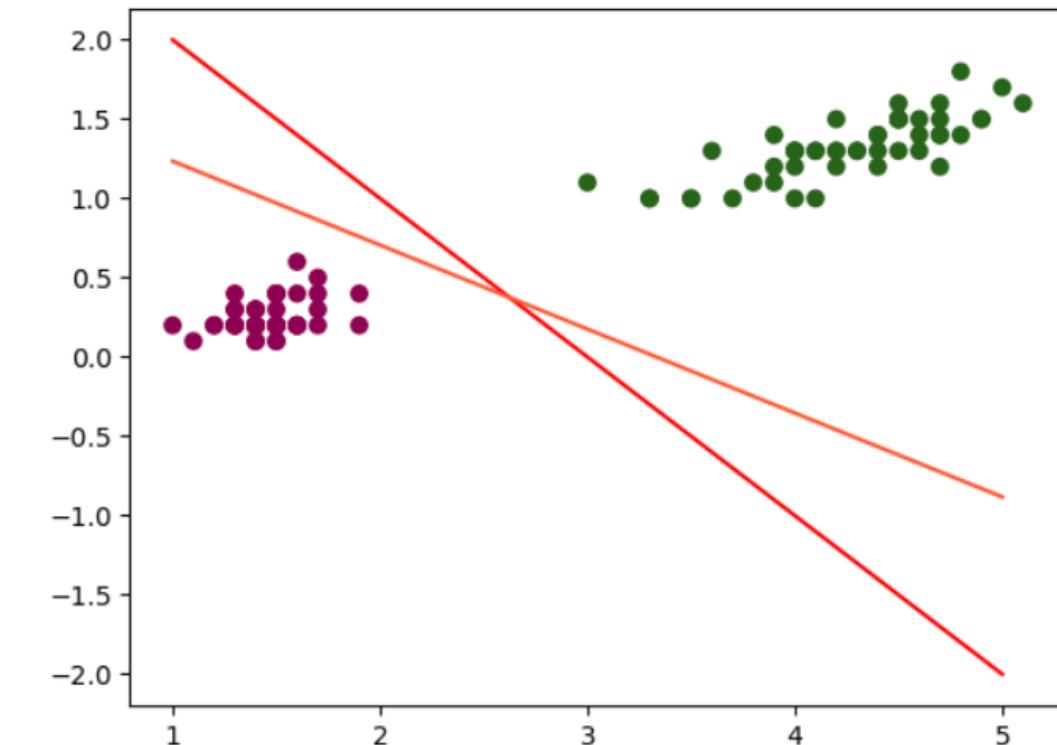
```
# Traçando novamente a reta e adicionando o ponto que acima
fig, ax = plt.subplots()

# Aqui estamos fazendo um gráfico de dispersão
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')

# Aqui estamos fazendo um gráfico de linha
x_reta3 = np.arange(1,6)
y_reta3 = (-1)*x_reta3+3
ax.plot(x_reta3,y_reta3,c='r')

# Aqui estamos fazendo um gráfico de linha
x_perc = np.arange(1,6)
y_perc = (-clf.intercept_-clf.coef_[0][0]*x_perc)/clf.coef_[0][1]
ax.plot(x_perc,y_perc,c='#FF5733')

plt.show()
```



## Módulo 12 – Perceptron

Para entender a importância de ter uma boa reta de separação, vamos colocar manualmente dois novos pontos em nossa base de dados.

```
# Adicionando uma nova base com os dados mostrados abaixo
base2 = pd.DataFrame([[0,0,3,0.75,0], [0,0,2.5,1,0]])
base2.columns = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)', 'Target']
```

```
base2.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
0	0	0	3.0	0.75	0
1	0	0	2.5	1.00	0

```
# Adicionado esses dados ao dataset iris
iris = iris.append(base2)
```

```
iris.tail()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
97	6.2	2.9	4.3	1.30	1
98	5.1	2.5	3.0	1.10	1
99	5.7	2.8	4.1	1.30	1
0	0.0	0.0	3.0	0.75	0
1	0.0	0.0	2.5	1.00	0

## Módulo 12 – Perceptron

Agora, vamos treinar nosso modelo com essa nova base de dados, com os dois pontos adicionados:

```
# Executando o Perceptron pra essa nova base
X = iris[["petal length (cm)", "petal width (cm)"]]
y = iris.Target

clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(X, y)

Perceptron()
```

## Módulo 12 – Perceptron

Veja que a reta do modelo (a laranja) é diferente da reta do modelo anterior, quando não havia os dois pontos. O modelo efetivamente foi alterado pela presença destes pontos na base de dados. Enquanto isso, nossa reta criada manualmente (a vermelha) permanece inalterada e classifica os novos pontos erroneamente. Percebemos, portanto, como o machine learning é importante por efetivamente aprender com novos dados.

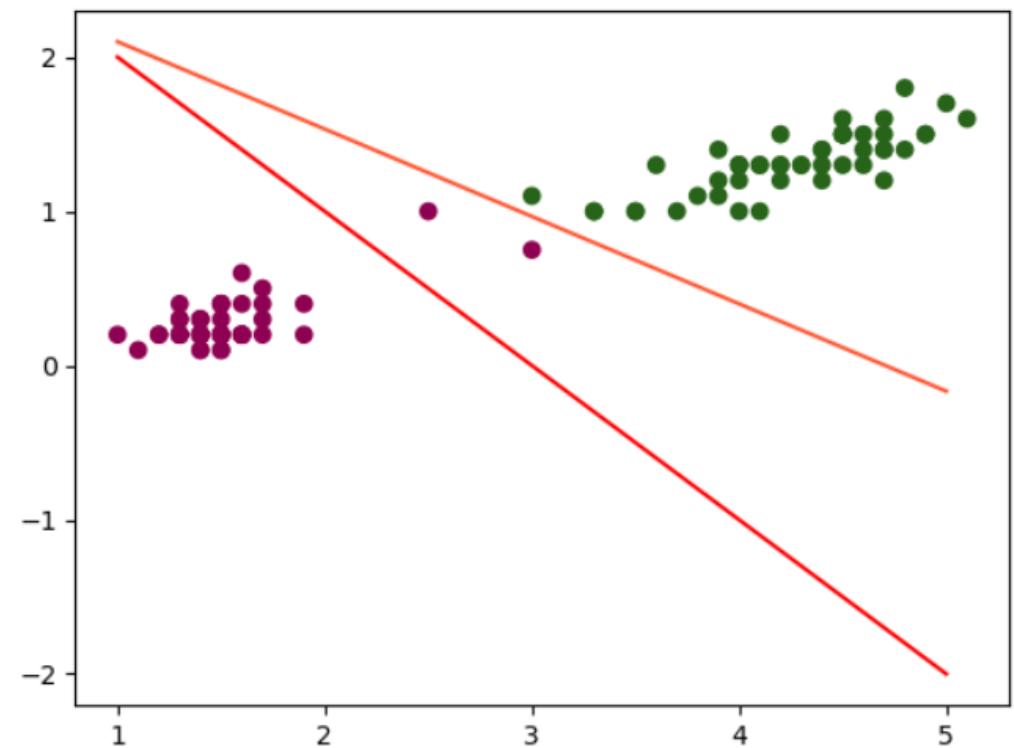
```
# Traçando novamente a reta e adicionando o ponto que acima
fig, ax = plt.subplots()

# Aqui estamos fazendo um gráfico de dispersão
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')

# Aqui estamos fazendo um gráfico de linha
x_reta3 = np.arange(1,6)
y_reta3 = (-1)*x_reta3+3
ax.plot(x_reta3,y_reta3,c='r')

# Aqui estamos fazendo um gráfico de linha
x_perc = np.arange(1,6)
y_perc = (-clf.intercept_-clf.coef_[0][0]*x_perc)/clf.coef_[0][1]
ax.plot(x_perc,y_perc,c='#FF5733')

plt.show()
```



## MÓDULO 13

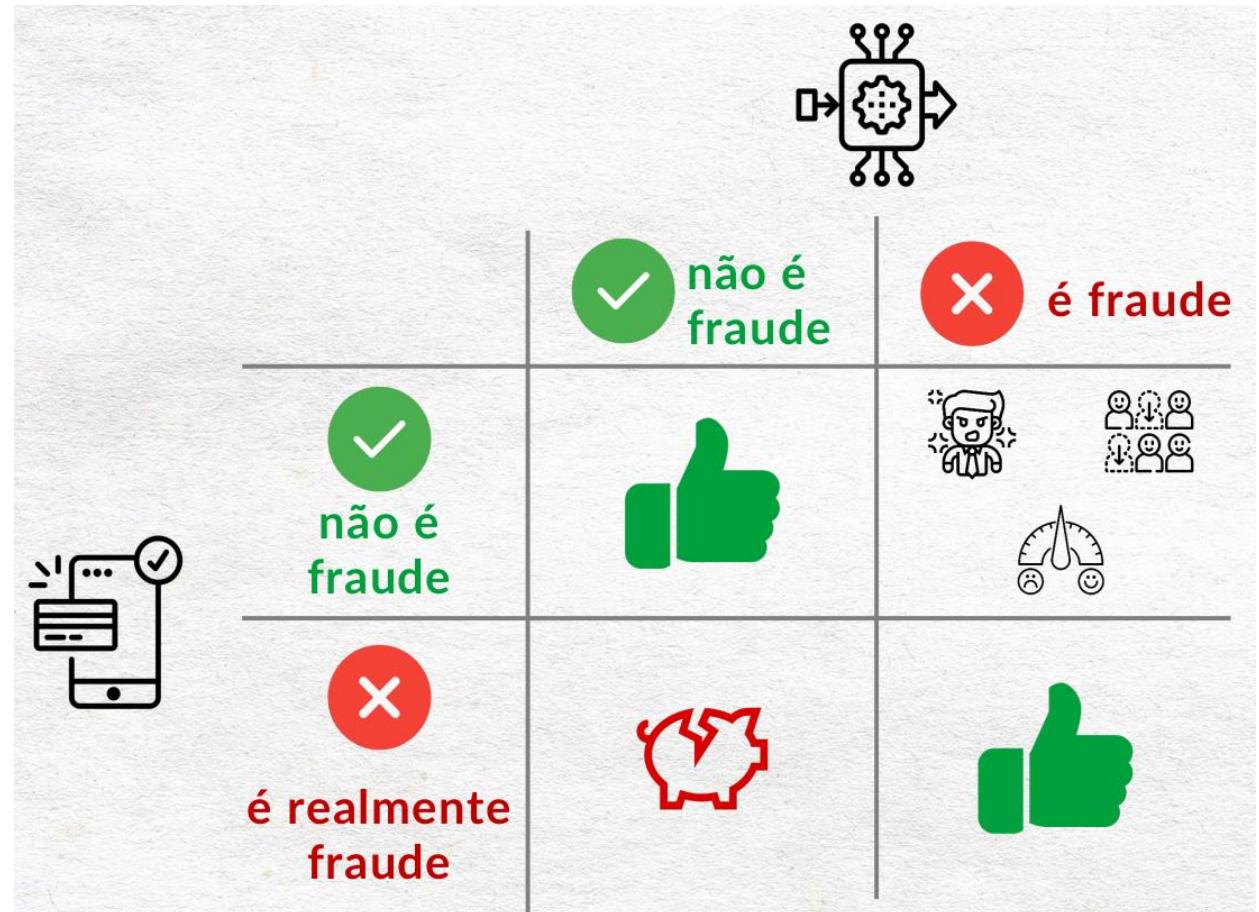
# Utilizando o aprendizado de máquinas

## Módulo 13 – Matriz de confusão

Em módulos anteriores, vimos que podemos classificar transações financeiras como fraudulentas ou não.

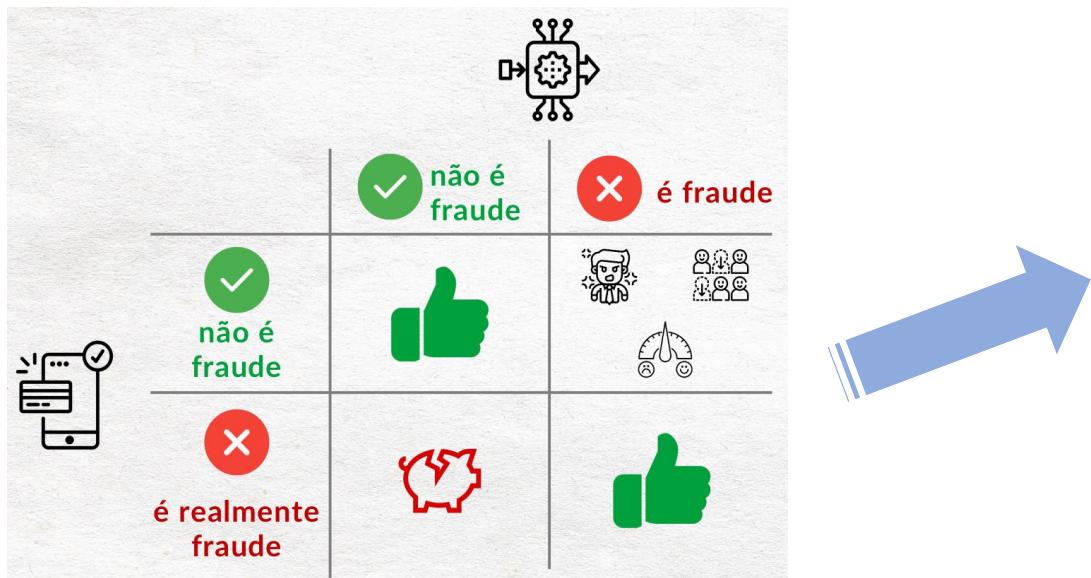
Vamos retomar esta aplicação para conhecer melhor alguns termos utilizados em aprendizado de máquinas.

Começaremos entendendo o que são resultados positivos e negativos em uma matriz de confusão.



## Módulo 13 – Matriz de confusão

Na vida real, é muito difícil que um modelo acerte todas as previsões. Assim, teremos os chamados falsos positivos (transações não fraudulentas erroneamente classificadas como fraude) e os falsos negativos (transações fraudulentas erroneamente classificadas como legítimas):



		MODELO	
REAL (DADOS)	+	+	-
	-	Verdadeiro positivo	Falso negativo
+	+	Verdadeiro negativo	Falso positivo
-	-		

## Módulo 13 – Matriz de confusão

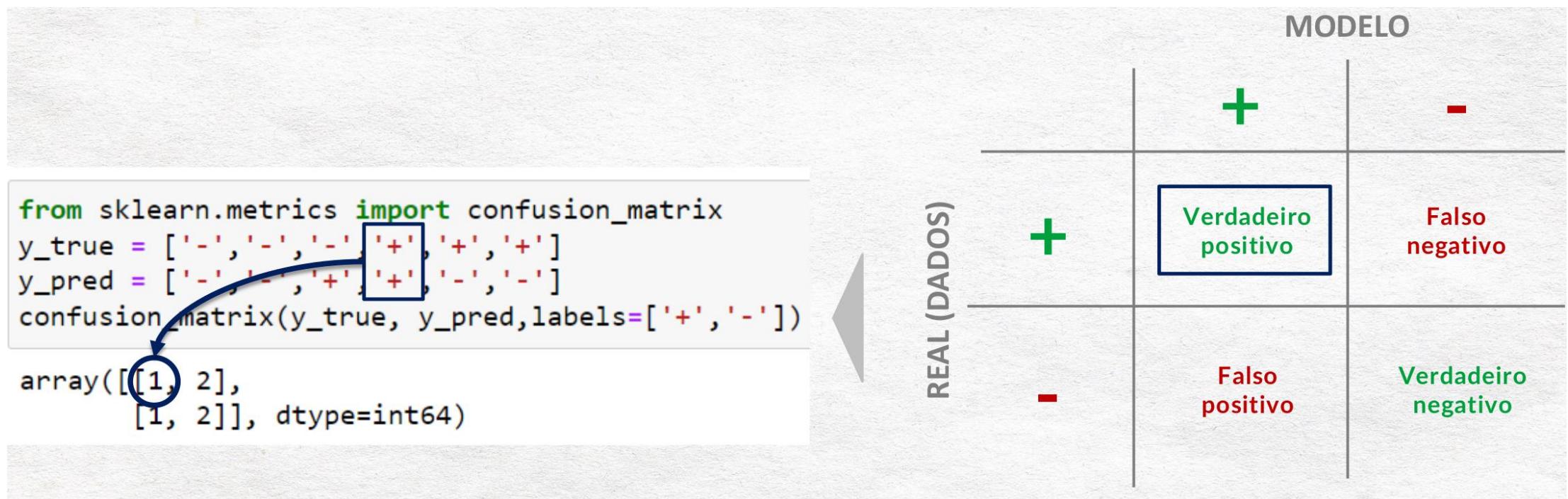
Esse arranjo das figuras apresentadas é conhecido como matriz de confusão. Vamos construir um pequeno código Python importando o método `confusion_matrix` do Scikit-Learn. Passaremos 6 dados reais (`y_true`) e 6 resultados de um modelo de previsão (`y_pred`).

```
from sklearn.metrics import confusion_matrix
y_true = ['-', '-', '-', '+', '+', '+']
y_pred = ['-', '-', '+', '+', '-', '-']
confusion_matrix(y_true, y_pred, labels=['+', '-'])

array([[1, 2],
       [1, 2]], dtype=int64)
```

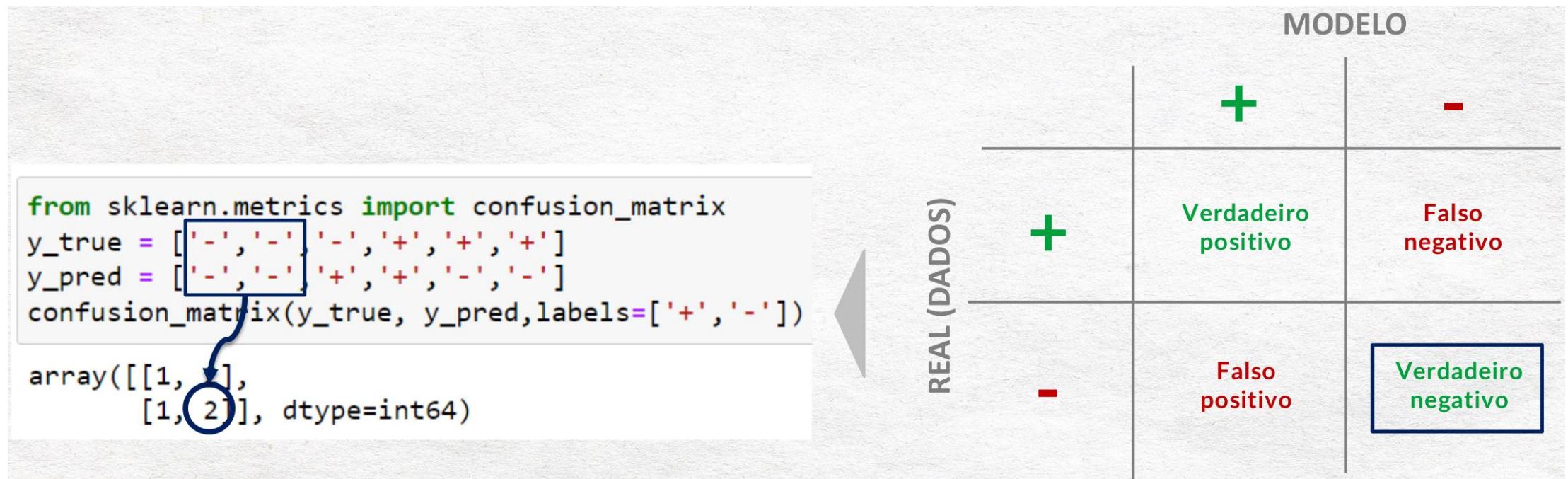
## Módulo 13 – Matriz de confusão

Observe, comparando visualmente as duas listas, que há apenas uma situação com sinal positivo nos dados que também apresenta sinal positivo na previsão. Logo, há um verdadeiro positivo.



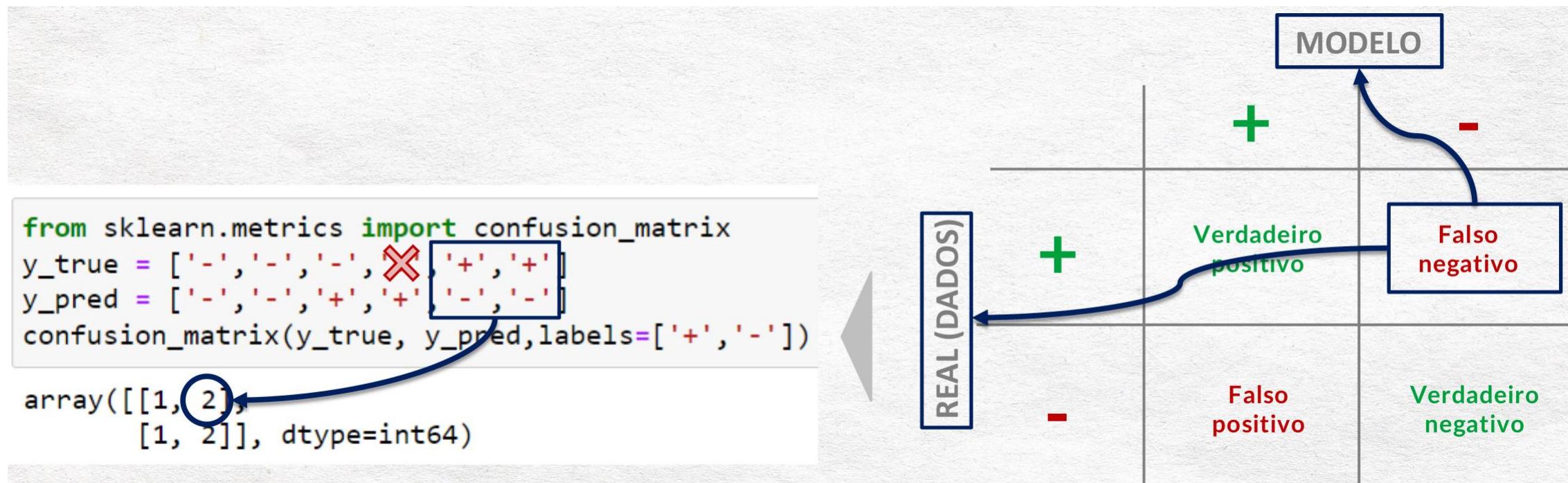
## Módulo 13 – Matriz de confusão

Seguindo, vemos que há duas situações coincidentes de sinais negativos. Logo, há 2 verdadeiros negativos.



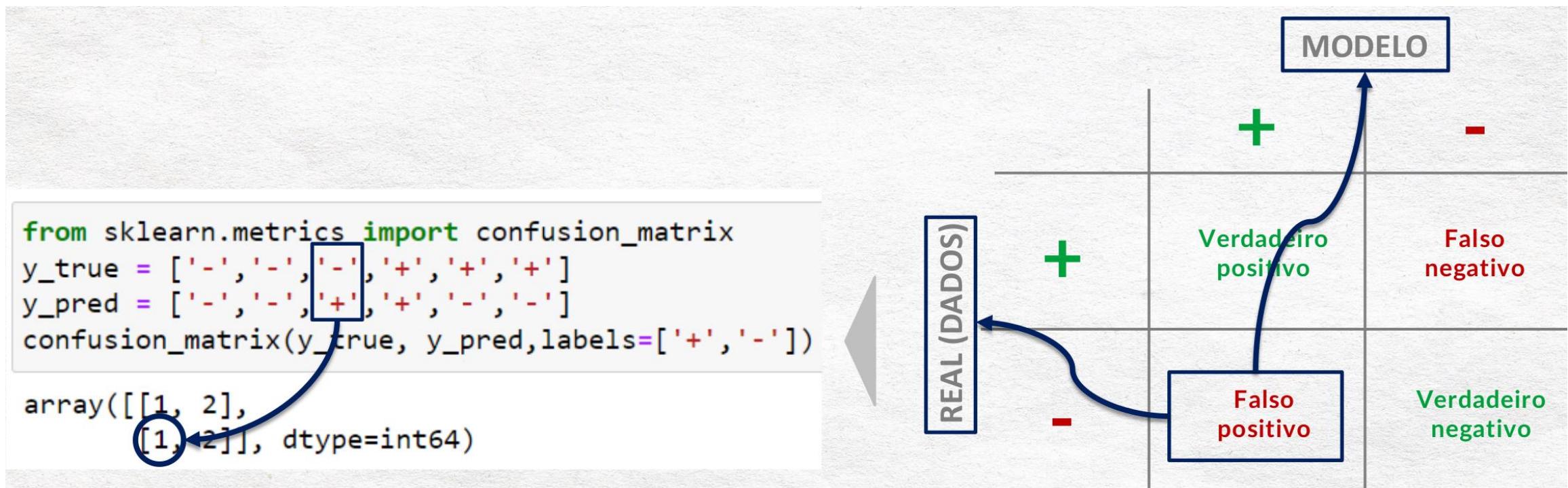
## Módulo 13 – Matriz de confusão

Agora, temos os erros do modelo. Há duas situações que são de sinal positivo na vida real, mas que o modelo previu como negativas. Logo, há dois falsos negativos.



## Módulo 13 – Matriz de confusão

Há uma situação de sinal negativo na vida real que o modelo classificou como positivo. Logo, há um falso positivo.



## Módulo 13 – Matriz de confusão

Observe nas imagens que o Scikit-Learn apresenta os números na forma de uma matriz. Essa é a forma usual de representar, com os números indicando cada situação.

		MODELO	
		+	-
REAL (DADOS)	+	1	2
	-	1	2

## Módulo 13 – Matriz de confusão

Nas páginas anteriores, usamos os símbolos + e - por questões didáticas. Em projetos reais, usualmente as classificações são representadas pelos números 0 e 1. Perceba que, por padrão, o Scikit-Learn coloca o 0 antes do 1, trocando as posições das classificações. Não há problema, o importante é saber o significado da matriz e estar atento a este detalhe.

		MODELO	
AL (DADOS)	+	-	
	+	Verdadeiro positivo	Falso negativo

```
from sklearn.metrics import confusion_matrix
y_true = ['-', '-', '-', '+', '+', '+']
y_pred = ['-', '-', '+', '+', '-', '-']
confusion_matrix(y_true, y_pred, labels=['+', '-'])

array([[1, 2],
       [1, 2]], dtype=int64)
```

		MODELO	
REAL (DADOS)	0	1	
	0	Verdadeiro negativo	Falso positivo

```
# Dados
y_true = [0,0,0,1,1,1]
y_pred = [0,0,1,1,0,0]

confusion_matrix(y_true, y_pred)

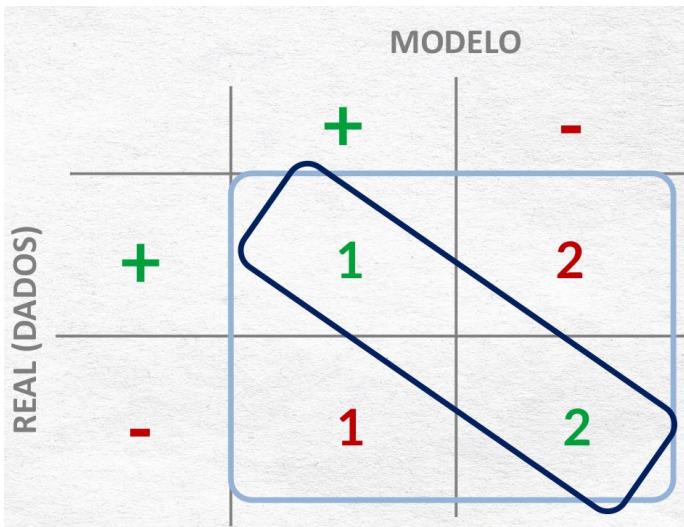
array([[2, 1],
       [2, 1]], dtype=int64)
```

## Módulo 13 – Acurácia de um modelo de classificação

Quando desenvolvemos um modelo, nossa maior preocupação é saber o quanto bom ele é. Daí a necessidade de métricas. A matriz de confusão nos ajuda a entender algumas das principais métricas.

Comecemos com a acurácia:  $\text{Acurácia} = \frac{\text{Verdadeiros positivos} + \text{Verdadeiros negativos}}{\text{Total}}$

Com os valores do nosso exemplo:  $\text{Acurácia} = \frac{1 + 2}{1 + 2 + 1 + 2} = 0,5 \therefore 50\%$



O Scikit-Learn possui uma vasta coleção de métodos para métricas. Podemos importar o `accuracy_score` e obter o mesmo valor acima:

```
from sklearn.metrics import accuracy_score  
accuracy_score(y_true, y_pred)
```

0.5

## Módulo 13 – Acurácia de um modelo de classificação

Um cuidado que devemos ter é o de não confiar cegamente em uma única métrica. Por exemplo, uma acurácia de 90 % é boa?

$$\text{Acurácia} = \frac{\text{Verdadeiros positivos} + \text{Verdadeiros negativos}}{\text{Total}}$$

Observe acima novamente a definição de acurácia. Perceba que é uma métrica que considera a fração de acertos no total de previsões. Logo, se a base de dados for muito desbalanceada, como no nosso caso ao lado onde 90 % das entradas são de uma das classes, altas acuráncias serão obtidas facilmente. Afinal, basta prever que todas as entradas são da classe dominante e já teremos acuráncias dessa ordem de grandeza.

Este fenômeno é chamado de **paradoxo da acurácia**. Vale uma busca para ver mais casos interessantes.

```
from sklearn.metrics import confusion_matrix  
confusion_matrix(y_true,y_pred,labels=[1,0])
```

```
array([[ 0,  2],  
       [ 0, 18]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score  
accuracy_score(y_true,y_pred)
```

8

**1: é fraude**  
**0: não é fraude**

## Módulo 13 – Precisão de um modelo de classificação

A precisão é a fração de verdadeiros positivos frente o total de previsões positivas.

$$\text{Precisão} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos positivos}}$$

Com os valores do nosso exemplo:  $\text{Precisão} = \frac{1}{1+1} = 0,5 \therefore 50\%$

		MODELO	
		+	-
REAL (DADOS)	+	1	2
	-	1	2

No Scikit-Learn, podemos importar o `precision_score` e obter o mesmo valor acima:

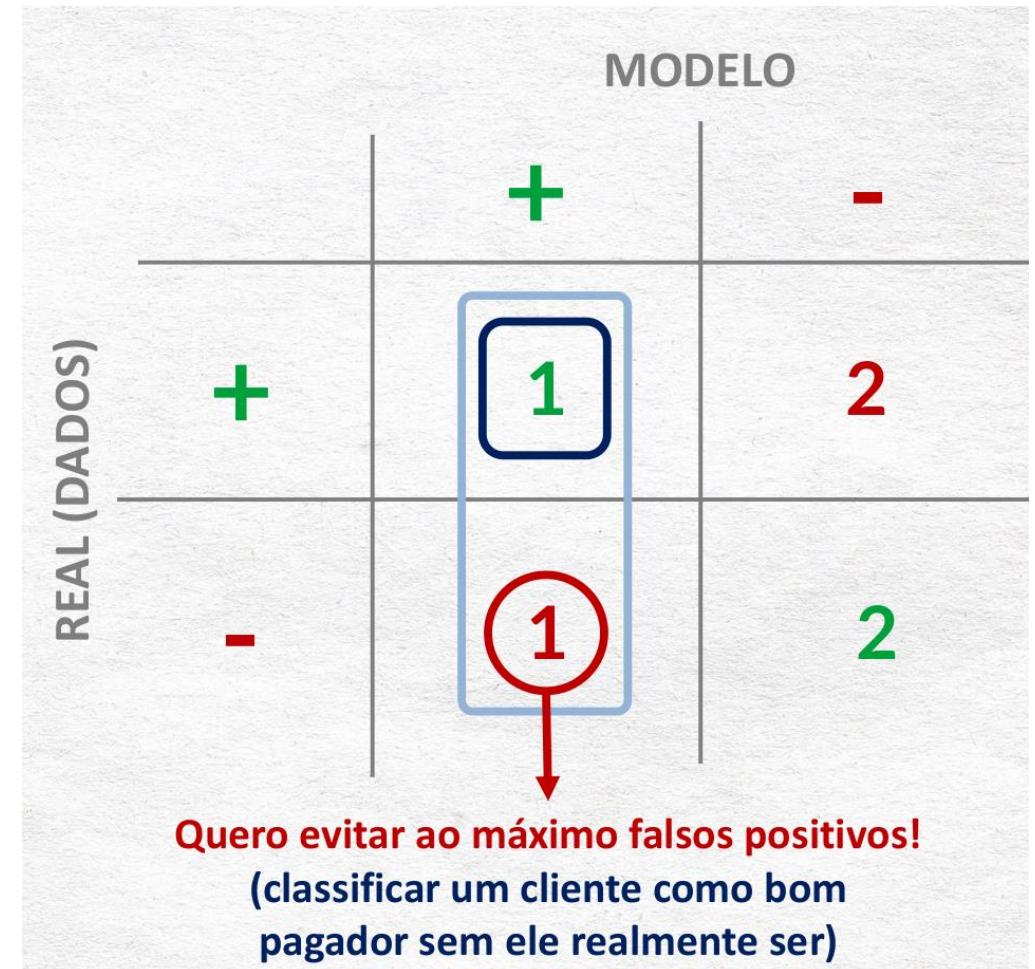
```
from sklearn.metrics import precision_score  
precision_score(y_true, y_pred, pos_label='+')
```

0.5

## Módulo 13 – Precisão de um modelo de classificação

Pela definição, vemos que a precisão mede o quanto seu modelo é bom em prever positivos. Buscar maximizar a precisão irá minimizar o número de erros de falsos positivos.

No nosso contexto, uma transação legítima ser considerada fraudulenta significa que o usuário terá seu cartão bloqueado indevidamente, causando conflito e desconforto no cliente.



## Módulo 13 – Recall de um modelo de classificação

O recall é fração de verdadeiros positivos frente ao total de positivos reais. Afinal, os falsos negativos são, na realidade, positivos.

$$\text{Recall} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos negativos}}$$

Com os valores de nosso exemplo:

$$\text{Recall} = \frac{1}{1+2} = 0,333\dots \therefore 33,3\%$$

		MODELO	
		+	-
REAL (DADOS)	+	1	2
	-	1	2

No Scikit-Learn, podemos importar o `recall_score` e obter o mesmo valor acima:

```
from sklearn.metrics import recall_score  
recall_score(y_true, y_pred, pos_label='+')
```

```
0.3333333333333333
```

## Módulo 13 – Recall de um modelo de classificação

Pela definição, vemos que o recall quantifica a proporção de reais positivos que foram identificados. Buscar maximizar o recall irá minimizar o número de erros de falsos negativos.

No nosso contexto, ao classificar erroneamente uma transação fraudulenta como legítima, um usuário teve seu dinheiro roubado via, por exemplo, clonagem do cartão. A companhia do cartão terá que reembolsar o usuário.



## Módulo 13 – Recall de um modelo de classificação

Como já visto, não se deve confiar em uma única métrica. Vamos considerar um caso onde temos um modelo com recall de 100 %. Será que é um bom modelo?

Pela definição, vemos que o recall quantifica a proporção de reais positivos que foram identificados. Buscar maximizar o recall irá minimizar o número de erros de falsos negativos. No nosso caso, maximizar a detecção de fraudes.

Mas observe como a precisão foi muito penalizada. Ou seja, o número de falsos positivos aumentou. Isto também possui um custo associado. Logo, a empresa deve encontrar um modelo que seja um bom custo-benefício entre precisão e recall.

```
# Dados
y_true = [0,0,0,1,1,1]
y_pred = [1,1,1,1,1,1]

from sklearn.metrics import confusion_matrix
confusion_matrix(y_true,y_pred,labels=[1,0])
array([[3, 0],
       [3, 0]], dtype=int64)

from sklearn.metrics import precision_score
precision_score(y_true,y_pred)
0.5

from sklearn.metrics import recall_score
recall_score(y_true,y_pred)
1.0
```

		MODELO	
		+	-
REAL (DADOS)	+	3	0
	-	3	0
		RECALL	PRECISÃO
		1: é fraude	0: não é fraude

## Módulo 13 – Conclusão de métricas de um modelo de classificação

Como conclusão dessa parte de métricas, temos que nenhuma métrica possui valor isoladamente.

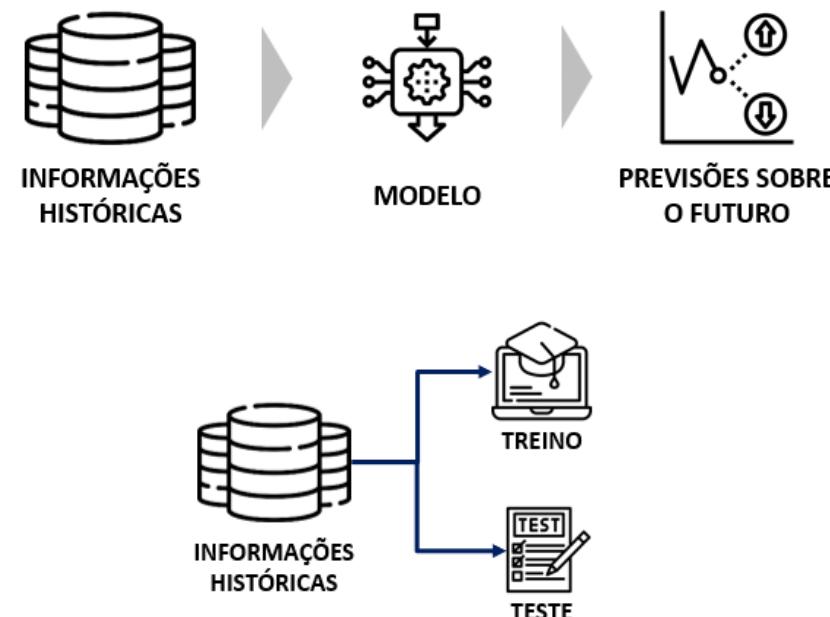
Acabamos de ver as três principais métricas e já percebemos a importância de considerar o problema a ser resolvido e os eventuais custos de cada decisão. Isto é uma constante em ciência de dados e sempre será discutido quando aprendermos novas métricas no decorrer do curso.

Percebemos claramente a importância de sermos cientistas de dados impressionadores!

## Módulo 13 – Separando os dados em treino e teste

Durante toda a explicação até o momento, comparamos um modelo com dados reais. Mas como fazer essa comparação quando recebemos uma base de dados fechada, sem ter uma forma de adquirir novos dados reais posteriormente?

Uma forma é separar a base em partes, uma para treinar o modelo e outra para testar o modelo, esta última fazendo o papel do que seria os novos dados reais. Os dados de treino farão o papel das informações históricas que treinam o modelo enquanto que os de teste irão servir para avaliar qual a qualidade do modelo para previsões.



## Módulo 13 – Separando os dados em treino e teste

Para ver como fazer essa separação com o Scikit-Learn, vamos usar uma base de dados que mostra se determinados clientes de um banco são inadimplentes ou não junto com a informação de saldo em conta e de investimentos.

Começamos importando o Pandas e a base da mesma forma que fizemos em outros módulos:

```
1 import pandas as pd  
2 base = pd.read_excel("BaseInadimplencia.xlsx")  
3 base.head()
```

	SaldoConta	SaldoInvestimento	Situacao
0	-1.365390	-3.280278	0
1	-1.992492	-4.158429	0
2	-3.910816	-0.874096	0
3	-2.745822	-2.250098	0
4	-1.352205	-1.280924	0

## Módulo 13 – Separando os dados em treino e teste

Assim como já feito em outros módulos, separamos a base em X (atributos) e y (alvo) e, com o método `train_test_split`, separamos em dados de treino (*train*) e teste (*test*):

```
1 from sklearn.model_selection import train_test_split  
2 X = baseCompleta[['SaldoConta', 'SaldoInvestimento']]  
3 y = baseCompleta['Situacao']  
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Podemos verificar o tamanho de cada divisão feita:

```
1 X_train.shape
```

```
(670, 2)
```

```
1 X_test.shape
```

```
(330, 2)
```

## Módulo 13 – Avaliando nosso modelo

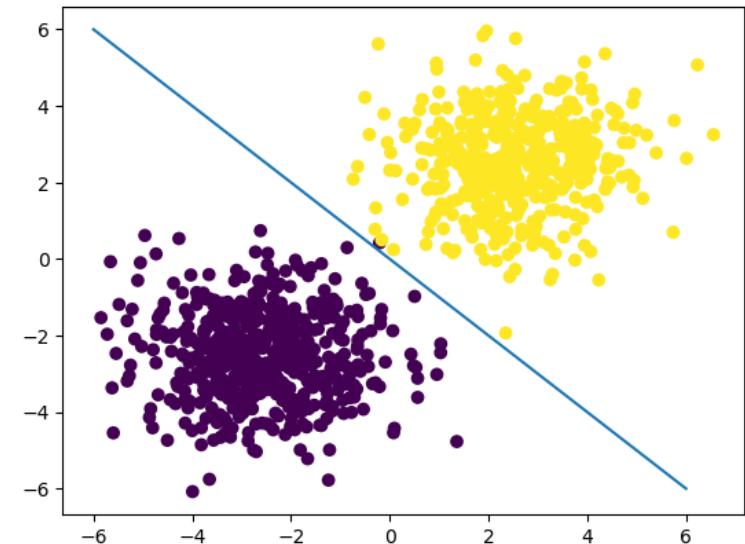
Vamos usar a mesma ideia de traçar uma reta que pode ser utilizada como fronteira de classificação dos nossos dados.

Essa reta pode ser transformada em função e aplicada nos dados, gerando previsões em cima da base de treino e da base de testes:

```
1 # Transformando essa reta em uma função para classificar os pontos
2 def clf(x,y):
3     # y_modelo = -x_modelo
4     y_modelo = -x
5
6     # Se y_modelo > y_dado -> class: 0
7     if y_modelo >= y:
8         return 0
9     # Se y_modelo < y_dado -> class: 1
10    elif y_modelo < y:
11        return 1
```

```
1 # Aplicando a função aos dados de treino
2 y_pred_train = X_train.apply(lambda x:clf(x['SaldoConta'],x['SaldoInvestimento']),axis=1)
3 y_pred_test = X_test.apply(lambda x:clf(x['SaldoConta'],x['SaldoInvestimento']),axis=1)
```

```
1 # Criando uma reta capaz de separar esses pontos
2 import matplotlib.pyplot as plt
3 fig, ax = plt.subplots()
4
5 ax.scatter(X_train.SaldoConta, X_train['SaldoInvestimento'],c=y_train)
6 ax.scatter(X_test.SaldoConta, X_test['SaldoInvestimento'],c=y_test)
7 x = [6,-6]
8 y = [-6,6]
9 ax.plot(x, y)
10
11 plt.show()
```



## Módulo 13 – Avaliando nosso modelo

Com o método `head` do Pandas, podemos ver alguns exemplos da classificação feita em cada base. E, com o método `confusion_matrix` do Scikit-Learn, conseguimos construir as matrizes de confusão para treino e teste.

```
1 # Verificando a base com a nova coluna  
2 y_pred_train.head(5)
```

```
703    1  
311    0  
722    1  
629    1  
0      0  
dtype: int64
```

```
1 y_pred_test.head(5)
```

```
521    0  
737    1  
740    1  
660    1  
411    0  
dtype: int64
```

```
1 from sklearn.metrics import confusion_matrix
```

```
1 # Matriz de confusão para a base de treino  
2 confusion_matrix(y_train,y_pred_train)
```

```
array([[359,    1],  
       [ 0, 310]])
```

```
1 # Matriz de confusão para a base de teste  
2 confusion_matrix(y_test,y_pred_test)
```

```
array([[183,    0],  
       [ 0, 147]])
```

## Módulo 13 – Avaliando nosso modelo

Agora, podemos utilizar os métodos das métricas que estudamos anteriormente no módulo.

```
1 from sklearn.metrics import accuracy_score  
2 # Acurácia para a base de treino  
3 accuracy_score(y_train,y_pred_train)
```

0.9985074626865672

```
1 from sklearn.metrics import precision_score  
2 # Precisão para a base de treino  
3 precision_score(y_train,y_pred_train)
```

0.9967845659163987

```
1 from sklearn.metrics import recall_score  
2 # Recall para a base de treino  
3 recall_score(y_train,y_pred_train)
```

1.0

```
1 # Acurácia para a base de teste  
2 accuracy_score(y_test,y_pred_test)
```

1.0

```
1 # Precisão para a base de teste  
2 precision_score(y_test,y_pred_test)
```

1.0

```
1 # Recall para a base de teste  
2 recall_score(y_test,y_pred_test)
```

1.0

Como vemos, os valores são elevados, na base de teste todas as métricas foram de 100 %. Isto é coerente com o que vimos graficamente, onde a reta separava bem as duas classes.

Mas será esse o melhor modelo? Como podemos comparar modelos?

## Módulo 13 – Avaliando nosso modelo

Em um projeto real, usualmente testamos mais de um modelo para vermos qual é o mais adequado. Não apenas sob o ponto de vista de métricas, mas também do ponto de vista de performance (velocidade e consumo de recursos computacionais).

Agora, iremos conhecer um outro modelo, o de árvore de decisão, e comparar as métricas obtidas.

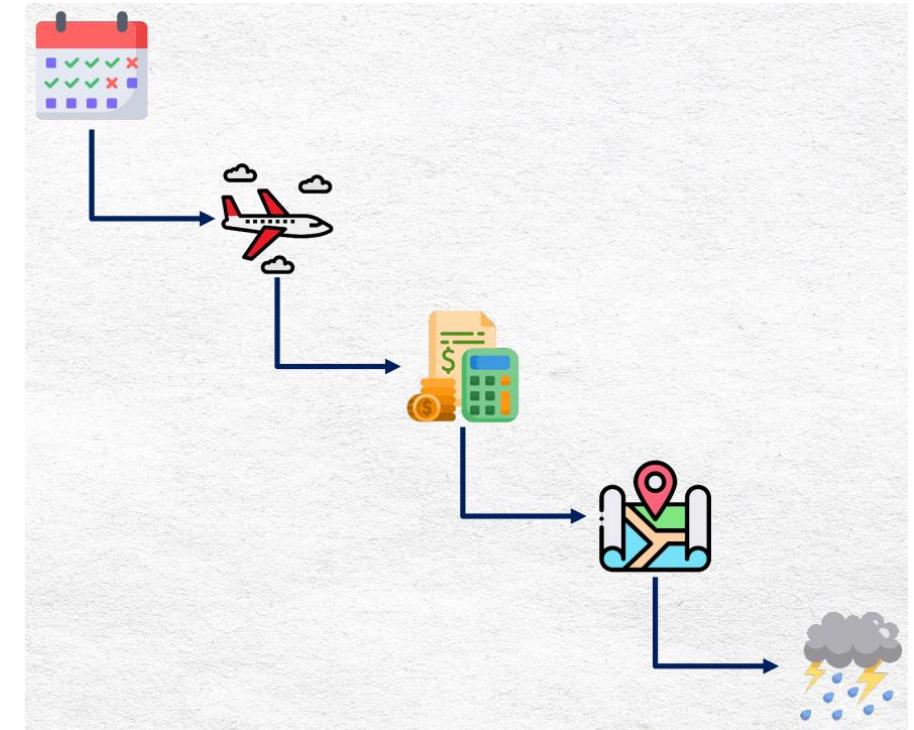


## Módulo 13 – O que é uma árvore de decisão?

O modelo que vamos estudar agora é o árvore de decisão.

Para entender a ideia geral do modelo, imagine que você foi convidado para uma festa. Provavelmente, você irá considerar alguns fatores para decidir se vai ou não ao evento. Por exemplo:

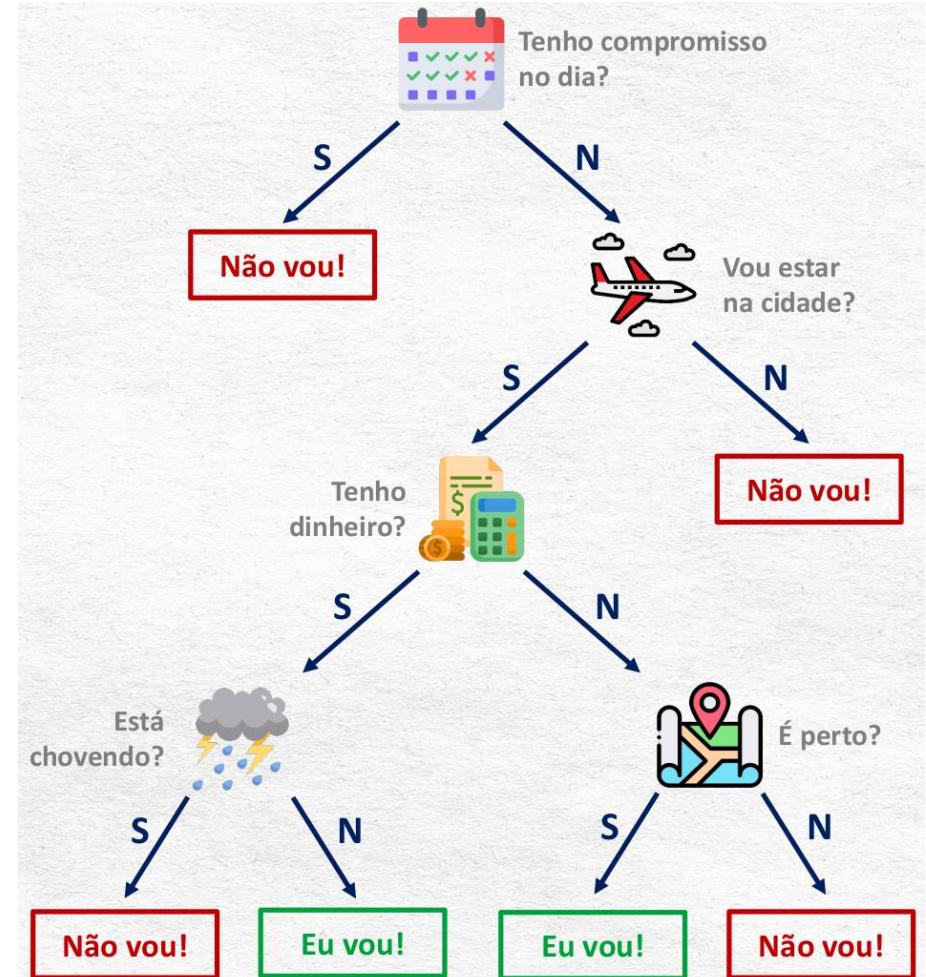
- Está livre no dia?
- Precisa viajar?
- Tem dinheiro?
- O local é de fácil acesso e seguro?
- Qual a previsão do tempo para o dia?



## Módulo 13 – O que é uma árvore de decisão?

Podemos elencar estes fatores na forma de uma árvore de decisões, cujos **ramos** vão crescendo a depender das respostas que damos a cada um dos questionamentos feitos nos **nós**.

Temos o **nó raiz** (compromisso no dia?), passando pelos **nós folha** (perguntas intermediárias), até chegar aos **nós finais** (perguntas finais). Também são utilizados os termos **nós internos** e **nós terminais** para os nós folha e finais, respectivamente.



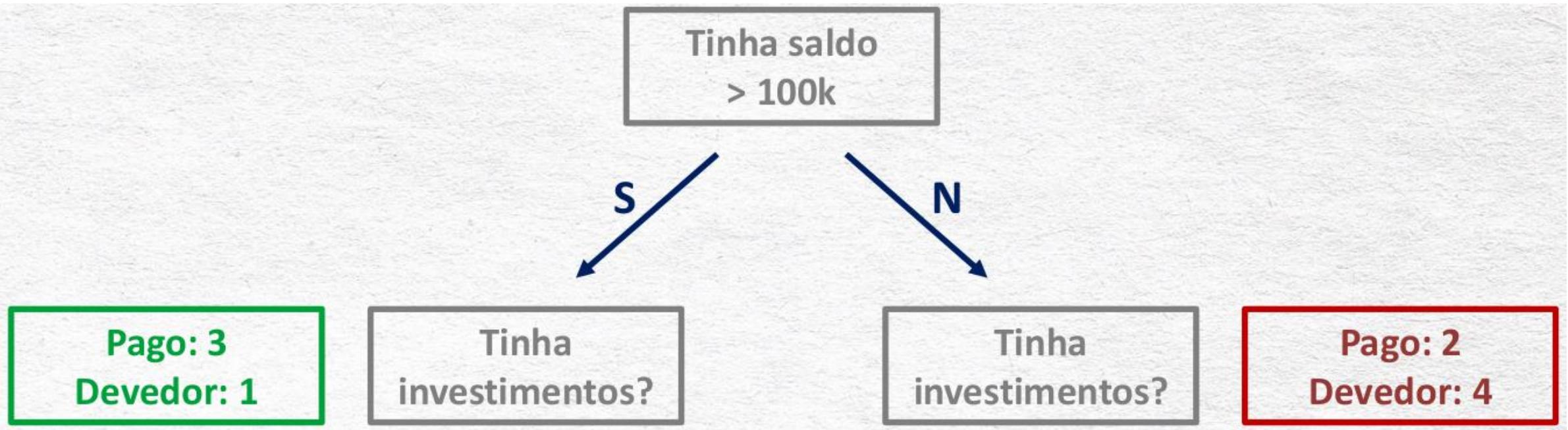
## Módulo 13 – O que é uma árvore de decisão?

Vamos considerar o seguinte exemplo: uma instituição financeira quer desenvolver um modelo de previsão de possíveis clientes devedores. Ela possui dados históricos de clientes de saldo em conta, se os clientes possuem investimentos ou não, e se possuem casa própria ou não.

Tinha saldo > 100k	Tinha investimentos	Tinha casa própria	Situação empréstimo
Sim	Sim	Sim	Pago
Não	Sim	Sim	Pago
Não	Não	Sim	Pago
Sim	Não	Sim	Pago
Não	Não	Não	Devedor
Não	Sim	Não	Devedor
Não	Sim	Não	Devedor
Não	Não	Não	Devedor
Sim	Sim	Não	Pago
Sim	Não	Não	Devedor

## Módulo 13 – O que é uma árvore de decisão?

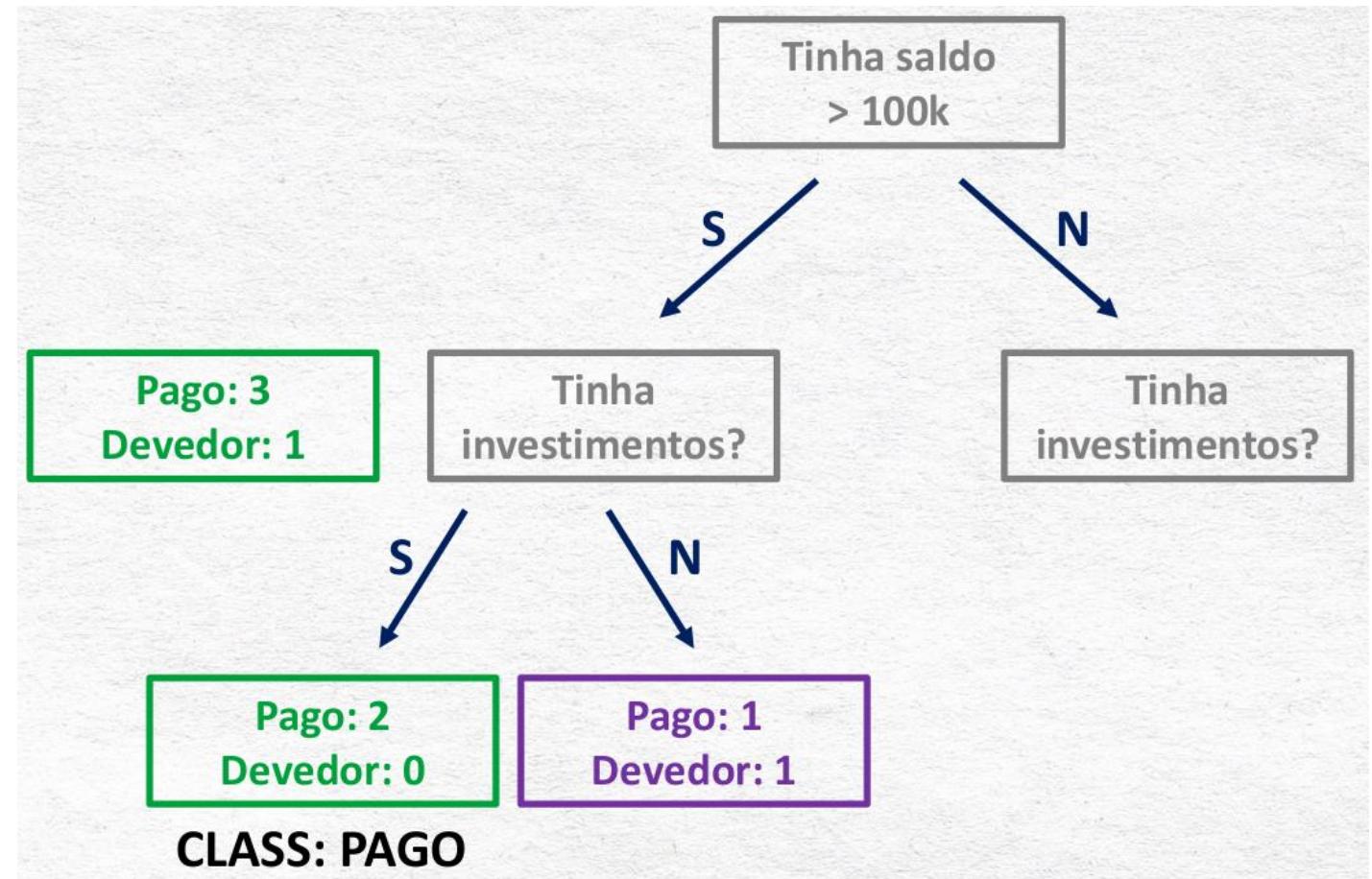
Podemos tentar construir uma árvore de decisão com base nos dados que temos. Primeiro, temos que decidir qual será nossa primeira pergunta. Vamos começar com o saldo bancário:



Agora temos dois ramos e nenhum deles chegou em uma classificação definitiva.

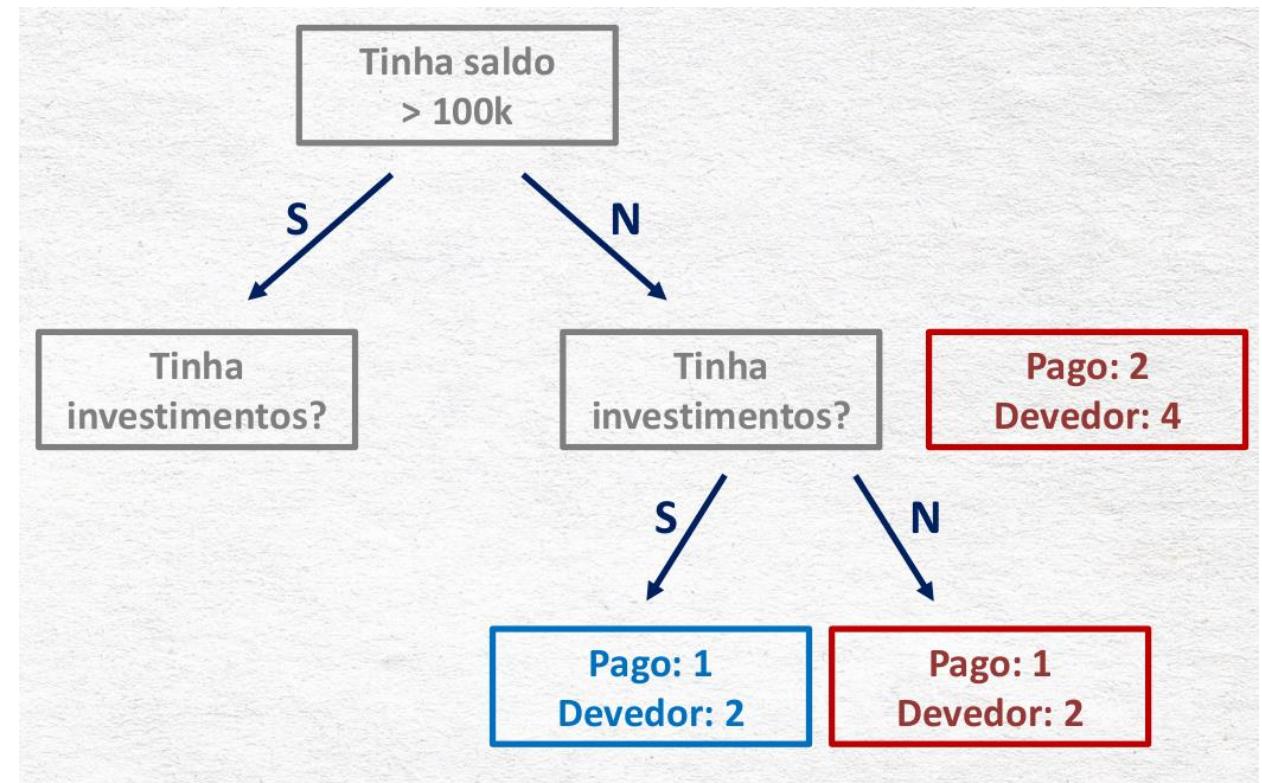
## Módulo 13 – O que é uma árvore de decisão?

Se estendermos o ramo da esquerda, chegamos em uma classificação definitiva, onde identificamos um caminho que permite identificar clientes bom pagadores.



## Módulo 13 – O que é uma árvore de decisão?

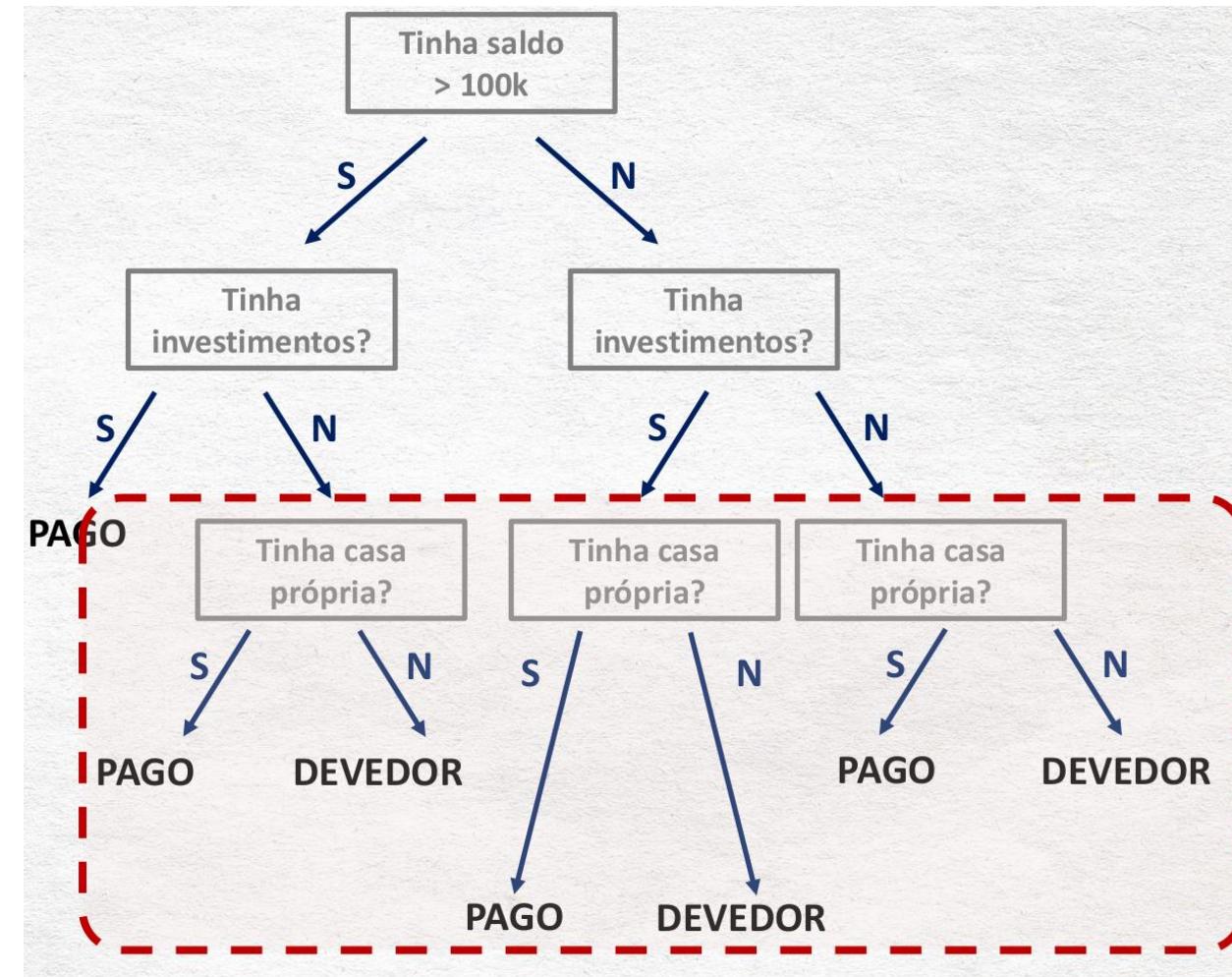
Indo para o ramo da direita, ainda não chegamos em uma classificação final ao responder o questionamento sobre investimentos. Ainda há mais um parâmetro para adicionar à árvore: a posse ou não de casa própria pelo cliente.



## Módulo 13 – O que é uma árvore de decisão?

Veja que a adição destes nós permitiu classificar todos os clientes.

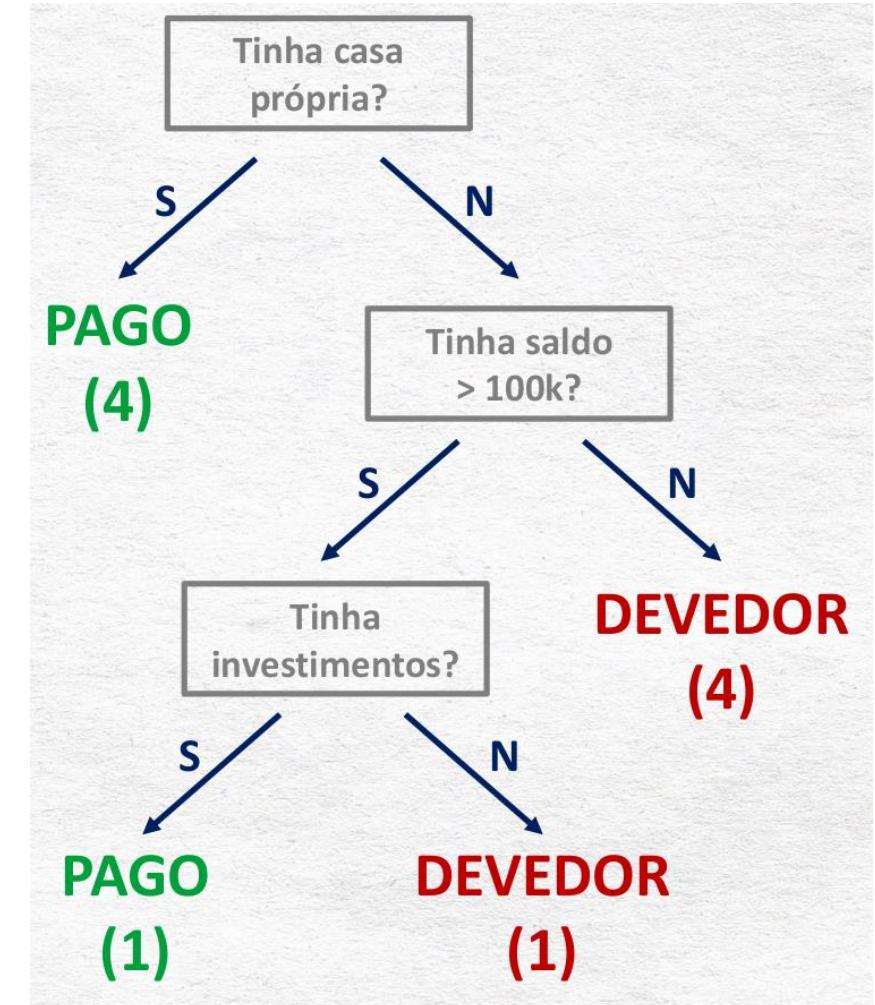
Isso nos faz pensar: será que se essa pergunta fosse nosso nó raiz, teríamos uma árvore menor?



## Módulo 13 – O que é uma árvore de decisão?

Veja como colocando a análise da casa logo no início, temos quase todos os bons pagadores classificados rapidamente.

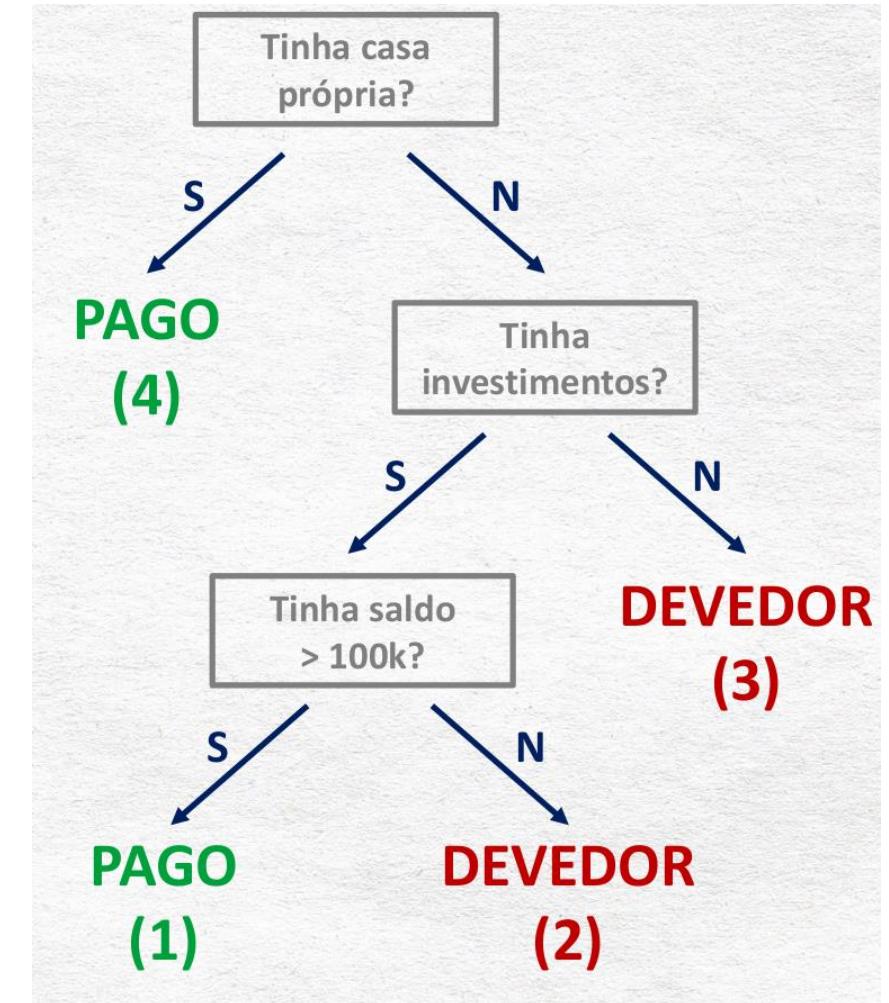
Ao questionar o saldo de investimentos, todos com menos de 100 mil são devedores. Por fim, a análise de investimentos classifica os demais.



## Módulo 13 – O que é uma árvore de decisão?

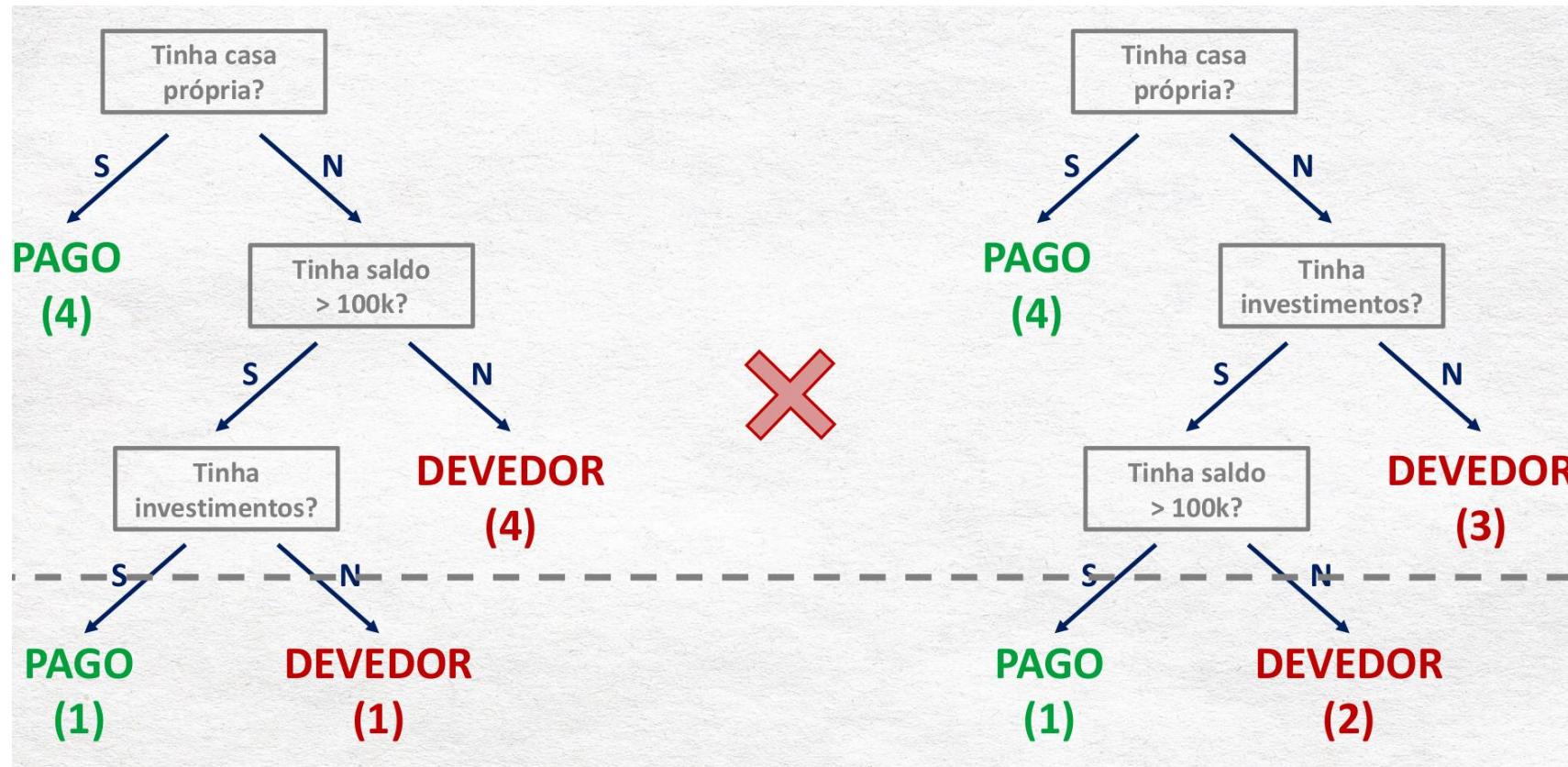
Trocando a ordem dos dois últimos nós, temos um resultado similar.

Mas qual é o melhor modelo? O da página anterior ou o ao lado?



## Módulo 13 – O que é uma árvore de decisão?

Considere os dois lado a lado. Vemos que o da esquerda com duas perguntas já classificou 8 dos 10 clientes. Isto significa que, se precisarmos diminuir o número de nós para tornar o modelo mais rápido, ele será um modelo provavelmente mais eficaz.



## Módulo 13 – Reutilizando o dataset íris

Vamos novamente importar o dataset íris para fazer a comparação entre o modelo de regressão linear e o de árvore de decisão.

Assim como anteriormente, vamos deixar com duas classes apenas e fazer nossa separação entre base de treino e de teste. A única diferença é que desta vez excluiremos a classe 0, pois as 1 e 2 não são tão facilmente separadas por uma reta. Dessa forma, será uma boa forma de comparar os modelos:

```
1 import pandas as pd  
  
1 # Importando o dataset iris  
2 from sklearn.datasets import load_iris  
3 data = load_iris()
```

```
1 # Transformando em um DataFrame  
2 iris = pd.DataFrame(data.data)  
3 iris.columns = data.feature_names  
4 iris['target'] = data.target  
5 iris = iris[iris.target != 0]  
6 iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

```
1 X = iris.drop('target',axis=1)  
2 y = iris.target
```

```
1 from sklearn.model_selection import train_test_split  
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

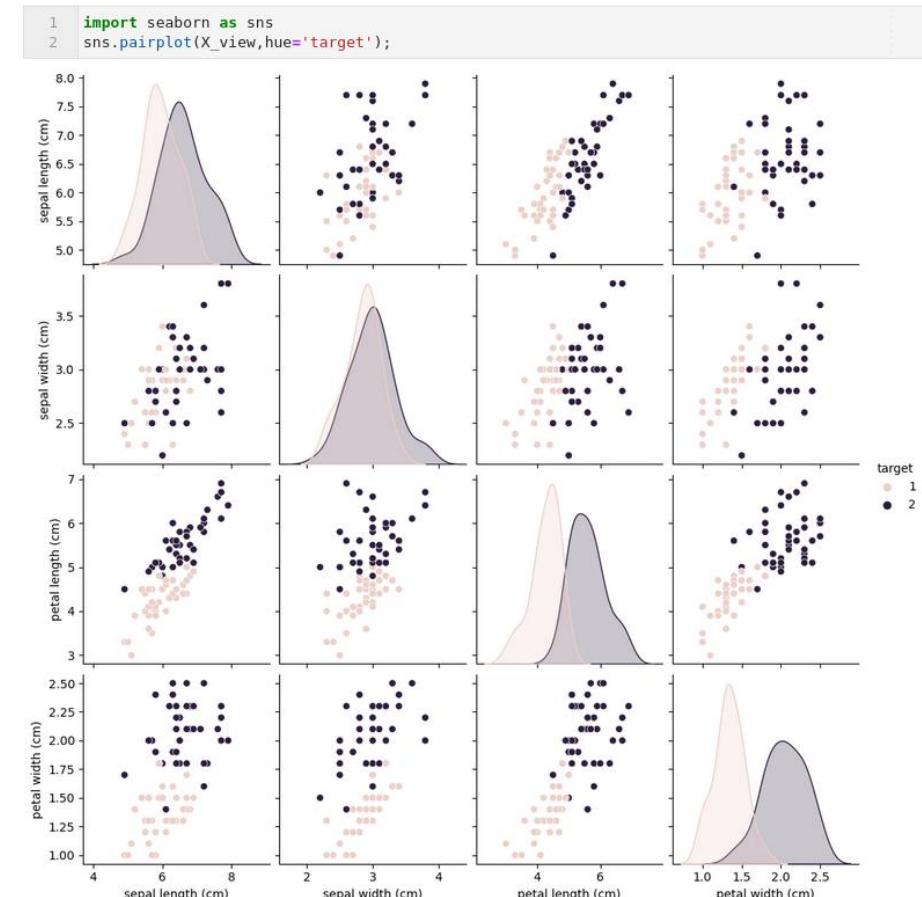
## Módulo 13 – Reutilizando o dataset íris

Apenas para visualização, vamos concatenar os X e y de teste e criar um pairplot.

Vemos como, de uma forma geral, é possível distinguir as duas classes, com pouca sobreposição das mesmas.

```
1 X_view = pd.concat([X_train,y_train],axis=1)
2 X_view.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
105	7.6	3.0	6.6	2.1	2
138	6.0	3.0	4.8	1.8	2
76	6.8	2.8	4.8	1.4	1
92	5.8	2.6	4.0	1.2	1
119	6.0	2.2	5.0	1.5	2

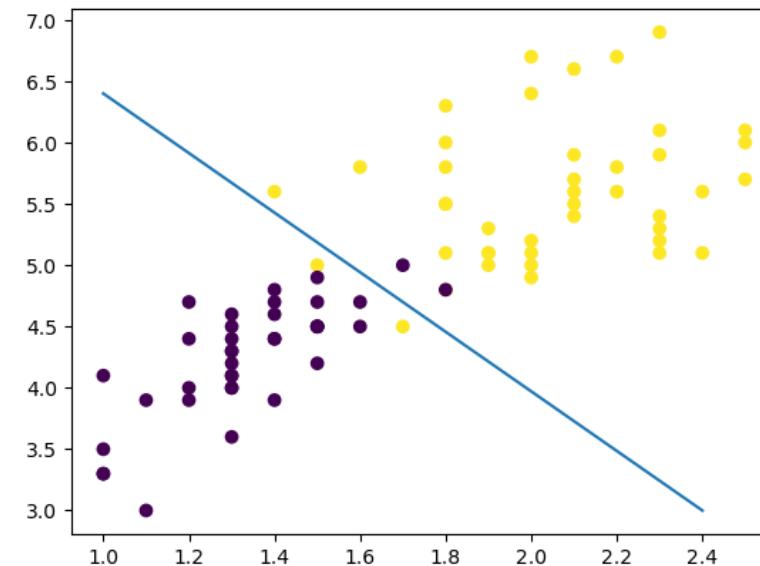


## Módulo 13 – Criando uma reta

De forma similar ao que já fizemos em outros datasets, vamos visualmente traçar uma reta que separa boa parte dos pontos de acordo com a coloração das classes e criar uma função de classificação de acordo com a posição dos pontos:

```
1 a=-1.7/0.7
2 b=3-2.4*a
3
4 def clf_reta(x,y):
5     y_modelo = a*x + b
6
7     # Se y_ponto > y_funcao: class 2
8     if y >= y_modelo:
9         return 2
10    # Se y_ponto < y_funcao: class 1
11    elif y < y_modelo:
12        return 1
```

```
1 # plot
2 fig, ax = plt.subplots()
3
4 ax.scatter(X_train['petal width (cm)'], X_train['petal length (cm)'], c=y_train.values)
5
6 # Criar o plot de uma reta
7 x_reta = [2.4,1]
8 y_reta = [3,6.4]
9 ax.plot(x_reta,y_reta)
10
11 plt.show()
```



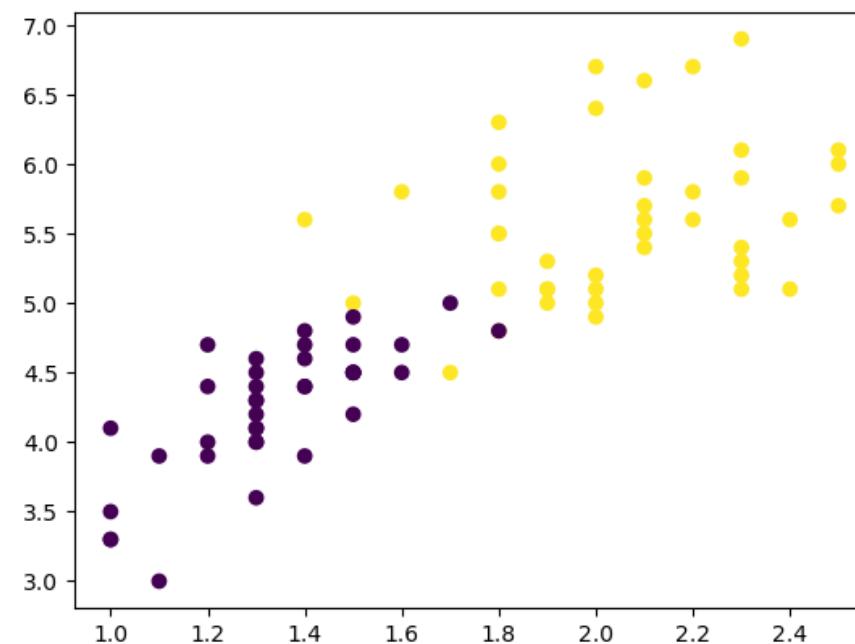
## Módulo 13 – Criando uma árvore de decisão

Para a árvore de decisões, precisamos definir alguns nós. Ou seja, alguns valores de x ou y que permitem separar nossas classes.

Visualmente, vemos que sempre que  $x \geq 1.9$  temos a classe amarela (2) e sempre que  $x \leq 1.3$  temos a classe roxa (1). Como essas duas comparações separam as classes por completo, colocamos elas no início, como os nós de cima da árvore. Para y, vemos que o valor de  $y = 5$  separa razoavelmente as classes. Mas colocando essa comparação ao final, as comparações de x já terão sido feitas e temos uma classificação boa das classes:

```
1 def clf_arvore(x,y):
2     if x >= 1.9:
3         return 2
4     elif x <= 1.3:
5         return 1
6     elif y <= 5:
7         return 1
8     elif y > 5:
9         return 2
```

```
1 # plot
2 fig, ax = plt.subplots()
3
4 ax.scatter(X_train['petal width (cm)'], X_train['petal length (cm)'], c=y_train.values)
5
6 # ax.set(xlim=(1.35,1.85), ylim=(5, 8))
7
8 plt.show()
```



## Módulo 13 – Comparando os modelos

Com nossas funções criadas, aplicamos aos nossos dados de treino e de teste e podemos, finalmente, começar a avaliar nossas métricas:

```
1 # Dados de treino
2 y_modelo_reta_treino = X_train.apply(lambda x:clf_reta(x['petal width (cm)'],x['petal length (cm)']),axis=1)
3 y_modelo_arvore_treino = X_train.apply(lambda x:clf_arvore(x['petal width (cm)'],x['petal length (cm)']),axis=1)
```

```
1 # Dados de teste
2 y_modelo_reta_teste = X_test.apply(lambda x:clf_reta(x['petal width (cm)'],x['petal length (cm)']),axis=1)
3 y_modelo_arvore_teste = X_test.apply(lambda x:clf_arvore(x['petal width (cm)'],x['petal length (cm)']),axis=1)
```

Como vimos, as métricas são decorrentes da matriz de confusão. Desta forma, vamos começar construindo as matrizes para os modelos:

```
1 # Para a reta
2 from sklearn.metrics import confusion_matrix
3 confusion_matrix(y_test, y_modelo_reta_teste)
```

```
array([[11,  1],
       [ 1,  7]])
```

```
1 # Para a árvore
2 from sklearn.metrics import confusion_matrix
3 confusion_matrix(y_test, y_modelo_arvore_teste)
```

```
array([[11,  1],
       [ 3,  5]])
```

## Módulo 13 – Comparando os modelos

Agora, podemos verificar as três métricas que estudamos: acurácia, precisão e recall.

Como vemos, a reta é consistentemente melhor que a árvore em todas as métricas. Ou seja, para este caso específico, comparando as funções que criamos, a função de separação linear ter melhor resultado que a função de árvore de decisão.

```
1 # Para a reta
2 from sklearn.metrics import accuracy_score
3 accuracy_score(y_test, y_modelo_reta_teste)
```

0.9

```
1 # Para a árvore
2 from sklearn.metrics import accuracy_score
3 accuracy_score(y_test, y_modelo_arvore_teste)
```

0.8

```
1 # Para a reta
2 from sklearn.metrics import precision_score
3 precision_score(y_test, y_modelo_reta_teste, pos_label=2)
```

0.875

```
1 # Para a árvore
2 from sklearn.metrics import precision_score
3 precision_score(y_test, y_modelo_arvore_teste, pos_label=2)
```

0.8333333333333334

```
1 # Para a reta
2 from sklearn.metrics import recall_score
3 recall_score(y_test, y_modelo_reta_teste, pos_label=2)
```

0.875

```
1 # Para a árvore
2 from sklearn.metrics import recall_score
3 recall_score(y_test, y_modelo_arvore_teste, pos_label=2)
```

0.625

Isto não significa que sempre será assim. Estamos olhando estas funções neste dataset. Mudar as funções e o dataset podem gerar resultados bem distintos.

Daí o papel do cientista de dados, avaliar os resultados e a necessidade de novos estudos a depender dos resultados e da necessidade do cliente.

## MÓDULO 14

# Análise exploratória de dados

## Módulo 14 – Entendendo o dataset Titanic

Neste módulo, utilizaremos a base de dados de passageiros do famoso Titanic. É uma excelente base para aprendermos o básico de tratamento de dados e de visualização.



SEA TRIALS OF R.M.S. TITANIC, BELFAST LOUGH 2<sup>ND</sup> APRIL 1912

## Módulo 14 – Entendendo o dataset Titanic

O primeiro passo é conhecer o dicionário de dados da base. O dicionário descreve o nome de cada coluna e sua respectiva descrição. Como mostrado na aula, tal base foi obtida do Kaggle, de onde traduzimos o conteúdo. As colunas dessa base de dados e seus significados são:

- **Passenger ID**: ID do passageiro (número único para cada um dos passageiros)
- **Survived**: sobrevivente (0 = Não, 1 = Sim)
- **Pclass**: Classe da passagem (1 = primeira classe, 2 = segunda classe, 3 = terceira classe)
- **Name**: nome do passageiro
- **Sex**: Gênero do passageiro
- **Age**: Idade (em anos) do passageiro
- **SibSp**: número de irmãos / cônjuges a bordo do Titanic
- **Parch**: número de pais / filhos a bordo do Titanic
- **Ticket**: código do bilhete de embarque
- **Fare**: tarifa da passagem
- **Cabin**: código da cabine
- **Embarked**: porto de embarque (C = Cherbourg, Q = Queenstown, S = Southampton)

## Módulo 14 – Analisando as informações da base

Já vimos alguns projetos antes, e o começo é igual aos demais. Iremos importar nossa base com o `read_csv` do Pandas:

```
1 # Importando o pandas  
2 import pandas as pd
```

```
1 # Importando a base de dados  
2 base = pd.read_csv('train.csv')
```

E agora começaremos um procedimento de reconhecimento de nossa base, que deve ser feito sempre, em qualquer projeto de ciência de bases que estiver trabalhando.

## Módulo 14 – Analisando as informações da base

É sempre importante verificar se a importação foi bem feita. Para isso, olhamos as primeiras e últimas linhas do dataframe criado com o intuito de ver se há algo estranho que possa indicar problemas no arquivo de origem da base. No caso, aparentemente está tudo dentro do que se espera para o conteúdo das colunas conforme suas definições:

```
1 # Visualizando as 3 primeiras linhas  
2 base.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3		female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

```
1 # Visualizando as 3 últimas linhas  
2 base.tail(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

## Módulo 14 – Analisando as informações da base

Outra verificação importante é o tamanho da base com o atributo `shape`. Usualmente recebemos esta informação da fonte da base, de forma que temos mais um indício se a importação funcionou.

Além disso, com o método `info`, podemos verificar o tipo de cada coluna. Isto é importante, especialmente para se certificar de que colunas numéricas foram efetivamente reconhecidas como tal (inteiro, `int64`, ou float, `float64`), para fazer operações numéricas com as mesmas. Colunas de texto são ditas tipo `object` no Pandas. Compare os tipos com as descrições das colunas para se certificar de que estão corretas.

```
1 # Verificando o tamanho da base
2 base.shape
```

(891, 12)

```
1 # Verificando as informações.
2 base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId 891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object  
 4   Sex          891 non-null    object  
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object  
 9   Fare          891 non-null    float64 
 10  Cabin         204 non-null    object  
 11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

## Módulo 14 – Analisando as informações da base

Agora, vejamos se há valores nulos nas colunas. Dessa forma, podemos verificar se nossa base tem muitos dados ausentes e começar a pensar em como tratar estas situações. Como vemos, a coluna *Cabin* possui diversos valores ausentes, e também há na coluna *Age*:

```
1 # Contando a quantidade de valores nulos  
2 base.isnull().sum()
```

```
PassengerId      0  
Survived         0  
Pclass           0  
Name             0  
Sex              0  
Age            177  
SibSp            0  
Parch            0  
Ticket           0  
Fare             0  
Cabin          687  
Embarked        2  
dtype: int64
```

## Módulo 14 – Analisando as informações da base

Uma atenção especial deve ser dada aos dados numéricos. É sempre importante verificar se há valores negativos, zeros, muito pequenos, muito grandes, dispersos... Uma primeira noção pode ser obtida com o método `describe`. Ele apresenta uma estatística básica das colunas numéricas com: contagem, média, desvio padrão, valor mínimo, quartis, e valor máximo.

1	# Verificando as informações estatísticas							
2	base.describe()							
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000	
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208	
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429	
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000	
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400	
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200	
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000	
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200	

## Módulo 14 – Analisando as informações da base

Outra análise importante é de valores únicos por coluna. Aqui, verificamos que algumas colunas possuem apenas 2 ou 3 valores, coerente com o dicionário de dados, e que a coluna *PassengerId* é realmente uma identificação única dos passageiros, sem duplicatas, já que o tamanho da base é de 891 linhas.

```
1 # Verificando a quantidade de valores diferentes na base
2 base.nunique()
```

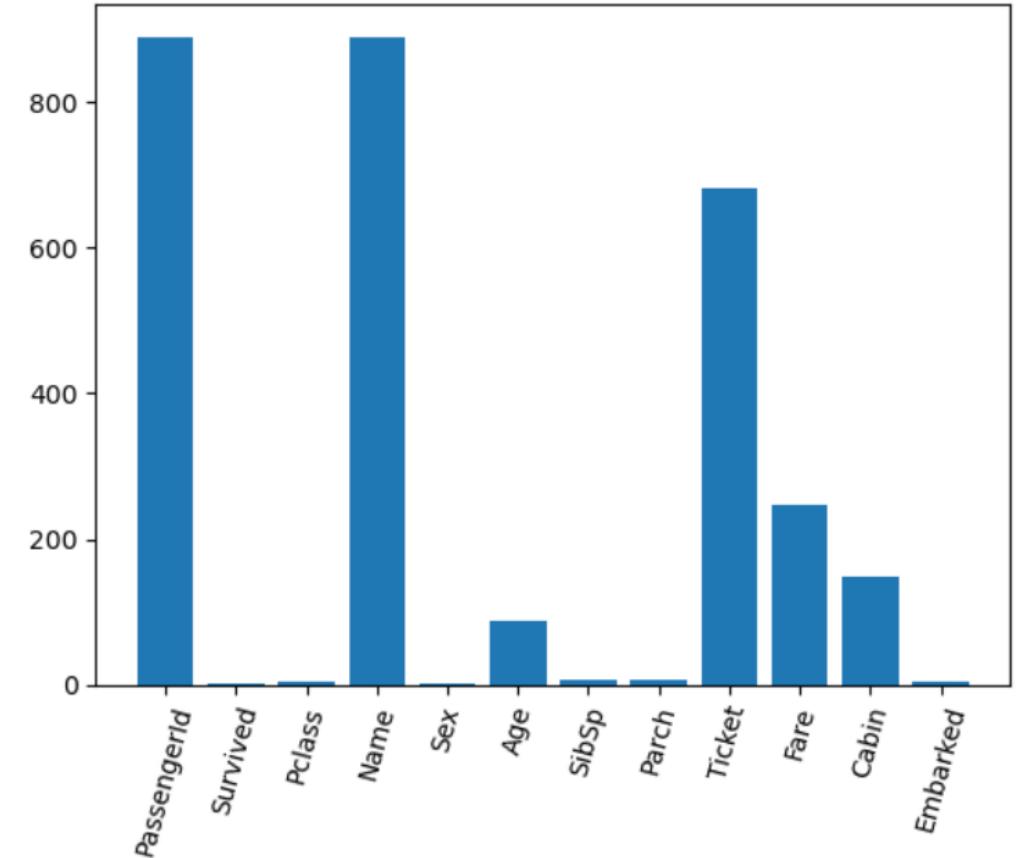
```
PassengerId    891
Survived       2
Pclass          3
Name           891
Sex            2
Age            88
SibSp          7
Parch          7
Ticket         681
Fare           248
Cabin          147
Embarked       3
dtype: int64
```

## Módulo 14 – Criando visualizações

Aqui vamos fazer nossa primeira visualização. Podemos pegar o `index` e o `values` dos valores únicos e construir um gráfico de barras de valores únicos por coluna:

```
1 # Importando o matplotlib
2 import matplotlib.pyplot as plt
```

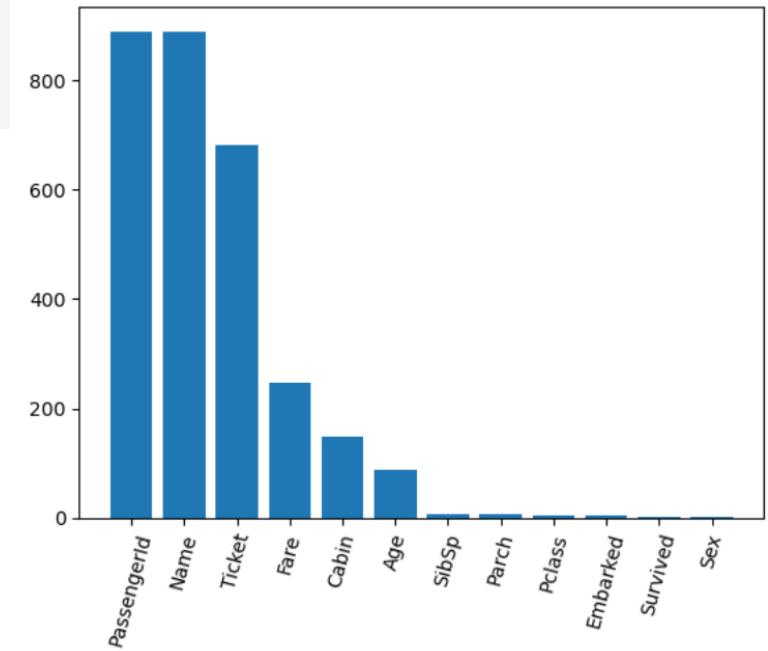
```
1 # Visualizando a informação acima de forma gráfica
2 fig, ax = plt.subplots()
3
4 ax.bar(base.nunique().index, base.nunique().values)
5
6 ax.tick_params(axis='x', labelrotation=75)
7
8 plt.show()
```



## Módulo 14 – Criando visualizações

A visualização é mais efetiva se as barras estiverem em ordem, seja crescente ou decrescente. No caso, com o parâmetro `ascending=False`, as barras ficarão em ordem decrescente de valores únicos.

```
1 # Visualizando a informação acima de forma gráfica em ordem decrescente
2 fig, ax = plt.subplots()
3
4 ax.bar(base.nunique().sort_values(ascending=False).index, base.nunique().sort_values(ascending=False).values)
5
6 ax.tick_params(axis='x',labelrotation=75)
7
8 plt.show()
```

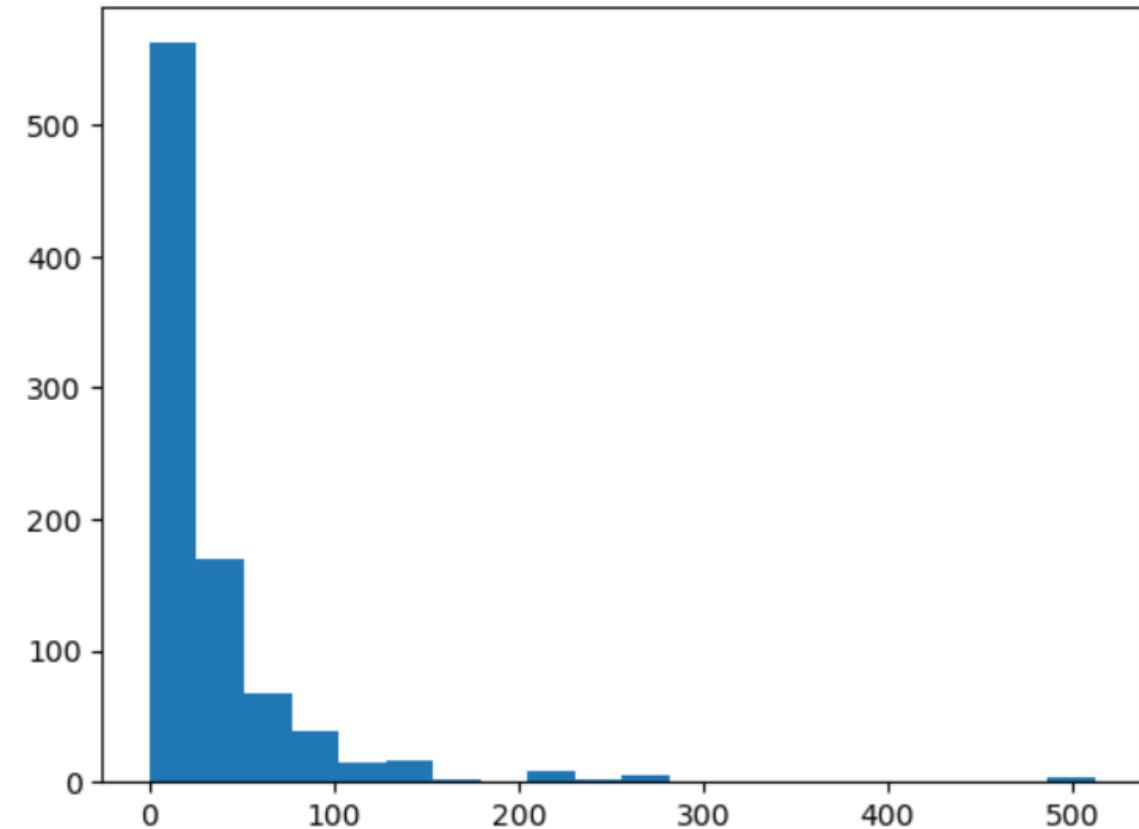


## Módulo 14 – Criando visualizações

Uma análise interessante é a de preço das passagens. O Matplotlib possui o método `hist` para construir histogramas. No código abaixo, construímos um histograma solicitando que os valores da coluna `Fare` sejam divididos em 20 grupos (`bins`) de igual largura.

```
1 # Verificando o histograma das tarifas
2 fig, ax = plt.subplots()
3
4 ax.hist(base.Fare, bins=20)
5
6 plt.show()
```

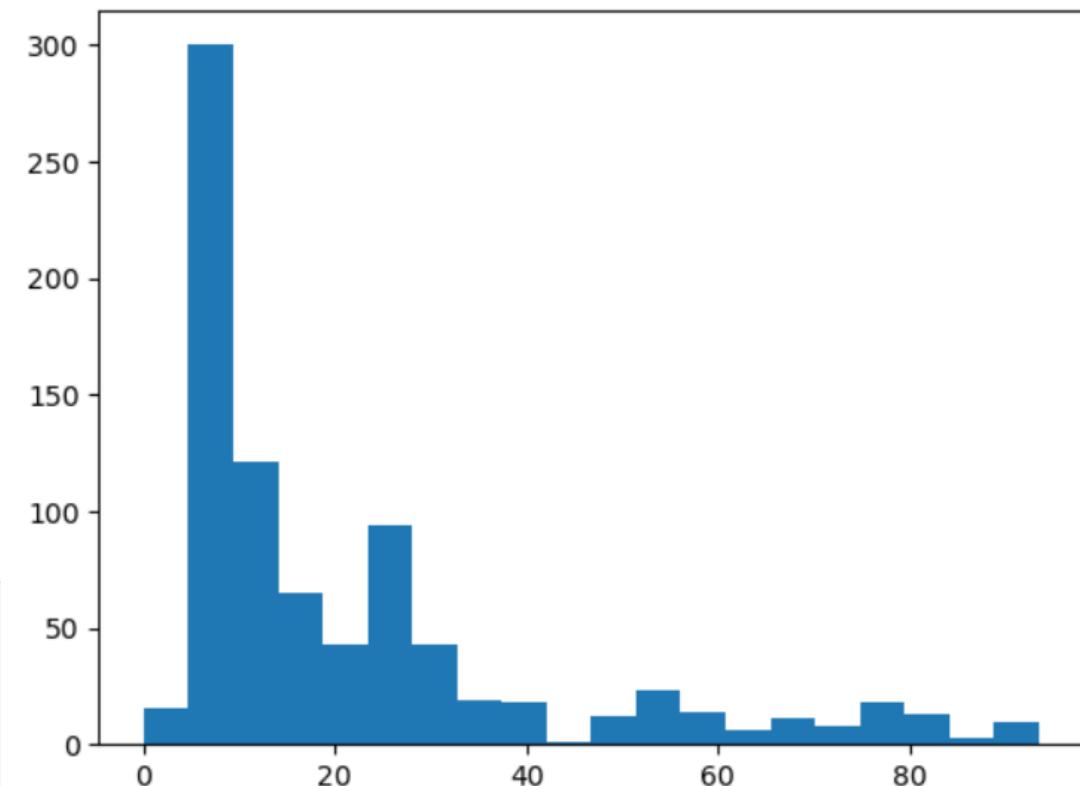
Observe como há uma grande concentração de valores abaixo de 100. Podemos, então, avaliar com mais detalhes essa faixa de valores.



## Módulo 14 – Criando visualizações

Com o conhecimento de Pandas que já adquirimos em outros módulos, podemos aplicar um filtro na coluna *Fare* de forma a pegar apenas valores menores do que 100. Assim, teremos um histograma para essa faixa de valores, permitindo um maior detalhamento. Veja como há uma maior concentração em valores abaixo de 20, com o pico por volta de 5.

```
1 # Verificando o histograma das tarifas apenas para tarifas menores que 100
2 fig, ax = plt.subplots()
3
4 ax.hist(base[base.Fare <100].Fare, bins=20)
5
6 plt.show()
```



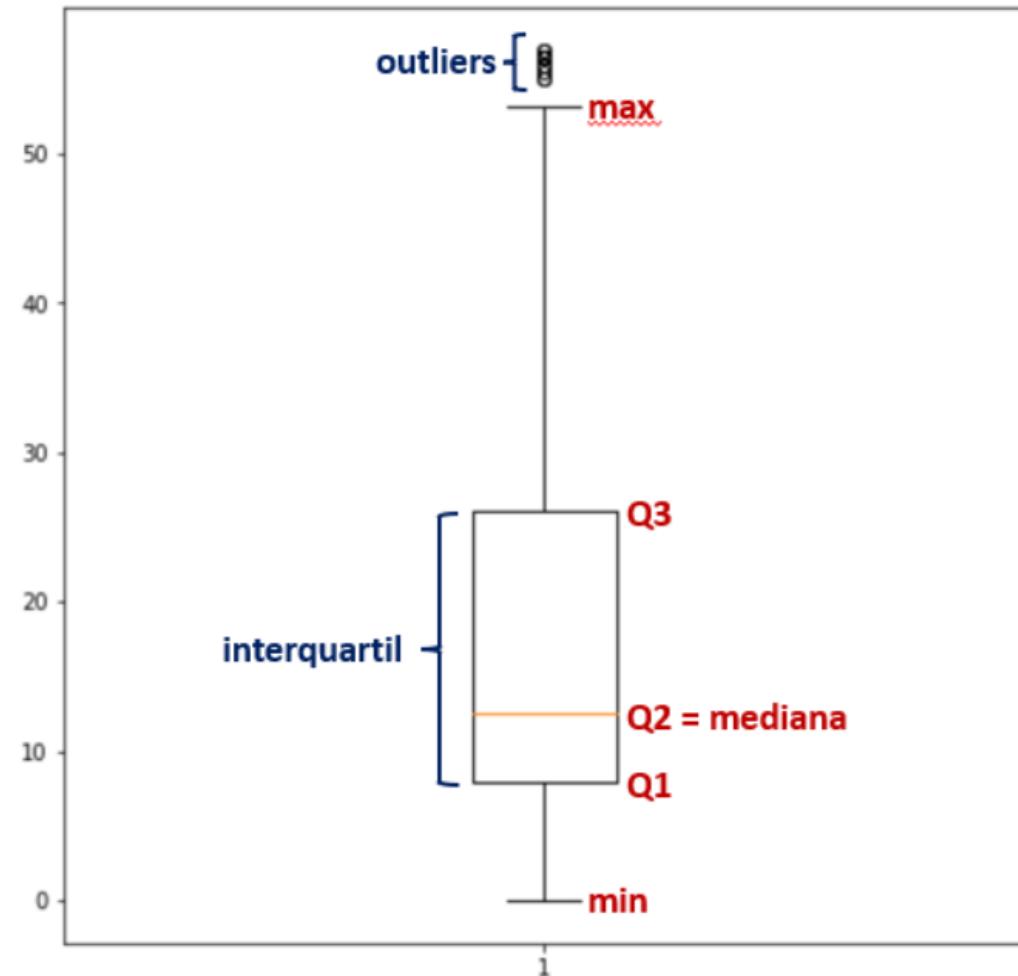
## Módulo 14 – Criando visualizações

Vimos que o método `describe` fornece um resumo estatístico das colunas numéricas de nossa base na forma de tabela. Um gráfico **boxplot** é uma maneira visual de obter boa parte destas informações estatísticas.

Em um boxplot temos um resumo visual da distribuição dos dados a partir de 5 valores numéricos: limite mínimo, quartis 1, 2 (mediana) e 3, e limite máximo.

A amplitude interquartil, AIQ, é a diferença  $Q3 - Q1$ , representando 50 % de todos os valores observados, concentradas na tendência central dos valores.

Os chamados *outliers* são valores que se encontram abaixo ou acima de  $1,5 \times AIQ$ . No exemplo ao lado, temos outliers superiores, acima do limite máximo.

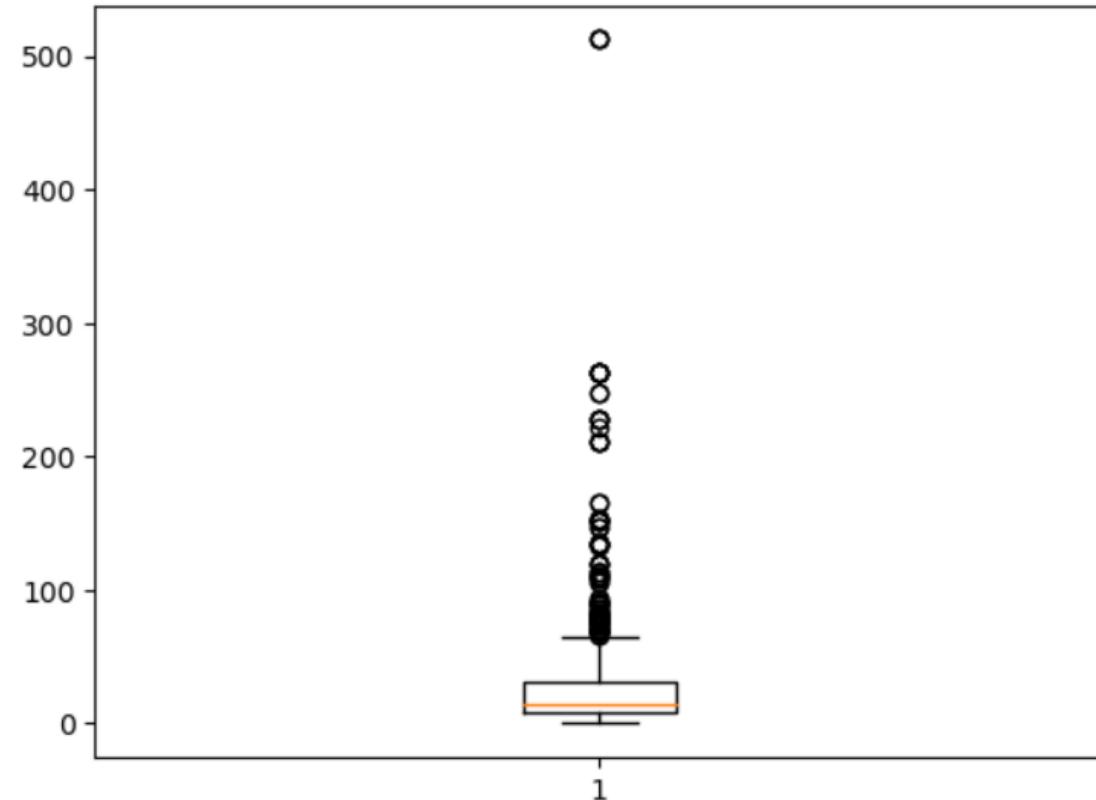


## Módulo 14 – Criando visualizações

O Matplotlib possui o método `boxplot`, para o qual podemos passar a coluna da nossa base que queremos estudar.

Veja se consegue interpretar o boxplot com base no que foi resultado do método `describe` anteriormente e com base no histograma já visto. Na próxima página, esta comparação será feita e você pode verificar se sua interpretação foi correta.

```
1 # Verificando para a coluna Fare
2 fig, ax = plt.subplots()
3
4 ax.boxplot(base.Fare)
5
6 plt.show()
```

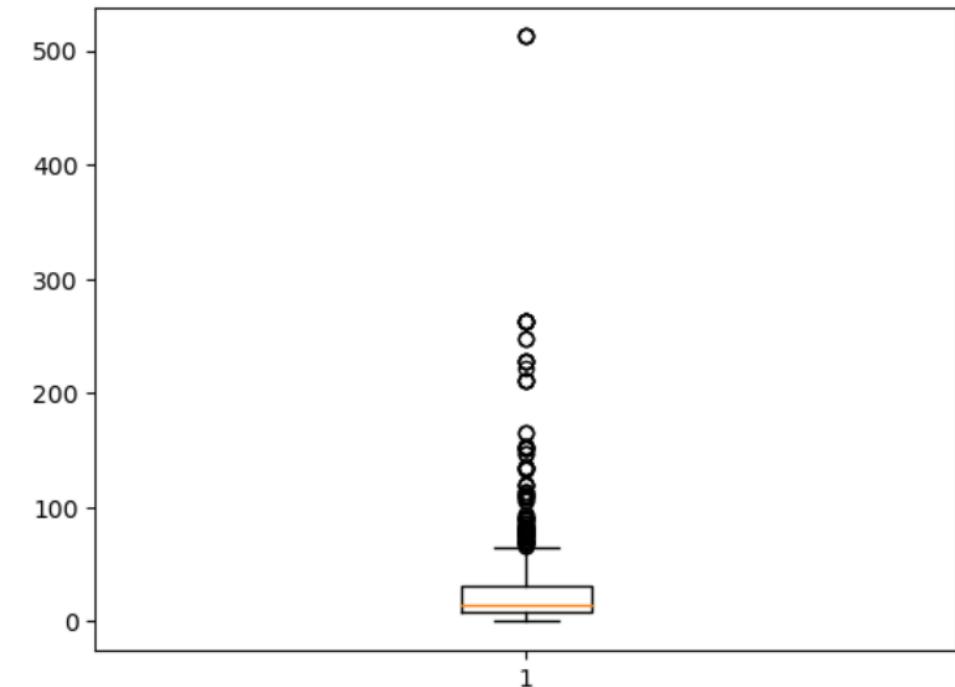


## Módulo 14 – Criando visualizações

Compare o gráfico com a tabela do `describe` que vimos anteriormente. Verifique que há uma concordância entre os valores da tabela e as linhas do boxplot. Observe, também, que o boxplot indica que mais de 50 % dos valores são baixos, algo que o histograma já nos dava indícios. Comece a se acostumar a reconhecer essa complementariedade das diferentes visualizações.

```
1 # Podemos acessar qualquer um dos valores do .describe()  
2 base.Fare.describe()
```

```
count    891.000000  
mean     32.204208  
std      49.693429  
min      0.000000  
25%     7.910400  
50%     14.454200  
75%     31.000000  
max     512.329200  
Name: Fare, dtype: float64
```



## Módulo 14 – Criando visualizações

Na página onde apresentamos um boxplot, falamos do conceito de amplitude interquartil e de como ele se relaciona com o conceito de outliers. Podemos calcular esta amplitude e verificar qual seria o limite máximo de valor que não se considera outlier:

```
1 # Determinando o interquartil  
2 Q1 = base.Fare.describe()['25%']  
3 Q3 = base.Fare.describe()['75%']  
4 interquartil = Q3 - Q1  
5  
6 # Calculando o valor máximo  
7 vlr_max = Q3 + 1.5*interquartil
```

Com isso, podemos olhar quais entradas de nossa base são outliers.

## Módulo 14 – Criando visualizações

Para olhar os outliers, basta filtrar com o Pandas por valores acima deste limite máximo:

```
1 # Agora filtrando os valores acima do máximo
2 base[base.Fare > vlr_max]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Fortune, Mr. Charles Alexander	female male	38.0 19.0	1 3	0 2	PC 17599 19950	71.2833 263.0000	C85 C23 C25 C27	C S
27	28	0	1	Spencer, Mrs. William Augustus (Marie Eugenie) Meyer, Mr. Edgar Joseph	female male	Nan 28.0	1 1	0 0	PC 17569 PC 17604	146.5208 82.1708	B78 NaN	C C
31	32	1	1	Harper, Mrs. Henry Sleeper (Myra Haxton)	female	49.0	1	0	PC 17572	76.7292	D33	C
34	35	0	1	Sage, Mr. Douglas Bullen	male	Nan	8	2	CA. 2343	69.5500	Nan	S
52	53	1	1	Goldenberg, Mrs. Samuel L (Edwiga Grabowska) Wick, Mrs. George Dennick (Mary Hitchcock)	female female	NaN 45.0	1 1	0 1	17453 36928	89.1042 164.8667	C92 NaN	C S
...	...	...	...	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.5500	Nan	S
846	847	0	3	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583	C50	C
849	850	1	1									
856	857	1	1									
863	864	0	3									
879	880	1	1									

116 rows × 12 columns

## Módulo 14 – Criando visualizações

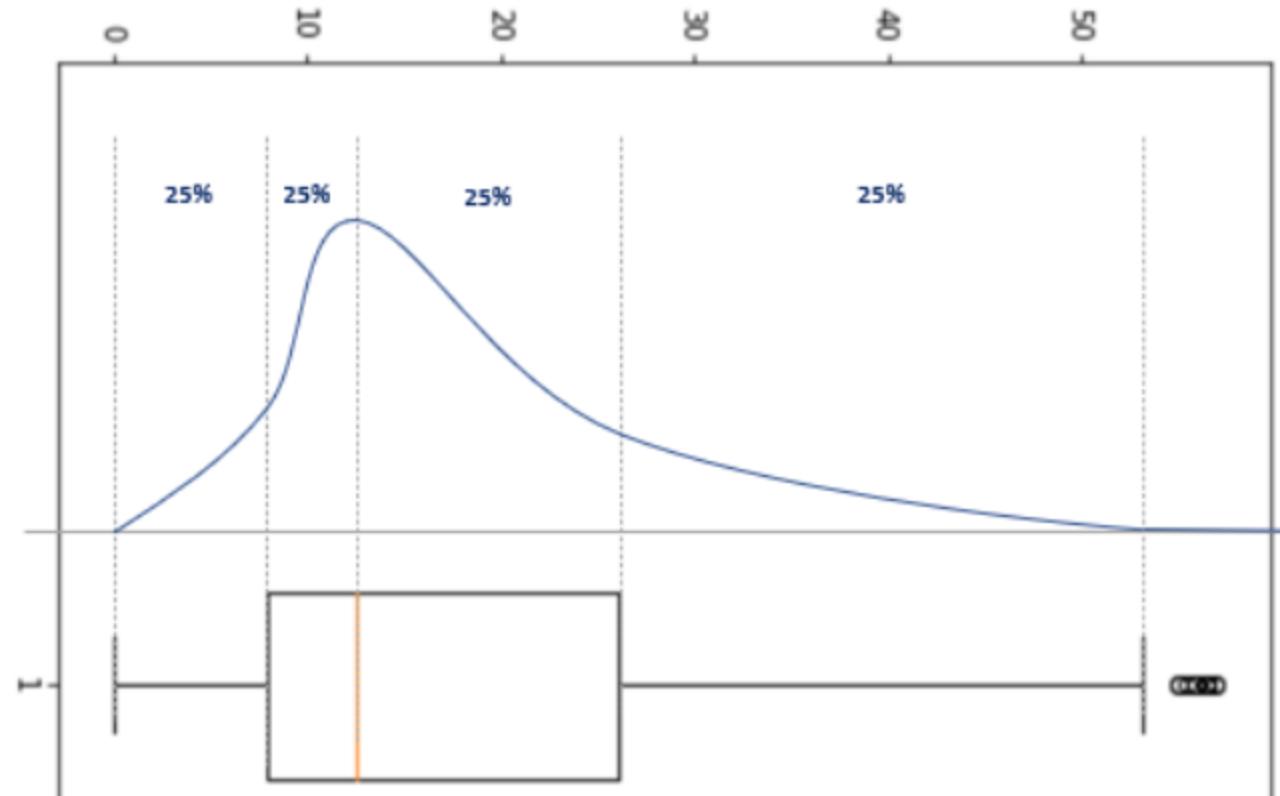
Também podemos olhar para cada intervalo entre os quartis de nossos valores:

```
1 # Verificando a quantidade de registros em cada intervalo (<Q1, entre Q1 e Q2, entre Q2 e Q3,>Q3)
2 Q2 = base.Fare.describe()['50%']
3 base[(base.Fare > Q2) & (base.Fare < Q3)]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230433	26.0000	NaN	S
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250	NaN	Q
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S

## Módulo 14 – Criando visualizações

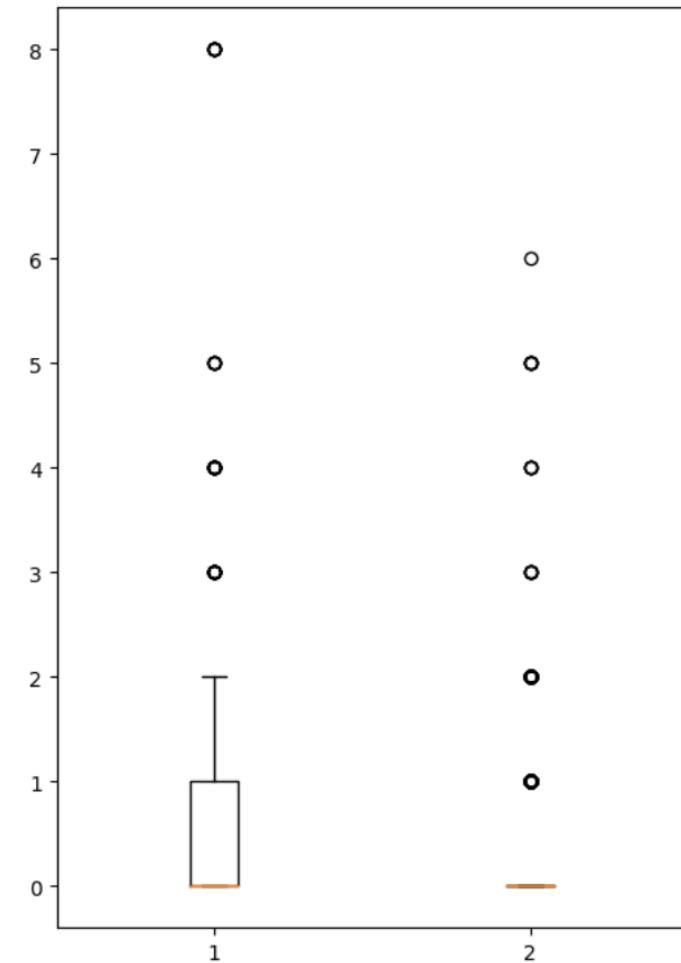
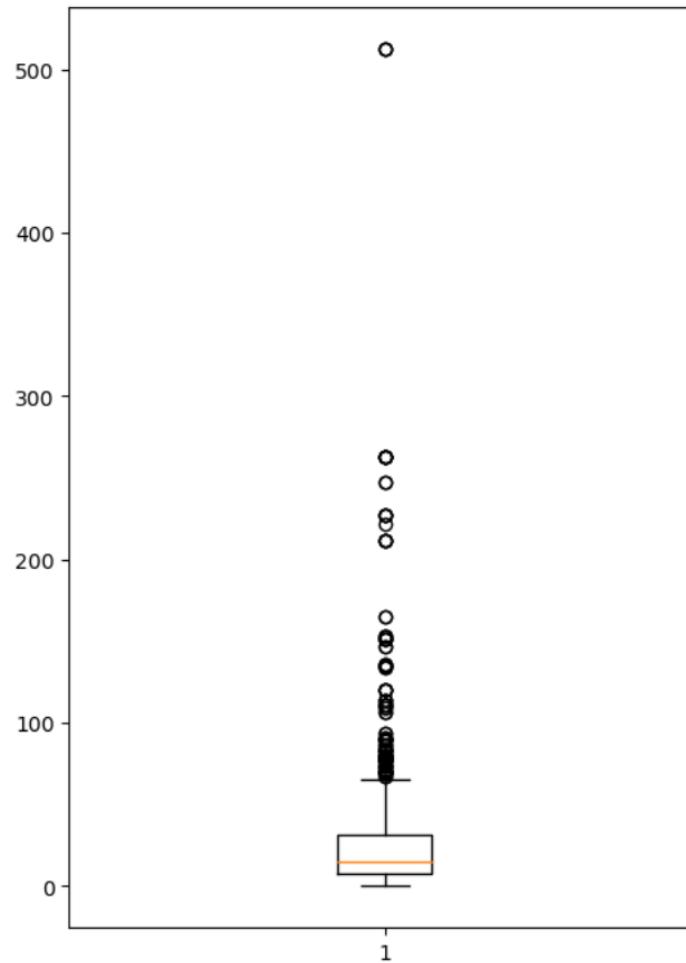
O nome “quartil” vem do fato de estarmos dividindo nossa base em 4 partes. Cada parte contendo 25 % dos dados, como mostrado a seguir. A curva azul indica a densidade dos dados, e veremos como a construir mais adiante.



## Módulo 14 – Criando visualizações

Tudo que fizemos pode ser reproduzido para outras colunas numéricas que desejarmos. Observe para as colunas referentes a familiares a bordo e compare com o resumo estatístico fornecido pelo `describe` para cada coluna.

```
1 # Verificando para as colunas SibSp e Parch
2 fig, ax = plt.subplots(ncols=2,figsize=(12,8))
3
4 ax[0].boxplot(base.Fare)
5 ax[1].boxplot([base.SibSp,base.Parch])
6
7 plt.show()
```



## Módulo 14 – Criando visualizações

Como já vimos, o `describe` pode mostrar um resumo estatístico por coluna. Para uma simples contagem de cada valor possível em uma coluna, pode-se utilizar o método `value_counts`. Perceba que, para a coluna `SibSp`, parte significativa das entradas são para os valores 0 e 1, justificando o boxplot visto na página anterior.

```
1 # Verificando o resumo estatístico para essa coluna  
2 base.SibSp.describe()
```

```
count    891.000000  
mean      0.523008  
std       1.102743  
min      0.000000  
25%     0.000000  
50%     0.000000  
75%     1.000000  
max      8.000000  
Name: SibSp, dtype: float64
```

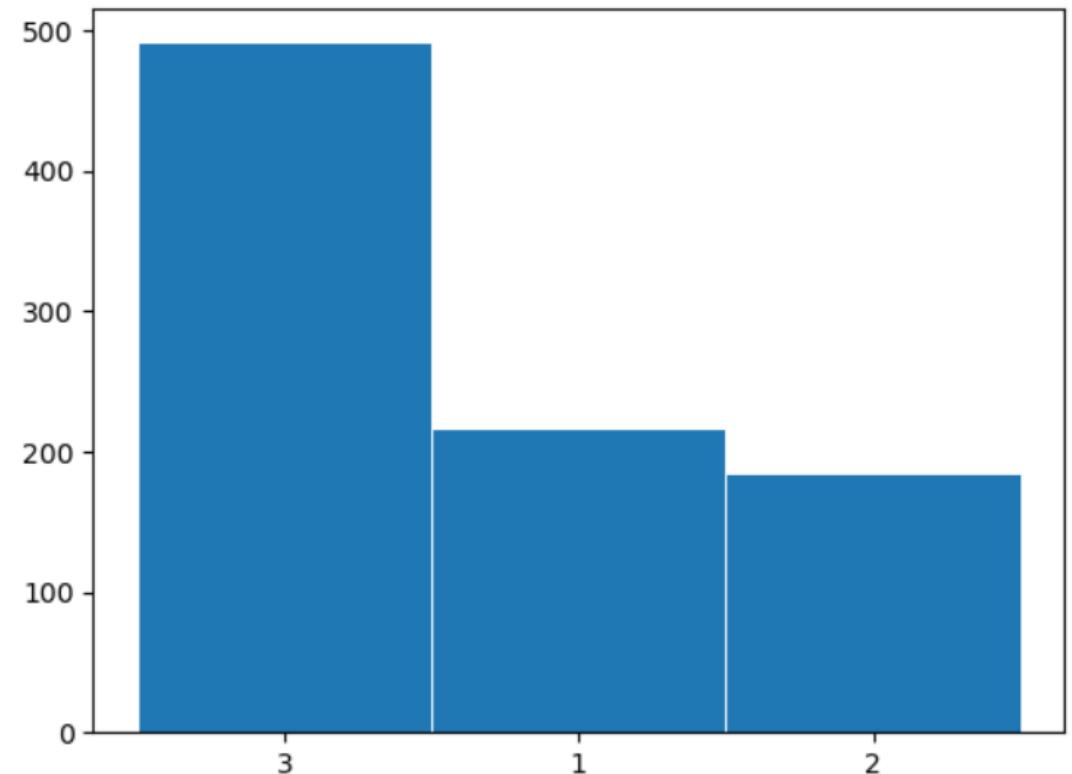
```
1 # Verificando o value_counts()  
2 base.SibSp.value_counts()
```

```
0      608  
1      209  
2      28  
4      18  
3      16  
8       7  
5       5  
Name: SibSp, dtype: int64
```

## Módulo 14 – Criando visualizações

Com o `value_counts`, podemos construir gráficos de barras para as colunas de interesse. Por exemplo, podemos ter uma visualização mostrando qual a classe que tinha mais passageiros.

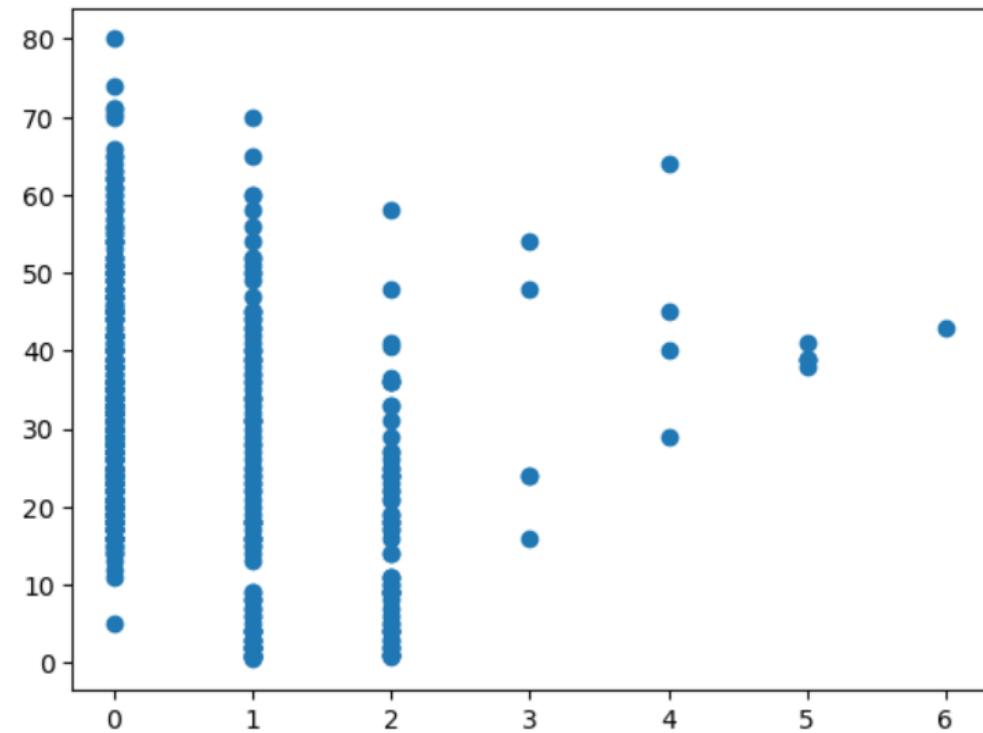
```
1 # Gerando um gráfico de barras
2 fig, ax = plt.subplots()
3
4 x=base.Pclass.value_counts().index.astype('str')
5 y=base.Pclass.value_counts().values
6
7 ax.bar(x, y, width=1, edgecolor="white", linewidth=0.7)
8
9 plt.show()
```



## Módulo 14 – Correlação entre colunas

Em Ciência de Dados, uma parte importante da análise é a de relação entre colunas. Um bom gráfico para isso é o de dispersão, *scatter* em inglês. Com ele, podemos ver o número de registros a partir de pontos. Por exemplo, podemos verificar que famílias maiores usualmente possuem alguém com mais idade e são menos frequentes:

```
1 # Verificando se existe relação entre a coluna Parch e a idade
2 fig, ax = plt.subplots()
3
4 x=base.Parch
5 y=base.Age
6
7 ax.scatter(x, y)
8
9 plt.show()
```



## Módulo 14 – Correlação entre colunas

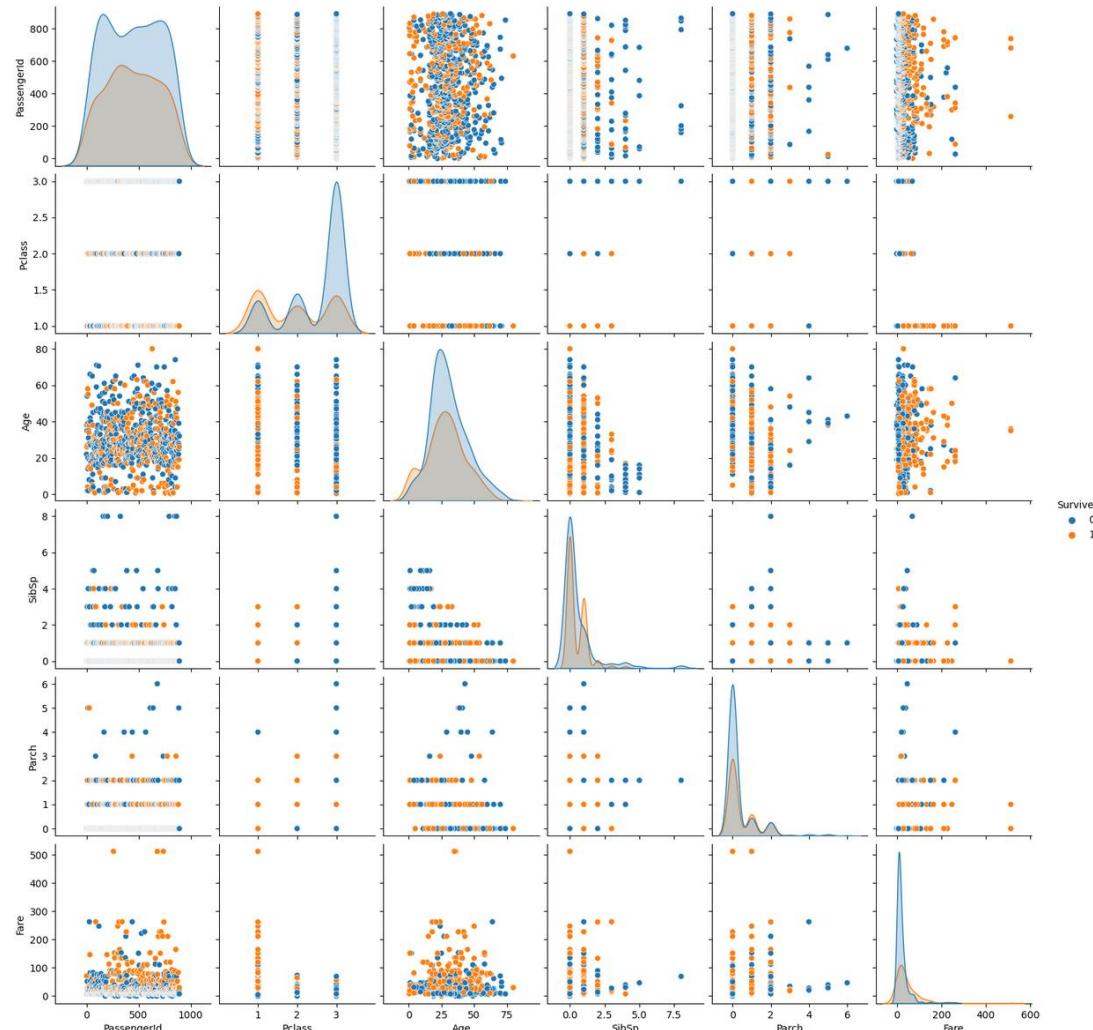
Ao invés de fazer esta análise coluna a coluna, podemos utilizar o método `scatter_matrix`, que constrói esta análise para os pares de colunas da base de dados. Na diagonal, onde haveria repetição das colunas, é mostrado um histograma dos valores da coluna em questão.

```
1 # Utilizando o pandas
2 pd.plotting.scatter_matrix(base, figsize=(16,12))
3 plt.show()
```



## Módulo 14 – Correlação entre colunas

Um dos aspectos mais interessantes desta base de dados é entender quais aspectos foram os mais importantes para a sobrevivência dos passageiros quando houve o acidente. O Seaborn possui o método `pairplot`, que também faz uma matriz de gráficos de dispersão, mas permite diferenciar os pontos com cores a partir de uma coluna. Assim, podemos passar a coluna `Survived`. Veja como há mais coloração laranja (sobrevivente) em tarifas maiores e na primeira classe.

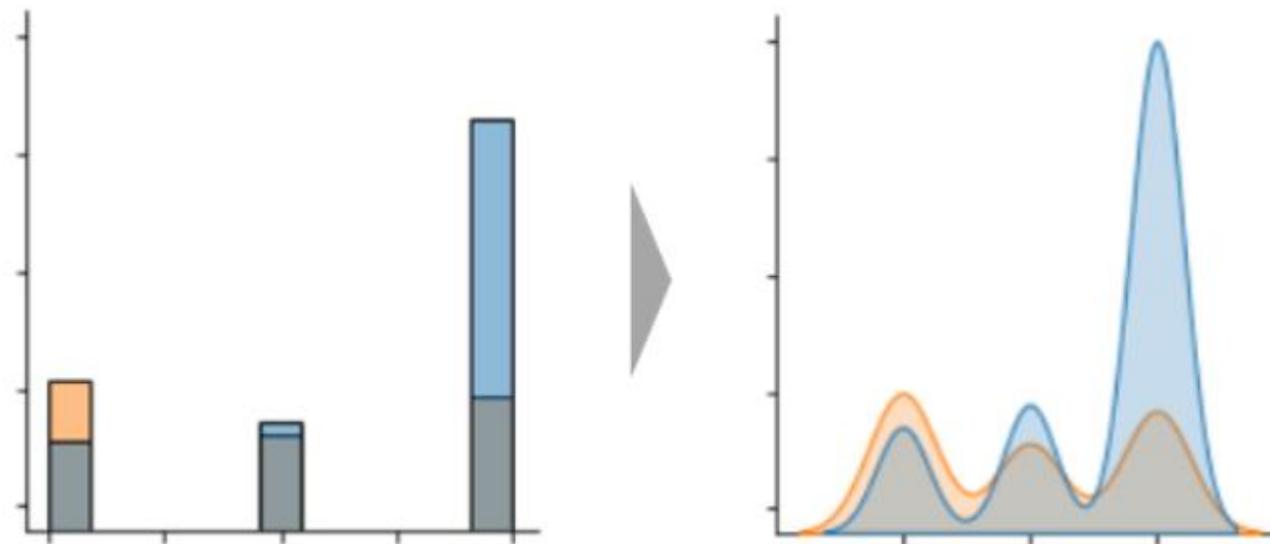


```
1 # Utilizando o seaborn
2 import seaborn as sns
3 sns.pairplot(base,hue="Survived")# ,diag_kind='hist')
4 plt.show()
```

## Módulo 14 – Correlação entre colunas

Observe, na página anterior, que a diagonal não apresenta mais um histograma\* mas, sim, um gráfico KDE. KDE vem do inglês *Kernel Density Estimation*, e mede a chance de uma variável aleatória assumir um determinado valor. De forma simplificada, é como se fosse um histograma suavizado. Observe na figura abaixo um histograma e o KDE correspondente lado a lado.

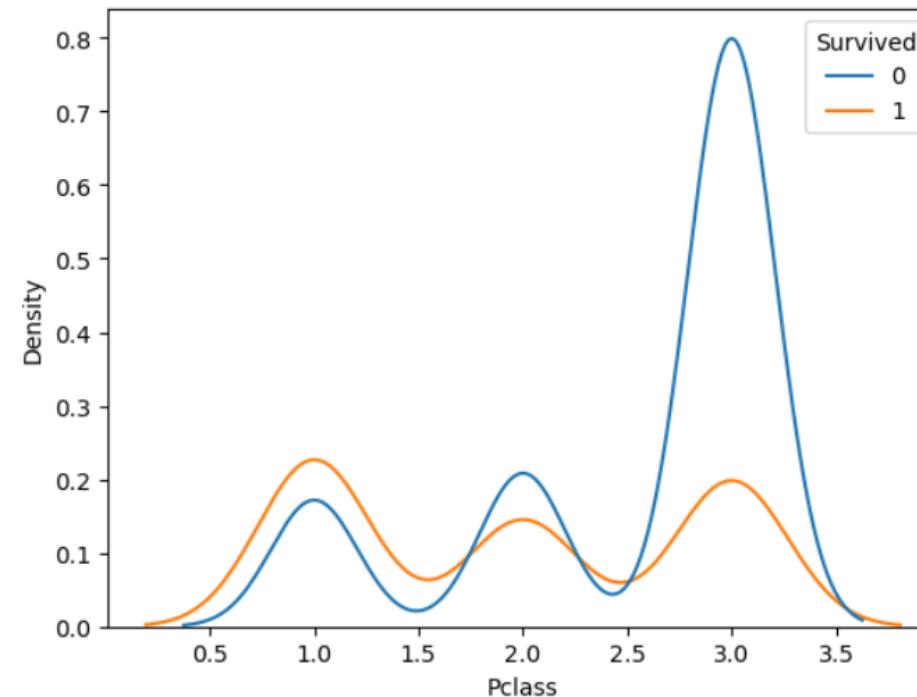
\*Para exibir um histograma, passe o parâmetro `diag_kind='hist'`.



## Módulo 14 – Correlação entre colunas

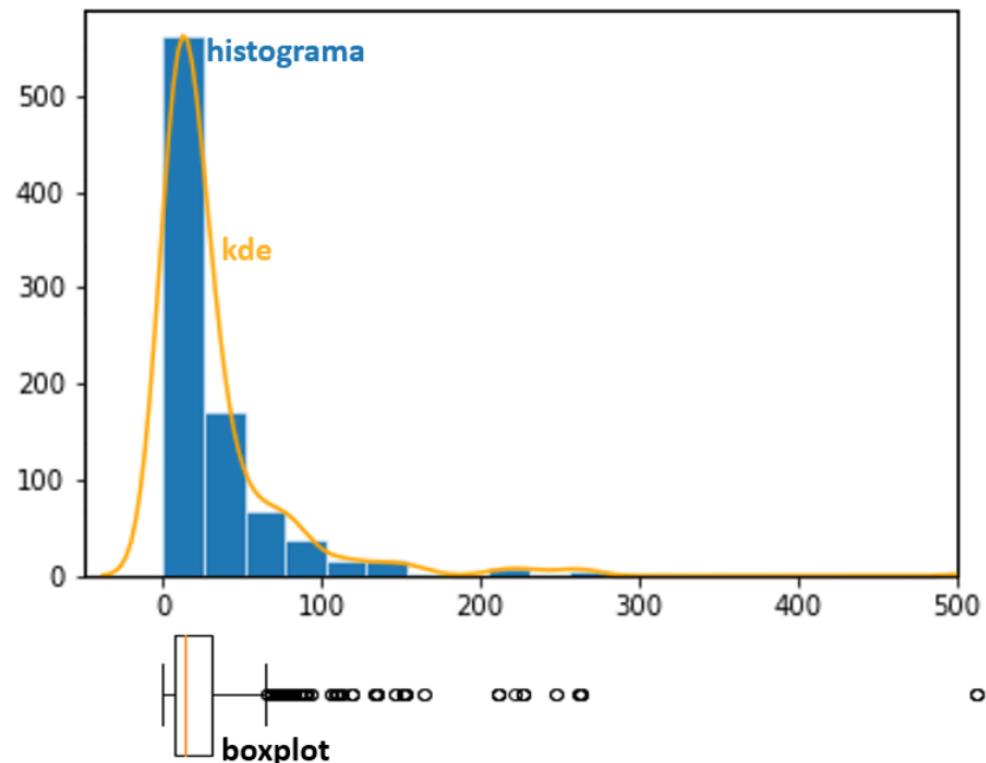
Podemos construir um gráfico KDE para cada coluna com o método `kdeplot`. Por exemplo, podemos ver como é o gráfico KDE da coluna correspondente à classe de cada passageiro, verificando novamente a diferença na sobrevivência ao acidente a depender da classe.

```
1 # Fazendo o kde para o Pclass
2 sns.kdeplot(base.Pclass,hue=base.Survived)
3 plt.show()
```



## Módulo 14 – Correlação entre colunas

O KDE é mais uma visualização que complementa o visto nos histogramas e nos boxplots. Por exemplo, para a coluna *Fare*, vemos que o KDE acompanha o histograma, suavizando-o, e vemos que onde há baixos valores eram os outliers do boxplot.



## Módulo 14 – Correlação entre colunas

Em uma análise exploratória, é sempre importante verificar se as colunas possuem algum tipo de correlação. Para isso, podemos utilizar o método `corr` do Pandas.

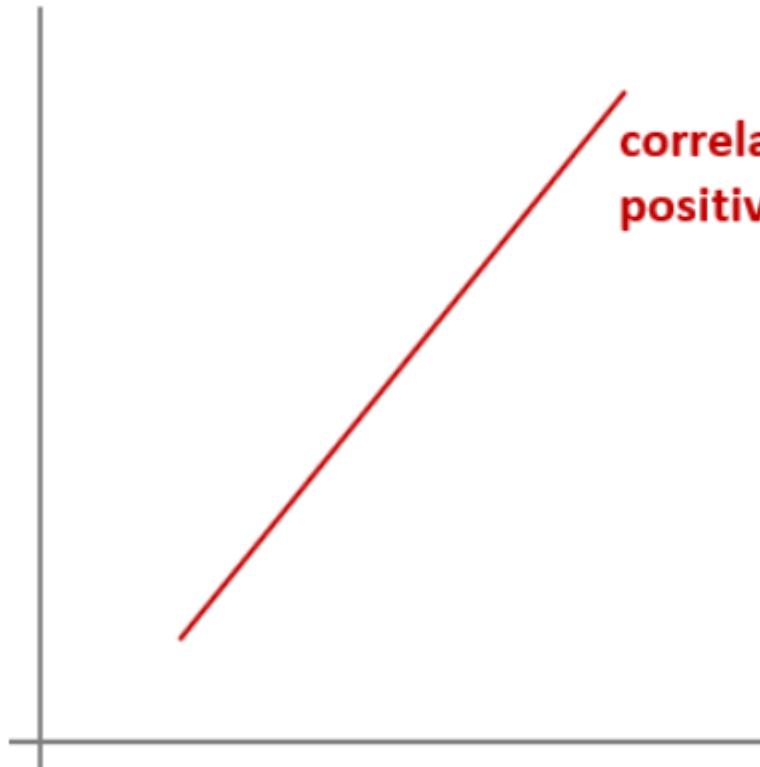
```
1 # Verificando a correlação entre os dados  
2 base.corr()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

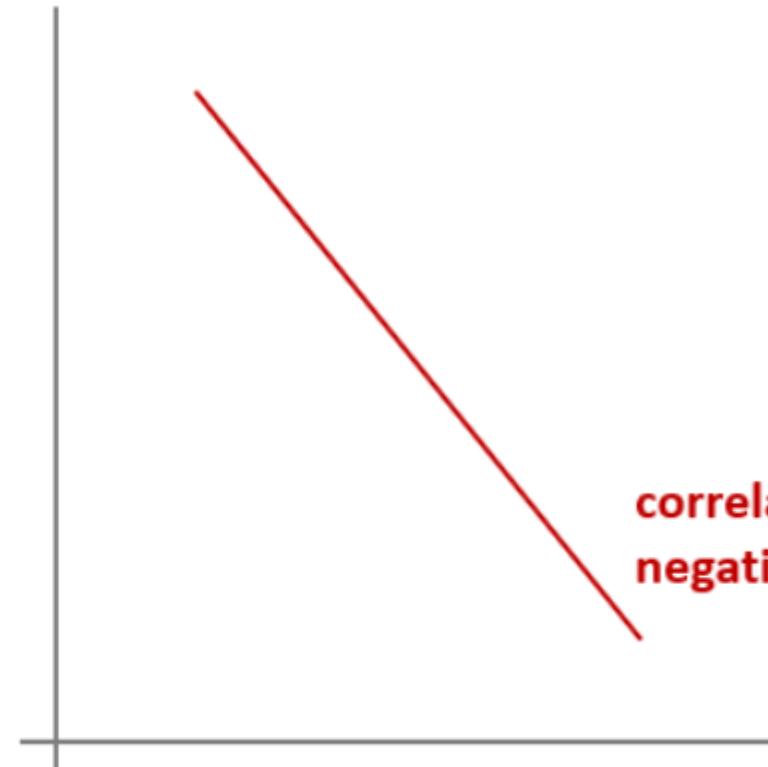
Observe que há valores positivos e negativos. Estes valores podem variar de -1 a +1. Um valor positivo indica correlação positiva (quando um valor aumenta, o outro também aumenta). E um valor negativo indica o oposto (quando um valor aumenta, o outro diminui). Quanto mais se aproximam de -1 ou +1 maior a correlação, seja negativa ou positiva.

## Módulo 14 – Correlação entre colunas

Graficamente, as correlações seriam como a inclinação de uma reta. O valor da inclinação indica o quanto de correlação há entre as variáveis.



**correlação  
positiva**

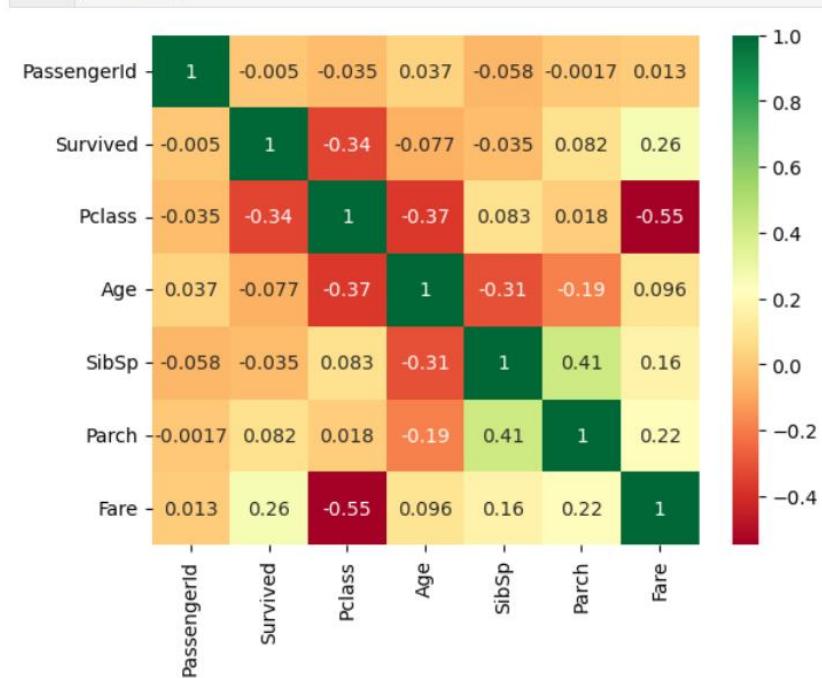


**correlação  
negativa**

## Módulo 14 – Correlação entre colunas

Podemos passar o dataframe de correlação para o método heatmap do Seaborn para facilitar a visualização com cores. As cores indicam a correlação, se positiva ou negativa, e a intensidade mostra a intensidade da correlação. Por exemplo, veja como há uma correlação negativa entre *Pclass* e *Fare*. Faz sentido, pois as passagens mais caras correspondem à classe 1, menor número entre as classes.

```
1 # Tornando visual
2 # cmap: https://matplotlib.org/stable/tutorials/colors/colormaps.html
3 sns.heatmap(base.corr(), annot=True, cmap='RdYlGn')
4 plt.show()
```



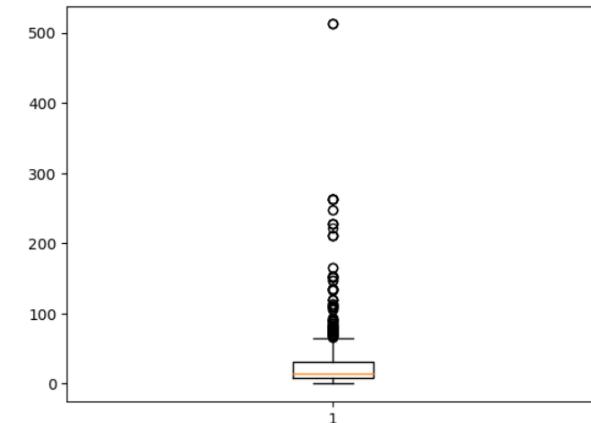
## Módulo 14 – Tratando valores ausentes e outliers

Agora que já vimos o básico de visualizações, vamos abordar outro assunto muito importante de análise de dados, o de tratamento de valores vazios e outliers.

Por exemplo, já vimos que a coluna *Fare* possui outliers. Mas será que são valores que fazem sentido? Procurando no dataframe, vemos que são pessoas da primeira classe, que realmente possui valores maiores, e com nome. Logo, parecem ser registros autênticos e não algum erro de preenchimento da base. Inclusive, se buscarmos na internet o nome dessas pessoas, achamos suas informações e das cabines de luxo.

```
1 # Verificando para a coluna Fare
2 base[base.Fare > 300]
```

```
1 # Verificando para a coluna Fare
2 fig, ax = plt.subplots()
3
4 ax.boxplot(base.Fare)
5
6 plt.show()
```



PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
258	259	1	Ward, Miss. Anna	female	35.0	0	0	PC 17755	512.3292	NaN	C
679	680	1	Cardeza, Mr. Thomas Drake Martinez	male	36.0	0	1	PC 17755	512.3292	B51 B53 B55	C
737	738	1	Lesurer, Mr. Gustave J	male	35.0	0	0	PC 17755	512.3292	B101	C

## Módulo 14 – Tratando valores ausentes e outliers

Seguindo a mesma lógica, podemos ver se há valores muito baixos de *Fare*. Aqui, temos algo estranho, já que há valores 0 mesmo para primeira classe.

1	base[base.Fare < 1]												
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
179	180	0	3	Leonard, Mr. Lionel	male	36.0	0	0	LINE	0.0	NaN	S	
263	264	0	1	Harrison, Mr. William	male	40.0	0	0	112059	0.0	B94	S	
271	272	1	3	Tornquist, Mr. William Henry	male	25.0	0	0	LINE	0.0	NaN	S	
277	278	0	2	Parkes, Mr. Francis "Frank"	male	NaN	0	0	239853	0.0	NaN	S	
302	303	0	3	Johnson, Mr. William Cahoon Jr	male	19.0	0	0	LINE	0.0	NaN	S	
413	414	0	2	Cunningham, Mr. Alfred Fleming	male	NaN	0	0	239853	0.0	NaN	S	
466	467	0	2	Campbell, Mr. William	male	NaN	0	0	239853	0.0	NaN	S	
481	482	0	2	Frost, Mr. Anthony Wood "Archie"	male	NaN	0	0	239854	0.0	NaN	S	
597	598	0	3	Johnson, Mr. Alfred	male	49.0	0	0	LINE	0.0	NaN	S	
633	634	0	1	Parr, Mr. William Henry Marsh	male	NaN	0	0	112052	0.0	NaN	S	
674	675	0	2	Watson, Mr. Ennis Hastings	male	NaN	0	0	239856	0.0	NaN	S	
732	733	0	2	Knight, Mr. Robert J	male	NaN	0	0	239855	0.0	NaN	S	
806	807	0	1	Andrews, Mr. Thomas Jr	male	39.0	0	0	112050	0.0	A36	S	
815	816	0	1	Fry, Mr. Richard	male	NaN	0	0	112058	0.0	B102	S	
822	823	0	1	Reuchlin, Jonkheer. John George	male	38.0	0	0	19972	0.0	NaN	S	

## Módulo 14 – Tratando valores ausentes e outliers

Provavelmente estes valores não são corretos, talvez na realidade não se tinha essa informação e foi colocado zero no lugar. Logo, podemos substituir esses valores por algo mais razoável como, por exemplo, a média:

```
1 # Calculando a média  
2 mean_Fare = base.Fare.mean()
```

```
1 # Substituindo esses valores  
2 base.loc[base.Fare < 1, 'Fare'] = mean_Fare
```

Após a substituição, não mais registros desses valores baixos:

```
1 base[base.Fare < 1]
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
-------------	----------	--------	------	-----	-----	-------	-------	--------	------	-------	----------

## Módulo 14 – Tratando valores ausentes e outliers

Anteriormente, vimos que há valores vazios/nulos na coluna Age:

```
1 # Verificando a base  
2 base[base.Age.isnull()]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.0000	NaN	S
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.2250	NaN	C
26	27	0	3	Emir, Mr. Farred Chehab	male	NaN	0	0	2631	7.2250	NaN	C
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	330959	7.8792	NaN	Q
...	...	...	...	...	...	...	...	...	...	...	...	...
859	860	0	3	Razi, Mr. Raihed	male	NaN	0	0	2629	7.2292	NaN	C
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.5500	NaN	S
868	869	0	3	van Melkebeke, Mr. Philemon	male	NaN	0	0	345777	9.5000	NaN	S
878	879	0	3	Laleff, Mr. Kristo	male	NaN	0	0	349217	7.8958	NaN	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S

177 rows × 12 columns

## Módulo 14 – Tratando valores ausentes e outliers

Podemos verificar se há mais valores ausentes a depender do status de sobrevivência:

```
1 # Entendendo melhor os valores
2 base.loc[base.Age.isnull(),'Survived'].value_counts()

0    125
1     52
Name: Survived, dtype: int64
```

Veja como há mais valores ausentes entre os que não sobreviveram.

Assim como anteriormente, vamos substituir estes valores ausentes pela média da coluna:

```
1 # Calculando a média das idades
2 base.Age.mean()
```

29.69911764705882

```
1 # Substituindo os valores nulos por essa média
2 base.loc[base.Age.isnull(),'Age'] = base.Age.mean()
```

```
1 # Verificando novamente a base
2 base[base.Age.isnull()]
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
-------------	----------	--------	------	-----	-----	-------	-------	--------	------	-------	----------

É importante ressaltar que pode-se escolher outro critério de substituição que não seja pela média.

## Módulo 14 – Pandas Profiling

Tudo que vimos até agora é o básico que deve ser feito em todo início de uma análise de base de dados. E é muito importante ter esses fundamentos. Mas, se é algo que sempre se faz, é natural que tenham desenvolvido uma biblioteca que automatiza esta análise toda. É a biblioteca *ydata-profiling*, mais conhecida como *Pandas Profiling*, já que ela mudou de nome há pouco tempo. Tal biblioteca gera um relatório com uma análise exploratória básica de toda a base de dados.

Para instalar, basta usar `pip install -U ydata-profiling`

Após instalar, importe a classe `ProfileReport`:

```
1 # Importando o pandas e o pandas profiling
2 import pandas as pd
3 from pandas_profiling import ProfileReport
```

Então, importe a base de dados para o Pandas e a passe para o `ProfileReport`, com um título de sua escolha para o relatório:

```
1 # Importando a base de dados
2 base = pd.read_csv('train.csv')
```

```
1 # Gerando o relatório
2 profile = ProfileReport(base, title="Pandas Profiling Report")
```

# Módulo 14 – Pandas Profiling

Agora, basta chamar a variável criada e um relatório será apresentado dentro de seu notebook:

```
1 # Visualizar o relatório
2 profile
```

Summarize dataset: 100% 47/47 [00:03<00:00, 8.19it/s, Completed]

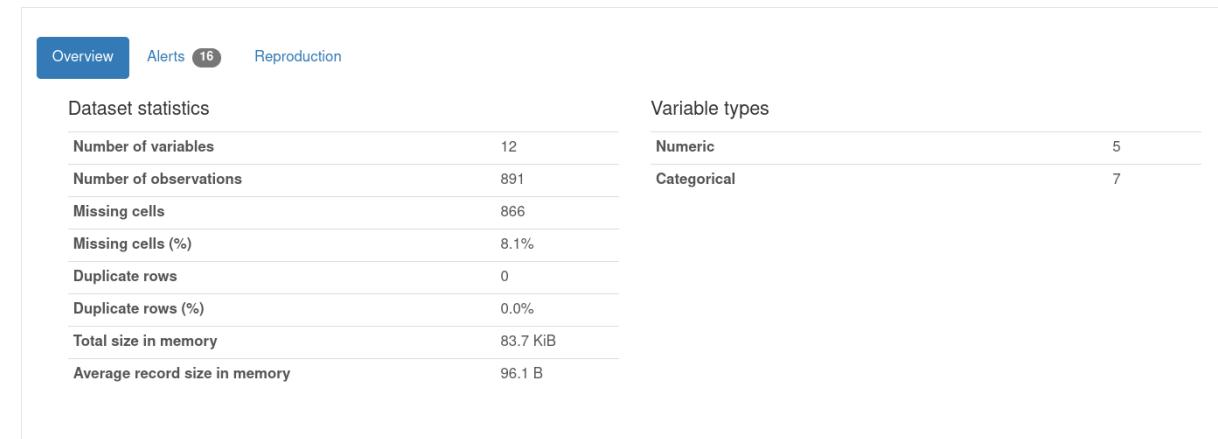
Generate report structure: 100% 1/1 [00:02<00:00, 2.47s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.32s/it]

Pandas Profiling Report

Overview Variables Interactions Correlations Missing values Sample

## Overview



## Módulo 14 – Pandas Profiling

O relatório pode ser extenso e seria melhor abri-lo em um arquivo a parte. A estrutura do relatório é em HTML, de forma que podemos usar o método `to_file` para exportar para um arquivo HTML que pode ser aberto em um navegador:

```
1 profile.to_file("análise.html")
```

Export report to file: 100%

1/1 [00:00<00:00, 22.63it/s]

Após o final da exportação, abra o arquivo em seu navegador de preferência.

# Módulo 14 – Pandas Profiling

O relatório gerado tem algumas seções. Recomendamos fortemente que veja todas com detalhes. Aqui iremos apresentar o básico de cada seção, destacando os conceitos que vimos passo a passo no decorrer deste módulo.

A primeira seção é um resumo, mostrando o número de colunas, de linhas, de valores ausentes e de duplicatas. Também mostra o tipo das colunas presentes.

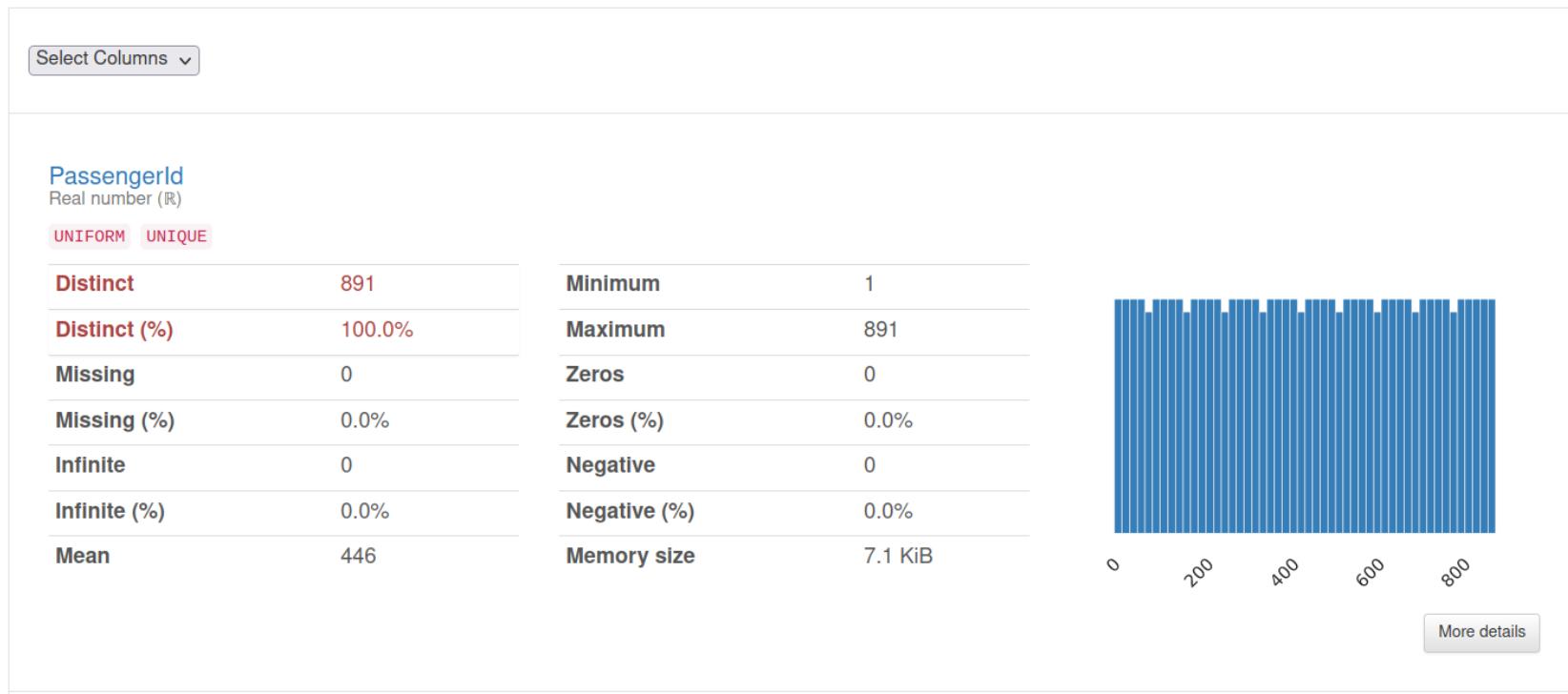
## Overview

Overview	Alerts 16	Reproduction
Dataset statistics		Variable types
<b>Number of variables</b>		5
<b>Number of observations</b>		7
<b>Missing cells</b>		
<b>Missing cells (%)</b>		
<b>Duplicate rows</b>		
<b>Duplicate rows (%)</b>		
<b>Total size in memory</b>		
<b>Average record size in memory</b>		

## Módulo 14 – Pandas Profiling

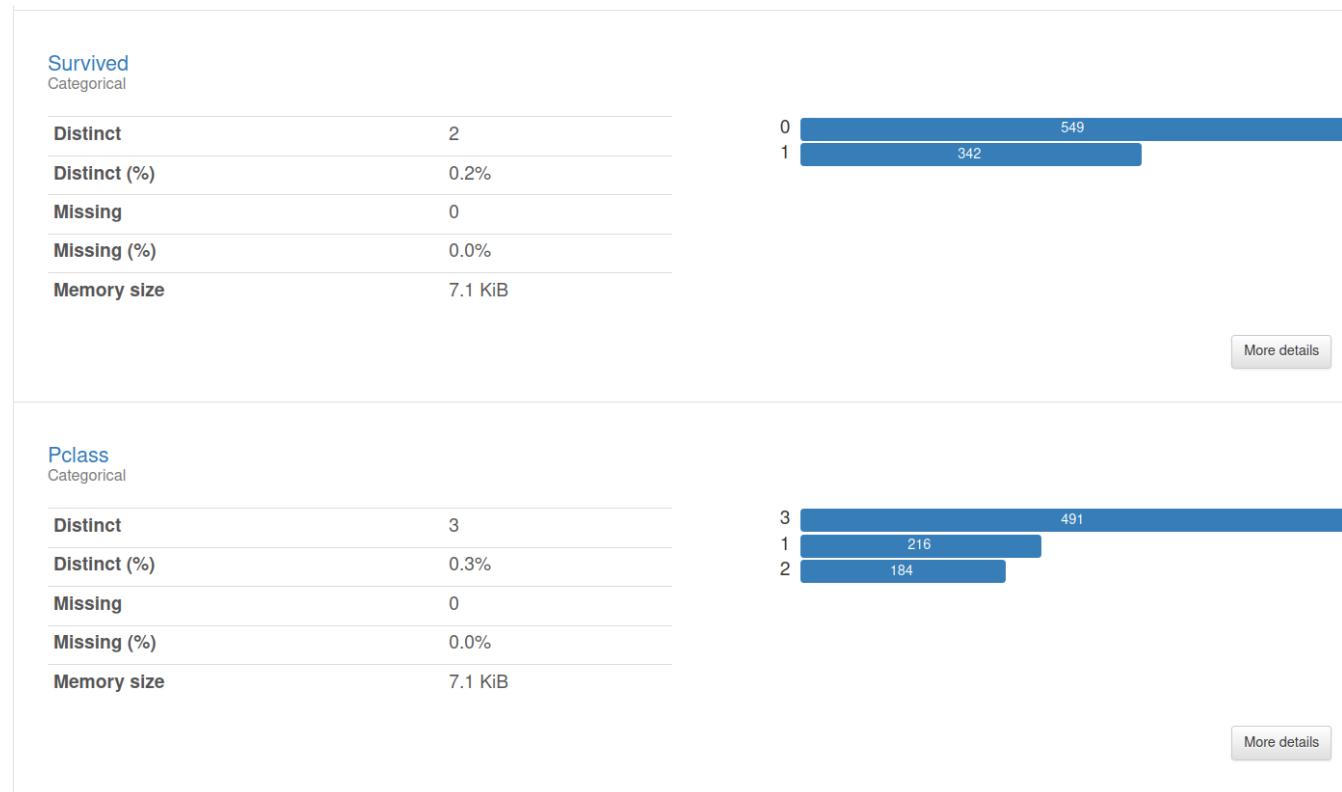
A próxima seção mostra, para cada coluna, um breve descrição estatística. Com valores ausentes e, no caso de colunas numéricas, valores mínimo, máximo, zeros e negativos, e média. Apresenta, também, um histograma. Abaixo, a coluna *PassengerId*.

### Variables



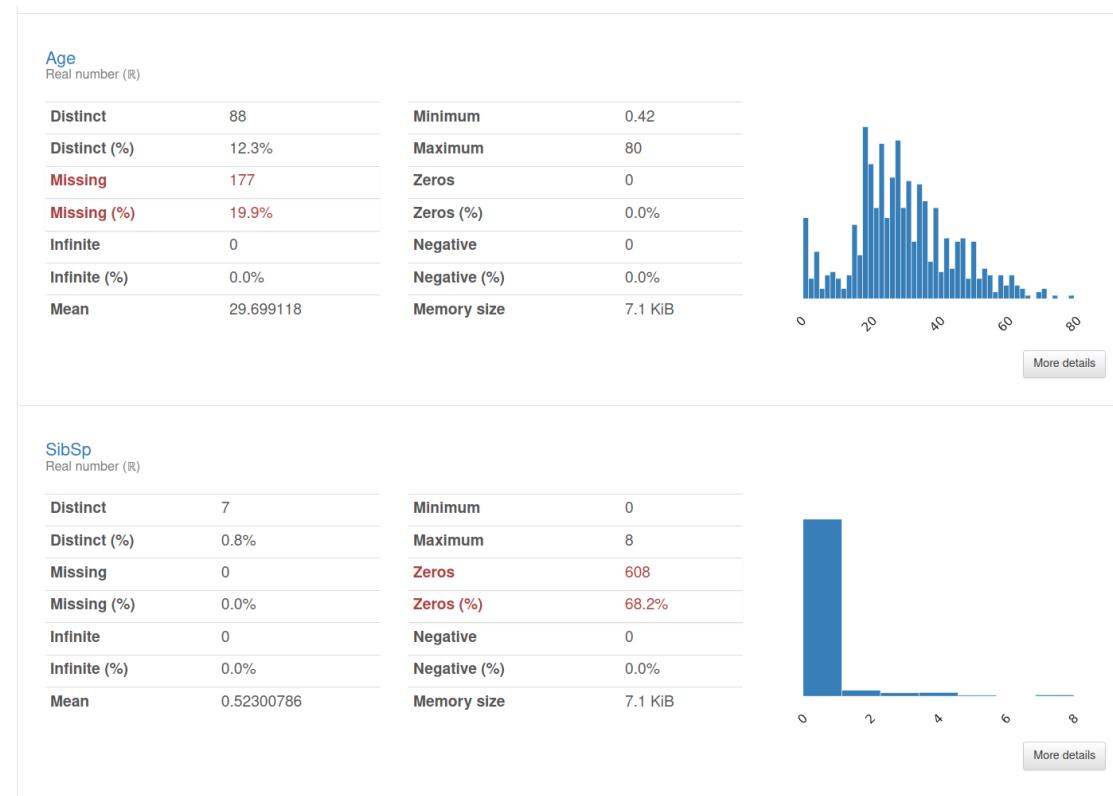
## Módulo 14 – Pandas Profiling

Observe que a biblioteca reconhece quando as colunas são categóricas, ou seja, quando possuem poucos valores que, na realidade, descrevem categorias. Abaixo, vemos as informações das colunas *Survived* e *Pclass*. Os gráficos já mostram claramente que temos mais mortos que sobreviventes e muito mais passageiros da classe 3 que das demais.



## Módulo 14 – Pandas Profiling

Abaixo, temos as colunas *Age* e *SibSp*. Observe que a biblioteca destaca estatísticas que podem ser importantes para o analista. Na coluna *Age*, destacou a quantidade de valores ausentes e, na coluna *SibSp*, a quantidade de valores erro. Desta forma, facilita o trabalho do analista e já incentiva o mesmo a pensar em estratégias para lidar com estes casos, como fizemos no decorrer do módulo.



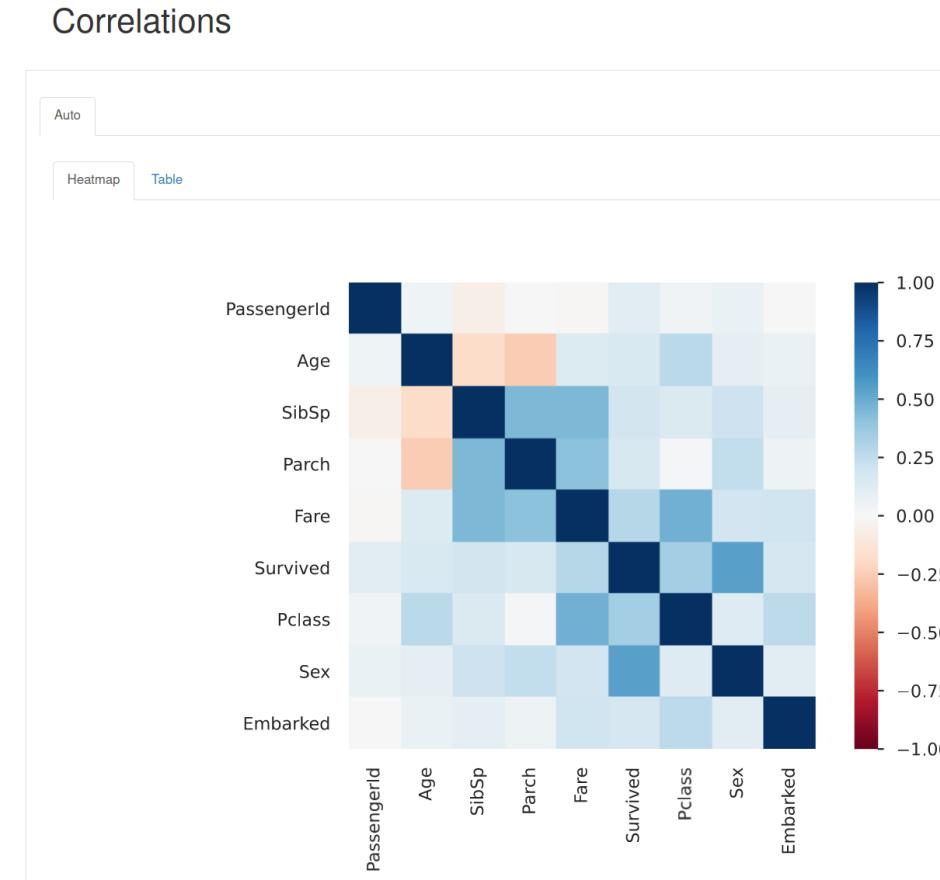
# Módulo 14 – Pandas Profiling

No decorrer do módulo, discutimos sobre correlações e gráficos de dispersão. Na seção *Interactions*, a biblioteca gera gráficos de dispersão para cada par de coluna numérica. Abaixo, um dos gráficos que construímos durante o módulo, construído automaticamente pela biblioteca.



# Módulo 14 – Pandas Profiling

Da mesma forma, constrói automaticamente mapas de calor:



## Módulo 14 – Finalizando a análise e apresentando os resultados

O relatório do Pandas Profiling pode ser personalizado de diversas formas, basta ver a documentação. Pode ser uma boa primeira apresentação aos clientes finais de sua análise.

Também é importante ir documentando sua análise no Jupyter Notebook, usando as células de texto Markdown. Assim, pode-se gerar um relatório bem explicado aos interessados. Neste caso, para apresentar a outros, pode-se remover células de código intermediário, deixando só as importantes para o contexto do cliente final. O Jupyter Notebook permite exportar para diversos formatos como, por exemplo, PDF. Dessa forma, o cliente final pode abrir sem ter a necessidade de precisar instalar o programa.

Tudo que foi visto neste módulo servirá de base aos demais, já que toda nova base deve ser explorada inicialmente, antes de se realizar estudos mais detalhados como aprendizado de máquinas. Assim, treine bem o conteúdo deste módulo! Bons estudos!

## MÓDULO 15

# O Scikit-Learn



# Módulo 15 – Conhecendo o Scikit-Learn

O Scikit-Learn é uma biblioteca open-source essencial para cientistas de dados que buscam implementar e aprimorar modelos de aprendizado de máquina.

Com uma ampla gama de algoritmos e ferramentas de pré-processamento, a biblioteca facilita a criação e a avaliação de modelos eficientes e robustos. Resumindo:

- Biblioteca de aprendizado de máquina em Python
- Ferramentas eficientes e fáceis de usar
- Aplicações em classificação, regressão, clusterização e redução de dimensionalidade
- Baseada em NumPy, SciPy e matplotlib

The screenshot shows the official scikit-learn website. At the top, there's a navigation bar with links for 'Install', 'User Guide', 'API', 'Examples', 'Community', and 'More'. Below the header, there's a main title 'scikit-learn' and a subtitle 'Machine Learning in Python'. A 'Getting Started' button is visible. To the right, there's a list of bullet points: 'Simple and efficient tools for predictive data analysis', 'Accessible to everybody, and reusable in various contexts', 'Built on NumPy, SciPy, and matplotlib', and 'Open source, commercially usable - BSD license'. The page is divided into several sections with sub-sections and examples:

- Classification**: Identifying which category an object belongs to. Applications: Spam detection, image recognition. Algorithms: SVM, nearest neighbors, random forest, and more... Examples: A grid of small plots showing classification results.
- Regression**: Predicting a continuous-valued attribute associated with an object. Applications: Drug response, Stock prices. Algorithms: SVR, nearest neighbors, random forest, and more... Examples: A line plot showing a regression model fit to data points.
- Clustering**: Automatic grouping of similar objects into sets. Applications: Customer segmentation, Grouping experiment outcomes. Algorithms: k-Means, spectral clustering, mean-shift, and more... Examples: A scatter plot with data points grouped into four clusters (red, green, blue, purple) with centroids marked by white crosses.
- Dimensionality reduction**: Reducing the number of random variables to consider. Applications: Visualization, Increased efficiency. Algorithms: PCA, feature selection, non-negative matrix factorization, and more... Examples: A 3D scatter plot of the Iris dataset labeled 'Virginica', 'Versicolor', and 'Setosa'.
- Model selection**: Comparing, validating and choosing parameters and models. Applications: Improved accuracy via parameter tuning. Algorithms: grid search, cross validation, metrics, and more... Examples: A line plot showing cross-validation scores for different models.
- Preprocessing**: Feature extraction and normalization. Applications: Transforming input data such as text for use with machine learning algorithms. Algorithms: preprocessing, feature extraction, and more... Examples: A grid of plots showing various preprocessing steps like scaling and normalization.

# Módulo 15 – Conhecendo o Scikit-Learn

A documentação do Scikit-Learn é notável por sua abrangência e clareza, fornecendo exemplos práticos, explicações detalhadas e orientações passo a passo. Isso facilita o aprendizado e a aplicação efetiva da biblioteca, mesmo para iniciantes na área de Ciência de Dados e aprendizado de máquina.

Em nossos vídeos, sempre mostramos como encontrar as informações na documentação.

Lembrando, se tiver dificuldade em inglês, clique com o botão direito do mouse na página e selecione a opção de tradução do navegador.

The screenshot shows the official scikit-learn documentation website. At the top, there's a navigation bar with links for 'Install', 'User Guide', 'API', 'Examples', 'Community', and 'More'. Below the navigation bar, there's a sidebar with links for 'scikit-learn 1.2.2' (highlighted), 'Other versions', 'Please cite us if you use the software.', and several sections under 'Getting Started': 'Fitting and predicting: estimator basics', 'Transformers and pre-processors', 'Pipelines: chaining pre-processors and estimators', 'Model evaluation', and 'Automatic parameter searches'. The main content area has a large blue header 'Getting Started'. Below it, a paragraph explains the purpose of the guide: 'The purpose of this guide is to illustrate some of the main features that scikit-learn provides. It assumes a very basic working knowledge of machine learning practices (model fitting, predicting, cross-validation, etc.). Please refer to our installation instructions for installing scikit-learn.' Another section, 'Fitting and predicting: estimator basics', provides an example of fitting a 'RandomForestClassifier' to data. A code block shows the Python code:

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> clf = RandomForestClassifier(random_state=0)
>>> X = [[1, 2, 3], # 2 samples, 3 features
...      [11, 12, 13]]
>>> y = [0, 1] # classes of each sample
>>> clf.fit(X, y)
RandomForestClassifier(random_state=0)
```

Below the code, it says: 'The `fit` method generally accepts 2 inputs:

- The samples matrix (or design matrix) `X`. The size of `X` is typically `(n_samples, n_features)`, which means that samples are represented as rows and features are represented as columns.
- The target values `y` which are real numbers for regression tasks, or integers for classification (or any other discrete set of values). For unsupervised learning tasks, `y` does not need to be specified. `y` is usually 1d array where the `i`th entry corresponds to the target of the `i`th sample (row) of `X`.

It also notes: 'Both `X` and `y` are usually expected to be numpy arrays or equivalent array-like data types, though some estimators work with other formats such as sparse matrices.'

Finally, it says: 'Once the estimator is fitted, it can be used for predicting target values of new data. You don't need to re-train the estimator:'

```
>>> clf.predict(X) # predict classes of the training data
array([0, 1])
>>> clf.predict([[4, 5, 6], [14, 15, 16]]) # predict classes of new data
array([0, 1])
```

## Módulo 15 – A base de dados Íris

No módulo 12, construímos um modelo de classificação manualmente, usando o dataset íris.

Recordando, naquele módulo retiramos uma das classes e mostramos que era possível separar as duas restantes a partir de uma reta.

No código ao lado, carregamos o dataset, removemos a classe 2, e mostramos as 5 primeiras linhas de nossa base.

```
1 # Importando o pandas  
2 import pandas as pd
```

```
1 # Importando o dataset iris  
2 from sklearn.datasets import load_iris  
3 data = load_iris()
```

```
1 # Transformando em um dataframe e usando somente target 0 e 1  
2 iris = pd.DataFrame(data.data)  
3 iris.columns = data.feature_names  
4 iris['target'] = data.target  
5 iris = iris[iris.target != 2]  
6 iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

## Módulo 15 – Entendendo o Perceptron

No citado módulo, após fazermos a separação com uma reta manualmente, vimos que há no Scikit-Learn o algoritmo Perceptron. Este algoritmo ajusta uma reta e permite classificar com base se o valor resultante está acima ou abaixo do limite imposto por esta reta.

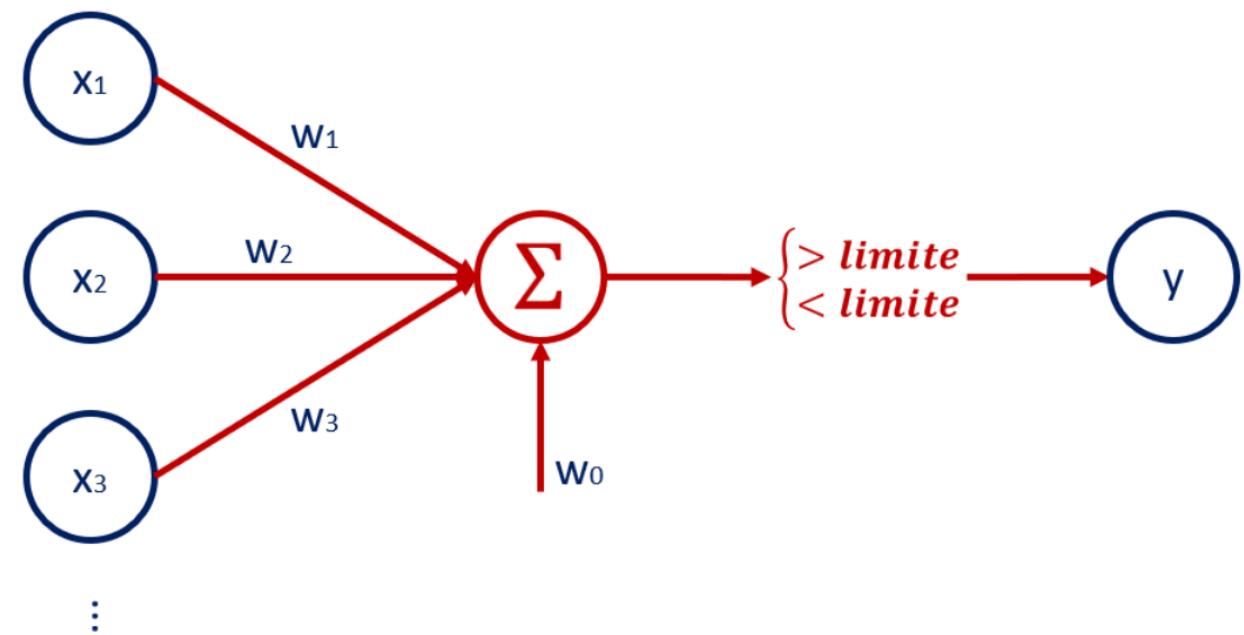
Lembrando, no contexto do Perceptron, os símbolos  $x$ ,  $w$  e  $y$  representam:

- $x$ : vetor de entrada, composto pelos atributos das amostras de dados (ex:  $x = [x_1, x_2, \dots, x_n]$ )
- $w$ : vetor de pesos, que são ajustados durante o treinamento do modelo (ex:  $w = [w_0, w_1, w_2, \dots, w_n]$ )
- $y$ : variável alvo, representando as classes (labels) das amostras de dados

A função linear do Perceptron é definida como:

$$y = w_i \cdot x + w_0$$

onde " $\cdot$ " denota um produto escalar.



## Módulo 15 – Entendendo o Perceptron

Vamos reproduzir rapidamente o que vimos no módulo 12. Abaixo, separamos duas colunas de nossa base de dados, que servirão de entrada para nosso modelo, e a coluna de classes:

```
1 # Importando o perceptron do scikit-learn  
2 from sklearn.linear_model import Perceptron
```

```
1 iris.head(2)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0

```
1 X = iris[['petal length (cm)', 'petal width (cm)']]  
2 y = iris.target
```

## Módulo 15 – Entendendo o Perceptron

Sem fazer a separação entre treino e teste (faremos mais adiante), vemos que nosso modelo é capaz de separar por completo as duas classes, já que resultou em um score de valor 1:

```
1 clf = Perceptron()
```

```
1 # Fazendo o fit do modelo
2 clf.fit(X,y)
```

Perceptron()

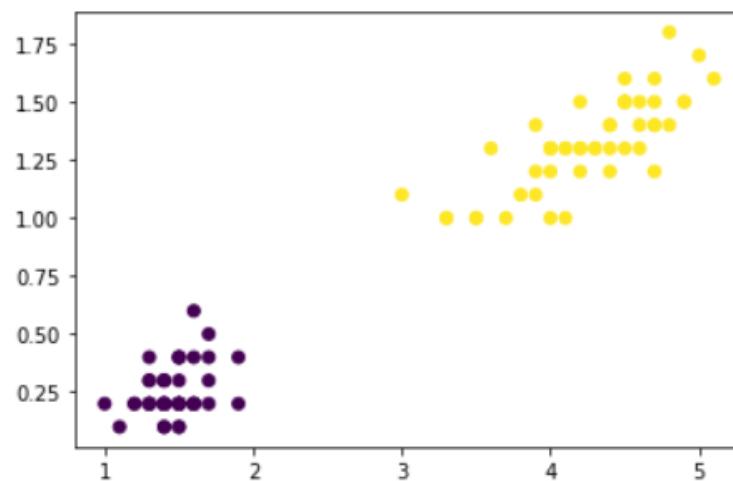
```
1 # Avaliando o modelo para a base completa
2 # (não separamos ainda em treino e teste)
3 clf.score(X, y)
```

1.0

## Módulo 15 – Entendendo o Perceptron

O significado disto pode ser averiguado visualmente. Abaixo, criamos um gráfico de dispersão das duas classes. Perceba que as duas são bem afastadas, permitindo uma completa separação por uma reta.

```
1 import matplotlib.pyplot as plt  
2 fig, ax = plt.subplots()  
3  
4 ax.scatter(iris['petal length (cm)'], iris['petal width (cm)'], c=iris.target)  
5  
6 plt.show()
```



## Módulo 15 – Entendendo o Perceptron

O algoritmo do Perceptron irá definir os parâmetros da reta que melhor separa as duas classes. Esses parâmetros podem ser obtidos com os atributos `coef_` e `intercept_`. De posse destes parâmetros, podemos construir a reta graficamente:

```
1 # w1 e w2  
2 clf.coef_
```

```
array([[0.9, 1.7]])
```

```
1 # w0  
2 clf.intercept_
```

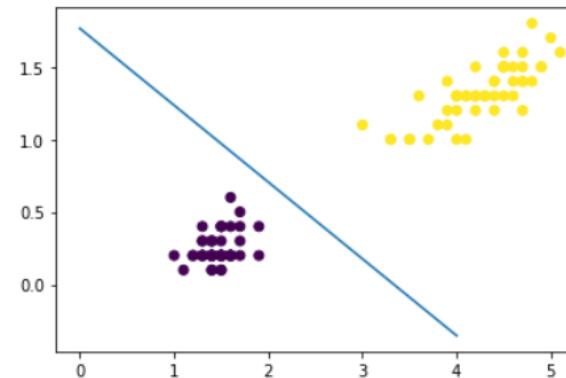
```
array([-3.])
```

Observe como a reta separa as duas classes perfeitamente, daí o `score` 1 apresentado anteriormente.

Dizemos que estas classes são linearmente separáveis, já que é possível traçar uma reta as separando.

```
1 # Criando a reta  
2 import numpy as np  
3  
4 w1 = clf.coef_[0][0]  
5 w2 = clf.coef_[0][1]  
6 w0 = clf.intercept_[0]  
7  
8 x = np.arange(0,5)  
9 y = (-w1*x-w0)/w2
```

```
1 # Visualizando de forma gráfica  
2 fig, ax = plt.subplots()  
3  
4 ax.scatter(iris['petal length (cm)'], iris['petal width (cm)'], c=iris.target)  
5 ax.plot(x,y)  
6  
7 plt.show()
```



## Módulo 15 – Entendendo árvore de decisão

Ainda com o módulo íris, no módulo 13 estudamos árvore de decisão. Veja, ao lado, que com as mesmas colunas a árvore também consegue separar completamente as classes:

```
1 # Importando a árvore de decisão
2 from sklearn import tree
```

```
1 # Definindo os dados
2 X = iris[['petal length (cm)', 'petal width (cm)']]
3 y = iris.target
```

```
1 # Criando o nosso classificador
2 clf = tree.DecisionTreeClassifier()
```

```
1 # Fazendo o fit com os dados
2 clf.fit(X, y)
```

DecisionTreeClassifier()

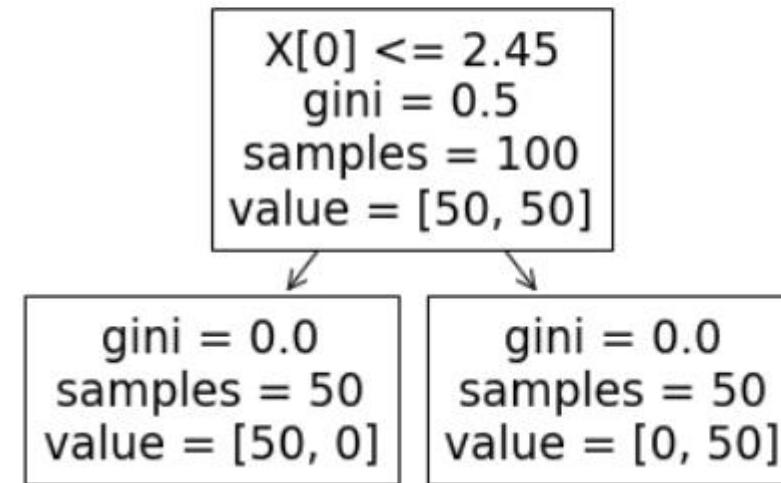
```
1 # Avaliando o modelo
2 clf.score(X, y)
```

1.0

## Módulo 15 – Entendendo árvore de decisão

Essa árvore pode ser visualizado com método `plot_tree`. Observe como com uma simples decisão já é possível separar as classes.

```
1 # Visualizando as decisões tomadas por essa árvore  
2 tree.plot_tree(clf);
```



Para facilitar a compreensão, usamos até este momento de nossos estudos apenas duas das três classes. Assim, nestes casos onde as classes são bem distintas, os modelos conseguem resolver rapidamente. Está na hora de colocarmos todas as três classes.

## Módulo 15 – Comparando os modelos com mais uma classe

Vamos importar novamente nossa base de dados, mas agora não iremos mais remover a classe 2.

Agora vamos resolver nosso problema de classificação, camada por camada.

```
1 # Importando o pandas  
2 import pandas as pd
```

```
1 # Importando o dataset iris  
2 from sklearn.datasets import load_iris  
3 data = load_iris()
```

```
1 # Transformando em um dataframe e usando somente target 0 e 1  
2 iris = pd.DataFrame(data.data)  
3 iris.columns = data.feature_names  
4 iris['target'] = data.target  
5 # iris = iris[iris.target != 2]  
6 iris.head(2)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0

## Módulo 15 – Comparando os modelos com mais uma classe

Comecemos verificando que realmente estamos com nossas 3 classes, cada uma com 50 amostras.

Novamente, criaremos nossos X e y para passar para o modelo. Usaremos as mesmas duas colunas de antes, para comparar os resultados.

```
1 # Verificando os valores na base
2 iris.target.value_counts()

2 50
1 50
0 50
Name: target, dtype: int64
```

```
1 # Definindo os valores de X e y do modelo
2 X = iris[['petal length (cm)', 'petal width (cm)']]
3 y = iris.target
```

## Módulo 15 – Comparando os modelos com mais uma classe

Veja como agora os modelos não possuem mais o score com valor 1. Ou seja, a presença de mais uma classe já dificulta, neste caso, a separação completa das classes.

```
1 # Importação do perceptron  
2 from sklearn.linear_model import Perceptron
```

```
1 # Criando o nosso classificador  
2 clfPercep = Perceptron()
```

```
1 # Fazendo o fit do modelo  
2 clfPercep.fit(X, y)
```

Perceptron()

```
1 # Avaliando o modelo para a base completa.  
2 # (não separamos ainda em treino e teste)  
3 clfPercep.score(X, y)
```

0.8733333333333333

```
1 # Importando a árvore de decisão  
2 from sklearn import tree
```

```
1 # Criando o nosso classificador  
2 clfArvore = tree.DecisionTreeClassifier()
```

```
1 # Fazendo o fit com os dados  
2 clfArvore.fit(X,y)
```

DecisionTreeClassifier()

```
1 # Avaliando o modelo  
2 clfArvore.score(X, y)
```

0.9933333333333333

## Módulo 15 – Comparando os modelos com mais uma classe

Aproveitando o conhecimento de módulos anteriores, podemos construir uma matriz de confusão para os dois modelos.

Veja que ambos os modelos classificaram corretamente as 50 amostras da classe 0. Já para a classe 1, a árvore de decisão foi bem mais performática que o perceptron. A árvore de decisão classificou corretamente as 50 amostras da classe 2.

```
1 from sklearn.metrics import confusion_matrix
2
1 predictArvore = clfArvore.predict(X)
2 predictPercep = clfPercep.predict(X)
3
1 # Perceptron
2 confusion_matrix(iris.target, predictPercep)
3
array([[50,  0,  0],
       [ 1, 33, 16],
       [ 0,  2, 48]])
4
1 # Árvore de Decisão
2 confusion_matrix(iris.target, predictArvore)
3
array([[50,  0,  0],
       [ 0, 49,  1],
       [ 0,  0, 50]])
```

## Módulo 15 – Comparando os modelos com mais uma classe

Já vimos o conceito de três métricas: acurácia, precisão e recall. O método `score` dos classificadores, em geral, equivale à acurácia. Podemos perceber isto abaixo, onde importamos o `accuracy_score` e obtemos os mesmos valores vistos anteriormente:

```
1 from sklearn.metrics import accuracy_score
```

```
1 # Perceptron  
2 accuracy_score(iris.target, predictPercep)
```

0.8733333333333333

```
1 # Árvore de Decisão  
2 accuracy_score(iris.target, predictArvore)
```

0.9933333333333333

## Módulo 15 – Comparando os modelos com mais uma classe

No caso da precisão, haverá uma pequena diferença comparando com o visto em módulos passados. Agora, temos 3 classes ao invés de 2, de forma que precisamos informar ao método como considerar os verdadeiros e falsos positivos, se por classe ou globalmente. Descreveremos melhor este aspecto mais adiante, por ora, passando o parâmetro `average='micro'` teremos a contagem feita globalmente:

```
1 | from sklearn.metrics import precision_score
```

```
1 | # Perceptron
```

```
2 | precision_score(iris.target, predictPercep,average='micro')
```

```
0.8733333333333333
```

```
1 | # Árvore de Decisão
```

```
2 | precision_score(iris.target, predictArvore,average='micro')
```

```
0.9933333333333333
```

## Módulo 15 – Comparando os modelos com mais uma classe

Seguindo a mesma lógica, no recall também é necessário passar o parâmetro `average`:

```
1 from sklearn.metrics import recall_score
```

```
1 # Perceptron  
2 recall_score(iris.target, predictPercep, average='micro')
```

0.8733333333333333

```
1 # Árvore de Decisão
```

```
2 recall_score(iris.target, predictArvore, average='micro')
```

0.9933333333333333

Observe que, nas três métricas apresentadas, a árvore de decisão apresentou melhores resultados.

## Módulo 15 – Entendendo o parâmetro *average*

Para entender um pouco melhor o parâmetro *average*, vamos criar alguns dados e avaliar a mudança na métrica a cada mudança deste parâmetro. Consideraremos 3 possíveis classes:

```
1 from sklearn.metrics import confusion_matrix  
2 from sklearn.metrics import precision_score  
  
1 y_true2 = [0, 0, 0, 1, 1, 1, 2, 2]  
2 y_pred2 = [0, 1, 2, 1, 2, 2, 2, 2]
```

```
1 confusion_matrix(y_true2, y_pred2)  
  
array([[1, 1, 1],  
       [0, 1, 2],  
       [0, 0, 2]])
```

## Módulo 15 – Entendendo o parâmetro *average*

Um dos possíveis valores para o parâmetro é `None`. Neste caso, a pontuação de cada classe é retornada. Relembrando do conceito de precisão:

$$\text{Precisão} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos positivos}}$$

```
1 confusion_matrix(y_true2, y_pred2)
array([[1, 1, 1],
       [0, 1, 2],
       [0, 0, 2]])
```

```
1 precision_score(y_true2, y_pred2, average=None)
array([1. , 0.5, 0.4])
```

Assim, vemos que para a primeira classe, temos 1 verdadeiro positivo (VP) e nenhum falso positivo (FP). Daí a pontuação 1. Para a segunda classe, temos 1 VP e 1 FP, resultando em 0.5. E, para a classe 3, temos 2 VP 3 FP, chegando em 0.4.

## Módulo 15 – Entendendo o parâmetro *average*

Quando o parâmetro tem valor `micro`, as métricas são calculadas globalmente, considerando o total de VPs e FPs.

```
1 confusion_matrix(y_true2, y_pred2)  
  
array([[1, 1, 1],  
       [0, 1, 2],  
       [0, 0, 2]])
```

```
1 precision_score(y_true2, y_pred2, average='micro')  
  
0.5  
  
1 4/8  
  
0.5
```

Veja, na matriz de confusão, que o modelo possui 4 VP em um total global de 8 amostras. Daí o resultado de 0.5.

## Módulo 15 – Entendendo o parâmetro *average*

Quando `macro`, calcula as métricas para cada classe e encontra sua média não ponderada. Na prática, é o valor médio daqueles encontrados em `None`:

```
1 confusion_matrix(y_true2, y_pred2)
array([[1, 1, 1],
       [0, 1, 2],
       [0, 0, 2]])
```

```
1 precision_score(y_true2, y_pred2, average='macro')
```

0.6333333333333333

```
1 (1+0.5+0.4)/3
```

0.6333333333333333

## Módulo 15 – Entendendo o parâmetro *average*

Por fim, quando o parâmetro é passado como `weighted`, o método calcula as métricas para cada classe resultando em sua média ponderada pelo suporte. Suporte é o nome dado ao número de instâncias verdadeiras para cada classe. Vejamos a seguir:

```
1 confusion_matrix(y_true2, y_pred2)  
  
array([[1, 1, 1],  
       [0, 1, 2],  
       [0, 0, 2]])
```

```
1 precision_score(y_true2, y_pred2, average='weighted')  
  
0.6625  
  
1 (3*1+3*0.5+2*0.4)/8  
  
0.6625
```

Veja que há 3 valores verdadeiros para a classe 0, igualmente para a classe 1, e 2 valores verdadeiros para a classe 2. Ao multiplicarmos estes valores pelos verdadeiros positivos de cada classe e dividirmos pelo total de amostras, chegamos ao mesmo valor do método.

## Módulo 15 – Separando os dados em treino e teste e avaliando os modelos

Até o momento expandimos nosso conhecimento de módulos anteriores sobre métodos de classificação e métricas para um caso multiclasse, com 3 classes.

No entanto, também vimos anteriormente que o mais correto em um projeto é separar a base em dados de treino e de teste, para se assegurar que o modelo não “decorou” a base de treino.

Assim, vamos acrescentar mais esta camada em nosso procedimento. O início é o mesmo, importar as bibliotecas e a base de dados:

```
1 # Importando o pandas
2 import pandas as pd

1 # Importando o dataset iris
2 from sklearn.datasets import load_iris
3 data = load_iris()

1 # Transformando em um dataframe e usando somente target 0 e 1
2 iris = pd.DataFrame(data.data)
3 iris.columns = data.feature_names
4 iris['target'] = data.target
5 # iris = iris[iris.target != 2]
6 iris.head(2)

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) target
0                 5.1             3.5            1.4            0.2       0
1                 4.9             3.0            1.4            0.2       0

1 # Verificando os valores na base
2 iris.target.value_counts()

2 50
1 50
0 50
Name: target, dtype: int64

1 # Definindo os valores de X e y do modelo
2 X = iris[['petal length (cm)', 'petal width (cm)']]
3 y = iris.target
```

## Módulo 15 – Separando os dados em treino e teste e avaliando os modelos

Agora, com o train\_test\_split, separamos nossa base em treino e teste e passamos os dados de treino para nossos modelos:

```
1 from sklearn.model_selection import train_test_split
```

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
1 # Importação do perceptron
2 from sklearn.linear_model import Perceptron
```

```
1 # Criando o nosso classificador
2 clfPercep = Perceptron()
```

```
1 # Fazendo o fit do modelo
2 clfPercep.fit(X_train, y_train)
```

Perceptron()

```
1 # Avaliando o modelo para a base completa
2 # (não separamos ainda em treino e teste)
3 clfPercep.score(X, y)
```

0.6666666666666666

```
1 # Importando a árvore de decisão
2 from sklearn import tree
```

```
1 # Criando o nosso classificador
2 clfArvore = tree.DecisionTreeClassifier()
```

```
1 # Fazendo o fit com os dados
2 clfArvore.fit(X_train, y_train)
```

DecisionTreeClassifier()

```
1 # Avaliando o modelo
2 clfArvore.score(X, y)
```

0.9933333333333333



## Módulo 15 – Separando os dados em treino e teste e avaliando os modelos

Com os modelos treinados, podemos aplica-los em nossos dados de teste e verificar as métricas.

Comecemos com as matrizes de confusão. Observe como os dois modelos classificam corretamente as classes 0 e 2. No entanto, o Perceptron não é um bom modelo para classificar a classe 1, enquanto que a árvore de decisão classifica corretamente todas as amostras da classe 1, assim como das demais.

```
1 from sklearn.metrics import confusion_matrix
```

```
1 predictArvore = clfArvore.predict(X_test)
2 predictPercep = clfPercep.predict(X_test)
```

```
1 # Perceptron
2 confusion_matrix(y_test, predictPercep)
```

```
array([[19,  0,  0],
       [ 7,  0,  6],
       [ 0,  0, 13]])
```

```
1 # Árvore de Decisão
2 confusion_matrix(y_test, predictArvore)
```

```
array([[19,  0,  0],
       [ 0, 13,  0],
       [ 0,  0, 13]])
```

# Módulo 15 – Separando os dados em treino e teste e avaliando os modelos

Agora, podemos avaliar as três métricas que vimos estudando: acurácia, precisão e recall. Veja que a árvore de decisão apresenta valor 1 em todas, já que classificou corretamente todas as amostras de todas as classes.

```
1 from sklearn.metrics import accuracy_score  
1 # Perceptron  
2 accuracy_score(y_test, predictPercep)
```

0.7111111111111111

```
1 # Árvore de Decisão  
2 accuracy_score(y_test, predictArvore)
```

1.0

```
1 from sklearn.metrics import precision_score  
1 # Perceptron  
2 precision_score(y_test, predictPercep,average='micro')
```

0.7111111111111111

```
1 # Árvore de Decisão  
2 precision_score(y_test, predictArvore,average='micro')
```

1.0

```
1 from sklearn.metrics import recall_score  
1 # Perceptron  
2 recall_score(y_test, predictPercep,average='micro')
```

0.7111111111111111

```
1 # Árvore de Decisão  
2 recall_score(y_test, predictArvore,average='micro')
```

1.0

## Módulo 15 – Adicionando um novo modelo – Regressão logística

Agora temos um procedimento padronizado, sendo bem simples aplicar a um novo modelo.

Suponha, por exemplo, que a árvore de decisão que vimos ter um score perfeito anteriormente, tenha se mostrada muito lenta em um ambiente de produção, seja pelo volume de dados ou por ter gerado muitos nós de decisão. Podemos facilmente treinar um novo modelo agora que criamos esta estrutura inicial de um projeto de ciência de dados.

Vamos ver o modelo de regressão logística, também muito aplicado em ciência de dados. O início é o mesmo, importar bibliotecas e nossa base de dados:

```
1 # Importando o pandas  
2 import pandas as pd
```

```
1 # Importando o dataset iris  
2 from sklearn.datasets import load_iris  
3 data = load_iris()
```

```
1 # Transformando em um dataframe e usando somente target 0 e 1  
2 iris = pd.DataFrame(data.data)  
3 iris.columns = data.feature_names  
4 iris['target'] = data.target  
5 # iris = iris[iris.target != 2]  
6 iris.head(2)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0		5.1	3.5	1.4	0
1		4.9	3.0	1.4	0

```
1 # Verificando os valores na base  
2 iris.target.value_counts()
```

```
2 50  
1 50  
0 50  
Name: target, dtype: int64
```

```
1 # Definindo os valores de X e y do modelo  
2 X = iris[['petal length (cm)', 'petal width (cm)']]  
3 y = iris.target
```

```
1 # Definindo os valores de X e y do modelo  
2 X2 = iris.drop('target', axis=1)  
3 y2 = iris.target
```

# Módulo 15 – Adicionando um novo modelo – Regressão logística

Depois, separamos nossa base em treino e teste e treinamos nossos modelos:

```
1 from sklearn.model_selection import train_test_split  
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
1 # Importação do perceptron  
2 from sklearn.linear_model import Perceptron  
  
1 # Criando o nosso classificador  
2 clfPercep = Perceptron()  
  
1 # Fazendo o fit do modelo  
2 clfPercep.fit(X_train2, y_train2)  
  
Perceptron()  
  
1 # Avaliando o modelo para a base completa.  
2 # (não separamos ainda em treino e teste)  
3 clfPercep.score(X_train2, y_train2)
```

0.8

```
1 # Importando a árvore de decisão  
2 from sklearn import tree  
  
1 # Criando o nosso classificador  
2 clfArvore = tree.DecisionTreeClassifier()  
  
1 # Fazendo o fit com os dados  
2 clfArvore.fit(X_train, y_train)  
  
DecisionTreeClassifier()  
  
1 # Avaliando o modelo  
2 clfArvore.score(X_train, y_train)
```

0.9904761904761905

```
1 # Importar a regressão logística  
2 from sklearn.linear_model import LogisticRegression  
  
1 # Criando o nosso classificador  
2 clfLogist = LogisticRegression(random_state=0).fit(X_train, y_train)  
  
1 # Avaliando o modelo  
2 clfLogist.score(X_train, y_train)
```

0.9523809523809523

# Módulo 15 – Adicionando um novo modelo – Regressão logística

Por fim, construímos as matrizes de confusão e avaliamos as métricas. Perceba que a regressão logística também classifica todas as classes corretamente. Logo, se ela tiver melhor eficiência em produção, pode substituir a árvore de decisão. Observe, também, como foi simples adaptar a estrutura para estudar um novo modelo.

```
1 from sklearn.metrics import confusion_matrix  
  
1 predictArvore = clfArvore.predict(X_test)  
2 predictPercep = clfPercep.predict(X_test2)  
3 predictLogist = clfLogist.predict(X_test)  
  
1 # Perceptron  
2 confusion_matrix(y_test2, predictPercep)  
  
array([[19,  0,  0],  
       [ 0, 13,  0],  
       [ 0,  0, 13]])  
  
1 # Árvore de Decisão  
2 confusion_matrix(y_test, predictArvore)  
  
array([[19,  0,  0],  
       [ 0, 13,  0],  
       [ 0,  0, 13]])  
  
1 # Regressão Logística  
2 confusion_matrix(y_test, predictLogist)  
  
array([[19,  0,  0],  
       [ 0, 13,  0],  
       [ 0,  0, 13]])
```

```
1 from sklearn.metrics import accuracy_score  
  
1 # Perceptron  
2 accuracy_score(y_test2, predictPercep)  
  
0.822222222222222  
  
1 # Árvore de Decisão  
2 accuracy_score(y_test, predictArvore)  
  
1.0  
  
1 # Regressão Logística  
2 accuracy_score(y_test, predictLogist)  
  
1.0
```

```
1 from sklearn.metrics import precision_score  
  
1 # Perceptron  
2 precision_score(y_test2, predictPercep, average='micro')  
0.822222222222222  
  
1 # Árvore de Decisão  
2 precision_score(y_test, predictArvore, average='micro')  
1.0  
  
1 # Regressão Logística  
2 precision_score(y_test, predictLogist, average='micro')  
1.0  
  
1 from sklearn.metrics import recall_score  
  
1 # Perceptron  
2 recall_score(y_test2, predictPercep, average='micro')  
0.822222222222222  
  
1 # Árvore de Decisão  
2 recall_score(y_test, predictArvore, average='micro')  
1.0  
  
1 # Regressão Logística  
2 recall_score(y_test, predictLogist, average='micro')  
1.0
```

# Módulo 15 – Entendendo regressão linear

Já nos aprofundamos em modelos de classificação. Está no momento de nos aprofundarmos no modelo de regressão linear, que vimos de forma introdutória em módulos passados.

A regressão linear é um método estatístico que busca modelar a relação entre uma variável dependente ( $y$ ) e uma ou mais variáveis independentes ( $x$ ). O objetivo é encontrar a melhor linha reta (ou hiperplano, no caso de múltiplas variáveis) que minimize a soma dos erros quadráticos entre os valores reais e os valores previstos pelo modelo. A regressão linear é amplamente utilizada na análise e previsão de dados devido à sua simplicidade, interpretabilidade e eficiência computacional.

Veja ao lado que criamos 3 pares ( $x, y$ ). No caso, eles são descritos perfeitamente pela reta  $y = 2500x + 4100$ . Usamos o `LinearRegression` do Scikit-Learn e os atributos `coef_` e `intercept_` para descobrir os coeficientes angular e linear, respectivamente, da reta. E percebemos que são perfeitamente lineares pelo `score` de valor 1.

```
1 import pandas as pd
2
3 # Dados 1
4 dados1 = {
5     'X': [1,2,3],
6     'Y': [6600,9100,11600]
7 }
8
9 dados1 = pd.DataFrame(dados1)
10
11 # Importando a Regressão Linear
12 from sklearn import linear_model
13
14 # Criando nosso algoritmo de regressão
15 reg = linear_model.LinearRegression()
16 reg.fit(dados1.X.values.reshape(-1, 1),dados1.Y)
17
18 LinearRegression()
19
20 # Avaliando o modelo
21 reg.score(dados1.X.values.reshape(-1, 1),dados1.Y)
22
23 1.0
24
25 # Determinando o coeficiente angular
26 reg.coef_
27
28 array([2500.])
29
30 # Determinando o coeficiente linear
31 reg.intercept_
32
33 4100.000000000001
```

## Módulo 15 – Entendendo regressão linear

O que ocorre se mudar um dos valores? Veja que, alterando o último valor da lista Y, já não temos mais uma relação perfeitamente linear pelo valor do score. No entanto, o valor ainda é alto o suficiente para podermos considerar que um modelo linear descreve bem o conjunto de pontos.

Podemos verificar de forma visual, a partir dos coeficientes.

```
1 # Dados 2
2 dados2 = {
3     'X': [1,2,3],
4     'Y': [6600,9100,11000]
5 }
6
7 dados2 = pd.DataFrame(dados2)
```

```
1 # Criando nosso algoritmo de regressão
2 reg2 = linear_model.LinearRegression()
3 reg2.fit(dados2.X.values.reshape(-1, 1),dados2.Y)
```

LinearRegression()

```
1 # Avaliando o modelo
2 reg2.score(dados2.X.values.reshape(-1, 1),dados2.Y)
```

0.9938398357289527

```
1 # Determinando o coeficiente angular
2 reg2.coef_
```

array([2200.])

```
1 # Determinando o coeficiente linear
2 reg2.intercept_
```

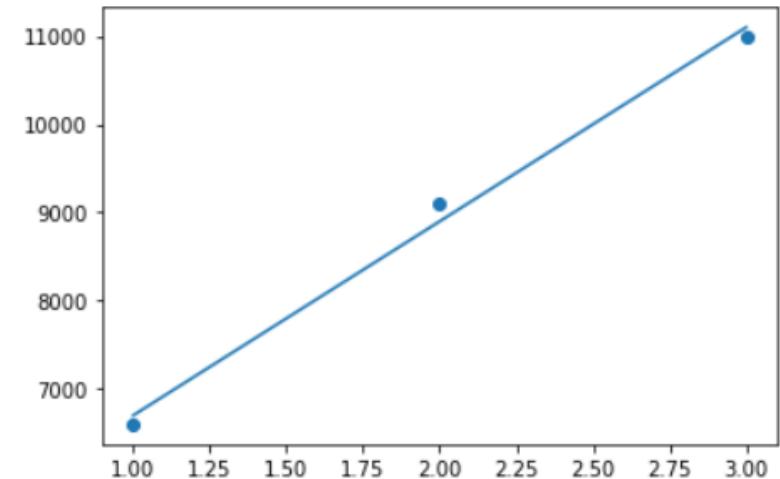
4500.000000000001

## Módulo 15 – Entendendo regressão linear

Observe que a reta, construída a partir dos coeficientes fornecidos pelo modelo, não passa por nenhum dos pontos, mas descreve uma tendência destes.

O que vimos nestas últimas páginas é similar ao que já havíamos visto em módulos anteriores. Agora, vamos aplicar o método de regressão linear em dados reais.

```
1 a = reg2.coef_[0]
2 b = reg2.intercept_
3
4 # plot
5 import matplotlib.pyplot as plt
6 import numpy as np
7 fig, ax = plt.subplots()
8
9 ax.scatter(dados2.X,dados2.Y)
10 x=np.arange(1,4)
11 y=a*x+b
12 ax.plot(x,y)
13
14 plt.show()
```



## Módulo 15 – Regressão linear no mercado de ações

Vamos pegar uma base de dados contendo a cotação das ações da empresa Tesla na bolsa americana entre 2010 e 2020. Nossa objetivo é estudar se há alguma relação entre o valor da ação e o volume negociado. Nesta base temos:

- Date: data
- Open: preço na abertura do pregão
- High: maior preço durante o dia
- Low: menor preço durante o dia
- Close: preço no fechamento do pregão
- Adj Close: preço após ajustes (split, dividendos, etc)
- Volume: volume de negociações

```
1 import pandas as pd
1 # Importando a base
2 base = pd.read_csv('TSLA.csv',parse_dates=['Date'])
1 # Visualizando as primeiras linhas
2 base.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	2010-07-01	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	2010-07-02	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	2010-07-06	20.000000	20.00	15.830000	16.110001	16.110001	6866900

## Módulo 15 – Regressão linear no mercado de ações

Como sempre, vamos começar verificando a integridade da base e os tipos de dados. No caso, vemos que a coluna de data está no formato adequado, acima como as colunas de preços e de volume. E não há problemas de duplicatas.

```
1 # Visualizando as informações da base  
2 base.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 2416 entries, 0 to 2415  
Data columns (total 7 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          -----            
 0   Date        2416 non-null    datetime64[ns]  
 1   Open         2416 non-null    float64  
 2   High         2416 non-null    float64  
 3   Low          2416 non-null    float64  
 4   Close        2416 non-null    float64  
 5   Adj Close    2416 non-null    float64  
 6   Volume       2416 non-null    int64  
dtypes: datetime64[ns](1), float64(5), int64(1)  
memory usage: 132.2 KB
```

```
1 # Verificando valores duplicados  
2 base.duplicated().sum()
```

0

## Módulo 15 – Regressão linear no mercado de ações

Continuando, com o `describe` temos um resumo estatístico de nossa base. Observe, pela diferença entre os valores mínimos e máximos, que as ações valorizaram significativamente no período. E, também, que houve um aumento no volume negociado.

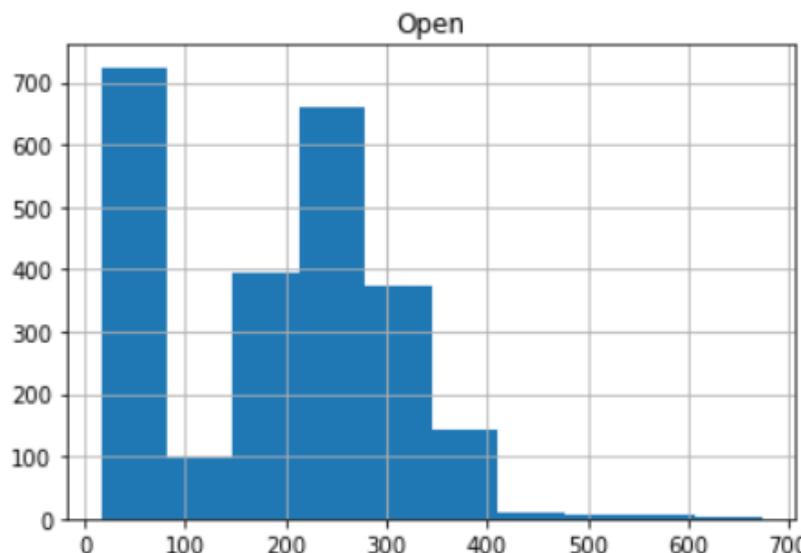
1	# Obtendo as informações estatísticas					
2	base.describe()					
	Open	High	Low	Close	Adj Close	Volume
<b>count</b>	2416.000000	2416.000000	2416.000000	2416.000000	2416.000000	2.416000e+03
<b>mean</b>	186.271147	189.578224	182.916639	186.403651	186.403651	5.572722e+06
<b>std</b>	118.740163	120.892329	116.857591	119.136020	119.136020	4.987809e+06
<b>min</b>	16.139999	16.629999	14.980000	15.800000	15.800000	1.185000e+05
<b>25%</b>	34.342498	34.897501	33.587501	34.400002	34.400002	1.899275e+06
<b>50%</b>	213.035003	216.745002	208.870002	212.960007	212.960007	4.578400e+06
<b>75%</b>	266.450012	270.927513	262.102501	266.774994	266.774994	7.361150e+06
<b>max</b>	673.690002	786.140015	673.520020	780.000000	780.000000	4.706500e+07

## Módulo 15 – Regressão linear no mercado de ações

Cada uma das colunas pode ser estudada graficamente, utilizando todas as visualizações que estudamos. Por exemplo, abaixo temos o histograma da coluna *Open*, mostrando que a maior parte das vezes o valor da ação na abertura foi abaixo de 100, mas há uma quantidade significativa entre 150 e 400. Na realidade, somando as barras, esta faixa de valores possui mais entradas correspondentes. Pratique fazendo o histograma das demais colunas.

```
1 # Visualizando o histograma dos dados  
2 base.hist('Open')
```

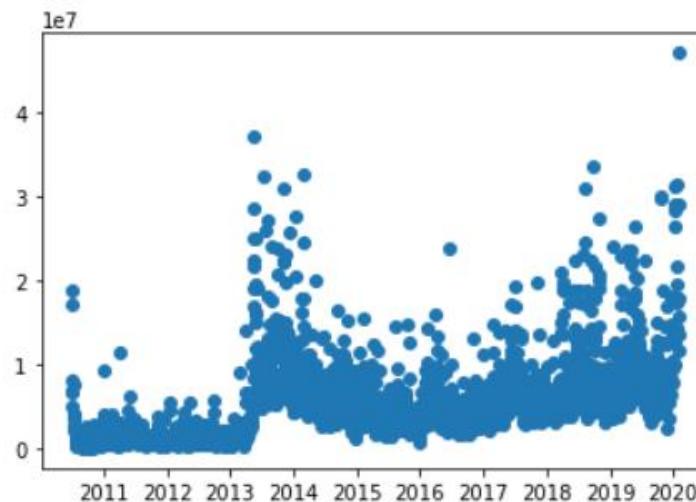
```
array([[<AxesSubplot:title={'center':'Open'}>]], dtype=object)
```



## Módulo 15 – Regressão linear no mercado de ações

O histograma avalia a coluna isoladamente. Podemos fazer gráficos de dispersão para avaliar pares de colunas, estudando a relação entre as mesmas. Por exemplo, considerando a variação do volume negociado no tempo, vemos um aumento significativo por volta de 2013 e uma tendência de alta ao se aproximar de 2020:

```
1 # Gerando um gráfico de dispersão para verificar a relação entre as informações
2 import matplotlib.pyplot as plt
3 fig, ax = plt.subplots()
4
5 ax.scatter(base.Date, base.Volume)
6
7 plt.show()
```



## Módulo 15 – Regressão linear no mercado de ações

Como queremos estudar a relação entre preço e volume, vamos criar um modelo de regressão. Vamos considerar o preço de abertura num primeiro momento. Considerando o que já aprendemos, vamos começar separando em base de treino e de teste:

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(base.Open,base.Volume, test_size=0.33, random_state=42)

1 X_train
```

1572	206.500000
393	27.190001
998	224.110001
534	29.520000
30	18.690001
	...
1638	218.559998
1095	243.000000
1130	223.809998
1294	260.329987
860	119.379997

Name: Open, Length: 1618, dtype: float64

## Módulo 15 – Regressão linear no mercado de ações

Agora, baste treinar o modelo e avaliar os resultados na base de teste. Observe que tivemos valores de score relativamente baixos, mas o modelo performou melhor, considerando esta métrica, na base de teste:

```
1 # Importando a Regressão Linear  
2 from sklearn.linear_model import LinearRegression
```

```
1 # Criando nosso algoritmo de regressão  
2 reg = LinearRegression().fit(X_train.values.reshape(-1,1), y_train)
```

```
1 # Avaliando o modelo nos dados de TREINO  
2 reg.score(X_train.values.reshape(-1,1), y_train)
```

0.21503603604976262

```
1 # Avaliando o modelo nos dados de TESTE  
2 reg.score(X_test.values.reshape(-1,1), y_test)
```

0.3284195874914201

```
1 # Determinando o coeficiente angular  
2 reg.coef_[0]
```

20195.187973651628

```
1 # Determinando o coeficiente linear  
2 reg.intercept_
```

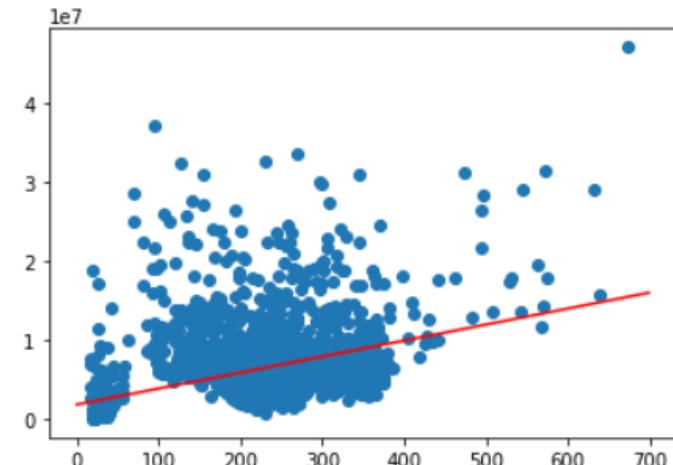
1904679.7631864208

## Módulo 15 – Regressão linear no mercado de ações

Com os valores dos coeficientes, podemos visualizar a reta do nosso modelo.

Como vemos, a reta mostra a tendência crescente dos valores, mas provavelmente geraria muitos erros se fosse utilizada isoladamente para previsões. Podemos aumentar a complexidade do nosso modelo passando mais colunas para treiná-lo.

```
1 # Visualizando de forma gráfica
2 import numpy as np
3
4 a = reg.coef_[0]
5 b = reg.intercept_
6
7 x = np.arange(0,700)
8 y = a*x+b
9
10 fig, ax = plt.subplots()
11
12 ax.scatter(base.Open, base.Volume)
13 ax.plot(x,y,c='r')
14
15 plt.show()
```



## Módulo 15 – Regressão linear no mercado de ações

Vamos, por exemplo, passar mais uma coluna, a *High*.

```
1 base.head(2)
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100

```
1 # Definindo X e y
2 X = base[['Open', 'High']]
3 y = base.Volume
```

## Módulo 15 – Regressão linear no mercado de ações

Observe que a simples adição de mais de uma coluna já fez melhorar o nosso **score** (compare com os valores das páginas anteriores). Você pode fazer de exercício a adição de mais colunas e, inclusive, passar todas as colunas da base.

```
1 # Separando novamente em treino e teste  
2 from sklearn.model_selection import train_test_split  
3 X_train2, X_test2, y_train2, y_test2 = train_test_split(X,y, test_size=0.33, random_state=42)
```

```
1 # Usando a regressão linear  
2 reg2 = LinearRegression().fit(X_train2, y_train2)
```

```
1 # Avaliando nos dados de treino  
2 reg2.score(X_train2, y_train2)
```

0.3492902359670137

```
1 # Avaliando nos dados de teste  
2 reg2.score(X_test2, y_test2)
```

0.39825327242737796

## Módulo 15 – Regressão linear no mercado de ações

Agora que temos uma regressão linear múltipla, é um pouco mais difícil ter uma visualização. Mas a lógica é a mesma para uma regressão linear simples. Podemos obter os coeficientes:

```
1 # Coeficiente angular  
2 reg2.coef_
```

```
array([-408577.50212954, 421052.85560007])
```

```
1 # Coeficiente linear  
2 reg2.intercept_
```

```
1928821.1465417575
```

Cada coeficiente angular é um peso passado para uma das colunas  $X$ , e temos nosso intercepto. Assim, nossa equação se torna

$$y = w_i \cdot X + w_0$$

Uma forma que já vimos em nosso material anteriormente.

## MÓDULO 16

# Criando um algoritmo de regressão

## Módulo 16 – Conhecendo a base de casas da Califórnia

Neste módulo, utilizaremos a base de dados California Housing, presente no Scikit-Learn.



# Módulo 16 – Conhecendo a base de casas da Califórnia

A base de dados California Housing é um conjunto de dados amplamente utilizado para estudar e aplicar técnicas de regressão. O conjunto contém informações demográficas e habitacionais de diferentes regiões da Califórnia, coletadas no Censo de 1990. As variáveis incluem características como renda média, idade média da habitação, população e número de cômodos. O objetivo é prever o valor médio de uma casa em uma determinada região.

```
1 # Importando a base
2 from sklearn.datasets import fetch_california_housing

1 # Visualizando a base importada
2 retorno = fetch_california_housing()
3 retorno.DESCR

... _california_housing_dataset:\n\nCalifornia Housing dataset\n-----\n**Data Set Characteristics:**\n  :Number of Instances: 20640\n  :Number of Attributes: 8 numeric, predictive attributes and the target\n  :Attribute Information:\n    - MedInc median income in block group\n    - HouseAge median house age in block group\n    - AveRooms average number of rooms per household\n    - AveBedrms average number of bedrooms per household\n    - Population block group population\n    - AveOccup average number of household members\n    - Latitude block group latitude\n    - Longitude block group longitude\n  :Missing Attribute Values: None\n\nThis dataset was obtained from the StatLib repository.\nhttps://www.dcc.fc.up.pt/~ltorgo/Regression/cal\_housing.html\nThe target variable is the median house value for California districts, expressed in hundreds of thousands of dollars ($100,000).\n\nThis dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).\n\nA household is a group of people residing within a home. Since the average number of rooms and bedrooms in this dataset are provided per household, these columns may take surprisingly large values for block groups with few households and many empty houses, such as vacation resorts.\n\nIt can be downloaded/loaded using the func:`sklearn.datasets.fetch_california_housing` function.\n.. topic:: References\n- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,\n  Statistics and Probability Letters, 33 (1997) 291-297
```

Com o atributo DESCR vemos, em forma de texto, o conteúdo. O que precisamos é transformar em forma de dataframe do Pandas.



## Módulo 16 – Conhecendo a base de casas da Califórnia

Outros atributos que utilizamos para a construção do dataframe: `data`, `feature_names` e `target`. Respectivamente, fornecendo as variáveis, seus nomes, e a variável alvo.

```
1 # Transformando em um DataFrame
2 import pandas as pd
3 casas = pd.DataFrame(retorno.data)
4 casas.columns = retorno.feature_names
5 casas['MedHouseVal'] = retorno.target
6 casas.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
1 # Criando as variáveis X e y
2 X = casas.drop('MedHouseVal',axis=1)
3 y = casas.MedHouseVal
```

## Módulo 16 – Conhecendo a base de casas da Califórnia

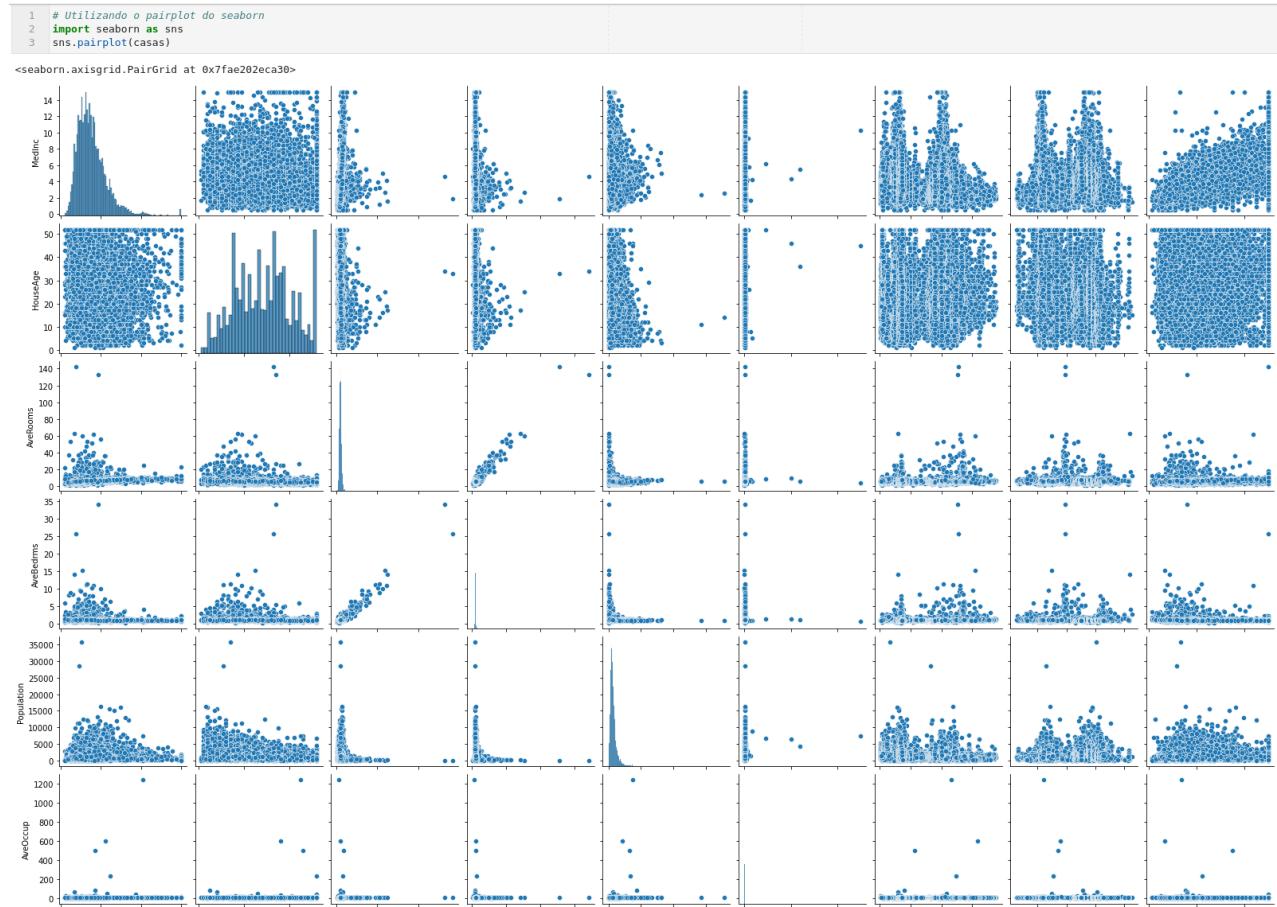
As colunas da base de dados e seus significados são:

- *MedInc*: renda média dos residentes de uma região (em dezenas de milhares de dólares)
- *HouseAge*: idade média das casas em uma região (em anos)
- *AveRooms*: número médio de cômodos por casa em uma região
- *AveBedrms*: número médio de quartos por casa em uma região
- *Population*: número total de pessoas residentes em uma região
- *AveOccup*: número médio de residents por casa em uma região
- *Latitude*: latitude da região
- *Longitude*: longitude da região
- *MedHouseVal*: valor médio das casas em uma região (em unidades de 100.000 dólares) – variável alvo a ser prevista

## Módulo 16 – Visualizando os dados

Aproveitando o que já vimos nos últimos módulos, podemos usar o método `pairplot` do Seaborn para ter uma ideia da distribuição dos valores de cada coluna.

Nesta visualização, busque identificar o formato das distribuições e os pontos que se destacam como outliers, praticando o que estamos estudando seguidamente no curso.

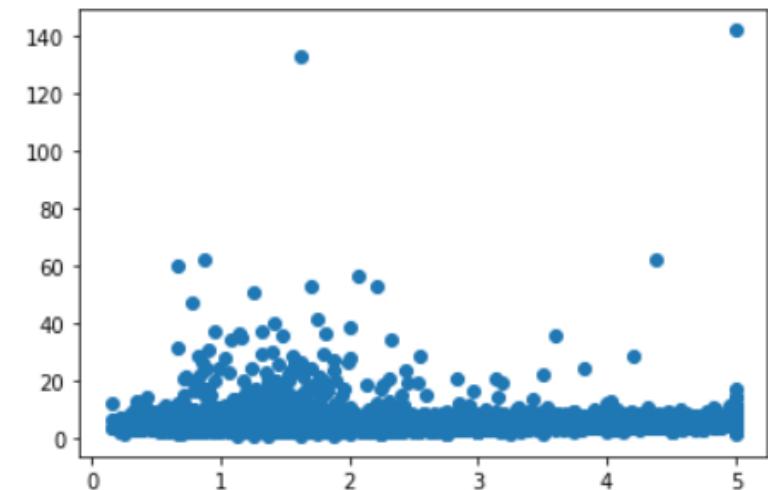


## Módulo 16 – Visualizando os dados

Para mais detalhes, podemos construir os gráficos de dispersão de pares específicos. Por exemplo, podemos verificar se há alguma relação entre o preço médio de uma casa e o número de cômodos.

O preço médio está no eixo horizontal, lembrando que está expresso em centenas de milhares de dólares (ou seja, 1 significa \$100.000,00). No eixo vertical, temos a média de cômodos. Observe que há valores elevados de cômodos para alguns pontos. Isto pode ser devido a problemas na base ou a situações específicas como, por exemplo citado na documentação, casas com quartos para alugar ou casas de repouso.

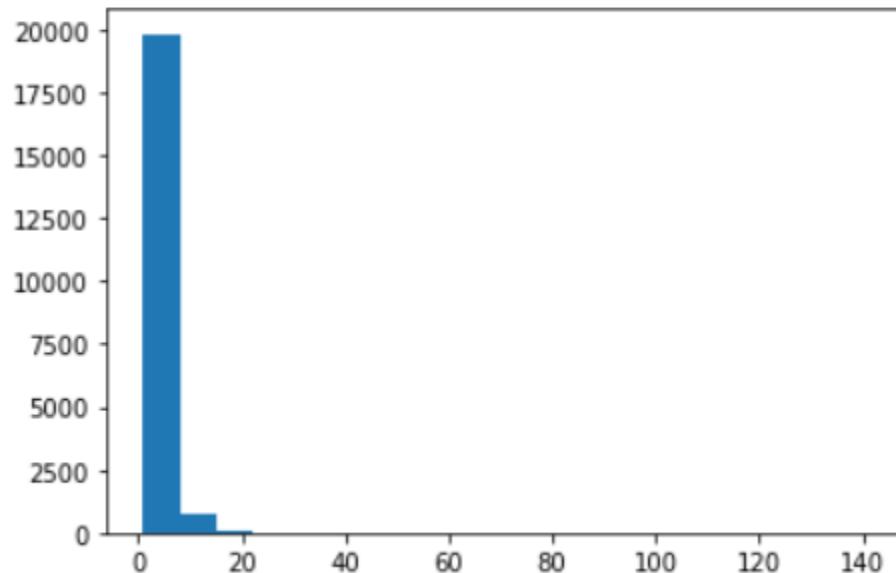
```
1 # Criando uma scatter plot
2 import matplotlib.pyplot as plt
3 fig,ax = plt.subplots()
4
5 ax.scatter(casas.MedHouseVal,casas.AveRooms)
6
7 plt.show()
```



## Módulo 16 – Visualizando os dados

Podemos perceber melhor esta questão dos cômodos com um histograma. Veja como boa parte está na faixa usual de valores baixos.

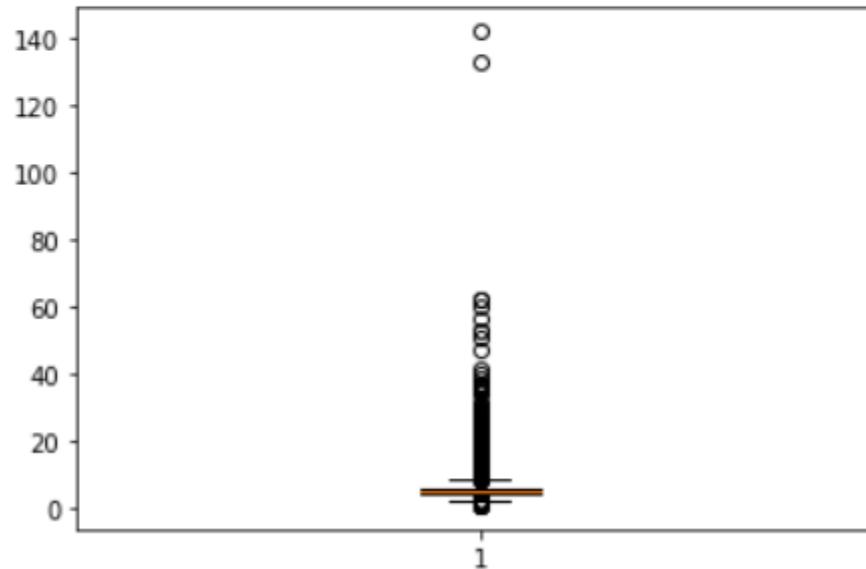
```
1 # Visualizando o histograma dos dados
2 fig,ax = plt.subplots()
3
4 ax.hist(casas.AveRooms,bins=20)
5
6 plt.show()
```



## Módulo 16 – Visualizando os dados

Outra visualização que mostra a distribuição de cômodos é o boxplot. Nele, fica evidente que há maior parte dos dados em valores pequenos e há dois pontos com valores muito elevados.

```
1 # Verificando outliers com o boxplot
2 fig,ax = plt.subplots()
3
4 ax.boxplot(casas.AveRooms)
5
6 plt.show()
```



## Módulo 16 – Visualizando os dados

Com o método `describe`, vemos que 75 % dos dados estão em até cerca de 6 cômodos.

```
1 casas.AveRooms.describe()  
  
count      20640.000000  
mean        5.429000  
std         2.474173  
min         0.846154  
25%        4.440716  
50%        5.229129  
75%        6.052381  
max        141.909091  
Name: AveRooms, dtype: float64
```

## Módulo 16 – Visualizando os dados

Com o método `corr`, podemos buscar quais variáveis tem maior correlação com a variável alvo.

```
1 # Criando uma matriz de correlação  
2 casas.corr()
```

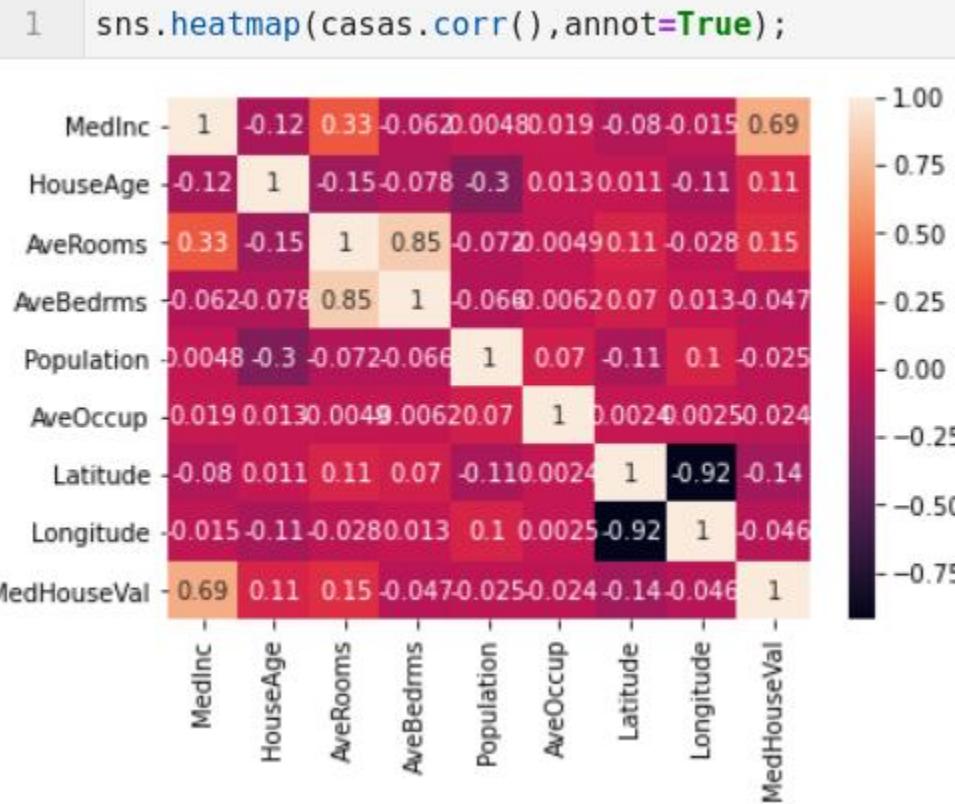
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
MedInc	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766	-0.079809	-0.015176	0.688075
HouseAge	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191	0.011173	-0.108197	0.105623
AveRooms	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852	0.106389	-0.027540	0.151948
AveBedrms	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181	0.069721	0.013344	-0.046701
Population	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863	-0.108785	0.099773	-0.024650
AveOccup	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000	0.002366	0.002476	-0.023737
Latitude	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366	1.000000	-0.924664	-0.144160
Longitude	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476	-0.924664	1.000000	-0.045967
MedHouseVal	0.688075	0.105623	0.151948	-0.046701	-0.024650	-0.023737	-0.144160	-0.045967	1.000000

Lembrando do que já estudamos, um valor positivo indica correlação positiva (quando um valor aumenta, o outro também aumenta). E um valor negativo indica o oposto (quando um valor aumenta, o outro diminui). Quanto mais se aproximam de -1 ou +1 maior a correlação, seja negativa ou positiva.

Veja a correlação elevada do valor médio da casa com a renda média.

## Módulo 16 – Visualizando os dados

Visualmente, as correlações ficam mais evidentes com um mapa de calor, criado a partir do método `heatmap` do Seaborn.



## Módulo 16 – Verificando valores duplicados e outliers

Agora que temos uma noção visual da base, vamos verificar o quanto a base está completa, se há valores duplicados e se iremos ou não remover outliers.

Com o método `info`, vemos que todas as colunas estão com os tipos numéricos e sem valores ausentes.

```
1 # Verificando as informações da base
2 casas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   MedInc       20640 non-null   float64
 1   HouseAge    20640 non-null   float64
 2   AveRooms    20640 non-null   float64
 3   AveBedrms   20640 non-null   float64
 4   Population  20640 non-null   float64
 5   AveOccup   20640 non-null   float64
 6   Latitude     20640 non-null   float64
 7   Longitude   20640 non-null   float64
 8   MedHouseVal 20640 non-null   float64
dtypes: float64(9)
memory usage: 1.4 MB
```

Com o método `duplicated`, vemos que não há linhas duplicadas nesta base de dados.

```
1 # Verificando se existe algum valor duplicado
2 casas.duplicated().sum()
```

0

## Módulo 16 – Verificando valores duplicados e outliers

Com o `describe`, temos um resumo estatístico de cada coluna. Perceba que em algumas há uma grande diferença entre o valor na linha 75% e o valor na linha `max`, o que talvez indique outliers superiores. Uma grande diferença entre 25% e `min` pode indicar outliers inferiores. Diagramas de boxplot podem confirmar se realmente são, mas aqui já temos alguns indícios.

1	# Adicionando as informações estatísticas								
2	casas.describe()								
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
<b>count</b>	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
<b>mean</b>	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35.631861	-119.569704	2.068558
<b>std</b>	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532	1.153956
<b>min</b>	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000	0.149990
<b>25%</b>	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000	1.196000
<b>50%</b>	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000	1.797000
<b>75%</b>	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000	2.647250
<b>max</b>	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000	5.000010

## Módulo 16 – Verificando valores duplicados e outliers

Uma outra análise interessante é a de valores únicos por coluna. Observe como há poucos valores únicos na coluna referente a idade da casa, enquanto que outras colunas como a de média de cômodos possui diversos valores.

Esta análise é interessante pois, em algumas bases de dados, algumas colunas possuem poucos valores únicos que podem ser trabalhadas como colunas de variáveis categóricas. Não é o caso desta base, mas aparecerão outras no curso.

```
1 # Verificando a cardinalidade  
2 casas.nunique()
```

```
MedInc      12928  
HouseAge     52  
AveRooms    19392  
AveBedrms   14233  
Population   3888  
AveOccup    18841  
Latitude     862  
Longitude    844  
MedHouseVal  3842  
dtype: int64
```

## Módulo 16 – Verificando valores duplicados e outliers

O método `value_counts` permite contar quantas vezes um determinado valor aparece. Abaixo, temos os 5 valores que mais aparecem (`head`) e os 5 que menos aparecem (`tail`) na coluna `HouseAge`. Ou seja, temos 1273 casas com 52 anos de idade, e 4 casas com 1 ano de idade.

```
1 casas.HouseAge.value_counts().head()
```

```
52.0    1273
36.0    862
35.0    824
16.0    771
17.0    698
Name: HouseAge, dtype: int64
```

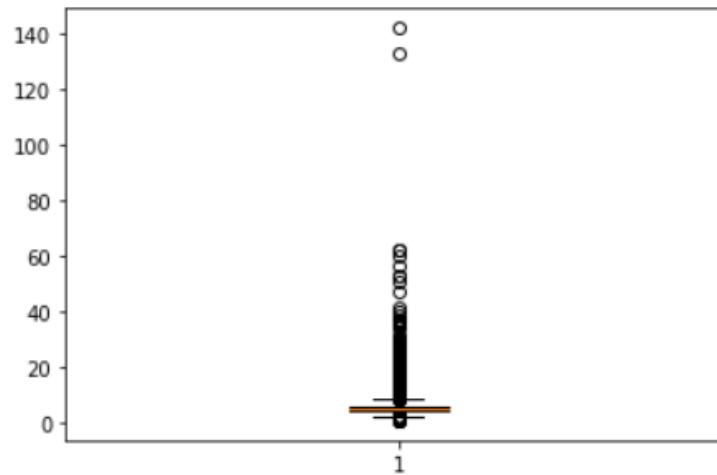
```
1 casas.HouseAge.value_counts().tail()
```

```
49.0    134
3.0     62
2.0     58
51.0    48
1.0     4
Name: HouseAge, dtype: int64
```

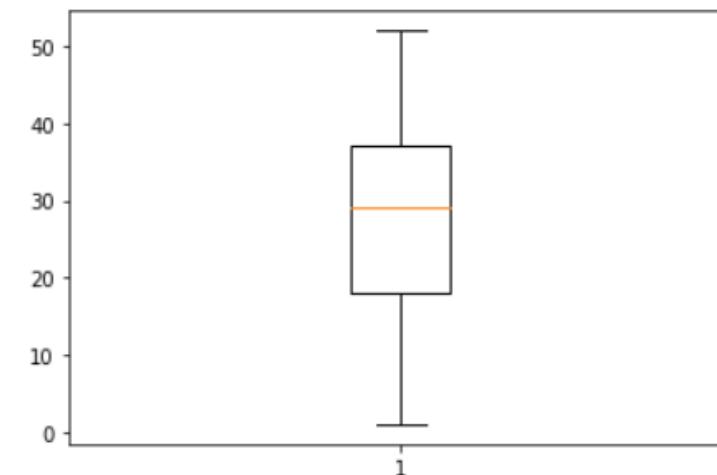
## Módulo 16 – Verificando valores duplicados e outliers

Como vimos no `describe`, algumas colunas parecem indicar a presença de outliers. A seguir, comparamos os diagramas de boxplot de `AveRooms` e de `HouseAge`. Compare os diagramas com o `describe` e perceba a diferença nos outliers entre os diagramas.

```
1 # Visualizando o boxplot para a coluna 'AveRooms'  
2 fig,ax = plt.subplots()  
3  
4 ax.boxplot(casas.AveRooms)  
5  
6 plt.show()
```



```
1 # Visualizando para a coluna 'HouseAge'  
2 fig,ax = plt.subplots()  
3  
4 ax.boxplot(casas.HouseAge)  
5  
6 plt.show()
```



## Módulo 16 – Verificando valores duplicados e outliers

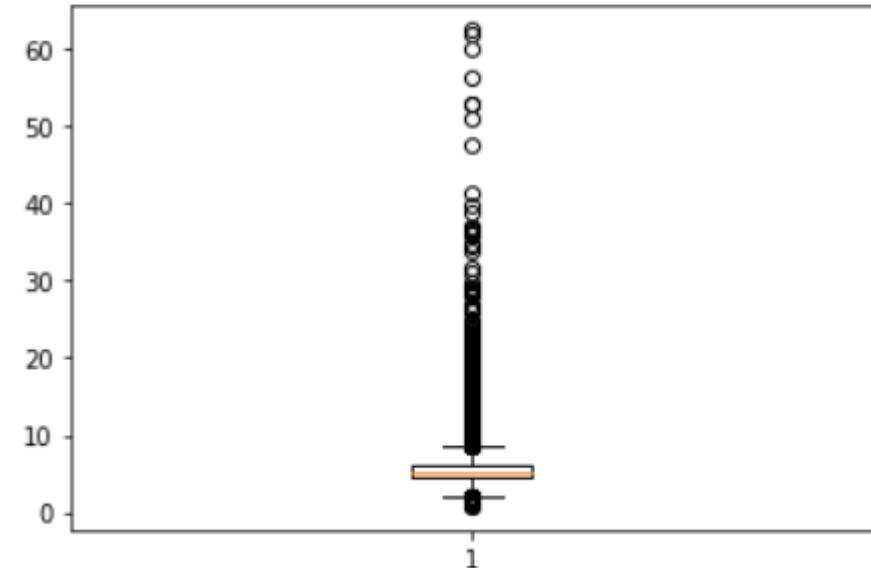
No diagrama de boxplot de AveRooms, vemos dois pontos muito acima dos demais. Podemos retirar estes de nossa base de dados:

```
1 # Tratando os dados  
2 casas = casas[casas.AveRooms < 100]
```

```
1 casas.shape
```

(20638, 9)

```
1 # Visualizando novamente  
2 fig,ax = plt.subplots()  
3  
4 ax.boxplot(casas.AveRooms)  
5  
6 plt.show()
```



## Módulo 16 – Separando a base em treino e teste e usando regressão linear simples

Agora que já fizemos o reconhecimento de nossa base e tratamento de dados básico, podemos começar o procedimento de treinamento de modelos. Como visto nos módulos anteriores, começando separando nossa base em duas, uma base de treino e outra base de teste:

```
1 from sklearn.model_selection import train_test_split
```

```
1 casas.head(2)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.02381	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.97188	2401.0	2.109842	37.86	-122.22	3.585

```
1 X = casas.drop('MedHouseVal',axis=1)
2 y = casas.MedHouseVal
```

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

## Módulo 16 – Separando a base em treino e teste e usando regressão linear simples

Quando fizemos a visualização de dispersão de nossa base, com o pairplot, vimos que algumas colunas tem comportamento aparentemente mais linear com a variável alvo do que outras. Podemos, por exemplo, comparar *MedInc* com *AveRooms* utilizando o modelo de regressão linear. Perceba como o score para *MedInc* é significativamente maior:

```
1 # Criando nossa regressão e fazendo o fit com os dados
2 from sklearn.linear_model import LinearRegression
3 RegSim1 = LinearRegression().fit(X_train.MedInc.values.reshape(-1,1), y_train)
```

```
1 # Verificando o coeficiente de determinação
2 print(RegSim1.score(X_train.MedInc.values.reshape(-1,1), y_train))
3 print(RegSim1.score(X_test.MedInc.values.reshape(-1,1), y_test))
```

```
0.4722764318735512
0.4780251100283853
```

```
1 # Escolhendo outra coluna e verificando se o resultado está melhor
2 RegSim2 = LinearRegression().fit(X_train.AveRooms.values.reshape(-1,1), y_train)
3 print(RegSim2.score(X_train.AveRooms.values.reshape(-1,1), y_train))
4 print(RegSim2.score(X_test.AveRooms.values.reshape(-1,1), y_test))
```

```
0.027892442711821674
0.033409092499478765
```

## Módulo 16 – Separando a base em treino e teste e usando regressão linear simples

Com um loop for, podemos fazer a análise para todas as colunas. Observe como a única que apresenta um valor significativo é a *MedInc*:

```
1 # Usando o for para fazer a regressão de todas as colunas
2 lista = X_train.columns
3
4 for i in lista:
5     print(i)
6     RegSim3 = LinearRegression().fit(X_train[i].values.reshape(-1,1), y_train)
7     print(RegSim3.score(X_train[i].values.reshape(-1,1), y_train))
8     print(RegSim3.score(X_test[i].values.reshape(-1,1), y_test))
```

MedInc  
0.4722764318735512  
0.4780251100283853  
HouseAge  
0.009901923713279026  
0.01435329520341111  
AveRooms  
0.027892442711821674  
0.033409092499478765  
AveBedrms  
0.0039958828973664096  
0.004431841062501274  
Population  
0.0004691454405161277  
0.0005852769184258033  
AveOccup  
0.000999915854954958  
-0.0037063533253798298  
Latitude  
0.021513560930008224  
0.018471709820245708  
Longitude  
0.0017303702792554887  
0.002805043235198257

## Módulo 16 – Entendendo o coeficiente de determinação

Mostramos os valores do método `score` da classe `LinearRegression`. Esse método fornece os mesmos valores que seriam obtidos utilizando diretamente o método `r2_score` do Scikit-Learn. Ou seja, estamos utilizando o **coeficiente de determinação**, R ao quadrado, como métrica. Veja abaixo que os valores obtidos pelos métodos são idênticos. Observe, também, que um dos valores é negativo, o que significa que o modelo é pior do que se considerássemos a média dos dados:

```
1 # Criando o modelo para o AveOccup
2 from sklearn.metrics import r2_score
3 RegSim4 = LinearRegression().fit(X_train.AveOccup.values.reshape(-1,1), y_train)

1 # Avaliando o score
2 print(RegSim4.score(X_train.AveOccup.values.reshape(-1,1), y_train))
3 print(RegSim4.score(X_test.AveOccup.values.reshape(-1,1), y_test))

0.000999915854954958
-0.0037063533253798298

1 predict_test = RegSim4.predict(X_test.AveOccup.values.reshape(-1,1))
2 predict_treino = RegSim4.predict(X_train.AveOccup.values.reshape(-1,1))

1 # Usando o r2_score do Scikit-Learn
2 r2_score(y_train, predict_treino)

0.000999915854954958

1 r2_score(y_test, predict_test)

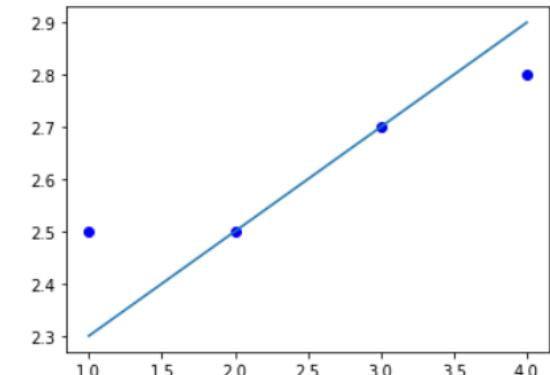
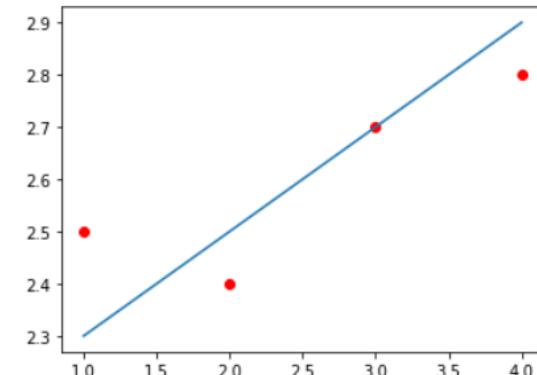
-0.0037063533253798298
```

## Módulo 16 – Entendendo o coeficiente de determinação

No entanto, não necessariamente esta é a melhor métrica para uma regressão linear. Na página anterior, vimos que o modelo pode até ser pior que a média dos dados. Veja abaixo que criamos dois conjuntos de dados fictícios, sendo a diferença entre eles no segundo ponto apenas. Observe que a segunda reta passa por mais pontos da base, e para os demais pontos as distâncias são as mesmas da primeira reta, mas possui menor valor de coeficiente de determinação.

```
1 x = [1,2,3,4]
2 y_true1 = [2.5,2.4,2.7,2.8]
3 y_true2 = [2.5,2.5,2.7,2.8]
4 y_pred = [2.3,2.5,2.7,2.9]

1 import matplotlib.pyplot as plt
2 fig,ax = plt.subplots(ncols=2,figsize=(12,4))
3
4 ax[0].scatter(x,y_true1,c='r')
5 ax[0].plot(x,y_pred)
6 ax[1].scatter(x,y_true2,c='b')
7 ax[1].plot(x,y_pred)
8
9 plt.show()
```



```
1 # Gerando o r2_score para os 2 conjuntos de dados
2 print(r2_score(y_true1,y_pred))
3 print(r2_score(y_true2,y_pred))
```

0.3999999999999999  
0.2592592592592575

## Módulo 16 – Entendendo o coeficiente de determinação

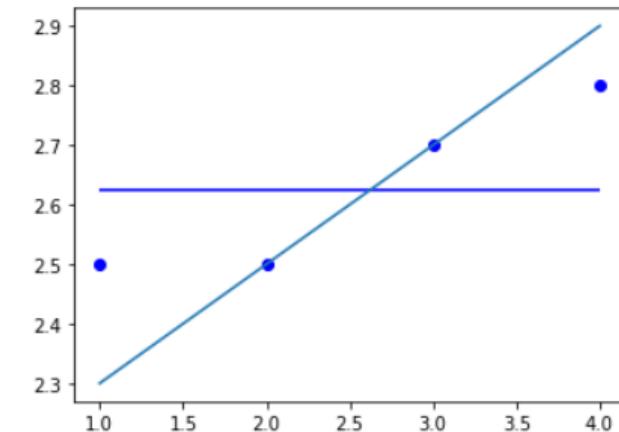
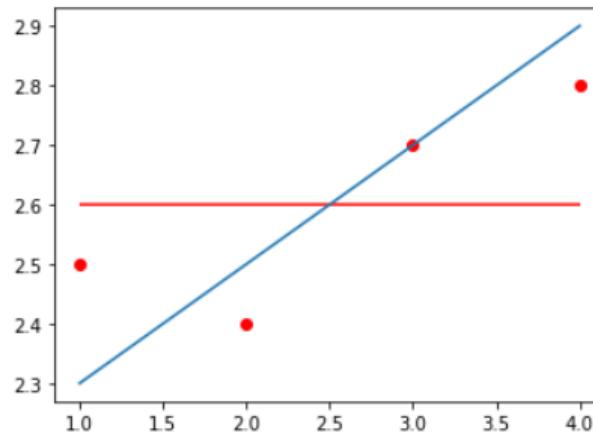
Como isto é possível? Vejamos a seguir a definição do coeficiente. Repare que no numerador comparamos os dados com os valores previstos, enquanto que no denominador comparamos com a média. Ou seja, a razão envolve uma comparação com o valor médio.

$$R^2 = 1 - \frac{\sum(y - \hat{y})^2}{\sum(y - \bar{y})^2}$$

↑ Valor previsto  
↑ Média

Agora faz sentido a primeira reta ter maior coeficiente. Observe que ela melhora significativamente a previsão versus a reta que representa a média (observe, por exemplo, que a distância do segundo ponto para reta é menor que para a média).

```
1 import numpy as np
2 fig,ax = plt.subplots(ncols=2,figsize=(12,4))
3
4 ax[0].scatter(x,y_true1,c='r')
5 ax[0].plot(x,y_pred)
6 ax[0].hlines(y=np.mean(y_true1),xmin=1,xmax=4,color='r')
7 ax[1].scatter(x,y_true2,c='b')
8 ax[1].plot(x,y_pred)
9 ax[1].hlines(y=np.mean(y_true2),xmin=1,xmax=4,color='b')
10
11 plt.show()
```



## Módulo 16 – Métricas de erro para regressão

Sabendo que o coeficiente de determinação não é necessariamente a melhor métrica, vejamos o que fazer neste tipo de situação. Primeiro, vamos gerar um novo conjunto de dados de exemplo:

```
1 import pandas as pd
2 dados = {
3     'X': [1,2,3,4],
4     'Y': [5,7,10,11]
5 }
6
7 dados = pd.DataFrame(dados)
8
9 X = dados.X.values.reshape(-1,1)
10 y = dados.Y
```

```
1 dados.shape[0] # Número de linhas da base
```

4

```
1 X.shape[1] # Número de colunas do modelo
```

1

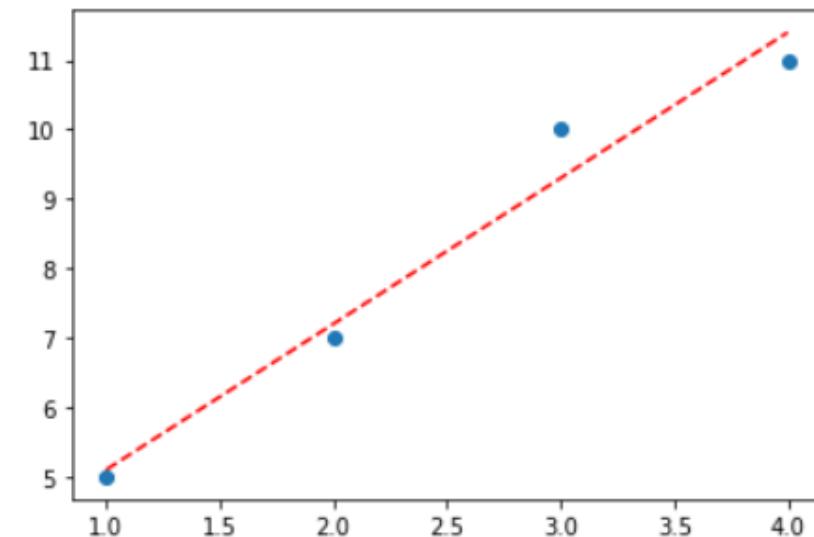
## Módulo 16 – Métricas de erro para regressão

Vamos fazer uma regressão no conjunto de dados e visualizar a reta gerada:

```
1 # Regressão Linear
2 from sklearn import linear_model
3 regLinear = linear_model.LinearRegression()
4 regLinear.fit(X,y)

LinearRegression()

1 # Visualizando essa reta
2 a=regLinear.coef_[0]
3 b=regLinear.intercept_
4 y_reta = a*X+b
5
6 import matplotlib.pyplot as plt
7 fig,ax = plt.subplots()
8
9 ax.scatter(X,y)
10 ax.plot(X,y_reta, '--r')
11
12 plt.show()
```



```
1 y_pred_reg = regLinear.predict(X)
```

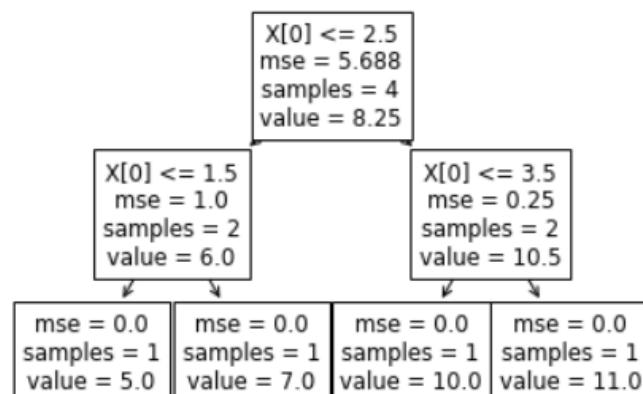
## Módulo 16 – Métricas de erro para regressão

Como já estudamos árvores em módulos anteriores, podemos comparar nosso modelo linear com um modelo de árvore de regressão. Desta forma, temos um padrão de comparação. Para este conjunto de exemplo, a árvore gerada é:

```
1 # Árvore de Decisão
2 from sklearn.tree import DecisionTreeRegressor
3 regArvore = DecisionTreeRegressor(random_state=0)
4 regArvore.fit(X,y)
```

DecisionTreeRegressor(random\_state=0)

```
1 from sklearn import tree
2 tree.plot_tree(regArvore);
```



```
1 y_pred_arvore = regArvore.predict(X)
```

## Módulo 16 – Métricas de erro para regressão

Observe que o score da árvore é melhor que o da regressão linear:

```
1 | from sklearn.metrics import r2_score
```

```
1 | y_true = y
```

```
1 | # Regressão Linear  
2 | r2_score(y_true, y_pred_reg)
```

0.9692307692307693

```
1 | # Árvore de Decisão  
2 | r2_score(y_true, y_pred_arvore)
```

1.0

## Módulo 16 – Métricas de erro para regressão

Uma outra métrica é o chamado coeficiente de determinação ajustado. Ele leva em consideração o tamanho da amostra e o número de variáveis utilizadas na previsão:

$$R^2_{Ajustado} = 1 - \frac{\sum(y - \hat{y})^2}{\sum(y - \bar{y})^2} * \frac{n - 1}{n - p - 1}$$

Tamanho da amostra  
Número de variáveis preditoras

## Módulo 16 – Métricas de erro para regressão

Como a árvore já tinha valor 1 de score, este se mantém. O score do modelo de regressão muda um pouco com esta nova métrica:

```
1 # Regressão Linear  
2 r2_score_modelo = r2_score(y_true, y_pred_reg)  
3 somatorio = 1 - r2_score_modelo  
4 n = dados.shape[0]  
5 p = X.shape[1]  
6  
7  
8 r2_ajus = 1 - somatorio*((n-1)/(n-p-1))  
9 print(r2_ajus)
```

0.953846153846154

```
1 # Árvore de Decisão  
2 r2_score_modelo = r2_score(y_true, y_pred_arvore)  
3 somatorio = 1 - r2_score_modelo  
4 n = dados.shape[0]  
5 p = X.shape[1]  
6  
7  
8 r2_ajus = 1 - somatorio*((n-1)/(n-p-1))  
9 print(r2_ajus)
```

1.0

## Módulo 16 – Métricas de erro para regressão

Existem, ainda, duas métricas muito relevantes para modelos lineares: o erro médio absoluto (EMA), e o erro quadrático médio (EQM). Diferentemente do coeficiente de determinação, eles consideram a distância dos pontos à reta de previsão (e não à média). No caso do EQM, as distâncias são elevadas ao quadrado, penalizando mais outliers presentes na amostra:

$$EMA = \frac{1}{n} \sum |\hat{y} - y|$$
$$EQM = \frac{1}{n} \sum (\hat{y} - y)^2$$

```
1 from sklearn.metrics import mean_absolute_error  
  
1 # Reg Linear  
2 mean_absolute_error(y_true, y_pred_reg)
```

0.3499999999999964

```
1 # Árvore de Decisão  
2 mean_absolute_error(y_true, y_pred_arvore)
```

0.0

```
1 from sklearn.metrics import mean_squared_error  
  
1 # Reg Linear  
2 mean_squared_error(y_true, y_pred_reg)
```

0.1749999999999995

```
1 # Árvore de Decisão  
2 mean_squared_error(y_true, y_pred_arvore)
```

0.0

## Módulo 16 – Métricas de erro para regressão

Assim como para o coeficiente de determinação, o Scikit-Learn possui métodos para estas duas métricas:

```
1 # Criando um novo modelo de regressão para a coluna 'MedInc'  
2 RegSim4 = LinearRegression().fit(X_train.MedInc.values.reshape(-1,1), y_train)
```

```
1 # Fazendo o predict  
2 y_pred = RegSim4.predict(X_test.MedInc.values.reshape(-1,1))
```

```
1 # Avaliando o erro médio absoluto dos dados de TESTE  
2 from sklearn.metrics import mean_absolute_error  
3 mean_absolute_error(y_test, y_pred)
```

0.6294060850621533

```
1 # Avaliando o erro quadrático médio para os dados de TESTE  
2 from sklearn.metrics import mean_squared_error  
3 mean_squared_error(y_test,y_pred)
```

0.700386804985532

## Módulo 16 – Métricas de erro para regressão

Seguindo a mesma rotina que fizemos anteriormente, podemos fazer um loop para verificar as métricas para cada um das colunas de nossa base de dados:

```
1 lista = X_train.columns
2
3 for i in lista:
4     print(i)
5     RegSim4 = LinearRegression().fit(X_train[i].values.reshape(-1,1), y_train)
6     y_pred = RegSim4.predict(X_test[i].values.reshape(-1,1))
7     print(mean_absolute_error(y_test, y_pred))
8     print(mean_squared_error(y_test,y_pred))

MedInc
0.6294060850621533
0.700386804985532
HouseAge
0.9112472186113978
1.3225424434775808
AveRooms
0.8980334210700883
1.2969733418961416
AveBedrms
0.9164225728049346
1.3358550677054248
Population
0.9184219850918659
1.3410163940887445
AveOccup
0.9186157888878567
1.3467749109300464
Latitude
0.9102081408848934
1.3170163476624213
Longitude
0.9150371204426578
1.3380379078276463
```

## Módulo 16 – Regressão linear múltipla

Até o momento, sempre escolhemos colunas individualmente para treinar nosso modelo. Mas, nada impede que sejam escolhidas mais colunas. Nestes casos, dizemos que estamos realizando uma regressão linear múltipla. A organização do código é similar a que vimos anteriormente, bastando passar uma lista com as colunas desejadas:

```
1 # Visualizando o X_train  
2 X_train.head(2)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
<b>18368</b>	5.3468	42.0	6.364322	1.087940	957.0	2.404523	37.16	-121.98
<b>19197</b>	3.9191	36.0	6.110063	1.059748	711.0	2.235849	38.45	-122.69

```
1 # Escolhendo 2 variáveis para fazer a regressão  
2 X_Mult_train = X_train[['AveRooms', 'MedInc']]  
3 X_Mult_test = X_test[['AveRooms', 'MedInc']]
```

```
1 # Fazendo a regressão dessas duas variáveis  
2 regMult = LinearRegression().fit(X_Mult_train, y_train)  
3 y_pred = regMult.predict(X_Mult_test)
```

```
1 # Avaliando o erro  
2 print(mean_absolute_error(y_test, y_pred))  
3 print(mean_squared_error(y_test,y_pred))**
```

0.6198538766617373  
0.6836236839427631

# Módulo 16 – Regressão linear múltipla

Com um loop, podemos verificar qual par de variáveis performa melhor:

```
1 # Usando o for para testar todos os pares de variáveis
2 lista = X_train.columns
3
4 for i in range(0,len(lista)):
5     for j in range(i+1,len(lista)):
6         print(lista[i],lista[j])
7
8         X_Mult_train = X_train[[lista[i],lista[j]]]
9         X_Mult_test = X_test[[lista[i],lista[j]]]
10
11         regMult = LinearRegression().fit(X_Mult_train, y_train)
12         y_pred = regMult.predict(X_Mult_test)
13
14         print(mean_absolute_error(y_test, y_pred))
15         print(mean_squared_error(y_test,y_pred))
```

MedInc HouseAge	0.6031032736795487
MedInc AveRooms	0.6469498044744214
MedInc AveBedrms	0.6198538766617373
MedInc AveOccup	0.6836236839427631
MedInc Population	0.6293603474891636
MedInc Latitude	0.7000859500219594
MedInc Longitude	0.629263237936552
HouseAge AveRooms	0.6989716091404197
HouseAge AveBedrms	0.6285791640431908
HouseAge AveOccup	0.6989059866213563
HouseAge Population	0.6919213687862203
HouseAge Latitude	0.6234988360932905
HouseAge Longitude	0.6280224867409706
HouseAge AveRooms	0.6977953219825376
HouseAge AveBedrms	0.8851613422391501
HouseAge AveOccup	1.2639976421592691
HouseAge Population	0.9097189234157886
HouseAge Latitude	1.3181806604731452
HouseAge Longitude	0.9110706637950305
HouseAge AveRooms	1.3225596984147312
HouseAge AveBedrms	0.9112283717210554
HouseAge AveOccup	1.3271932563519058
HouseAge Population	0.9036568258128097
HouseAge Latitude	1.2966063624435409
HouseAge Longitude	0.9087432485078941
	1.3200482235071982



## Módulo 16 – Regressão linear múltipla

E, claro, podemos treinar o modelo com todas as variáveis existentes:

```
1 # Usando todas as variáveis existentes no modelo
2 regMult = LinearRegression().fit(X_train, y_train)
3 y_pred = regMult.predict(X_test)
4
5 print(mean_absolute_error(y_test, y_pred))
6 print(mean_squared_error(y_test,y_pred))
```

```
0.5269635378186492
0.5122113387754117
```

Perceba que os erros são menores do que os vistos anteriormente com variáveis individuais e com pares de variáveis. No entanto, não necessariamente isto é verdade em todos os casos. Em bases de dados muito grandes, treinar o modelo com todas as variáveis pode ter um custo elevado. Nestes casos, faz sentido realizar estudos preliminares buscando selecionar melhor as variáveis. Deve-se também ter cuidado com overfitting ao se utilizar todas as variáveis.

## Módulo 16 – Comparando modelos

Para finalizar este módulo, vamos comparar nossa regressão com dois outros modelos. Primeiro, com uma árvore de regressão, utilizando todas as nossas variáveis:

```
1 # Importando a árvore de regressão
2 from sklearn import tree
```

```
1 # Criando o regressor
2 regArvore = tree.DecisionTreeRegressor(random_state=42)
```

```
1 # Fazendo o fit
2 regArvore = regArvore.fit(X_train, y_train)
```

```
1 # Avaliando o modelo
2 y_predArvore = regArvore.predict(X_test)
3
4 print(mean_absolute_error(y_test, y_predArvore))
5 print(mean_squared_error(y_test,y_predArvore))
```

0.4639818100775194

0.5301621113025969

Veja que nosso erro média absoluto é menor que o da regressão linear, mas o erro quadrático médio maior. Como o EQM é mais sensível a outliers, isto pode indicar que há alguns outliers na base que estão diminuindo a performance deste modelo.

## Módulo 16 – Comparando modelos

Vamos considerar agora outro modelo, o SVR – *Support Vector Regression*. Veremos detalhes deste modelo em módulos futuros, nosso objetivo no momento é apenas mostrar como é simples modificar o código para treinar um novo modelo. Veja como a estrutura é a mesma, precisamos apenas importar o regressor e fazer as devidas substituições no código:

```
1 # Importando o SVR  
2 from sklearn import svm
```

```
1 # Criando o regressor  
2 regSVR = svm.SVR()
```

```
1 # Fazendo o fit  
2 regSVR.fit(X_train, y_train)
```

SVR()

```
1 # Avaliando o modelo  
2 y_predSVR = regSVR.predict(X_test)  
3  
4 print(mean_absolute_error(y_test, y_predSVR))  
5 print(mean_squared_error(y_test,y_predSVR))
```

0.8701841836434215  
1.36186322768604

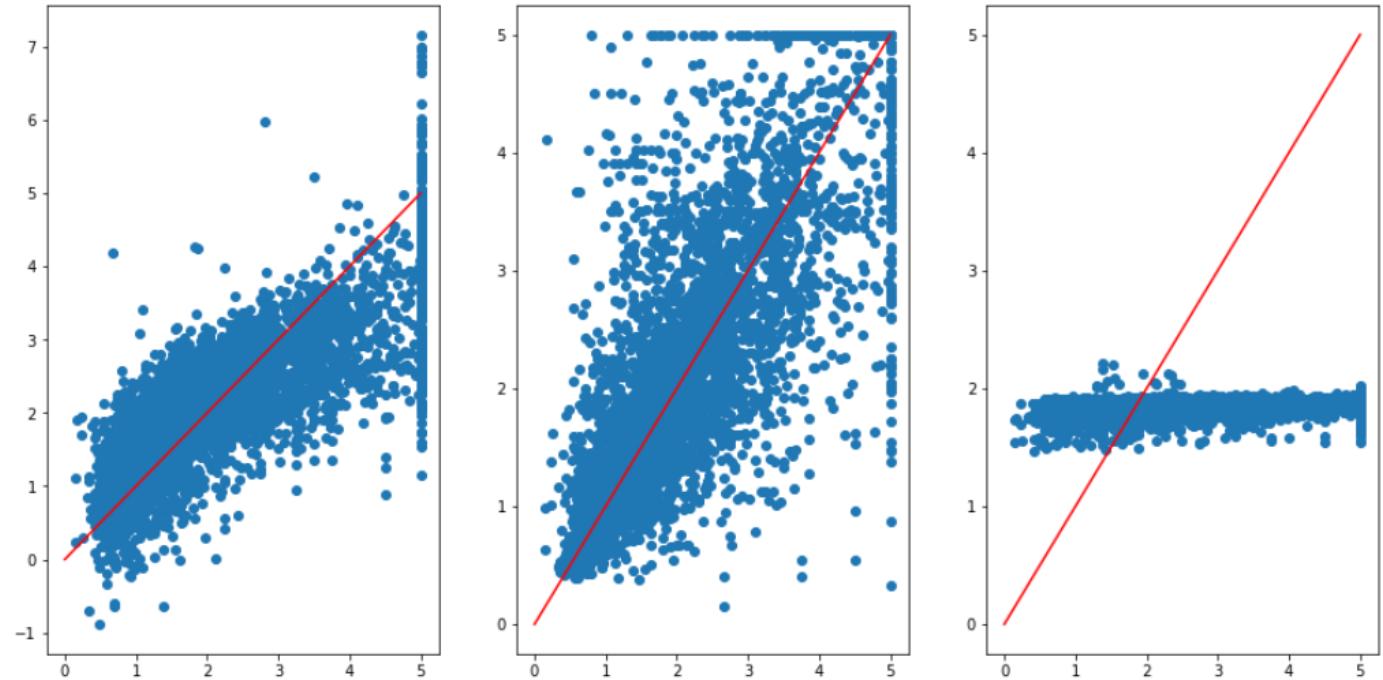
Observe como os erros foram significativamente maiores que os dos outros modelos. Desta forma, para esta base, os outros modelos são mais adequados.

Em módulos futuros, estudaremos o SVR e seu significado.

## Módulo 16 – Comparando modelos

Por fim, podemos visualizar as retas de cada modelo. Observe que o SVR realmente possui um pior ajuste que os demais. A árvore possui pontos mais dispersos que a regressão linear, daí o EQM maior da árvore comparada ao modelo de regressão linear.

```
1 import numpy as np
2
3 fig,ax = plt.subplots(ncols=3,figsize=(16,8))
4
5 x = np.arange(0,6)
6 y = x
7 ax[0].scatter(y_test,y_pred)
8 ax[0].plot(x,y,'r')
9 ax[1].scatter(y_test,y_predArvore)
10 ax[1].plot(x,y,'r')
11 ax[2].scatter(y_test,y_predSVR)
12 ax[2].plot(x,y,'r')
13
14 plt.show()
```



## MÓDULO 17

# Conceitos básicos de SQL para Ciência de Dados

## Módulo 17 – O que é SQL?

SQL vem de *Structured Query Language*, ou Linguagem de Consulta Estruturada. É uma linguagem usada para gerenciar e manipular bancos de dados relacionais.

- Permite criar, consultar, modificar e excluir tabelas e dados
- Possibilita filtrar, ordenar e agrupar dados
- Facilita a combinação e comparação de informações entre tabelas
- Funções e operadores para cálculos e manipulação de texto
- Segurança e controle de acesso aos dados
- Amplamente utilizada e suportada por diversos sistemas de gerenciamento de bancos de dados (SGBDs)

Neste módulo, usaremos o banco SQLite por ser mais simples, mas os fundamentos servem para qualquer outra vertente de SQL. A biblioteca sqlite3, padrão de uma distribuição Python, possibilita uma integração com este tipo de banco de dados, de forma que podemos continuar a utilizar o Jupyter Notebook como viemos fazendo no decorrer do curso.

# Módulo 17 – Usando SQL com o Pandas

Os primeiros passos para trabalhar com o SQLite são:

- Conectar ao banco de dados
- Criar um cursor

Com o cursor criado, podemos executar comandos SQL. E, com o método `fetchall`, resgatar todos os registros retornados pelo comando.

O comando executado no exemplo é

```
SELECT * FROM dados
```

O asterisco significa todas as colunas. Logo, estamos solicitando literalmente que selecione os registros (linhas) de todas as colunas da tabela `dados`.

Pode parecer confuso, mas o retorno é basicamente uma lista de tuplas. Cada tupla representa uma linha da tabela consultada. E esse retorno pode facilmente ser transformado em um dataframe do Pandas.

```
1 # Criando a conexão
2 import sqlite3
3 con = sqlite3.connect('BaseDados.db')

1 # Criando um cursor
2 cur = con.cursor()

1 # Buscando todas as informações da tabela
2 cur.execute('SELECT * FROM dados').fetchall()

[(0,
  1,
  'Maria Eduarda da Rocha',
  38273,
  'maria@hashtag.com',
  0,
  1,
  12,
  1,
  1,
  9.0),
(1, 2, 'Bárbara da Cunha', 63546, 'barbara@hashtag.com', 0, 1, 2, 1, 1, 10.0),
(2, 3, 'Kevin Melo', 80515, 'kevin@hashtag.com', 1, 1, 3, 1, 1, 7.0),
(3,
  4,
  'Pedro Henrique da Costa',
  68004,
  'pedro@hashtag.com',
  1,
  1,
  1,
  1,
  1,
  4.0),
(4, 5, 'Mirella Viana', 28421, 'mirella@hashtag.com', 1, 1, 1, 1, 1, 7.0),
(5, 6, 'Lívia Jesus', 22284, 'livia@hashtag.com', 0, 1, 14, 1, 1, 8.0),
(6, 7, 'Lara Lopes', 38362, 'lara@hashtag.com', 1, 1, 1, 1, 0, None),
(7, 8, 'Lucca Cardoso', 45109, 'lucca@hashtag.com', 1, 1, 3, 1, 0, None),
(8, 9, 'Isabelly Souza', 31859, 'isabelly@hashtag.com', 1, 1, 3, 1, 1, 7.0),
(9, 10, 'Cauã Porto', 42674, 'cauã@hashtag.com', 0, 1, 2, 1, 0, None),
(10, 11, 'Maria Ferreira', 86563, 'maria@hashtag.com', 0, 1, 1, 1, 1, 7.0),
(11, 12, 'Júlia Pinto', 47086, 'julia@hashtag.com', 1, 1, 2, 1, 1, 6.0),
```

## Módulo 17 – Usando SQL com o Pandas

Armazenando o retorno do comando um variável, `resultado`, podemos passar para um DataFrame do Pandas. O objeto cursor possui um atributo, `description`, que armazena diversas informações, dentre elas o nome das colunas. Assim, eis as 5 primeiras linhas de nossa tabela do banco de dados na forma de um DataFrame Pandas:

```
1 # Buscando todas as informações da tabela
2 cur.execute('SELECT * FROM dados')
3 resultado = cur.fetchall()
```

```
1 # Salvando em um DataFrame do pandas
2 import pandas as pd
3 resultado = pd.DataFrame(resultado)
4 resultado.columns = [i[0] for i in cur.description]
5 resultado.head()
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	2	3	Kevin Melo	80515	kevin@hashtag.com	1	1	3	1	1	7.0
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1	1	1	1	4.0
4	4	5	Mirella Viana	28421	mirella@hashtag.com	1	1	1	1	1	7.0

Esta é uma tabela **fictícia** de alunos da Hashtag. Apresenta uma ID, o nome, a matrícula, e o e-mail. Após, duas colunas representando se o aluno tem acesso à plataforma e se este está liberado, onde 0 é “não” e 1 é “sim”. Então, uma coluna contando os dias desde o último acesso do aluno. Por fim, colunas sobre provas com o número da prova feita (1, 2, 3, ...), se a prova foi feita ou não (0 “não” ou 1 “sim”) e a nota da prova.

## Módulo 17 – Usando SQL com o Pandas

Como de costume, de nossos outros módulos, podemos olhar também os últimos 5 registros, para se certificar que toda a base de dados foi obtida com sucesso. E olhar o atributo `shape` para se certificar do tamanho da base:

```
1 # Visualizando os resultados  
2 resultado.tail()
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
38	38	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	3	0	NaN
39	39	19	Melissa Ribeiro	38438	melissa@hashtag.com	1	1	3	3	0	NaN
40	40	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	4	0	NaN
41	41	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	4	1	7.0
42	42	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	5	0	NaN

```
1 # Obtendo o tamanho da base  
2 resultado.shape
```

(43, 11)

## Módulo 17 – Selecionando colunas específicas com SELECT

Ao invés de trazermos toda a base, podemos passar ao SELECT apenas as colunas desejadas. Por exemplo:

```
1 # Selecionando apenas as colunas nome_aluno e cod_matricula
2 cur.execute('SELECT nome_aluno,cod_matricula FROM dados')
3 resultado = cur.fetchall()
4 resultado = pd.DataFrame(resultado)
5 resultado.columns = [i[0] for i in cur.description]
6 resultado.head()
```

	nome_aluno	cod_matricula
0	Maria Eduarda da Rocha	38273
1	Bárbara da Cunha	63546
2	Kevin Melo	80515
3	Pedro Henrique da Costa	68004
4	Mirella Viana	28421

Na prática, quando lidamos com bases de dados muito extensas, este é o procedimento mais usual.

## Módulo 17 – Selecionando colunas específicas com SELECT

Algumas vezes o nome de uma dada coluna pode ser muito longo ou conter caracteres que causam problemas com demais comandos a serem utilizados. Assim, podemos utilizar o comando AS para renomear colunas. No exemplo, renomeamos e-mail para email, retirando o hífen que é inconveniente no código:

```
1 # Selecionando apenas as colunas nome_aluno e e-mail
2 comando = 'SELECT nome_aluno,[e-mail] as email FROM dados'
3 cur.execute(comando)
4 resultado = cur.fetchall()
5 resultado = pd.DataFrame(resultado)
6 resultado.columns = [i[0] for i in cur.description]
7 resultado.head()
```

	nome_aluno	email
0	Maria Eduarda da Rocha	maria@hashtag.com
1	Bárbara da Cunha	barbara@hashtag.com
2	Kevin Melo	kevin@hashtag.com
3	Pedro Henrique da Costa	pedro@hashtag.com
4	Mirella Viana	mirella@hashtag.com

## Módulo 17 – Criando uma função para nossos estudos de SQL com Pandas

Observe nas páginas anteriores que estamos sempre usando a mesma estrutura de criar um comando e gerar um dataframe, exibindo as 5 primeiras linhas deste dataframe. Para evitar mais repetição, podemos encapsular esta lógica em uma função. Tal função recebe o comando SQL, exibe as 5 primeiras linhas de um dataframe criado a partir do comando, assim como o atributo `shape` deste, e retorna o resultado completo caso este queira ser utilizado pelo usuário:

```
1 # Criando uma função para consultar os dados
2 def executa_sql(comando):
3     cur.execute(comando)
4     resultado = cur.fetchall()
5     resultado = pd.DataFrame(resultado)
6     resultado.columns = [i[0] for i in cur.description]
7     print(resultado.shape)
8     display(resultado.head())
9     return resultado
```

## Módulo 17 – Criando uma função para nossos estudos de SQL com Pandas

Assim, podemos repetir nosso último exemplo passando o comando SQL para nossa função:

1	resultado = executa_sql('SELECT nome_aluno,[e-mail] as email FROM dados')
	(43, 2)
nome_aluno	
0	Maria Eduarda da Rocha maria@hashtag.com
1	Bárbara da Cunha barbara@hashtag.com
2	Kevin Melo kevin@hashtag.com
3	Pedro Henrique da Costa pedro@hashtag.com
4	Mirella Viana mirella@hashtag.com

Temos a informação de que o comando retorna 43 linhas e 2 colunas no total. E exibimos os 5 primeiros resultados.

Utilizaremos esta função no restante do módulo para focarmos nos comandos SQL apenas.

## Módulo 17 – Criando uma função para nossos estudos de SQL com Pandas

A função exibe as 5 primeiras linhas, mas retorna o dataframe completo. Assim, se armazenarmos o resultado em uma variável, `resultado` no exemplo, podemos interagir com o dataframe. Abaixo, solicitamos com o método `tail` os 5 últimos registros do dataframe criado a partir de todas as colunas de nossa base de dados armazenado na variável `resultado`.

```
1 | resultado = executa_sql('SELECT * FROM dados')
```

(43, 11)

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	2	3	Kevin Melo	80515	kevin@hashtag.com	1	1	3	1	1	7.0
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1	1	1	1	4.0
4	4	5	Mirella Viana	28421	mirella@hashtag.com	1	1	1	1	1	7.0

```
1 | resultado.tail()
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
38	38	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	3	0	NaN
39	39	19	Melissa Ribeiro	38438	melissa@hashtag.com	1	1	3	3	0	NaN
40	40	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	4	0	NaN
41	41	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	4	1	7.0
42	42	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	5	0	NaN

## Módulo 17 – Lidando com valores repetidos com DISTINCT

Lembrando de nossos módulos anteriores, um dos primeiros cuidados que devemos ter é verificar se há dados duplicados em nossa base. No Pandas, utilizávamos o método `drop_duplicates`:

index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1 Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2 Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	2	3 Kevin Melo	80515	kevin@hashtag.com	1	1	3	1	1	7.0
3	3	4 Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1	1	1	1	4.0
4	4	5 Mirella Viana	28421	mirella@hashtag.com	1	1	1	1	1	7.0
5	5	6 Lívia Jesus	22284	livia@hashtag.com	0	1	14	1	1	8.0
6	6	7 Lara Lopes	38362	lara@hashtag.com	1	1	1	1	0	NaN
7	7	8 Lucca Cardoso	45109	lucca@hashtag.com	1	1	3	1	0	NaN
8	8	9 Isabelly Souza	31859	isabelly@hashtag.com	1	1	3	1	1	7.0
9	9	10 Cauã Porto	42674	cauã@hashtag.com	0	1	2	1	0	NaN
10	10	11 Maria Ferreira	86563	maria@hashtag.com	0	1	1	1	1	7.0
11	11	12 Júlia Pinto	47086	julia@hashtag.com	1	1	2	1	1	6.0
12	12	13 Antônio Azevedo	29022	lovedogs@hashtag.com	0	1	14	1	1	6.0
13	13	14 Laura Melo	50966	laura@hashtag.com	1	1	2	1	1	6.0
14	14	15 Gabriela Costela	21262	gabriela@hashtag.com	1	1	2	1	1	8.0
15	15	16 Eloah Aragão	65749	eloah@hashtag.com	1	1	3	1	1	8.0
16	16	17 Francisco Pires	59518	xlc0@hashtag.com	0	1	13	1	0	NaN
17	17	18 Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	1	1	6.0
18	18	19 Melissa Ribeiro	38438	melissa@hashtag.com	1	1	3	1	1	9.0
19	19	20 Lavinia Carvalho	89428	lavinia@hashtag.com	1	1	1	1	0	NaN
20	20	1 Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	2	1	8.0

Repare acima que ainda aparecem nomes e matrículas repetidos. Isto porque um mesmo aluno pode ter feito mais de uma prova, como denotado na coluna `nr_prova`.

## Módulo 17 – Lidando com valores repetidos com DISTINCT

Por exemplo, vemos abaixo o registro para um estudante específico:

1	resultado[resultado.nome_aluno == 'Eloah Aragão']									
index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
15	15	16 Eloah Aragão	65749	eloah@hashtag.com	1	1	3	1	1	8.0
31	31	16 Eloah Aragão	65749	eloah@hashtag.com	1	1	3	2	1	6.0
37	37	16 Eloah Aragão	65749	eloah@hashtag.com	1	1	3	3	1	9.0
41	41	16 Eloah Aragão	65749	eloah@hashtag.com	1	1	3	4	1	7.0
42	42	16 Eloah Aragão	65749	eloah@hashtag.com	1	1	3	5	0	NaN

Se quisermos considerar apenas o nome e matrícula como critério para descartar duplicatas, no Pandas fazemos:

```
1 # Selecionando apenas as colunas nome_aluno e cod_matricula e selecionando
2 # valores distintos com o pandas
3 resultado = resultado.loc[:,['nome_aluno','cod_matricula']]
4 resultado = resultado.drop_duplicates()
5 resultado.shape
```

(20, 2)

## Módulo 17 – Lidando com valores repetidos com DISTINCT

A lógica pode ser facilmente reproduzida com SQL. Basta passarmos o comando **DISTINCT** após **SELECT** e antes dos nomes das colunas. Assim, expressamos o interesse em selecionar apenas os dados distintos entre si das referidas colunas, o que descarta repetições:

```
1 # Selecionando valores distintos utilizando o SQL
2 resultado = executa_sql('SELECT DISTINCT nome_aluno,cod_matricula FROM dados')
```

(20, 2)

	nome_aluno	cod_matricula
0	Maria Eduarda da Rocha	38273
1	Bárbara da Cunha	63546
2	Kevin Melo	80515
3	Pedro Henrique da Costa	68004
4	Mirella Viana	28421

## Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Vamos novamente pegar toda a base:

```
1 # Filtrando apenas alunos com problema de acesso usando o pandas
2 resultado = executa_sql('SELECT * FROM dados')
```

(43, 11)

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	2	3	Kevin Melo	80515	kevin@hashtag.com	1	1	3	1	1	7.0
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1	1	1	1	4.0
4	4	5	Mirella Viana	28421	mirella@hashtag.com	1	1	1	1	1	7.0

Por vezes, queremos selecionar parte dos dados a partir de um filtro. Por exemplo, com o Pandas, se quiséssemos selecionar apenas os alunos sem acesso à plataforma, faríamos:

```
1 resultado = resultado[resultado.acesso_plataforma == 0]
2 resultado.shape
```

(18, 11)

## Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Com o SQL, temos o comando WHERE:

```
1 # Fazendo esse mesmo filtro utilizando o WHERE do SQL
2 resultado = executa_sql('SELECT * FROM dados WHERE acesso_plataforma = 0')
```

(18, 11)

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	5	6	Lívia Jesus	22284	livia@hashtag.com	0	1	14	1	1	8.0
3	9	10	Cauã Porto	42674	caua@hashtag.com	0	1	2	1	0	NaN
4	10	11	Maria Ferreira	86563	maria@hashtag.com	0	1	1	1	1	7.0

## Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Podemos restringir ainda mais nosso filtro com o operador AND. No caso, além de o aluno não ter mais acesso à plataforma, queremos que o último acesso tenha sido até 2 dias atrás:

```
1 # Verificando se existe algum aluno com problema de acesso E que entrou nos últimos 2 dias
2 resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail] as email FROM dados \
3                         WHERE acesso_plataforma = 0 AND dias_ultimo_acesso <= 2')
```

(3, 3)

	nome_aluno	cod_matricula	email
0	Bárbara da Cunha	63546	barbara@hashtag.com
1	Cauã Porto	42674	cauã@hashtag.com
2	Maria Ferreira	86563	maria@hashtag.com

## Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Também podemos ter filtros com alternativas usando o operador OR. Abaixo, selecionamos alunos sem acesso à plataforma ou que tenham acessado a mesma há mais de 10 dias:

```
1 # Verificando problema de acesso OU acesso acima de 10 dias
2 resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail] as email, \
3                           acesso_plataforma, dias_ultimo_acesso FROM dados \
4                           WHERE acesso_plataforma = 0 OR dias_ultimo_acesso > 10')
```

(8, 5)

	nome_aluno	cod_matricula	email	acesso_plataforma	dias_ultimo_acesso
0	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	12
1	Bárbara da Cunha	63546	barbara@hashtag.com	0	2
2	Lívia Jesus	22284	livia@hashtag.com	0	14
3	Cauã Porto	42674	cauã@hashtag.com	0	2
4	Maria Ferreira	86563	maria@hashtag.com	0	1

## Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Também podemos querer o contrário de um filtro, ou seja, sua negação. Para isso serve o operador NOT. Observe abaixo a comparação entre os dois comandos SQL. No primeiro, selecionamos registros com 10 dias ou mais do último acesso. No segundo, negamos o filtro, o que retorna apenas registros com menos de 10 dias do último acesso:

```
1 # Buscando alunos com mais de 10 dias de acesso
2 resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail] as email, \
3                           acesso_plataforma, dias_ultimo_acesso FROM dados \
4                           WHERE dias_ultimo_acesso >= 10')
```

(5, 5)

	nome_aluno	cod_matricula	email	acesso_plataforma	dias_ultimo_acesso
0	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	12
1	Lívia Jesus	22284	livia@hashtag.com	0	14
2	Antônio Azevedo	29022	lovedogs@hashtag.com	0	14
3	Francisco Pires	59518	xlc0@hashtag.com	0	13
4	Bárbara Freitas	19442	barbara@hashtag.com	0	12

```
1 # Negando essa condição acima com NOT
2 resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail] as email, \
3                           acesso_plataforma, dias_ultimo_acesso FROM dados \
4                           WHERE NOT dias_ultimo_acesso >= 10')
```

(15, 5)

	nome_aluno	cod_matricula	email	acesso_plataforma	dias_ultimo_acesso
0	Bárbara da Cunha	63546	barbara@hashtag.com	0	2
1	Kevin Melo	80515	kevin@hashtag.com	1	3
2	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1
3	Mirella Viana	28421	mirella@hashtag.com	1	1
4	Lara Lopes	38362	lara@hashtag.com	1	1

## Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Neste caso específico, também poderíamos ter feito um comando utilizando o sinal de menor e passando o valor 10:

```
1 ➔ resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail] as email, \  
2                               acesso_plataforma, dias_ultimo_acesso FROM dados \  
3                               WHERE dias_ultimo_acesso < 10')
```

(15, 5)

	nome_aluno	cod_matricula	email	acesso_plataforma	dias_ultimo_acesso
0	Bárbara da Cunha	63546	barbara@hashtag.com	0	2
1	Kevin Melo	80515	kevin@hashtag.com	1	3
2	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1
3	Mirella Viana	28421	mirella@hashtag.com	1	1
4	Lara Lopes	38362	lara@hashtag.com	1	1

## Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Vamos praticar mais um pouco a seleção com filtros e operadores. Vamos buscar os alunos que fizeram a prova 4:

```
1 # Filtrando apenas os alunos que FIZERAM a 4 prova
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, nr_prova, prova_feita, nota_prova \
3                               FROM dados WHERE nr_prova = 4 AND prova_feita = 1')
```

(1, 5)

	nome_aluno	cod_matricula	nr_prova	prova_feita	nota_prova
0	Eloah Aragão	65749	4	1	7.0

Agora, alunos que fizeram as provas 3 ou 4. Observe os parênteses utilizados, para deixar claro que as provas são alternativas, mas filtro de que a prova, seja a 3 ou a 4, tenha sido feita é uma restrição:

```
1 # Filtrando alunos que fizeram a 3 ou 4 prova
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, nr_prova, prova_feita, nota_prova \
3                               FROM dados WHERE (nr_prova = 4 OR nr_prova = 3) AND prova_feita = 1')
```

(3, 5)

	nome_aluno	cod_matricula	nr_prova	prova_feita	nota_prova
0	Maria Eduarda da Rocha	38273	3	1	6.0
1	Eloah Aragão	65749	3	1	9.0
2	Eloah Aragão	65749	4	1	7.0

## Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Por fim, vejamos alunos que tiveram nota maior que 5 na terceira prova:

```
1 # Filtrando os alunos com nota maior que 5 na terceira prova
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, nr_prova, prova_feita, nota_prova \
3                               FROM dados WHERE nr_prova = 3 AND nota_prova > 5')
```

(2, 5)

	nome_aluno	cod_matricula	nr_prova	prova_feita	nota_prova
0	Maria Eduarda da Rocha	38273	3	1	6.0
1	Eloah Aragão	65749	3	1	9.0

Acrescentando o operador NOT, teríamos os com nota menor do que 5. No caso, nenhum aluno.

```
1 # Algum aluno tirou menos de 5 na terceira prova?
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, nr_prova, prova_feita, nota_prova \
3                               FROM dados WHERE nr_prova = 3 AND NOT nota_prova > 5')
```

# Módulo 17 – Funções de agregação

Vamos olhar nossa base novamente, especificamente para a coluna de dias do último acesso:

Por vezes, precisamos de um estudo quantitativo de comportamento de usuários de uma determinada plataforma. Podemos estar interessados em verificar se os mesmos estão acessando a plataforma recorrentemente. Para isso, podemos utilizar funções de agregação, para verificar o valor mínimo, máximo, a média, dentre outras métricas. No nosso exemplo, vemos que o aluno que está a mais dias sem acessar não acessa a plataforma há 14 dias. E a média é de aproximadamente 5 dias:

Obviamente, podemos ter todas as funções utilizadas num mesmo retorno, como colunas distintas:

1 # Visualizando a base	2 resultado_sql = executa_sql('SELECT * FROM dados')	(43, 11)	index id_aluno nome_aluno cod_matricula e-mail acesso_plataforma acesso_liberado dias_ultimo_acesso nr_prova prova_feita nota_prova	0 0 1 Maria Eduarda da Rocha 38273 maria@hashtag.com 0 1 12 1 1 9.0 1 1 2 Bárbara da Cunha 63546 barbara@hashtag.com 0 1 2 1 1 10.0 2 2 3 Kevin Melo 80515 kevin@hashtag.com 1 1 3 1 1 7.0 3 3 4 Pedro Henrique da Costa 68004 pedro@hashtag.com 1 1 1 1 1 4.0 4 4 5 Mirella Viana 28421 mirella@hashtag.com 1 1 1 1 1 7.0
1 # Verificando o valor MÍNIMO da coluna dias_ultimo_acesso	2 resultado_sql = executa_sql('SELECT MIN(dias_ultimo_acesso) as minimo FROM dados')	(1, 1)	minimo	0 1
1 # Verificando o valor MÁXIMO da coluna dias_ultimo_acesso	2 resultado_sql = executa_sql('SELECT MAX(dias_ultimo_acesso) as maximo FROM dados')	(1, 1)	maximo	0 14
1 # Verificando a MÉDIA da coluna dias_ultimo_acesso	2 resultado_sql = executa_sql('SELECT AVG(dias_ultimo_acesso) as media FROM dados')	(1, 1)	media	0 5.116279
1 resultado_sql = executa_sql('SELECT AVG(dias_ultimo_acesso) as media, \\\nMIN(dias_ultimo_acesso) as minimo, \\\nMAX(dias_ultimo_acesso) as maximo \\\nFROM dados')		(1, 3)	media minimo maximo	0 5.116279 1 14

## Módulo 17 – Funções de agregação

No Pandas, para fazer agrupamentos, podemos utilizar o método `groupby`. Por exemplo, agrupando por aluno, contando quantas vezes cada aluno aparece na base de dados com a função de agregação `count`:

```
1 # Relembrando o groupby no pandas para verificar quantas vezes cada aluno aparece
2 resultado_sql.groupby('nome_aluno')['id_aluno'].count()
```

nome_aluno	
Antônio Azevedo	2
Bárbara Freitas	3
Bárbara da Cunha	2
Cauã Porto	1
Eloah Aragão	5
Francisco Pires	1
Gabriela Costela	2
Isabelly Souza	2
Júlia Pinto	3
Kevin Melo	2
Lara Lopes	1
Laura Melo	2
Lavínia Carvalho	1
Lucca Cardoso	1
Lívia Jesus	2
Maria Eduarda da Rocha	4
Maria Ferreira	3
Melissa Ribeiro	3
Mirella Viana	2
Pedro Henrique da Costa	1
Name: id_aluno, dtype: int64	

## Módulo 17 – Funções de agregação

No SQL é bastante similar. Passamos a instrução com a função agregadora e, após, o comando GROUP BY com o nome da coluna:

1	# Fazendo o group by no SQL para contar a quantidade de linhas por alunos
2	resultado_sql = executa_sql('SELECT nome_aluno, COUNT(id_aluno) as registros FROM dados GROUP BY nome_aluno')
(20, 2)	
	nome_aluno registros
0	Antônio Azevedo 2
1	Bárbara Freitas 3
2	Bárbara da Cunha 2
3	Cauã Porto 1
4	Eloah Aragão 5

## Módulo 17 – Funções de agregação

O uso de filtros também é possível. Veja, no exemplo, que estamos solicitando a média das provas com a condição de que exista uma nota (`IS NOT NULL` significa pegar apenas valores não nulos, ou seja, deve haver uma nota). Lembre-se que, em programação, zero e nulo são coisas distintas.

```
1 # Fazendo o group by para calcular a média das provas
2 resultado_sql = executa_sql('SELECT nome_aluno, AVG(nota_prova) as media FROM dados \
3                               where nota_prova is not null \
4                               GROUP BY nome_aluno')
```

(15, 2)

	nome_aluno	media
0	Antônio Azevedo	6.0
1	Bárbara Freitas	8.0
2	Bárbara da Cunha	10.0
3	Eloah Aragão	7.5
4	Gabriela Costela	8.0

## Módulo 17 – Ordenando registros com ORDER BY

Seria interessante poder ordenar os resultados. Para isso, podemos utilizar o comando **ORDER BY** seguido do que queremos ordenar e de que forma. No caso, queremos ordenar a média de notas **AVG(nota\_prova)** de forma decrescente **DESC**. Se quiséssemos crescente, utilizariamos o comando **ASC**.

```
1 # Ordenando as notas da maior para a menor
2 resultado_sql = executa_sql('SELECT nome_aluno, AVG(nota_prova) as media FROM dados \
3                               where nota_prova is not null \
4                               GROUP BY nome_aluno \
5                               ORDER BY AVG(nota_prova) desc')
```

(15, 2)

	nome_aluno	media
0	Bárbara da Cunha	10.0
1	Melissa Ribeiro	8.0
2	Lívia Jesus	8.0
3	Júlia Pinto	8.0
4	Gabriela Costela	8.0

## Módulo 17 – Limitando a base com LIMIT

Observe na página anterior que há 15 registros no total. Poderíamos limitar nosso retorno aos 5 primeiros com o comando **LIMIT 5**:

```
1 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, AVG(nota_prova) as media_prova, \
2                               count(nota_prova) as qtd_provas FROM dados \
3                               WHERE nota_prova IS NOT NULL \
4                               GROUP BY nome_aluno,cod_matricula \
5                               ORDER BY AVG(nota_prova) desc \
6                               LIMIT 5')
```

(5, 4)

	nome_aluno	cod_matricula	media_prova	qtd_provas
0	Bárbara da Cunha	63546	10.0	1
1	Bárbara Freitas	19442	8.0	2
2	Gabriela Costela	21262	8.0	1
3	Júlia Pinto	47086	8.0	2
4	Lívia Jesus	22284	8.0	1

## Módulo 17 – Filtros com HAVING

Vejamos o que ocorre quando tentamos realizar uma agregação utilizando WHERE na mesma query:

```
1 # Utilizando o WHERE para selecionar apenas alunos com média maior que 5
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, AVG(nota_prova) as media_prova, \
3                               count(nota_prova) as qtd_provas FROM dados \
4                               WHERE nota_prova IS NOT NULL AND AVG(nota_prova) > 5 \
5                               GROUP BY nome_aluno,cod_matricula \
6                               ORDER BY AVG(nota_prova) desc \
7                               LIMIT 5')
```

```
-----
OperationalError                                  Traceback (most recent call last)
Cell In[13], line 2
      1 # Utilizando o WHERE para selecionar apenas alunos com média maior que 5
----> 2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, AVG(nota_prova) as media_prova, \
      3                                     count(nota_prova) as qtd_provas FROM dados \
      4                                     WHERE nota_prova IS NOT NULL AND AVG(nota_prova) > 5 \
      5                                     GROUP BY nome_aluno,cod_matricula \
      6                                     ORDER BY AVG(nota_prova) desc \
      7                                     LIMIT 5')

Cell In[4], line 3, in executa_sql(comando)
      2 def executa_sql(comando):
----> 3     cur.execute(comando)
      4     resultado = cur.fetchall()
      5     resultado = pd.DataFrame(resultado)

OperationalError: misuse of aggregate: AVG()
```

## Módulo 17 – Filtros com HAVING

Isto ocorre pois há uma ordem de execução dos comandos em um query SQL:



Observe que o WHERE é avaliado antes de agrupamentos. Para resolver o erro, precisamos utilizar o comando HAVING:

```
1 # Trocando o WHERE pelo HAVING
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, AVG(notas_prova) as media_prova, \
3                               count(notas_prova) as qtd_provas FROM dados \
4                               WHERE notas_prova IS NOT NULL \
5                               GROUP BY nome_aluno,cod_matricula \
6                               HAVING AVG(notas_prova) > 7 \
7                               ORDER BY AVG(notas_prova) desc')
```

(8, 4)

	nome_aluno	cod_matricula	media_prova	qtd_provas
0	Bárbara da Cunha	63546	10.0	1
1	Bárbara Freitas	19442	8.0	2
2	Gabriela Costela	21262	8.0	1
3	Júlia Pinto	47086	8.0	2
4	Lívia Jesus	22284	8.0	1

## Módulo 17 – Filtros com HAVING

Podemos visualizar todos os registros para nos certificar que o filtro foi corretamente aplicado:

1	display(resultado_sql)	nome_aluno	cod_matricula	media_prova	qtd_provas
0	Bárbara da Cunha	63546	10.000000	1	
1	Bárbara Freitas	19442	8.000000	2	
2	Gabriela Costela	21262	8.000000	1	
3	Júlia Pinto	47086	8.000000	2	
4	Lívia Jesus	22284	8.000000	1	
5	Melissa Ribeiro	38438	8.000000	2	
6	Maria Eduarda da Rocha	38273	7.666667	3	
7	Eloah Aragão	65749	7.500000	4	

## Módulo 17 – Condicionais com CASE

Assim como em Python, com IF/ELIF/ELSE, também é possível construir estruturas condicionais em SQL com cláusulas CASE. A primeira condição que for verdadeira será retornada. Cada condição tem o formato

WHEN <condição> THEN <valor>

Após as condições, podemos colocar um ELSE caso nenhuma das condições anteriores seja verdadeira. E precisamos terminar um CASE com END. Vamos definir a seguinte regra para aplicar na nossa base de dados:

- Nota > 7: aprovado
- Nota > 5 e qtd\_provas = 1: fazer prova 2
- Nota > 5 e qtd\_provas = 2: fazer prova final
- Nota > 5 e qtd\_provas > 2: revisar matéria
- Nota < 5 e qtd\_provas = 1: revisar matéria
- Nota < 5 e qtd\_provas > 1: lista de exercícios

## Módulo 17 – Condicionais com CASE

Para começar, vamos implementar a primeira condição:

```
1 # Vamos utilizar o CASE para colocar apenas a primeira condição
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, \
3                               AVG(nota_prova) as media_prova, \
4                               count(nota_prova) as qtd_provas, \
5                               (CASE \
6                                 WHEN AVG(nota_prova) > 7 THEN "Aprovado" \
7                               END) as situacao_aluno \
8                               FROM dados \
9                               WHERE nota_prova IS NOT NULL \
10                              GROUP BY nome_aluno,cod_matricula \
11                              ORDER BY AVG(nota_prova) desc')
```

(15, 5)

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.0	1	Aprovado
1	Bárbara Freitas	19442	8.0	2	Aprovado
2	Gabriela Costela	21262	8.0	1	Aprovado
3	Júlia Pinto	47086	8.0	2	Aprovado
4	Lívia Jesus	22284	8.0	1	Aprovado

## Módulo 17 – Condicionais com CASE

Visualizando todo o retorno, vemos que os casos não verdadeiros para nossa condição ficaram com o valor `None`, já que não especificamos o valor desejado:

1	# Visualizando o resultado
2	display(resultado_sql)
nome_aluno cod_matricula media_prova qtd_provas situacao_aluno	
0 Bárbara da Cunha 63546 10.000000 1 Aprovado	
1	Bárbara Freitas 19442 8.000000 2 Aprovado
2	Gabriela Costela 21262 8.000000 1 Aprovado
3	Júlia Pinto 47086 8.000000 2 Aprovado
4	Lívia Jesus 22284 8.000000 1 Aprovado
5	Melissa Ribeiro 38438 8.000000 2 Aprovado
6	Maria Eduarda da Rocha 38273 7.666667 3 Aprovado
7	Eloah Aragão 65749 7.500000 4 Aprovado
8	Isabelly Souza 31859 7.000000 1 None
9	Kevin Melo 80515 7.000000 1 None
10	Mirella Viana 28421 7.000000 1 None
11	Maria Ferreira 86563 6.500000 2 None
12	Antônio Azevedo 29022 6.000000 1 None
13	Laura Melo 50966 6.000000 1 None
14	Pedro Henrique da Costa 68004 4.000000 1 None

## Módulo 17 – Condicionais com CASE

Vamos adicionar todas as condições. Observe que colocamos um ELSE para o caso de haver alguma condição que não se encaixe nas demais listadas.

```
1 # Adicionando TODAS as condições
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, \
3                             AVG(nota_prova) as media_prova, \
4                             count(nota_prova) as qtd_provas, \
5                             (CASE \
6                               WHEN AVG(nota_prova) > 7 THEN "Aprovado" \
7                               WHEN AVG(nota_prova) > 5 AND count(nota_prova) = 1 THEN "Fazer prova 2" \
8                               WHEN AVG(nota_prova) > 5 AND count(nota_prova) = 2 THEN "Fazer prova final" \
9                               WHEN AVG(nota_prova) > 5 AND count(nota_prova) > 2 THEN "Revisar matéria" \
10                              WHEN AVG(nota_prova) < 5 AND count(nota_prova) = 1 THEN "Revisar matéria" \
11                              WHEN AVG(nota_prova) < 5 AND count(nota_prova) > 1 THEN "Lista de exercícios" \
12                                ELSE "Verificar manualmente" \
13                            END) as situacao_aluno \
14                           FROM dados \
15                           WHERE nota_prova IS NOT NULL \
16                           GROUP BY nome_aluno,cod_matricula \
17                           ORDER BY AVG(nota_prova) desc')
```

(15, 5)

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.0	1	Aprovado
1	Bárbara Freitas	19442	8.0	2	Aprovado
2	Gabriela Costela	21262	8.0	1	Aprovado
3	Júlia Pinto	47086	8.0	2	Aprovado
4	Lívia Jesus	22284	8.0	1	Aprovado

## Módulo 17 – Condicionais com CASE

Observe, agora, que todas as situações foram contempladas com alguma condição:

1	display(resultado_sql)				
	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.000000	1	Aprovado
1	Bárbara Freitas	19442	8.000000	2	Aprovado
2	Gabriela Costela	21262	8.000000	1	Aprovado
3	Júlia Pinto	47086	8.000000	2	Aprovado
4	Lívia Jesus	22284	8.000000	1	Aprovado
5	Melissa Ribeiro	38438	8.000000	2	Aprovado
6	Maria Eduarda da Rocha	38273	7.666667	3	Aprovado
7	Eloah Aragão	65749	7.500000	4	Aprovado
8	Isabelly Souza	31859	7.000000	1	Fazer prova 2
9	Kevin Melo	80515	7.000000	1	Fazer prova 2
10	Mirella Viana	28421	7.000000	1	Fazer prova 2
11	Maria Ferreira	86563	6.500000	2	Fazer prova final
12	Antônio Azevedo	29022	6.000000	1	Fazer prova 2
13	Laura Melo	50966	6.000000	1	Fazer prova 2
14	Pedro Henrique da Costa	68004	4.000000	1	Revisar matéria

## Módulo 17 – Subquery

Vamos simplificar um pouco nossa condicional, conforme segue:

```
1 # Utilizando parte do CASE que vimos na última aula
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, \
3                               AVG(nota_prova) as media_prova, \
4                               count(nota_prova) as qtd_provas, \
5                               (CASE \
6                                 WHEN AVG(nota_prova) > 7 THEN "Aprovado" \
7                                 WHEN AVG(nota_prova) > 5 AND count(nota_prova) = 1 THEN "Fazer prova 2" \
8                               END) as situacao_aluno \
9                               FROM dados \
10                              WHERE nota_prova IS NOT NULL \
11                              GROUP BY nome_aluno,cod_matricula \
12                              ORDER BY AVG(nota_prova) desc')
```

(15, 5)

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.0	1	Aprovado
1	Bárbara Freitas	19442	8.0	2	Aprovado
2	Gabriela Costela	21262	8.0	1	Aprovado
3	Júlia Pinto	47086	8.0	2	Aprovado
4	Lívia Jesus	22284	8.0	1	Aprovado



## Módulo 17 – Subquery

Poderíamos estruturar a query conforme a seguir. Observe que estamos aplicando a estrutura condicional ao resultado de uma query apresentada após o comando FROM. Nestes casos, dizemos estar utilizando uma *subquery*. Uma subquery pode aparecer no lugar de uma expressão, se esta retornar um valor único.

```
1 # Transformando a query sem o CASE em uma subquery
2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, media_prova, qtd_provas, \
3                               (CASE \
4                                 WHEN media_prova > 7 THEN "Aprovado" \
5                                 WHEN media_prova > 5 AND qtd_provas = 1 THEN "Fazer prova 2" \
6                                 END) as situacao_aluno \
7                               FROM (SELECT nome_aluno, cod_matricula, \
8                                     AVG(notas_prova) as media_prova, \
9                                     count(notas_prova) as qtd_provas \
10                                FROM dados \
11                               WHERE notas_prova IS NOT NULL \
12                               GROUP BY nome_aluno,cod_matricula \
13                               ORDER BY AVG(notas_prova) desc)')
```

(15, 5)

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.0	1	Aprovado
1	Bárbara Freitas	19442	8.0	2	Aprovado
2	Gabriela Costela	21262	8.0	1	Aprovado
3	Júlia Pinto	47086	8.0	2	Aprovado
4	Lívia Jesus	22284	8.0	1	Aprovado

## Módulo 17 – Outros filtros - IN

Em alguns contextos, gostaríamos de fazer filtros de valores dentro de um dado intervalo ou conjunto. Com o visto até o momento, poderíamos fazer utilizando WHERE para dias do último acesso entre 10 e 12:

```
1 # Verificando os dias de último acesso iguais a 10,11,12 usando o WHERE + AND
2 resultado_sql = executa_sql('SELECT * FROM dados WHERE dias_ultimo_acesso >= 10 AND dias_ultimo_acesso <=12')
```

(7, 11)											
	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	17	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	1	1	6.0
2	20	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	2	1	8.0
3	32	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	2	1	10.0
4	34	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	3	1	6.0

Outra forma de fazer seria utilizar o comando IN e passar uma listagem de valores, indicando que gostaríamos de filtrar nossa base para apenas casos onde tal coluna apresenta valores neste conjunto:

```
1 # Alterando para usar o IN
2 resultado_sql = executa_sql('SELECT * FROM dados \
3 WHERE dias_ultimo_acesso IN (10,11,12)')
```

(7, 11)											
	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	17	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	1	1	6.0
2	20	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	2	1	8.0
3	32	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	2	1	10.0
4	34	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	3	1	6.0

## Módulo 17 – Outros filtros - LIKE

Combinando com DISTINCT, podemos pegar alunos por número de matrícula ou por nome:

```
1 # Utilizando o IN para filtrar as matrículas 28421,38273,42674,65749
2 resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula FROM dados \
3                               WHERE cod_matricula IN (28421,38273,42674,65749)')
```

(4, 2)

	nome_aluno	cod_matricula
0	Maria Eduarda da Rocha	38273
1	Mirella Viana	28421
2	Cauã Porto	42674
3	Eloah Aragão	65749

```
1 # Verificando alunos que o nome seja igual Maria Ferreira utilizando o IN
2 resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula FROM dados \
3                               WHERE nome_aluno IN ("Maria Ferreira")')
```

(1, 2)

	nome_aluno	cod_matricula
0	Maria Ferreira	86563



## Módulo 17 – Outros filtros - LIKE

O comando **LIKE** no SQL é utilizado para buscar registros em uma tabela que correspondam a um padrão específico. Ele é frequentemente usado em combinação com a cláusula **WHERE** para filtrar resultados com base em colunas de texto. Assim, o mesmo último exemplo da página anterior poderia ser feito como:

```
1 # Fazendo a mesma consulta utilizando o LIKE
2 resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula FROM dados \
3                               WHERE nome_aluno LIKE "Maria Ferreira"')
```

(1, 2)

	nome_aluno	cod_matricula
0	Maria Ferreira	86563

## Módulo 17 – Outros filtros - LIKE

Mas a vantagem do LIKE é possibilidade de utilizar caracteres curinga para definir padrões:

- %: Representa qualquer sequência de caracteres (incluindo nenhuma sequência)
- \_: Representa exatamente um caractere

```
1 # Verificando todas as alunas chamadas Maria
2 resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula FROM dados \
3 WHERE nome_aluno LIKE "Maria%"')
```

(2, 2)

	nome_aluno	cod_matricula
0	Maria Eduarda da Rocha	38273
1	Maria Ferreira	86563

```
1 # Verificando se o e-mail possui o caracter !
2 resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula,[e-mail],acesso_plataforma,acesso_liberado FROM dados \
3 WHERE [e-mail] LIKE "%!%"')
```

(1, 5)

	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado
0	Francisco Pires	59518	xlc0@hashtag.com	0	1

```
1 # Verificando se possui !,ã ou í
2 resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula,[e-mail],acesso_plataforma,acesso_liberado FROM dados \
3 WHERE [e-mail] LIKE "%!%" \
4 OR [e-mail] LIKE "%ã%" \
5 OR [e-mail] LIKE "%í%"')
```

(3, 5)

	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado
0	Lívia Jesus	22284	livia@hashtag.com	0	1
1	Cauã Porto	42674	cauã@hashtag.com	0	1
2	Francisco Pires	59518	xlc0@hashtag.com	0	1

## Módulo 17 – Unindo bases com JOIN

Em uma situação real, por vezes temos mais de uma tabela, cada uma com dados dentro de um determinado contexto de nosso problema. Por exemplo, nossa base provavelmente seria melhor aplicada numa situação real com três tabelas. Uma para dados cadastrais dos alunos, outra para informações sobre a plataforma, e outra para informações sobre provas:

```
1 # Visualizando a tabela alunos  
2 alunos = executa_sql('SELECT * FROM alunos')
```

	(20, 5)	index	id_aluno	nome_aluno	cod_matricula	e-mail
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	
2	2	3	Kevin Melo	80515	kevin@hashtag.com	
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	
4	4	5	Mirella Viana	28421	mirella@hashtag.com	

```
1 # Visualizando a tabela plataforma  
2 plataforma = executa_sql('SELECT * FROM plataforma')
```

	(20, 6)	index	id_plataforma	cod_matricula	acesso_plataforma	acesso_liberado	dias_ultimo_acesso
0	0	0	1	38273	0	1	12
1	1	1	2	63546	0	1	2
2	2	2	3	80515	1	1	3
3	3	3	4	68004	1	1	1
4	4	4	5	28421	1	1	1

```
1 # Visualizando a tabela provas  
2 provas = executa_sql('SELECT * FROM provas')
```

	(43, 6)	index	id_prova	cod_matricula	nr_prova	prova_feita	nota_prova
0	0	0	1	38273	1	1	9.0
1	1	1	2	63546	1	1	10.0
2	2	2	3	80515	1	1	7.0
3	3	3	4	68004	1	1	4.0
4	4	4	5	28421	1	1	7.0

## Módulo 17 – Unindo bases com JOIN

Esta divisão permite trabalhar com cada tabela isoladamente de acordo com o contexto desejado, diminuindo riscos de alterações indevidas em dados não diretamente relacionados a um determinado contexto. Observe que, em cada tabela, há uma coluna de identificação *id\_*, e que todas elas podem ser relacionadas a partir do código de matrícula do aluno:

```
1 # Visualizando a tabela alunos
2 alunos = executa_sql('SELECT * FROM alunos')
```

	(20, 5)	index	id_aluno	nome_aluno	cod_matricula	e-mail
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	
2	2	3	Kevin Melo	80515	kevin@hashtag.com	
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	
4	4	5	Mirella Viana	28421	mirella@hashtag.com	

```
1 # Visualizando a tabela provas
2 provas = executa_sql('SELECT * FROM provas')
```

	(43, 6)	index	id_prova	cod_matricula	nr_prova	prova_feita	nota_prova
0	0	0	1	38273	1	1	9.0
1	1	1	2	63546	1	1	10.0
2	2	2	3	80515	1	1	7.0
3	3	3	4	68004	1	1	4.0
4	4	4	5	28421	1	1	7.0

```
1 # Visualizando a tabela plataforma
2 plataforma = executa_sql('SELECT * FROM plataforma')
```

	(20, 6)	index	id_plataforma	cod_matricula	acesso_plataforma	acesso_liberado	dias_ultimo_acesso
0	0	0	1	38273	0	1	12
1	1	1	2	63546	0	1	2
2	2	2	3	80515	1	1	3
3	3	3	4	68004	1	1	1
4	4	4	5	28421	1	1	1

## Módulo 17 – Unindo bases com JOIN

Seguindo o padrão que adotamos até aqui, vamos verificar um exemplo de junção de duas tabelas com o Pandas e, depois, replicar o conceito com SQL.

Para começar, vamos criar dois dataframes simples:

```
1 # Criando 2 dataframes
2 dic1 = {
3     "nomes": ['Nome1', 'Nome2', 'Nome3'],
4     "valores": [1,2,3]
5 }
6
7 base_dic1 = pd.DataFrame(dic1)
8
9 dic2 = {
10    "nomes": ['Nome1', 'Nome2', 'Nome4'],
11    "valores": [9,8,7]
12 }
13
14 base_dic2 = pd.DataFrame(dic2)
```

```
1 base_dic2
```

	nomes	valores
0	Nome1	9
1	Nome2	8
2	Nome4	7

## Módulo 17 – Unindo bases com JOIN

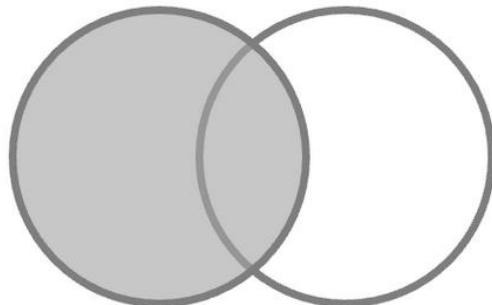
No Pandas, há o método `merge`, que permite juntar, ou mergear, dois dataframes. Passamos cada dataframe, o tipo de junção, e a coluna que deve ser considerada referência para junção.

No caso da junção do tipo `outer`, temos uma união das duas bases (diagrama abaixo).

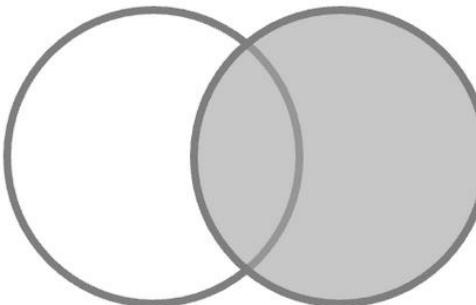
```
1 base_merge = pd.merge(  
2     base_dic1, # <- primeira base  
3     base_dic2, # <- segunda base  
4     how='outer', # <- tipo de junção que vamos fazer  
5     on="nomes" # <- coluna que vamos usar para fazer essa junção das bases  
6 ).  
7  
8 display(base_merge)
```

	nomes	valores_x	valores_y
0	Nome1	1.0	9.0
1	Nome2	2.0	8.0
2	Nome3	3.0	NaN
3	Nome4	NaN	7.0

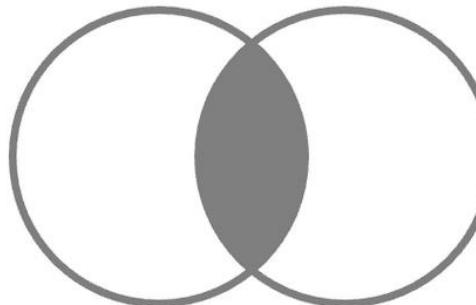
**LEFT JOIN**



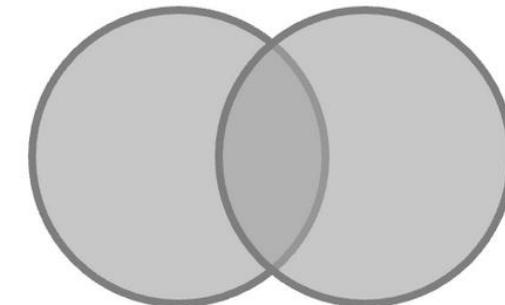
**RIGHT JOIN**



**INNER JOIN**



**FULL (OUTER) JOIN**



## Módulo 17 – Unindo bases com JOIN

Replicando a lógica para nosso caso de estudo, podemos unir as bases *alunos* e *plataforma*, considerando a coluna *cod\_matricula*.

Observe, ao lado, que fizemos uma junção do tipo **right**. Ou seja, retorna todos os registros da segunda tabela (*plataforma*) e os correspondentes da primeira tabela (*alunos*). Se não houver correspondência, os campos da primeira tabela aparecerão como nulos no resultado.

```
1 # Unindo a tabela alunos com a tabela plataforma
2 base_merge = pd.merge(
3     alunos, # <- primeira base
4     plataforma, # <- segunda base
5     how='right', # <- tipo de junção que vamos fazer
6     on="cod_matricula" # <- coluna que vamos usar para fazer essa junção das bases
7 )
8
9 display(base_merge)
```

	index_x	id_aluno	nome_aluno	cod_matricula	e-mail	index_y	id_plataforma	acesso_plataforma	acesso_liberado	dias_ultimo_acesso
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	0	1	12
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	1	2	0	1	2
2	2	3	Kevin Melo	80515	kevin@hashtag.com	2	3	1	1	3
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	3	4	1	1	1
4	4	5	Mirella Viana	28421	mirella@hashtag.com	4	5	1	1	1
5	5	6	Lívia Jesus	22284	livia@hashtag.com	5	6	0	1	14
6	6	7	Lara Lopes	38362	lara@hashtag.com	6	7	1	1	1
7	7	8	Lucca Cardoso	45109	lucca@hashtag.com	7	8	1	1	3
8	8	9	Isabelly Souza	31859	isabelly@hashtag.com	8	9	1	1	3
9	9	10	Cauã Porto	42674	cauã@hashtag.com	9	10	0	1	2
10	10	11	Maria Ferreira	86563	maria@hashtag.com	10	11	0	1	1
11	11	12	Júlia Pinto	47086	julia@hashtag.com	11	12	1	1	2
12	12	13	Antônio Azevedo	29022	lovedogs@hashtag.com	12	13	0	1	14
13	13	14	Laura Melo	50966	laura@hashtag.com	13	14	1	1	2
14	14	15	Gabriela Costela	21262	gabriela@hashtag.com	14	15	1	1	2
15	15	16	Eloah Aragão	65749	eloah@hashtag.com	15	16	1	1	3
16	16	17	Francisco Pires	59518	xlc0@hashtag.com	16	17	0	1	13
17	17	18	Bárbara Freitas	19442	barbara@hashtag.com	17	18	0	1	12
18	18	19	Melissa Ribeiro	38438	melissa@hashtag.com	18	19	1	1	3
19	19	20	Lavinia Carvalho	89428	lavinia@hashtag.com	19	20	1	1	1

## Módulo 17 – Unindo bases com JOIN

Podemos verificar que a junção partiu da tabela *plataforma* verificando que não há registros nulos oriundo desta tabela na base após o merge. Podemos, também, verificar que uma determinada matrícula, que está em *plataforma*, não está em *alunos*:

```
1 # Verificando se algum aluno não possui registro na tabela plataforma  
2 base_merge[base_merge.id_plataforma.isnull()]
```

index_x	id_aluno	nome_aluno	cod_matricula	e-mail	index_y	id_plataforma	acesso_plataforma	acesso_liberado	dias_ultimo_acesso
---------	----------	------------	---------------	--------	---------	---------------	-------------------	-----------------	--------------------

```
1 # Verificando se algum registro da tabela plataforma não possui a matrícula na tabela aluno  
2 base_merge[base_merge.id_aluno.isnull()]
```

index_x	id_aluno	nome_aluno	cod_matricula	e-mail	index_y	id_plataforma	acesso_plataforma	acesso_liberado	dias_ultimo_acesso
---------	----------	------------	---------------	--------	---------	---------------	-------------------	-----------------	--------------------

```
1 alunos[alunos.cod_matricula == 91047]
```

index	id_aluno	nome_aluno	cod_matricula	e-mail
-------	----------	------------	---------------	--------

```
1 base_merge.shape
```

(20, 10)

## Módulo 17 – Unindo bases com JOIN

Agora que vimos no Pandas, vamos ver no SQL. Abaixo, estamos fazendo um LEFT JOIN da tabela *alunos*, chamada de *a*, com a tabela *plataforma*, chamada de *p*. Estes “apelidos” *a* e *p* ajudam a indicar a origem das colunas em nossa base após a junção. Observe que indicamos no SELECT a origem de cada coluna com *a.* ou *p.* antes do nome da coluna. O comando ON serve para indicar a coluna que servirá de referência para a junção.

```
1 # Trazendo as informações da tabela aluno e fazendo o join com a tabela plataforma
2 aluno_plataforma = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
3                                     a.cod_matricula \
4                                     FROM alunos a \
5                                     LEFT JOIN plataforma p \
6                                     ON a.cod_matricula = p.cod_matricula')
```

(20, 4)

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0	63546
2	Kevin Melo	kevin@hashtag.com	1	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1	68004
4	Mirella Viana	mirella@hashtag.com	1	28421

## Módulo 17 – Unindo bases com JOIN

No exemplo abaixo, trocamos a ordem das tabelas plataforma e alunos. Lembre-se que há uma matrícula presente em plataforma que não está presente em alunos. Podemos indicar em nossa query que queremos que apareça uma matrícula, independente de qual a tabela de origem. O COALESCE no SQL é uma função que ajuda a lidar com valores NULL em consultas. Ele recebe uma lista de argumentos e retorna o primeiro valor não NULL encontrado na lista.

```
1 # Trazendo as informações da tabela plataforma e fazendo o join com a tabela alunos
2 ▼ plataforma_aluno = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
3                                     coalesce(a.cod_matricula,p.cod_matricula) as cod_matricula \
4                                     FROM plataforma p \
5                                     LEFT JOIN alunos a \
6                                     ON a.cod_matricula = p.cod_matricula')
```

(20, 4)

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0	63546
2	Kevin Melo	kevin@hashtag.com	1	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1	68004
4	Mirella Viana	mirella@hashtag.com	1	28421

## Módulo 17 – Unindo bases com JOIN

Verifique que não há matrículas nulas em nosso retorno:

1	display(plataforma_aluno)	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0	38273	
1	Bárbara da Cunha	barbara@hashtag.com	0	63546	
2	Kevin Melo	kevin@hashtag.com	1	80515	
3	Pedro Henrique da Costa	pedro@hashtag.com	1	68004	
4	Mirella Viana	mirella@hashtag.com	1	28421	
5	Lívia Jesus	livia@hashtag.com	0	22284	
6	Lara Lopes	lara@hashtag.com	1	38362	
7	Lucca Cardoso	lucca@hashtag.com	1	45109	
8	Isabelly Souza	isabelly@hashtag.com	1	31859	
9	Cauã Porto	cauã@hashtag.com	0	42674	
10	Maria Ferreira	maria@hashtag.com	0	86563	
11	Júlia Pinto	julia@hashtag.com	1	47086	
12	Antônio Azevedo	lovedogs@hashtag.com	0	29022	
13	Laura Melo	laura@hashtag.com	1	50966	
14	Gabriela Costela	gabriela@hashtag.com	1	21262	
15	Eloah Aragão	eloah@hashtag.com	1	65749	
16	Francisco Pires	xlc0@hashtag.com	0	59518	
17	Bárbara Freitas	barbara@hashtag.com	0	19442	
18	Melissa Ribeiro	melissa@hashtag.com	1	38438	
19	Lávinia Carvalho	lavinia@hashtag.com	1	89428	

# Módulo 17 – O UNION e o FULL JOIN no SQL

Vamos relembrar nossa base *alunos*. Observe que há 21 registros:

# Visualizando a tabela alunos				
(21, 4)				
	<b>id_aluno</b>	<b>nome_aluno</b>	<b>cod_matricula</b>	<b>e-mail</b>
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	2	Bárbara da Cunha	63546	barbara@hashtag.com
2	3	Kevin Melo	80515	kevin@hashtag.com
3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
4	5	Mirella Viana	28421	mirella@hashtag.com

1	display(alunos)			
	<b>id_aluno</b>	<b>nome_aluno</b>	<b>cod_matricula</b>	<b>e-mail</b>
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	2	Bárbara da Cunha	63546	barbara@hashtag.com
2	3	Kevin Melo	80515	kevin@hashtag.com
3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
4	5	Mirella Viana	28421	mirella@hashtag.com
5	6	Lívia Jesus	22284	livia@hashtag.com
6	7	Lara Lopes	38362	lara@hashtag.com
7	8	Lucca Cardoso	45109	lucca@hashtag.com
8	9	Isabelly Souza	31859	isabelly@hashtag.com
9	10	Cauã Porto	42674	cauã@hashtag.com
10	11	Maria Ferreira	86563	maria@hashtag.com
11	12	Júlia Pinto	47086	julia@hashtag.com
12	13	Antônio Azevedo	29022	lovedogs@hashtag.com
13	14	Laura Melo	50966	laura@hashtag.com
14	15	Gabriela Costela	21262	gabriela@hashtag.com
15	16	Eloah Aragão	65749	eloah@hashtag.com
16	17	Francisco Pires	59518	xlc0@hashtag.com
17	18	Bárbara Freitas	19442	barbara@hashtag.com
18	19	Melissa Ribeiro	38438	melissa@hashtag.com
19	20	Lavânia Carvalho	89428	lavinia@hashtag.com
20	21	Jorge Henrique Gomes	89791	jorge@hashtag.com



## Módulo 17 – O UNION e o FULL JOIN no SQL

Considere que há uma outra base com o registro de novos alunos e que você deseja adicionar estes registros à base *alunos*, atualizando-a. Observe que o aluno Jorge Henrique Gomes, presente nesta tabela de alunos novos, já está presente na tabela *alunos* mostrada na página anterior:

1	# Visualizando a tabela <i>alunos_novos</i>			
2	alunos_novos = executa_sql("SELECT * FROM <i>alunos_novos</i> ")			
(4, 4)				
	<b>id_aluno</b>	<b>nome_aluno</b>	<b>cod_matricula</b>	<b>e-mail</b>
0	1	Pedro Carlos	115640	pedro@hashtag.com
1	2	Beatriz Rezende	100310	bia@hashtag.com
2	3	Sheila Costa	112366	sheila@hashtag.com
3	21	Jorge Henrique Gomes	89791	jorge@hashtag.com

# Módulo 17 – O UNION e o FULL JOIN no SQL

O UNION é um operador usado para combinar os resultados de duas ou mais consultas SELECT em um único conjunto de resultados. Ele elimina linhas duplicadas, retornando apenas registros únicos. Antes do UNION, tínhamos 21 registros. Agora temos 24, pois o registro do aluno José Henrique Gomes já existia e não foi duplicado.

```
1 # Unindo as duas tabelas de alunos utilizando o UNION
2 alunos = executa_sql("SELECT * FROM alunos \
3                         UNION \
4                         SELECT * FROM alunos_novos")
```

(24, 4)				
	<b>id_aluno</b>	<b>nome_aluno</b>	<b>cod_matricula</b>	<b>e-mail</b>
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	1	Pedro Carlos	115640	pedro@hashtag.com
2	2	Beatriz Rezende	100310	bia@hashtag.com
3	2	Bárbara da Cunha	63546	barbara@hashtag.com
4	3	Kevin Melo	80515	kevin@hashtag.com

1	display(alunos)			
	<b>id_aluno</b>	<b>nome_aluno</b>	<b>cod_matricula</b>	<b>e-mail</b>
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	1	Pedro Carlos	115640	pedro@hashtag.com
2	2	Beatriz Rezende	100310	bia@hashtag.com
3	2	Bárbara da Cunha	63546	barbara@hashtag.com
4	3	Kevin Melo	80515	kevin@hashtag.com
5	3	Sheila Costa	112366	sheila@hashtag.com
6	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
7	5	Mirella Viana	28421	mirella@hashtag.com
8	6	Lívia Jesus	22284	livia@hashtag.com
9	7	Lara Lopes	38362	lara@hashtag.com
10	8	Lucca Cardoso	45109	lucca@hashtag.com
11	9	Isabelly Souza	31859	isabelly@hashtag.com
12	10	Cauã Porto	42674	cauã@hashtag.com
13	11	Maria Ferreira	86563	maria@hashtag.com
14	12	Júlia Pinto	47086	julia@hashtag.com
15	13	Antônio Azevedo	29022	lovedogs@hashtag.com
16	14	Laura Melo	50966	laura@hashtag.com
17	15	Gabriela Costela	21262	gabriela@hashtag.com
18	16	Eloah Aragão	65749	eloah@hashtag.com
19	17	Francisco Pires	59518	xlc0@hashtag.com
20	18	Bárbara Freitas	19442	barbara@hashtag.com
21	19	Melissa Ribeiro	38438	melissa@hashtag.com
22	20	Lavinia Carvalho	89428	lavinia@hashtag.com
23	21	Jorge Henrique Gomes	89791	jorge@hashtag.com

## Módulo 17 – O UNION e o FULL JOIN no SQL

Se, por qualquer motivo, você quiser atualizar com todos os registros, mesmo duplicados, há o UNION ALL. Observe que o registro do aluno Jorge Henrique Gomes está duplicado agora:

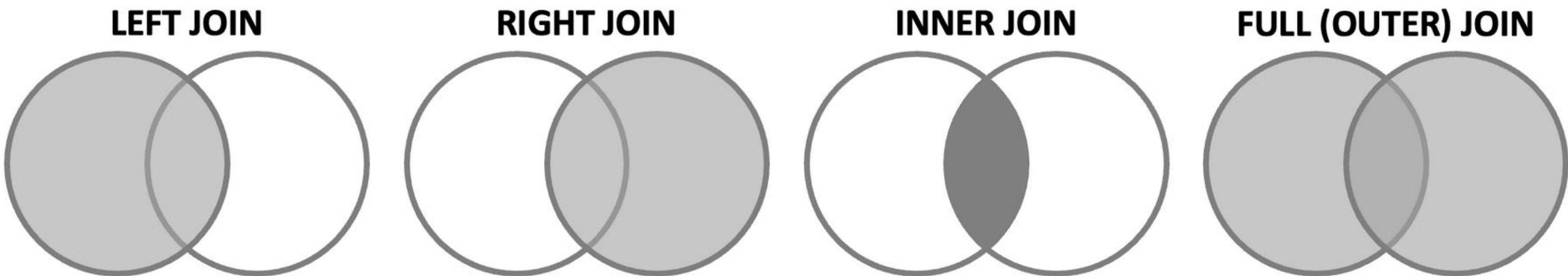
```
1 # Unindo as duas tabelas de alunos utilizando o UNION ALL
2 alunos = executa_sql("SELECT * FROM alunos \
3                         UNION ALL \
4                         SELECT * FROM alunos_novos")
```

	(25, 4)	id_aluno	nome_aluno	cod_matricula	e-mail
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	
1	2	Bárbara da Cunha	63546	barbara@hashtag.com	
2	3	Kevin Melo	80515	kevin@hashtag.com	
3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	
4	5	Mirella Viana	28421	mirella@hashtag.com	

1	display(alunos)			
	id_aluno	nome_aluno	cod_matricula	e-mail
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	2	Bárbara da Cunha	63546	barbara@hashtag.com
2	3	Kevin Melo	80515	kevin@hashtag.com
3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
4	5	Mirella Viana	28421	mirella@hashtag.com
5	6	Lívia Jesus	22284	livia@hashtag.com
6	7	Lara Lopes	38362	lara@hashtag.com
7	8	Lucca Cardoso	45109	lucca@hashtag.com
8	9	Isabelly Souza	31859	isabelly@hashtag.com
9	10	Cauã Porto	42674	cauã@hashtag.com
10	11	Maria Ferreira	86563	maria@hashtag.com
11	12	Júlia Pinto	47086	julia@hashtag.com
12	13	Antônio Azevedo	29022	lovedogs@hashtag.com
13	14	Laura Melo	50966	laura@hashtag.com
14	15	Gabriela Costela	21262	gabriela@hashtag.com
15	16	Eloah Aragão	65749	eloah@hashtag.com
16	17	Francisco Pires	59518	xlc0@hashtag.com
17	18	Bárbara Freitas	19442	barbara@hashtag.com
18	19	Melissa Ribeiro	38438	melissa@hashtag.com
19	20	Lavânia Carvalho	89428	lavinia@hashtag.com
20	21	Jorge Henrique Gomes	89791	jorge@hashtag.com
21	1	Pedro Carlos	115640	pedro@hashtag.com
22	2	Beatriz Rezende	100310	bia@hashtag.com
23	3	Sheila Costa	112366	sheila@hashtag.com
24	21	Jorge Henrique Gomes	89791	jorge@hashtag.com

## Módulo 17 – O UNION e o FULL JOIN no SQL

O sqlite3 não possui nativamente um comando que possibilite fazer o OUTER JOIN, mostrado no diagrama abaixo. Mas, com o que vimos de UNION, podemos construir uma query que gera exatamente o mesmo resultado que seria obtido por tal tipo de junção.



## Módulo 17 – O UNION e o FULL JOIN no SQL

Já vimos como realizar um LEFT JOIN entre as tabelas *alunos* e *plataforma*. Observe o valor nulo na coluna *acesso\_plataforma* para o aluno Jorge Henrique Gomes.

```
1 # Vamos selecionar todos os valores da tabela alunos fazendo o join com a tabela plataforma
2 aluno_plataforma = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
3                                     a.cod_matricula \
4                                     FROM alunos a \
5                                     LEFT JOIN plataforma p \
6                                     ON a.cod_matricula = p.cod_matricula')
```

(21, 4)

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0.0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0.0	63546
2	Kevin Melo	kevin@hashtag.com	1.0	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1.0	68004
4	Mirella Viana	mirella@hashtag.com	1.0	28421

1	display(aluno_plataforma)	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0.0	38273	
1	Bárbara da Cunha	barbara@hashtag.com	0.0	63546	
2	Kevin Melo	kevin@hashtag.com	1.0	80515	
3	Pedro Henrique da Costa	pedro@hashtag.com	1.0	68004	
4	Mirella Viana	mirella@hashtag.com	1.0	28421	
5	Lívia Jesus	livia@hashtag.com	0.0	22284	
6	Lara Lopes	lara@hashtag.com	1.0	38362	
7	Lucca Cardoso	lucca@hashtag.com	1.0	45109	
8	Isabelly Souza	isabelly@hashtag.com	1.0	31859	
9	Cauã Porto	cauã@hashtag.com	0.0	42674	
10	Maria Ferreira	maria@hashtag.com	0.0	86563	
11	Júlia Pinto	julia@hashtag.com	1.0	47086	
12	Antônio Azevedo	lovedogs@hashtag.com	0.0	29022	
13	Laura Melo	laura@hashtag.com	1.0	50966	
14	Gabriela Costela	gabriela@hashtag.com	1.0	21262	
15	Eloah Aragão	eloah@hashtag.com	1.0	65749	
16	Francisco Pires	xlc0@hashtag.com	0.0	59518	
17	Bárbara Freitas	barbara@hashtag.com	0.0	19442	
18	Melissa Ribeiro	melissa@hashtag.com	1.0	38438	
19	Lavinia Carvalho	lavinia@hashtag.com	1.0	89428	
20	Jorge Henrique Gomes	jorge@hashtag.com	NaN	89791	

## Módulo 17 – O UNION e o FULL JOIN no SQL

Assim como já vimos o inverso, um LEFT JOIN entre as tabelas *plataforma* e *alunos*. Observe os valores nulos na linha da matrícula 91047.

```
1 # Agora vamos fazer o join da tabela plataforma com a tabela aluno
2 plataforma_aluno = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
3 coalesce(a.cod_matricula,p.cod_matricula) as cod_matricula \
4 FROM plataforma p \
5 LEFT JOIN alunos a \
6 ON a.cod_matricula = p.cod_matricula')
```

(21, 4)

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0	63546
2	Kevin Melo	kevin@hashtag.com	1	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1	68004
4	Mirella Viana	mirella@hashtag.com	1	28421

1	display(plataforma_aluno)	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com		0	38273
1	Bárbara da Cunha	barbara@hashtag.com		0	63546
2	Kevin Melo	kevin@hashtag.com		1	80515
3	Pedro Henrique da Costa	pedro@hashtag.com		1	68004
4	Mirella Viana	mirella@hashtag.com		1	28421
5	Lívia Jesus	livia@hashtag.com		0	22284
6	Lara Lopes	lara@hashtag.com		1	38362
7	Lucca Cardoso	lucca@hashtag.com		1	45109
8	Isabelly Souza	isabelly@hashtag.com		1	31859
9	Cauã Porto	cauã@hashtag.com		0	42674
10	Maria Ferreira	maria@hashtag.com		0	86563
11	Júlia Pinto	julia@hashtag.com		1	47086
12	Antônio Azevedo	lovedogs@hashtag.com		0	29022
13	Laura Melo	laura@hashtag.com		1	50966
14	Gabriela Costela	gabriela@hashtag.com		1	21262
15	Eloah Aragão	eloah@hashtag.com		1	65749
16	Francisco Pires	xlc0@hashtag.com		0	59518
17	Bárbara Freitas	barbara@hashtag.com		0	19442
18	Melissa Ribeiro	melissa@hashtag.com		1	38438
19	Lavinia Carvalho	lavinia@hashtag.com		1	89428
20	None	None		0	91047

## Módulo 17 – O UNION e o FULL JOIN no SQL

O UNION dos JOINS anteriores equivale a um OUTER JOIN das duas tabelas. Observe que duas as linhas destacadas nas páginas anteriores estão presentes no resultado:

```
1 # Para fazer o full join, podemos exatamente pegar esses dois left joins e fazer um UNION
2 full_join = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
3                         coalesce(a.cod_matricula,p.cod_matricula) as cod_matricula \
4                         FROM plataforma p \
5                         LEFT JOIN alunos a \
6                         ON a.cod_matricula = p.cod_matricula \
7                         UNION \
8                         SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
9                         a.cod_matricula \
10                        FROM alunos a \
11                        LEFT JOIN plataforma p \
12                        ON a.cod_matricula = p.cod_matricula')
```

(22, 4)

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	None	None	0.0	91047
1	Antônio Azevedo	lovedogs@hashtag.com	0.0	29022
2	Bárbara Freitas	barbara@hashtag.com	0.0	19442
3	Bárbara da Cunha	barbara@hashtag.com	0.0	63546
4	Cauã Porto	cauã@hashtag.com	0.0	42674

1	# Visualizando o full join			
2	display(full_join)			
	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	None	None	0.0	91047
1	Antônio Azevedo	lovedogs@hashtag.com	0.0	29022
2	Bárbara Freitas	barbara@hashtag.com	0.0	19442
3	Bárbara da Cunha	barbara@hashtag.com	0.0	63546
4	Cauã Porto	cauã@hashtag.com	0.0	42674
5	Eloah Aragão	eloah@hashtag.com	1.0	65749
6	Francisco Pires	xlc0@hashtag.com	0.0	59518
7	Gabriela Costela	gabriela@hashtag.com	1.0	21262
8	Isabelly Souza	isabelly@hashtag.com	1.0	31859
9	Jorge Henrique Gomes	jorge@hashtag.com	NaN	89791
10	Júlia Pinto	julia@hashtag.com	1.0	47086
11	Kevin Melo	kevin@hashtag.com	1.0	80515
12	Lara Lopes	lara@hashtag.com	1.0	38362
13	Laura Melo	laura@hashtag.com	1.0	50966
14	Lavínia Carvalho	lavinia@hashtag.com	1.0	89428
15	Lucca Cardoso	lucca@hashtag.com	1.0	45109
16	Lívia Jesus	livia@hashtag.com	0.0	22284
17	Maria Eduarda da Rocha	maria@hashtag.com	0.0	38273
18	Maria Ferreira	maria@hashtag.com	0.0	86563
19	Melissa Ribeiro	melissa@hashtag.com	1.0	38438
20	Mirella Viana	mirella@hashtag.com	1.0	28421
21	Pedro Henrique da Costa	pedro@hashtag.com	1.0	68004

## MÓDULO 18

# Técnicas de storytelling com dados: Utilizando o SQL com dados reais de venda

## Módulo 18 – Apresentando a base de dados que vamos utilizar nesse módulo

Neste módulo utilizaremos uma base de dados da loja Olist, disponibilizada no Kaggle.

A base contém informações de aproximadamente 100 mil pedidos feitos entre 2016 e 2018 em diversos *marketplaces* no Brasil.

Com as informações da base, um mesmo pedido pode ser analisado por diferentes pontos de vista:

- Status do pedido
- Preço
- Forma de pagamento
- Status da entrega
- Comentários dos clientes

Os dados são reais, tendo sido apenas anonimizados. Eventuais referências a empresas nos comentários foram substituídas por nomes das grandes casas do seriado *Game of Thrones*.

Durante o módulo, mostraremos a base com Pandas e, então, mostraremos como explorar a base com SQL e apresentar os resultados de uma forma atraente com métodos de storytelling.

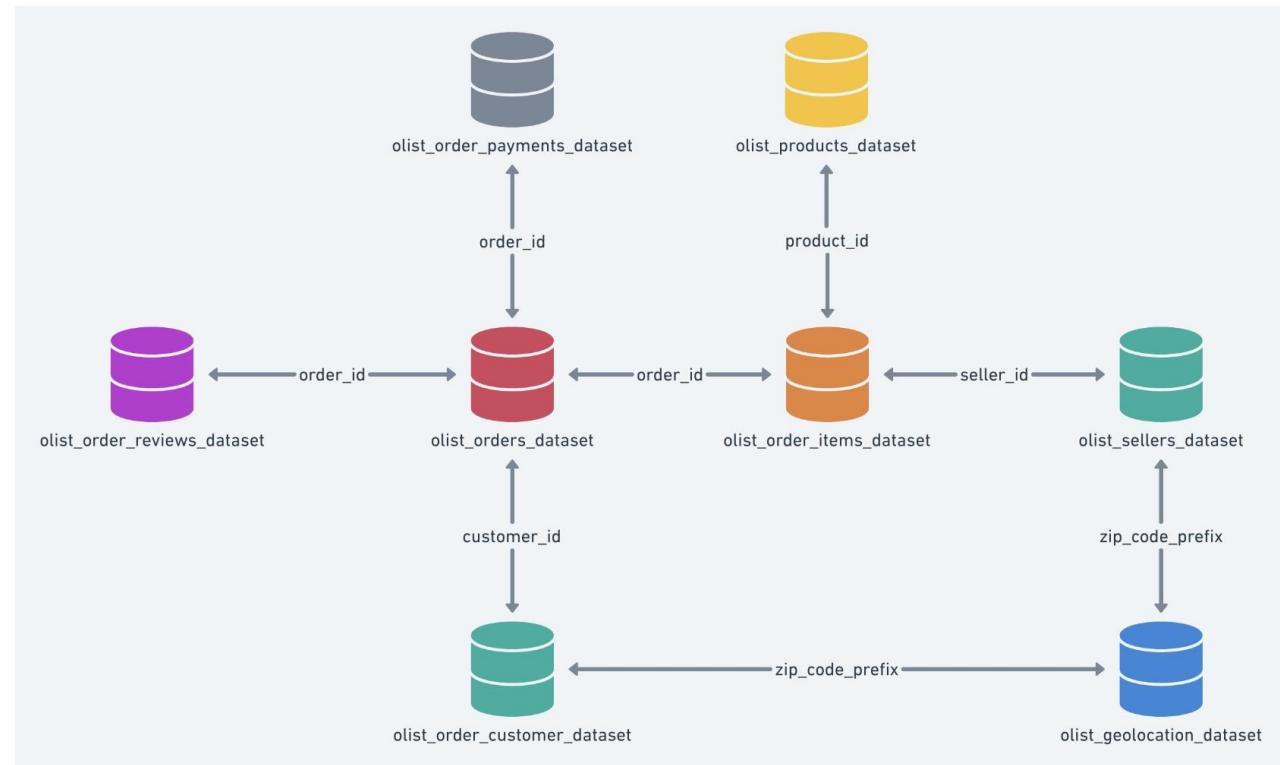
## Módulo 18 – Apresentando a base de dados que vamos utilizar nesse módulo

A base é subdividida em datasets específicos para cada etapa do processamento do pedido. O esquema ao lado mostra os datasets e os atributos que relacionam os datasets entre si.

Sempre que tiver alguma dúvida sobre como estes datasets se relacionam, ou porque um determinado atributo foi usado em um *merge* ou *join* no decorrer do módulo, consulte este esquema.

Algumas observações importantes:

- Um pedido pode ter vários itens
- Cada item pode ser vendido por um vendedor distinto



## Módulo 18 – Buscando os arquivos de nossa base

Como visto, a base foi dividida em datasets, disponibilizados em arquivos de formato csv. Os mesmos podem ser baixados da nossa plataforma de cursos. É recomendável que fiquem em uma mesma pasta, separada dos arquivos de código. Vamos chamar esta pasta de *bases*.

Com auxílio da biblioteca `os`, já disponível no Python, podemos listar e armazenar o conteúdo de uma determinada pasta em uma variável:

```
1 # Importando a biblioteca
2 import os

1 # Verificando a pasta que estamos atualmente
2 os.getcwd()

1 # Guardando essa pasta em uma variável
2 pasta = os.getcwd()

1 # Verificando os arquivos nessa pasta
2 os.listdir(pasta+'/bases')

['olist_customers_dataset.csv',
 'product_category_name_translation.csv',
 'olist_order_reviews_dataset.csv',
 'olist_geolocation_dataset.csv',
 'olist_order_items_dataset.csv',
 'olist_orders_dataset.csv',
 'olist_sellers_dataset.csv',
 'olist_products_dataset.csv',
 'olist_order_payments_dataset.csv']

1 # Guardando os arquivos em uma variável
2 arquivos = os.listdir(pasta+'/bases')

1 # Visualizando os arquivos
2 arquivos

['olist_customers_dataset.csv',
 'product_category_name_translation.csv',
 'olist_order_reviews_dataset.csv',
 'olist_geolocation_dataset.csv',
 'olist_order_items_dataset.csv',
 'olist_orders_dataset.csv',
 'olist_sellers_dataset.csv',
 'olist_products_dataset.csv',
 'olist_order_payments_dataset.csv']
```

## Módulo 18 – Passando a base para o Pandas

Desta forma, podemos facilmente criar dataframes do Pandas a partir destes arquivos. Por exemplo, veja a criação de um dataframe para as ordens e de um para os itens de cada ordem:

```
1 # Importando o pandas  
2 import pandas as pd
```

```
1 # Visualizando a olist_orders_dataset  
2 orders = pd.read_csv(pasta+'/bases/olist_orders_dataset.csv')  
3 order_items = pd.read_csv(pasta+'/bases/olist_order_items_dataset.csv')
```

```
1 # Visualizando as 5 primeiras linhas  
2 orders.head()
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15	2017-10-04 19:55:00	2017-10-10 21:25:13	2017-10-18 00:00:00
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	2018-07-24 20:41:37	2018-07-26 03:24:27	2018-07-26 14:31:00	2018-08-07 15:27:45	2018-08-13 00:00:00
2	47770eb9100c2d0c44946d9cf07ec65d	41ce2a54c0b03bf3443c3d931a367089	delivered	2018-08-08 08:38:49	2018-08-08 08:55:23	2018-08-08 13:50:00	2018-08-17 18:06:29	2018-09-04 00:00:00
3	949d5b44dbf5de918fe9c16f97b45f8a	f88197465ea7920adcdbec7375364d82	delivered	2017-11-18 19:28:06	2017-11-18 19:45:59	2017-11-22 13:39:59	2017-12-02 00:28:42	2017-12-15 00:00:00
4	ad21c59c0840e6cb83a9ceb5573f8159	8ab97904e6daeaa8866dbdbc4fb7aad2c	delivered	2018-02-13 21:18:39	2018-02-13 22:20:29	2018-02-14 19:46:34	2018-02-16 18:17:02	2018-02-26 00:00:00

## Módulo 18 – Passando a base para o Pandas

Como temos todas as bases em nossa variável **arquivos**, podemos fazer um loop **for** nesta variável, criando dinamicamente nossos dataframes.

No código, interagimos com o dicionário de variáveis do Python, o **globals**. Em cada passagem do loop, adicionamos a este dicionário a variável que corresponde ao dataframe que criamos para uma base.

E, também a cada passagem, exibimos o nome do arquivo csv original e o nome do dataframe gerado a partir do arquivo, além de mostrar que as 5 primeiras linhas de ambos são iguais, com o método **head**. Ao lado, a base de clientes **customers**.

```
1 # Percorrendo toda a lista de arquivos
2 for i in arquivos:
3     print(i)
4     nome_base = i.replace('olist_','').replace('_dataset','').replace('.csv','')
5     print(nome_base)
6     base = pd.read_csv(pasta+'/bases/'+i)
7     display(base.head())
8     globals()[nome_base] = base
9     display(globals()[nome_base].head())
```

olist\_customers\_dataset.csv  
customers

	customer_id	customer_unique_id	customer_zip_code_prefix	customer_city	customer_state
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	14409	franca	SP
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	9790	sao bernardo do campo	SP
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	1151	sao paulo	SP
3	b2b6027bc5c5109e529d4dc6358b12c3	259dac757896d24d7702b9acbbff3f3c	8775	mogi das cruzes	SP
4	4f2d8ab171c80ec8364f7c12e35b23ad	345ecd01c38d18a9036ed96c73b8d066	13056	campinas	SP

	customer_id	customer_unique_id	customer_zip_code_prefix	customer_city	customer_state
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	14409	franca	SP
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	9790	sao bernardo do campo	SP
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	1151	sao paulo	SP
3	b2b6027bc5c5109e529d4dc6358b12c3	259dac757896d24d7702b9acbbff3f3c	8775	mogi das cruzes	SP
4	4f2d8ab171c80ec8364f7c12e35b23ad	345ecd01c38d18a9036ed96c73b8d066	13056	campinas	SP

## Módulo 18 – Passando a base para o Pandas

Mais alguns dataframes: o de tradução dos nomes das categorias, o de geolocalização, e o de comentários dos usuários.

product\_category\_name\_translation.csv

product\_category\_name\_translation

product\_category\_name product\_category\_name\_english

0	beleza_saude	health_beauty
1	informatica_acessorios	computers_accessories
2	automotivo	auto
3	cama_mesa_banho	bed_bath_table
4	moveis_decoracao	furniture_decor

olist\_order\_reviews\_dataset.csv

order\_reviews

	review_id	order_id	review_score	review_comment_title	review_comment_message	review_creation_date	review_answer_timestamp
0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4	NaN	NaN	2018-01-18 00:00:00	2018-01-18 21:46:59
1	80e641a11e56f04c1ad469d5645fdfde	a548910a1c6147796b98fdf73dbeba33	5	NaN	NaN	2018-03-10 00:00:00	2018-03-11 03:05:13
2	228ce5500dc1d8e020d8d1322874b6f0	f9e4b658b201a9f2ecdecbb34bed034b	5	NaN	NaN	2018-02-17 00:00:00	2018-02-18 14:36:24
3	e64fb393e7b32834bb789ff8bb30750e	658677c97b385a9be170737859d3511b	5	NaN	Recebi bem antes do prazo estipulado.	2017-04-21 00:00:00	2017-04-21 22:02:06
4	f7c4243c7fe1938f181bec41a392bdeb	8e6bf81e283fa7e4f11123a3fb894f1	5	NaN	Parabéns lojas lannister adorei comprar pela I...	2018-03-01 00:00:00	2018-03-02 10:26:53

## Módulo 18 – Passando a base para o Pandas

Seguindo nos dataframes: o de items de cada ordem, e o de ordens.

olist\_order\_items\_dataset.csv  
order\_items

	order_id	order_item_id	product_id	seller_id	shipping_limit_date	price	freight_value
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	48436dade18ac8b2bce089ec2a041202	2017-09-19 09:45:35	58.90	13.29
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	dd7ddc04e1b6c2c614352b383efe2d36	2017-05-03 11:05:13	239.90	19.93
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fd	5b51032eddd242adc84c38acab88f23d	2018-01-18 14:48:30	199.00	17.87
3	00024acbcdf0a6daa1e931b038114c75	1	7634da152a4610f1595efa32f14722fc	9d7a1d34a5052409006425275ba1c2b4	2018-08-15 10:10:18	12.99	12.79
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	ac6c3623068f30de03045865e4e10089	df560393f3a51e74553ab94004ba5c87	2017-02-13 13:57:51	199.90	18.14

olist\_orders\_dataset.csv  
orders

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15	2017-10-04 19:55:00	2017-10-10 21:25:13	2017-10-18 00:00:00
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	2018-07-24 20:41:37	2018-07-26 03:24:27	2018-07-26 14:31:00	2018-08-07 15:27:45	2018-08-13 00:00:00
2	47770eb9100c2d0c44946d9cf07ec65d	41ce2a54c0b03bf3443c3d931a367089	delivered	2018-08-08 08:38:49	2018-08-08 08:55:23	2018-08-08 13:50:00	2018-08-17 18:06:29	2018-09-04 00:00:00
3	949d5b44dbf5de918fe9c16f97b45f8a	f88197465ea7920adcdbec7375364d82	delivered	2017-11-18 19:28:06	2017-11-18 19:45:59	2017-11-22 13:39:59	2017-12-02 00:28:42	2017-12-15 00:00:00
4	ad21c59c0840e6cb83a9ceb5573f8159	8ab97904e6daeaa8866dbdbc4fb7aad2c	delivered	2018-02-13 21:18:39	2018-02-13 22:20:29	2018-02-14 19:46:34	2018-02-16 18:17:02	2018-02-26 00:00:00

# Módulo 18 – Passando a base para o Pandas

Por fim, os dataframes: de vendedores, de pagamentos, e de produtos.

olist\_sellers\_dataset.csv  
sellers

	seller_id	seller_zip_code_prefix	seller_city	seller_state
0	3442f8959a84dea7ee197c632cb2df15	13023	campinas	SP
1	d1b65fc7debc3361ea86b5f14c68d2e2	13844	mogi guacu	SP
2	ce3ad9de960102d0677a81f5d0bb7b2d	20031	rio de janeiro	RJ
3	c0f3eea2e14555b6faeea3dd58c1b1c3	4195	sao paulo	SP
4	51a04a8a6bdcb23deccc82b0b80742cf	12914	braganca paulista	SP

olist\_order\_payments\_dataset.csv  
order\_payments

	order_id	payment_sequential	payment_type	payment_installments	payment_value
0	b81ef226f3fe1789b1e8b2acac839d17	1	credit_card	8	99.33
1	a9810da82917af2d9aefd1278f1dcfa0	1	credit_card	1	24.39
2	25e8ea4e93396b6fa0d3dd708e76c1bd	1	credit_card	1	65.71
3	ba78997921bbcddc1373bb41e913ab953	1	credit_card	8	107.78
4	42fdf880ba16b47b59251dd489d4441a	1	credit_card	2	128.45

olist\_products\_dataset.csv  
products

	product_id	product_category_name	product_name_lenght	product_description_lenght	product_photos_qty	product_weight_g	product_length_cm	product_height_cm	product_width_cm
0	1e9e8ef04dbcff4541ed26657ea517e5	perfumaria	40.0	287.0	1.0	225.0	16.0	10.0	14.0
1	3aa071139cb16b67ca9e5dea641aaa2f	artes	44.0	276.0	1.0	1000.0	30.0	18.0	20.0
2	96bd76ec8810374ed1b65e291975717f	esporte_lazer	46.0	250.0	1.0	154.0	18.0	9.0	15.0
3	cef67bcfe19066a932b7673e239eb23d	bebes	27.0	261.0	1.0	371.0	26.0	4.0	26.0
4	9dc1a7de274444849c219cff195d0b71	utilidades_domesticas	37.0	402.0	4.0	625.0	20.0	17.0	13.0

## Módulo 18 – Analisando ordens, itens e pagamentos

Dedique um certo tempo analisando cada base, reconhecendo os dados que vêm em cada uma a partir de suas colunas. E lembre-se das observações que fizemos no início do módulo:

- Um pedido pode ter vários itens
- Cada item pode ser vendido por um vendedor distinto

Vamos começar nosso estudo pela base de ordens:

1	# Vamos começar olhando a tabela de ordens							
2	orders.head()							
order_id customer_id order_status order_purchase_timestamp order_approved_at order_delivered_carrier_date order_delivered_customer_date order_estimated_delivery_date								
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15	2017-10-04 19:55:00	2017-10-10 21:25:13	2017-10-18 00:00:00
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	2018-07-24 20:41:37	2018-07-26 03:24:27	2018-07-26 14:31:00	2018-08-07 15:27:45	2018-08-13 00:00:00
2	47770eb9100c2d0c44946d9cf07ec65d	41ce2a54c0b03bf3443c3d931a367089	delivered	2018-08-08 08:38:49	2018-08-08 08:55:23	2018-08-08 13:50:00	2018-08-17 18:06:29	2018-09-04 00:00:00
3	949d5b44dbf5de918fe9c16f97b45f8a	f88197465ea7920adcdbec7375364d82	delivered	2017-11-18 19:28:06	2017-11-18 19:45:59	2017-11-22 13:39:59	2017-12-02 00:28:42	2017-12-15 00:00:00
4	ad21c59c0840e6cb83a9ceb5573f8159	8ab97904e6daeaa8866dbdbc4fb7aad2c	delivered	2018-02-13 21:18:39	2018-02-13 22:20:29	2018-02-14 19:46:34	2018-02-16 18:17:02	2018-02-26 00:00:00

## Módulo 18 – Analisando ordens, itens e pagamentos

Como já dito, cada ordem pode ter diversos itens. Então, é natural que busquemos correlacionar as duas bases, de ordens e de itens.

Ao lado, uma visão das 50 primeiras entradas da base de itens:

1	# E então analisar a tabela de ordem x itens							
2	order_items.head(50)							
		order_id	order_item_id	product_id	seller_id	shipping_limit_date	price	freight_value
0	00010242fe8c5a6d1ba2d792cb16214	1	4244733e06e7ebc4970a6e2683c13e61	48436dade18ac8b2bc0e99ec2a041202	2017-09-19 09:45:35	58.90	13.29	
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	dd7ddc04e1b6c2c614352b383efe2d36	2017-05-03 11:05:13	239.90	19.93	
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fd	5b51032eddd242adc84c38acab88f23d	2018-01-18 14:48:30	199.00	17.87	
3	00024acbcd0fa6daaa1e931b038114c75	1	7634da152a4610f1595efa32f14722fc	9d7a1d3a5052409006425275ba1c2b4	2018-08-15 10:10:18	12.99	12.79	
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	ac6c36230368f30de03045865e4e10089	df560393f3a51e74553ab94004b5c87	2017-02-13 13:57:51	199.90	18.14	
5	00048cc3ae7776c5dbb72d2a0634bc1ea	1	e9f2defde45ab8450fd70c526ef70f	6426d21aca402a131fc0a5d0960a3c90	2017-05-23 03:55:27	21.90	12.69	
6	00054e8431b9d7675808bbcb819fb4a32	1	8d4f2bb7e93e6710a28f34fa83ee7d28	7040e82f899a04d1b434bf795a43b4617	2017-12-14 12:10:31	19.90	11.85	
7	000576fe39319847ccb9d288c5617fa6	1	557d850972a7d6f792fd18ae1400d9b6	5996cdab893a4652a15592fb58ab8db	2018-07-10 12:30:45	810.00	70.75	
8	0005a1a1728c9d785b8e2b08b904576c	1	310ae3c140ff94b03219ad0ad3c778f	a416b6a846a111724393025641d4edd5e	2018-03-26 18:31:29	145.95	11.65	
9	0005f0442cb953cd1d21e1fb923495	1	4535b0e1091c278dfd193e5a1d63b39f	ba143b0f0110f0dc71ad71b4466ce92	2018-07-06 14:10:56	53.99	11.40	
10	00061f2a7cc09da83e415a52dc8a4af1	1	d63c1011f49d98b976c352955b1c4bea	cc419e0650a3c5ba77189a1882b7556a	2018-03-29 22:28:09	59.99	8.88	
11	00063b381e2406b52ad429470734bed5	1	f177554ea93259a5b282f24e33f65ab5	8602a61d680a10a82cceeeda0d99ea3d	2018-07-31 17:30:39	45.00	12.98	
12	0006ec9db01a64e59a68b2c340bf65a7	1	99a4788cb24856965c36a24e339b6058	4a3ca9315b744ce9f8e9374361493884	2018-07-26 17:24:20	74.00	23.32	
13	0008288aa423d2a3f0fcb17cd7d8719	1	368c6c730842d78016ad823897a372d8	1f50f920176fa81dab994f9023523100	2018-02-21 02:55:52	49.90	13.37	
14	0008288aa423d2a3f0fcb17cd7d8719	2	368c6c730842d78016ad823897a372d8	1f50f920176fa81dab994f9023523100	2018-02-21 02:55:52	49.90	13.37	
15	0009792311464db532ff765bf7b182ae	1	8cab8abac59158715e0d70a3c6807415	530ec6109d11eaaf87999465c6afe001	2018-08-17 12:15:10	99.90	27.65	
16	0009c9a17f916a706d71784483a5d643	1	3f27ac8e699df3d300ec4a5d8c5f0b2	fcb5ace8bcc9f27570d0f01a27d269	2018-05-02 09:31:53	639.00	11.34	
17	000aed2e25dbad2f9db70584c5a2ded	1	4fa33915031a8cde03dd0d3e8fb27f01	fe2032dab1a61af8794248c8196565c9	2018-05-16 20:57:03	144.00	8.77	
18	000c3e6612759851cc3ccbb4b83257986	1	b50c950aba0dcead2c48032a690ce817	218d46b86c1881d022bce9c68a7d4b15	2017-08-21 03:33:13	99.00	13.71	
19	000e562887b1f2006d75e0be9558292e	1	5ed9eaf534f693b51d0b6c5e4d5c2e9	8bacat7e12637ed9cffa18c7875207478	2018-02-28 12:08:37	25.00	16.11	
20	000e63d38ae8c00bbcb5a30573b99628	1	553e0e7590d3116a072507a3635d2877	1c129092bf23f28a5930387c980c0dfc	2018-03-29 20:07:49	47.90	8.88	
21	000e906b789b55f64edcb1f8403090d	1	5d79905de06d8897872c551bf0d9358	ea8482cd71df3c1969d7b9473ff13abc	2017-11-27 19:09:02	21.99	11.85	
22	000f25f4d72195062c040b12dce9a18a	1	1c05e0964302b6cf68ca0d15f326c6ba	7c67e1448b00f6e969d365cea6b010ab	2018-03-21 11:10:11	119.99	44.40	
23	001021efaa8636c29475e7734483457d	1	5d7c23067ed3fc8c6e699b9373d5890b	6560211a19b47992c3666c44a7e94c0	2018-03-05 09:35:41	49.00	15.10	
24	0010b2e5201cc5f1ae7e9c6cc8f5bd00	1	5a419dbf24a8c9718f522b81c69f61a	3504c0cb71d7fa48d967e0e4c94d59d9	2017-09-15 18:04:37	48.90	16.60	
25	00119ff934e539cf26f92b9ef0cdfed8	1	21b1c2f67a9aaf5bf0eb06c13b9fdbda	c864036feaabb8c1659f65ea4faebe1da	2017-08-11 00:35:12	219.90	16.98	
26	0011d82c4b53e22e84023405fb467e57	1	c389f712c4b4510bc997ce93e8b1a28	bfd27a966d91cfaafdb25d0765850fda	2018-01-29 21:51:25	289.00	26.33	
27	00125cb692d04887809806618a2a145f	1	1c0c0093a48f13ba70d0c6b0a9157cb7	41b39e28db005d9731d9d485a83b4c38	2017-03-29 13:05:42	109.90	25.51	
28	00130c0eee84a3d909e75bc08c5c3ca1	1	8932194e35fc6d7903d36f74e351d40	16090f2ca825584b5a147ab24aa30c86	2018-06-14 05:16:24	27.90	7.94	
29	0013503b13da1eac686219390b7d641b	1	38afdf723b95d455b418a0f57d623c6b	1554a68530182680adsc8b042c3ab563	2017-12-13 03:16:36	119.90	17.32	
30	00137e170939bba5a3134e2386413108	1	672e757f331900b9dea127a27b79fd	e59aa562b9f8076dd550fcddfe073491	2017-11-30 06:30:55	397.00	24.65	

## Módulo 18 – Analisando ordens, itens e pagamentos

Vamos pegar uma ordem qualquer e buscar os itens relacionados:

```
1 # Vamos selecionar uma ordem para analisar  
2 ordem = '001d8f0e34a38c37f7dba2a37d4eba8b'
```

```
1 # E filtrar na tabela de ordens  
2 orders[orders.order_id == ordem]
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
42205	001d8f0e34a38c37f7dba2a37d4eba8b	d987da9fb4086ab7c2c0f83963cd6722	delivered	2017-05-14 17:19:44	2017-05-14 17:35:11	2017-05-24 15:45:01	2017-05-26 13:14:50	2017-05-24 00:00:00

```
1 # Na tabela de ordem x itens  
2 order_items[order_items.order_id == ordem]
```

	order_id	order_item_id	product_id	seller_id	shipping_limit_date	price	freight_value
48	001d8f0e34a38c37f7dba2a37d4eba8b	1	e67307ff0f15ade43fcb6e670be7a74c	f4aba7c0bca51484c30ab7bdc34bcdd1	2017-05-18 17:35:11	18.99	7.78
49	001d8f0e34a38c37f7dba2a37d4eba8b	2	e67307ff0f15ade43fcb6e670be7a74c	f4aba7c0bca51484c30ab7bdc34bcdd1	2017-05-18 17:35:11	18.99	7.78

Como vemos, esta ordem possui dois itens iguais. Vemos que há um preço e um valor de frete.

## Módulo 18 – Analisando ordens, itens e pagamentos

Buscando na tabela de pagamentos pela mesma ordem, vemos que a mesma foi paga com cartão de crédito, em duas parcelas. O valor apresentado é a soma dos dois itens e seus respectivos fretes:

```
1 # E na tabela de pagamentos  
2 order_payments[order_payments.order_id == ordem]
```

		order_id	payment_sequential	payment_type	payment_installments	payment_value
24695	001d8f0e34a38c37f7dba2a37d4eba8b		1	credit_card	2	53.54

```
1 18.99+7.78+18.99+7.78
```

53.54

## Módulo 18 – Analisando ordens, itens e pagamentos

Podemos verificar quais ordens possuem mais de um item fazendo um groupby com contagem:

```
1 # Fazendo o group by das ordens e verificando aquelas que tiveram mais de 1 item
2 order_items.groupby('order_id')['order_id'].count().sort_values(ascending=False).head(15)

order_id
8272b63d03f5f79c56e9e4120aec44ef    21
1b15974a0141d54e36626dca3fdc731a    20
ab14fdcfbe524636d65ee38360e22ce8    20
9ef13efd6949e4573a18964dd1bbe7f5    15
428a2f660dc84138d969ccd69a0ab6d5    15
9bdc4d4c71aa1de4606060929dee888c    14
73c8ab38f07dc94389065f7eba4f297a    14
37ee401157a3a0b28c9c6d0ed8c3b24b    13
2c2a19b5703863c908512d135aa6accc    12
c05d6a79e55da72ca780ce90364abed9    12
3a213fcdf7d98be74ea0dc05a8b31ae    12
637617b3ffe9e2f7a2411243829226d0    12
af822dacd6f5cff7376413c03a388bb7    12
7f2c22c54cbae55091a09a9653fd2b8a    11
71dab1155600756af6de79de92e712e3    11
Name: order_id, dtype: int64
```

# Módulo 18 – Analisando ordens, itens e pagamentos

Vamos pegar uma dessas ordens:

```
1 # Vamos selecionar uma ordem para analisar
2 ordem = '1b15974a0141d54e36626dca3fdc731a'

1 # E filtrar na tabela de ordens
2 orders[orders.order_id == ordem]

order_id          customer_id  order_status  order_purchase_timestamp  order_approved_at  order_delivered_carrier_date  order_delivered_customer_date  order_estimated_delivery_date
86978  1b15974a0141d54e36626dca3fdc731a  be1b70680b9f9694d8c70f41fa3dc92b  delivered  2018-02-22 15:30:41  2018-02-24 03:20:27  2018-03-02 00:18:01  2018-03-05 15:22:27  2018-03-08 00:00:00

1 # Na tabela de ordem x itens
2 order_items[order_items.order_id == ordem].groupby(['order_id','product_id'])['order_id'].count()

order_id          product_id
1b15974a0141d54e36626dca3fdc731a  ee3d532c8a438679776d222e997606b3  20
Name: order_id, dtype: int64

1 products[products.product_id == 'ee3d532c8a438679776d222e997606b3']

product_id  product_category_name  product_name_lenght  product_description_lenght  product_photos_qty  product_weight_g  product_length_cm  product_height_cm  product_width_cm
27921  ee3d532c8a438679776d222e997606b3  informatica_acessorios  43.0  452.0  1.0  360.0  19.0  18.0  15.0
```

Vemos que esta ordem também possui apenas um produto, tendo sido comprado 20 itens do mesmo produto. Com a base de produtos, vemos que se trata de um acessório de informática.

Mas será que existem ordens com mais de um item, sendo os itens produtos distintos?

## Módulo 18 – Pivot

Vamos começar olhando novamente nossa base de itens:

```
1 order_items.head()
```

	order_id	order_item_id	product_id	seller_id	shipping_limit_date	price	freight_value
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	48436dade18ac8b2bce089ec2a041202	2017-09-19 09:45:35	58.90	13.29
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	dd7ddc04e1b6c2c614352b383efe2d36	2017-05-03 11:05:13	239.90	19.93
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fd	5b51032eddd242adc84c38acab88f23d	2018-01-18 14:48:30	199.00	17.87
3	00024acbcdf0a6daa1e931b038114c75	1	7634da152a4610f1595efa32f14722fc	9d7a1d34a5052409006425275ba1c2b4	2018-08-15 10:10:18	12.99	12.79
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	ac6c3623068f30de03045865e4e10089	df560393f3a51e74553ab94004ba5c87	2017-02-13 13:57:51	199.90	18.14

Veja que há a coluna *order\_item\_id* que, em ordens com mais de um item, vai aumentando seu valor para cada item (1, 2, 3, ...).

Para nossa análise, não precisamos das informações de vendedor, data, preço e frete. Assim, podemos nos restringir a apenas colunas sobre ordem e produto:

```
1 # Criando um novo DataFrame apenas com o id da ordem, do item e da ordem_item
2 tabela = order_items[['order_id','order_item_id','product_id']]
```

## Módulo 18 – Pivot

Agora podemos fazer o que chamamos de tabela pivot. Uma tabela pivot é uma ferramenta que permite resumir, analisar, explorar e apresentar grandes quantidades de dados em uma tabela compacta e fácil de entender, organizando e agrupando os dados selecionados de acordo com critérios específicos.

No nosso caso, buscamos ordens que possuam itens distintos. Assim, podemos colocar *order\_item\_id* como colunas, onde os valores *product\_id* preencherão as colunas:

```
1 # Fazendo o pivot da tabela
2 tabela_pivot = tabela.pivot(index='order_id',columns='order_item_id').reset_index()
3 tabela_pivot.head()
```

order_item_id	order_id	product_id																			
		1	2	3	4	5	6	7	8	9	...	12	13	14	15	16	17	18	19	20	21
0	00010242fe8c5a6d1ba2dd792cb16214	4244733e06e7ecb4970a6e2683c13e61	NaN	...	NaN																
1	00018f77f2f0320c557190d7a144bdd3	e5f2d52b802189ee658865ca93d83a8f	NaN	...	NaN																
2	000229ec398224ef6ca0657da4fc703e	c777355d18b72b67abbeef9df44fd0fd	NaN	...	NaN																
3	00024acbcdf0a6daa1e931b038114c75	7634da152a4610f1595efa32f14722fc	NaN	...	NaN																
4	00042b26cf59d7ce69dfabb4e55b4fd9	ac6c3623068f30de03045865e4e10089	NaN	...	NaN																

5 rows × 22 columns

## Módulo 18 – Pivot

Apenas para facilitar, podemos renomear as colunas:

```
1 # Renomeando as colunas para os nomes abaixo
2 colunas = ['order_id', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21']
3 tabela_pivot.columns = colunas
4 tabela_pivot.head()
```

	order_id	1	2	3	4	5	6	7	8	9	...	12	13	14	15	16	17	18	19	20	21
0	00010242fe8c5a6d1ba2dd792cb16214	4244733e06e7ecb4970a6e2683c13e61	NaN	...	NaN																
1	00018f77f2f0320c557190d7a144bdd3	e5f2d52b802189ee658865ca93d83a8f	NaN	...	NaN																
2	000229ec398224ef6ca0657da4fc703e	c777355d18b72b67abbeef9df44fd0fd	NaN	...	NaN																
3	00024acbcdf0a6daa1e931b038114c75	7634da152a4610f1595efa32f14722fc	NaN	...	NaN																
4	00042b26cf59d7ce69dfabb4e55b4fd9	ac6c3623068f30de03045865e4e10089	NaN	...	NaN																

## Módulo 18 – Pivot

Agora, podemos verificar quando o item 2 do pedido existe (não é nulo) e é diferente do item 1:

```
1 # Retirando as linhas onde o id 2 é vazio
2 tabela_pivot = tabela_pivot[tabela_pivot['2'].notnull()]
```

```
1 # Buscando valores onde o id 1 seja diferente do id 2
2 tabela_pivot[tabela_pivot['1'] != tabela_pivot['2']]
```

	order_id	1	2	3	4	5	6	7	8	9	...	12	13	14	15	16	17	18	19	20	21
73	002f98c0f7efd42638ed6100ca699b42	d41dc2f2979f52d75d78714b378d4068	880be32f4db1d9f6e2bec38fb6ac23ab	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN										
82	00337fe25a3780b3424d9ad7c5a4b35e	1f9799a175f50c9fa725984775cac5c5	13944d17b257432717fd260e69853140	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN										
134	005d9a5423d47281ac463a968b3936fb	fb7a100ec8c7b34f60cec22b1a9a10e0	4c3ae5db49258df0784827bdacf3b396	4c3ae5db49258df0784827bdacf3b396	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN									
198	00946f674d880be1f188abc10ad7cf46	4dcba9b9ca7e48d2f108d40caa77caa2	9bb2d066e4b33b624cbdfec7d50b3dcb	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN										
208	0097f0545a302aafa32782f1734ff71c	b6397895a17ce86decd60b898b459796	636598095d69a5718e67d2c9a3c7dde6	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN										
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
98540	ffa5e4c604dea4f0a59d19cc2322ac19	59d8034bb2a82cd5092e78d42148fa44	bd421826916d3e1d445cb860cea3c0fb	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN										
98555	ffb18bf111fa70edf316eb0390427986	599dc392f7a23273471b068d72408224	e86b81dcac341ea01df0260077cdf082	599dc392f7a23273471b068d72408224	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN									
98563	ffb8f7de8940249a3221252818937ecb	bd6e8cf9fe4122c385da2bcb9f979d5d	803f77475e1b51b47f1bfec4f2ec353f	bd0ac51dc93e62c4dbe6ca9d70a9b311	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN									
98575	ffc16cecff8dc037f60458f28d1c1ba5	241e398aacc909372622952b2ec6f954	7c1043bb5837db0c6bc1953419a18628	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN										
98596	ffd543c2b60842e148a86870dc60e212	3ce943997ff85cad84ec6770b35d6bcd	b7d94dc0640c7025dc8e3b46b52d8239	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN										

3104 rows x 22 columns

Vemos que há 3104 ordens com esta característica.

## Módulo 18 – Analisando pagamentos, vendedores e avaliações

Aprofundando mais em nossa base, podemos verificar se há alguma ordem com mais de um tipo de pagamento:

order_id	payment_sequential	payment_type	payment_installments	payment_value
b81ef226f3fe1789b1e8b2acac839d17	1	credit_card	8	99.33
a9810da82917af2d9aefcd1278f1dcfa0	1	credit_card	1	24.39
25e8ea4e93396b6fa0d3dd708e76c1bd	1	credit_card	1	65.71
ba78997921bbcdc1373bb41e913ab953	1	credit_card	8	107.78
42fdf880ba16b47b59251dd489d4441a	1	credit_card	2	128.45

1	# Filtrando na base de pagamentos se existem itens repetidos
2	order_payments.groupby('order_id')['order_id'].count().sort_values()

order_id	
00010242fe8c5a6d1ba2dd792cb16214	1
a9fa8567c21a5f8e4450af02a87f26cb	1
a9fa7ea99c0724de00e9d31f2a39a723	1
a9fa4232de4a87398cfab2a5bcbecc56	1
a9f99ebcef2a23dd06a0bd5ed649080e	1
	..
fedcd9f7ccdc8cba3a18defedd1a5547	19
895ab968e7bb0d5659d16cd74cd1650c	21
285c2e15beb4ac83635ccc563dc71f4	22
ccf804e764ed5650cd8759557269dc13	26
fa65dad1b0e818e3ccc5cb0e39231352	29
Name: order_id, Length: 99440, dtype: int64	

## Módulo 18 – Analisando pagamentos, vendedores e avaliações

Vamos pegar uma ordem com contagem maior do que 1.

Veja que, no caso selecionado, o cliente usou cartão de crédito e diversos vouchers com valores distintos.

		# Analisando os itens com mais de 1 linha				
1	2	order_payments[order_payments.order_id == 'fedcd9f7ccdc8cba3a18defedd1a5547']				
		order_id	payment_sequential	payment_type	payment_installments	payment_value
	7548	fedcd9f7ccdc8cba3a18defedd1a5547		15	voucher	1 10.66
	9665	fedcd9f7ccdc8cba3a18defedd1a5547		1	credit_card	1 1.67
	14352	fedcd9f7ccdc8cba3a18defedd1a5547		9	voucher	1 5.30
	15264	fedcd9f7ccdc8cba3a18defedd1a5547		11	voucher	1 8.42
	15930	fedcd9f7ccdc8cba3a18defedd1a5547		8	voucher	1 31.43
	18175	fedcd9f7ccdc8cba3a18defedd1a5547		14	voucher	1 7.63
	27087	fedcd9f7ccdc8cba3a18defedd1a5547		2	voucher	1 7.76
	32396	fedcd9f7ccdc8cba3a18defedd1a5547		13	voucher	1 5.84
	35657	fedcd9f7ccdc8cba3a18defedd1a5547		19	voucher	1 8.42
	36869	fedcd9f7ccdc8cba3a18defedd1a5547		5	voucher	1 9.76
	44641	fedcd9f7ccdc8cba3a18defedd1a5547		10	voucher	1 13.27
	45197	fedcd9f7ccdc8cba3a18defedd1a5547		4	voucher	1 10.33
	48288	fedcd9f7ccdc8cba3a18defedd1a5547		17	voucher	1 7.51
	62558	fedcd9f7ccdc8cba3a18defedd1a5547		12	voucher	1 9.76
	64520	fedcd9f7ccdc8cba3a18defedd1a5547		16	voucher	1 9.54
	87292	fedcd9f7ccdc8cba3a18defedd1a5547		18	voucher	1 11.12
	87783	fedcd9f7ccdc8cba3a18defedd1a5547		3	voucher	1 26.94
	95493	fedcd9f7ccdc8cba3a18defedd1a5547		7	voucher	1 11.78
	103569	fedcd9f7ccdc8cba3a18defedd1a5547		6	voucher	1 8.60

## Módulo 18 – Analisando pagamentos, vendedores e avaliações

Podemos, também, validar a informação passada que um mesmo item pode ser vendido por mais de um vendedor. Vamos criar uma tabela, a partir da base de itens, com a identificação dos produtos e dos vendedores:

```
1 | order_items.head()
```

	order_id	order_item_id	product_id	seller_id	shipping_limit_date	price	freight_value
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	48436dade18ac8b2bce089ec2a041202	2017-09-19 09:45:35	58.90	13.29
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	dd7ddc04e1b6c2c614352b383efe2d36	2017-05-03 11:05:13	239.90	19.93
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fd	5b51032eddd242adc84c38acab88f23d	2018-01-18 14:48:30	199.00	17.87
3	00024acbcdf0a6daa1e931b038114c75	1	7634da152a4610f1595efa32f14722fc	9d7a1d34a5052409006425275ba1c2b4	2018-08-15 10:10:18	12.99	12.79
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	ac6c3623068f30de03045865e4e10089	df560393f3a51e74553ab94004ba5c87	2017-02-13 13:57:51	199.90	18.14

```
1 | # Selecionando uma base com apenas o product_id e o seller_id  
2 | tabela = order_items[['product_id','seller_id']]
```

## Módulo 18 – Analisando pagamentos, vendedores e avaliações

Agora, podemos eliminar as duplicatas (informações que mostram mesmo produto para mesmo vendedor) e contar as entradas para um mesmo produto:

```
1 # Eliminando valores duplicados
2 tabela = tabela.drop_duplicates()
3 tabela.head()
```

	product_id	seller_id
0	4244733e06e7ecb4970a6e2683c13e61	48436dade18ac8b2bce089ec2a041202
1	e5f2d52b802189ee658865ca93d83a8f	dd7ddc04e1b6c2c614352b383efe2d36
2	c777355d18b72b67abbeef9df44fd0fd	5b51032eddd242adc84c38acab88f23d
3	7634da152a4610f1595efa32f14722fc	9d7a1d34a5052409006425275ba1c2b4
4	ac6c3623068f30de03045865e4e10089	df560393f3a51e74553ab94004ba5c87

```
1 # Contando a quantidade de product_id para verificar se existe mais de 1 vendedor
2 tabela.groupby('product_id')['product_id'].count().sort_values()
```

```
product_id
00066f42aeeb9f3007548bb9d3f33c38      1
aa0d48704d551d1247be98ca06dfa990      1
aa0c047aa75f7c8e437acd62601055fe      1
aa0816cf69c9faa95e573541be5c0c32      1
aa07dadb101ca2d652eaaec5c33c37af      1
..                                           ..
656e0eca68dcecf6a31b8ececfcabe3e8      7
36f60d45225e60c7da4558b070ce4b60      7
4298b7e67dc399c200662b569563a2b2      7
69455f41626a745aea9ee9164cb9eafd      8
d285360f29ac7fd97640bf0baef03de0      8
Name: product_id, Length: 32951, dtype: int64
```

## Módulo 18 – Analisando pagamentos, vendedores e avaliações

Podemos pegar algum dos produtos com contagem maior do que 1 e verificar que, realmente, há mais de um vendedor:

```
1 # Filtrando na base qualquer um desses produtos
2 tabela[tabela.product_id == '656e0eca68dcecf6a31b8ecefabe3e8']
```

	product_id	seller_id
111	656e0eca68dcecf6a31b8ecefabe3e8	0b90b6df587eb83608a64ea8b390cf07
1462	656e0eca68dcecf6a31b8ecefabe3e8	5f67c6082caacb26e431a7b17940cece
1510	656e0eca68dcecf6a31b8ecefabe3e8	e9bc59e7b60fc3063eb2290deda4cced
10924	656e0eca68dcecf6a31b8ecefabe3e8	9c0e69c7bf2619675bbadf47b43f655a
16932	656e0eca68dcecf6a31b8ecefabe3e8	8e6d7754bc7e0f22c96d255ebda59eba
21716	656e0eca68dcecf6a31b8ecefabe3e8	00fc707aaaad2d31347cf883cd2dfe10
41805	656e0eca68dcecf6a31b8ecefabe3e8	6973a06f484aacf400ece213dbf3d946

## Módulo 18 – Analisando pagamentos, vendedores e avaliações

Agora, vejamos a tabela de reviews, de avaliações e comentários dos clientes.

1	# Visualizando a base							
2	order_reviews.head()							
		review_id	order_id	review_score	review_comment_title	review_comment_message	review_creation_date	review_answer_timestamp
0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4	NaN		NaN	2018-01-18 00:00:00	2018-01-18 21:46:59
1	80e641a11e56f04c1ad469d5645fdfde	a548910a1c6147796b98fdf73dbeba33	5	NaN		NaN	2018-03-10 00:00:00	2018-03-11 03:05:13
2	228ce5500dc1d8e020d8d1322874b6f0	f9e4b658b201a9f2ecdecbb34bed034b	5	NaN		NaN	2018-02-17 00:00:00	2018-02-18 14:36:24
3	e64fb393e7b32834bb789ff8bb30750e	658677c97b385a9be170737859d3511b	5	NaN	Recebi bem antes do prazo estipulado.	2017-04-21 00:00:00	2017-04-21 22:02:06	
4	f7c4243c7fe1938f181bec41a392bdeb	8e6fb81e283fa7e4f11123a3fb894f1	5	NaN	Parabéns lojas lannister adorei comprar pela I...	2018-03-01 00:00:00	2018-03-02 10:26:53	

Com uma contagem, podemos ver que boa parte das reviews foram com notas altas (4 e 5).

1	# Verificando as notas
2	order_reviews.review_score.value_counts()
	5 57328
	4 19142
	1 11424
	3 8179
	2 3151
	Name: review_score, dtype: int64

## Módulo 18 – Analisando pagamentos, vendedores e avaliações

Somando os valores, temos 99224 avaliações. Considerando que há 99441 entradas na tabela de ordens, temos que boa parte dos clientes avaliam seus pedidos.

```
1 # Analisando quantas pessoas avaliam suas compras
2 order_reviews.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99224 entries, 0 to 99223
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   review_id        99224 non-null   object  
 1   order_id         99224 non-null   object  
 2   review_score     99224 non-null   int64  
 3   review_comment_title  11568 non-null   object  
 4   review_comment_message  40977 non-null   object  
 5   review_creation_date  99224 non-null   object  
 6   review_answer_timestamp  99224 non-null   object  
dtypes: int64(1), object(6)
memory usage: 5.3+ MB
```

```
1 57328+19142+11424+8179+3151
```

99224

## Módulo 18 – Criando um banco de dados com sqlite3

Em módulos anteriores, já vimos um básico de SQL. Aqui, vamos aprender como criar uma base de dados com a biblioteca `sqlite3` e como inserir, atualizar e deletar dados. Desta forma, poderemos, depois, pegar os arquivos csv e passá-los para um banco de dados, facilitando a exploração da base.

Seguinte o tutorial da documentação do `sqlite3`, podemos criar uma base para filmes com uma tabela chamada `movie`, com três colunas: `title`, `year`, e `score`.

```
1 # Importando o sqlite
2 import sqlite3
```

```
1 # Criando uma conexão
2 con = sqlite3.connect("tutorial2.db")
```

```
1 # Criando o cursor
2 cur = con.cursor()
```

```
1 # Criando a tabela da documentação
2 cur.execute("CREATE TABLE movie(title, year, score)")
```

```
<sqlite3.Cursor at 0x7fd4cc6248f0>
```

Quando não existe uma base de dados com o nome passado ao método `connect`, a biblioteca cria o arquivo da base na mesma pasta do arquivo Jupyter Notebook que está sendo utilizado.

## Módulo 18 – Criando um banco de dados com sqlite3

Se tentarmos pegar os dados desta tabela com `fetchall`, teremos como resultado uma lista vazia. Afinal, não foram inseridos dados na tabela.

Para inserir, usamos a sintaxe

```
INSERT INTO <tabela> VALUES  
    (valor1, valor2, valor3),  
    ...
```

No nosso caso, os valores correspondem ao título, ano e nota de um dado filme.

Após a inserção, `fetchall` retorna as entradas inseridas. Para garantir a gravação dos dados, usamos o método `commit` na conexão e fechamos esta com o método `close`.

```
1 cur.execute('SELECT * FROM movie').fetchall()  
[]
```

```
1 # Inserindo uma linha conforme a documentação  
2 cur.execute(  
3     "INSERT INTO movie VALUES  
4         ('Monty Python and the Holy Grail', 1975, 8.2),  
5         ('And Now for Something Completely Different', 1971, 7.5)  
6     "")
```

```
<sqlite3.Cursor at 0x7fd4cc6248f0>
```

```
1 # Selecionando todos os dados dessa tabela criada  
2 cur.execute('SELECT * FROM movie').fetchall()
```

```
[('Monty Python and the Holy Grail', 1975, 8.2),  
 ('And Now for Something Completely Different', 1971, 7.5)]
```

```
1 # Salvando as mudanças  
2 con.commit()
```

```
1 # Fechando a conexão  
2 con.close()
```

## Módulo 18 – Criando um banco de dados com sqlite3

Também podemos criar uma base a partir de um dataframe do Pandas. Abaixo, criamos um dataframe *dados* e, então, criamos uma base chamada *exemplo.db*. Com o método `to_sql` do Pandas, passamos nosso dataframe para uma tabela chamada *dados* para uma conexão da nossa base. Observe que as entradas retornadas por `fetchall` são as do nosso dataframe.

```
1 # Importando o pandas
2 import pandas as pd
```

```
1+ dados = {
2     'X': [1,2,3,4,5],
3     'Y': [10,9,8,7,6],
4     'Z': ['a','b','c','d','e']
5 }
6
7 dados = pd.DataFrame(dados)
```

```
1 # Visualizando esse DataFrame
2 dados.head()
```

	X	Y	Z
0	1	10	a
1	2	9	b
2	3	8	c
3	4	7	d
4	5	6	e

```
1 # Abrindo novamente a conexão
2 import sqlite3
3 con = sqlite3.connect('exemplo.db')
```

```
1 # Adicionando essa base como uma tabela
2 dados.to_sql('dados',con)
```

5

```
1 # Criando o cursor
2 cur = con.cursor()
```

```
1 # Selecionando todos os dados dessa tabela criada
2 cur.execute('SELECT * FROM dados').fetchall()
```

```
[(0, 1, 10, 'a'),
 (1, 2, 9, 'b'),
 (2, 3, 8, 'c'),
 (3, 4, 7, 'd'),
 (4, 5, 6, 'e')]
```

## Módulo 18 – Criando um banco de dados com sqlite3

Observando com mais cuidado o retorno da página anterior, vemos que as numerações das linhas (0, 1, 2, ...) foram inseridas. Estes números são os índices, como podemos ver no atributo `description` do cursor criado. Podemos evitar isto passando o parâmetro `index=False` para o método `to_sql`. O parâmetro `if_exists='replace'` evita erro caso a tabela já exista, dando a instrução de substituir a tabela existente com a que está sendo passada.

```
1 # Verificando o nome das colunas
2 cur.description

((('index', None, None, None, None, None, None),
 ('X', None, None, None, None, None, None),
 ('Y', None, None, None, None, None, None),
 ('Z', None, None, None, None, None, None))

1 # Adicionando como tabela mas sem o index
2 dados.to_sql('dados',con,index=False,if_exists='replace')
```

5

```
1 # Selecionando todos os dados dessa tabela criada
2 cur.execute('SELECT * FROM dados').fetchall()

[(1, 10, 'a'), (2, 9, 'b'), (3, 8, 'c'), (4, 7, 'd'), (5, 6, 'e')]
```

```
1 # Verificando o nome das colunas
2 cur.description

((('X', None, None, None, None, None, None),
 ('Y', None, None, None, None, None, None),
 ('Z', None, None, None, None, None, None))
```

Ao tentar criar uma nova tabela com o mesmo nome, teremos o erro mostrado abaixo:

```
ValueError: Table 'dados' already exists.
```

## Módulo 18 – Criando um banco de dados com sqlite3

Uma tabela pode ser excluída com o comando

DROP TABLE <nome da tabela>

Se tentarmos um fetchall após o drop, um erro será exibido.

Inserindo novamente os dados, o fetchall passa a exibir normalmente os dados inseridos.

```
1 # Executando o DROP dessa tabela
2 cur.execute('DROP TABLE dados')
```

```
<sqlite3.Cursor at 0x7f2cd8d033b0>
```

```
1 # Tentando selecionar os dados dessa tabela novamente
2 cur.execute('SELECT * FROM dados').fetchall()
```

```
-----
OperationalError                                     Traceback (most recent call last)
Cell In[10], line 2
      1 # Tentando selecionar os dados dessa tabela novamente
----> 2 cur.execute('SELECT * FROM dados').fetchall()

OperationalError: no such table: dados
```

```
1 # Inserindo novamente os dados (sem o index)
2 dados.to_sql('dados',con,index=False,if_exists='replace')
```

5

```
1 # Selecionando os dados
2 cur.execute('SELECT * FROM dados').fetchall()
```

```
[(1, 10, 'a'), (2, 9, 'b'), (3, 8, 'c'), (4, 7, 'd'), (5, 6, 'e')]
```

## Módulo 18 – Criando um banco de dados com sqlite3

Se colocarmos o parâmetro `if_exists='append'`, podemos atualizar uma tabela com dados de outro dataframe ao invés de substituir os dados existentes.

No caso, criamos outro dataframe e atualizamos nossa tabela `dados` presente no banco de dados.

```
1  dados2 = {  
2      'X': [6,7,8],  
3      'Y': [5,4,3],  
4      'Z': ['f','l','h']  
5  }  
6  
7  dados2 = pd.DataFrame(dados2)  
8  
9  dados2.head()
```

	X	Y	Z
0	6	5	f
1	7	4	l
2	8	3	h

```
1  # Usando o append da base  
2  dados2.to_sql('dados',con,index=False,if_exists='append')
```

3

```
1  # Fazendo o SELECT  
2  cur.execute('SELECT * FROM dados').fetchall()
```

```
[(1, 10, 'a'),  
(2, 9, 'b'),  
(3, 8, 'c'),  
(4, 7, 'd'),  
(5, 6, 'e'),  
(6, 5, 'f'),  
(7, 4, 'l'),  
(8, 3, 'h')]
```

## Módulo 18 – Criando um banco de dados com sqlite3

Usando apenas SQL, também podemos inserir novos dados em uma tabela já existente.

Uma forma é usar o `INSERT INTO` da forma já mostrada anteriormente:

```
1 # Podemos adicionar diretamente os valores
2 # Adicionando o valor (9,2,'j')
3 cur.execute('INSERT INTO dados values (9,2,"j")')
```

```
<sqlite3.Cursor at 0x7f2cd8d033b0>
```

```
1 # Fazendo o SELECT
2 cur.execute('SELECT * FROM dados').fetchall()
```

```
[(1, 10, 'a'),
 (2, 9, 'b'),
 (3, 8, 'c'),
 (4, 7, 'd'),
 (5, 6, 'e'),
 (6, 5, 'f'),
 (7, 4, 'l'),
 (8, 3, 'h'),
 (9, 2, 'j')]
```

## Módulo 18 – Criando um banco de dados com sqlite3

O `INSERT INTO` também aceita marcadores de posição (*placeholders* em inglês). Em SQL, tais marcadores são indicados com pontos de interrogação:

```
1 # Utilizando placeholders para adicionar os valores (10,1,'j')
2 cur.execute('INSERT INTO dados values (?,?,?)',(10,1,'j'))
```

```
<sqlite3.Cursor at 0x7f2cd8d033b0>
```

```
1 # Fazendo o SELECT
2 cur.execute('SELECT * FROM dados').fetchall()
```

```
[(1, 10, 'a'),
 (2, 9, 'b'),
 (3, 8, 'c'),
 (4, 7, 'd'),
 (5, 6, 'e'),
 (6, 5, 'f'),
 (7, 4, 'l'),
 (8, 3, 'h'),
 (9, 2, 'j'),
 (10, 1, 'j')]
```

## Módulo 18 – Criando um banco de dados com sqlite3

A vantagem de utilizar *placeholders* é que podemos utilizar o método `executemany` e passar uma lista de tuplas com os valores a serem inseridos:

```
1 adicionar = [
2     (11, 0, 'k'),
3     (12, -1, 'l'),
4     (13, -2, 'm')
5 ]
```

```
1 # Adicionando esses valores
2 cur.executemany('INSERT INTO dados values (?,?,?)',adicionar)
```

<sqlite3.Cursor at 0x7f2cd8d033b0>

```
1 # Fazendo o SELECT
2 cur.execute('SELECT * FROM dados').fetchall()
```

```
[(1, 10, 'a'),
(2, 9, 'b'),
(3, 8, 'c'),
(4, 7, 'd'),
(5, 6, 'e'),
(6, 5, 'f'),
(7, 4, 'l'),
(8, 3, 'h'),
(9, 2, 'j'),
(10, 1, 'j'),
(11, 0, 'k'),
(12, -1, 'l'),
(13, -2, 'm')]
```

```
1 # Salvando e fechando a conexão
2 con.commit()
3 con.close()
```

## Módulo 18 – Atualizando e deletando registros

E se quisermos atualizar ou corrigir algum dado? Podemos utilizar o comando UPDATE. No exemplo, atualizamos toda a coluna Z para apresentar a letra g:

```
1 # Fazendo o SELECT  
2 cur.execute('SELECT * FROM dados').fetchall()
```

```
[(1, 10, 'a'),  
(2, 9, 'b'),  
(3, 8, 'c'),  
(4, 7, 'd'),  
(5, 6, 'e'),  
(6, 5, 'f'),  
(7, 4, 'l'),  
(8, 3, 'h'),
```

```
1 # Atualizando a linha 7 para a letra g ao invés da letra l  
2 cur.execute('UPDATE dados SET Z = "g"')
```

```
<sqlite3.Cursor at 0x7f4339c08a40>
```

```
1 # Retornando todos os valores  
2 cur.execute('SELECT * FROM dados').fetchall()
```

```
[(1, 10, 'g'),  
(2, 9, 'g'),  
(3, 8, 'g'),  
(4, 7, 'g'),  
(5, 6, 'g'),  
(6, 5, 'g'),  
(7, 4, 'g'),  
(8, 3, 'g')]
```

## Módulo 18 – Atualizando e deletando registros

Para deletar, utilizamos o comando **DELETE**. Repare na importância de utilizar **WHERE**, para não correr o risco de deletar todos os dados:

```
1 import sqlite3  
2 con = sqlite3.connect('exemplo.db')  
  
1 cur = con.cursor()  
  
1 # Visualizando a nossa tabela  
2 cur.execute('SELECT * FROM dados').fetchall()  
  
[(1, 10, 'a'),  
(2, 9, 'b'),  
(3, 8, 'c'),  
(4, 7, 'd'),  
(5, 6, 'e'),  
(6, 5, 'f'),  
(7, 4, 'l'),  
(8, 3, 'h'),  
(9, 2, 'j'),  
(10, 1, 'j'),  
(11, 0, 'k'),  
(12, -1, 'l'),  
(13, -2, 'm')]
```

```
1 # Deletando as últimas 2 linhas  
2 cur.execute('DELETE FROM dados \  
WHERE Y < 0')  
  
<sqlite3.Cursor at 0x7f2e072a8420>  
  
1 # Visualizando os dados  
2 cur.execute('SELECT * FROM dados').fetchall()  
  
[(1, 10, 'a'),  
(2, 9, 'b'),  
(3, 8, 'c'),  
(4, 7, 'd'),  
(5, 6, 'e'),  
(6, 5, 'f'),  
(7, 4, 'g'),  
(8, 3, 'h'),  
(9, 2, 'j'),  
(10, 1, 'j'),  
(11, 0, 'k')]
```

Para deletar uma tabela por completo da base, utilizamos **DROP TABLE**:

```
1 # Apagando a tabela  
2 cur.execute('DROP TABLE dados')  
  
<sqlite3.Cursor at 0x7f4339c08a40>
```

# Módulo 18 – Criando um banco de dados para nosso estudo

Agora que já vimos como criar uma base de dados com SQLite, vamos voltar ao nosso estudo da Olist. Lembrando, temos os seguintes arquivos e podemos passá-los para nosso banco de dados como já visto anteriormente:

```
1 # Importando a biblioteca
2 import os
```

```
1 # Guardando a pasta atual em uma variável
2 pasta = os.getcwd()
```

```
1 # Guardando os arquivos em uma variável
2 arquivos = os.listdir(pasta + '/bases')
```

```
1 # Visualizando os arquivos
2 arquivos
```

```
['olist_customers_dataset.csv',
 'product_category_name_translation.csv',
 'olist_order_reviews_dataset.csv',
 'olist_geolocation_dataset.csv',
 'olist_order_items_dataset.csv',
 'olist_orders_dataset.csv',
 'olist_sellers_dataset.csv',
 'olist_products_dataset.csv',
 'olist_order_payments_dataset.csv']
```

```
1 # Importando o sqlite3
2 import sqlite3
```

```
1 # Criando a conexão e o nosso DB
2 con = sqlite3.connect("vendas_db.db")
```

```
1 base.to_sql('orders',con)
```

99441

```
1 # Percorrendo toda a lista de arquivos
2 for i in arquivos:
3     print(i)
4     nome_base = i.replace('.csv','').replace('olist_','').replace('_dataset','')
5     print(nome_base)
6     base = pd.read_csv(pasta + '/bases' + '/' + i)
7     display(base.head(3))
8     base.to_sql(nome_base,con)
```

olist\_customers\_dataset.csv  
customers

	customer_id	customer_unique_id	customer_zip_code_prefix	customer_city	customer_state	
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0		14409	franca	SP
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3		9790	sao bernardo do campo	SP
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e		1151	sao paulo	SP

product\_category\_name\_translation.csv  
product\_category\_name\_translation

product_category_name	product_category_name_english
beleza_saude	health_beauty
informatica_acessorios	computers_accessories
automotivo	auto

# Módulo 18 – Criando um banco de dados para nosso estudo

Assim como já fizemos anteriormente no módulo, todos os arquivos geram dataframes:

olist\_order\_reviews\_dataset.csv  
order\_reviews

	review_id	order_id	review_score	review_comment_title	review_comment_message	review_creation_date	review_answer_timestamp
0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4	NaN	NaN	2018-01-18 00:00:00	2018-01-18 21:46:59
1	80e641a11e56f04c1ad469d5645fdfde	a548910a1c6147796b98fdf73dbeba33	5	NaN	NaN	2018-03-10 00:00:00	2018-03-11 03:05:13
2	228ce5500dc1d8e020d8d1322874b6f0	f9e4b658b201a9f2ecdecbb34bed034b	5	NaN	NaN	2018-02-17 00:00:00	2018-02-18 14:36:24

olist\_geolocation\_dataset.csv  
geolocation

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolocation_state
0	1037	-23.545621	-46.639292	sao paulo	SP
1	1046	-23.546081	-46.644820	sao paulo	SP
2	1046	-23.546129	-46.642951	sao paulo	SP

olist\_order\_items\_dataset.csv  
order\_items

	order_id	order_item_id	product_id	seller_id	shipping_limit_date	price	freight_value
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	48436dade18ac8b2bce089ec2a041202	2017-09-19 09:45:35	58.9	13.29
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	dd7ddc04e1b6c2c614352b383efe2d36	2017-05-03 11:05:13	239.9	19.93
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fd	5b51032eddd242adc84c38acab88f23d	2018-01-18 14:48:30	199.0	17.87

olist\_orders\_dataset.csv  
orders

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15	2017-10-04 19:55:00	2017-10-10 21:25:13	2017-10-18 00:00:00
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	2018-07-24 20:41:37	2018-07-26 03:24:27	2018-07-26 14:31:00	2018-08-07 15:27:45	2018-08-13 00:00:00
2	47770eb9100c2d0c44946d9cf07ec65d	41ce2a54c0b03bf3443c3d931a367089	delivered	2018-08-08 08:38:49	2018-08-08 08:55:23	2018-08-08 13:50:00	2018-08-17 18:06:29	2018-09-04 00:00:00

# Módulo 18 – Criando um banco de dados para nosso estudo

No caso da base de ordens, é necessário passar o tipo adequando para as colunas de datas:

```
1 # Visualizando as 2 primeiras linhas.
2 base = pd.read_csv(pasta+'bases'+'/olist_orders_dataset.csv')

1 # Verificando as informações da base
2 base.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   order_id         99441 non-null   object  
 1   customer_id      99441 non-null   object  
 2   order_status      99441 non-null   object  
 3   order_purchase_timestamp 99441 non-null   object  
 4   order_approved_at 99281 non-null   object  
 5   order_delivered_carrier_date 97658 non-null   object  
 6   order_delivered_customer_date 96476 non-null   object  
 7   order_estimated_delivery_date 99441 non-null   object  
dtypes: object(8)
memory usage: 6.1+ MB
```

```
1 # Fazendo o tratamento para as colunas de datas
2 datas = ['order_purchase_timestamp','order_approved_at','order_delivered_carrier_date',
           'order_delivered_customer_date','order_estimated_delivery_date']
4
5 for i in datas:
6     base[i] = pd.to_datetime(base[i], format='%Y-%m-%d %H:%M:%S')

1 base.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   order_id         99441 non-null   object  
 1   customer_id      99441 non-null   object  
 2   order_status      99441 non-null   object  
 3   order_purchase_timestamp 99441 non-null   datetime64[ns]
 4   order_approved_at 99281 non-null   datetime64[ns]
 5   order_delivered_carrier_date 97658 non-null   datetime64[ns]
 6   order_delivered_customer_date 96476 non-null   datetime64[ns]
 7   order_estimated_delivery_date 99441 non-null   datetime64[ns]
dtypes: datetime64[ns](5), object(3)
memory usage: 6.1+ MB
```

	base.head()							
	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15	2017-10-04 19:55:00	2017-10-10 21:25:13	2017-10-18
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	2018-07-24 20:41:37	2018-07-26 03:24:27	2018-07-26 14:31:00	2018-08-07 15:27:45	2018-08-13
2	47770eb9100c2d0c44946d9cf07ec65d	41ce2a54c0b03bf3443c3d931a367089	delivered	2018-08-08 08:38:49	2018-08-08 08:55:23	2018-08-08 13:50:00	2018-08-17 18:06:29	2018-09-04
3	949d5b44dbf5de918fe9c16f97b45f8a	f88197465ea7920adcdbec7375364d82	delivered	2017-11-18 19:28:06	2017-11-18 19:45:59	2017-11-22 13:39:59	2017-12-02 00:28:42	2017-12-15
4	ad21c59c0840e6cb83a9ceb5573f8159	8ab97904e6daeaa8866dbdbc4fb7aad2c	delivered	2018-02-13 21:18:39	2018-02-13 22:20:29	2018-02-14 19:46:34	2018-02-16 18:17:02	2018-02-26

# Módulo 18 – Criando um banco de dados para nosso estudo

Com esta atualização no dataframe, podemos atualizar a base correspondente em nosso banco de dados:

```
1 # Substituindo a base antiga por essa no sql
2 base.to_sql('orders',con,if_exists='replace')
```

99441

```
1 # Fazendo qualquer consulta nessa tabela e guardando na variável resultado
2 consulta = 'SELECT * FROM orders LIMIT 5'
3 resultado = cur.execute(consulta).fetchall()
```

```
1 # Visualizando esse resultado
2 resultado
[(0,
  'e481f51cbdc54678b7cc49136f2d6af7',
  '9ef432eb6251297304e76186b10a928d',
  'delivered',
  '2017-10-02 10:56:33',
  '2017-10-02 11:07:15',
  '2017-10-04 19:55:00',
  '2017-10-10 21:25:13',
  '2017-10-18 00:00:00'),
 (1,
  '53cdb2fc8bc7dce0b6741e2150273451',
  'b0830fb4747a6c6d20dea0b8c802d7ef',
  'delivered',
  '2018-07-24 20:41:37',
  '2018-07-26 03:24:27',
  '2018-07-26 14:31:00',
  '2018-08-07 15:27:45',
  '2018-08-13 00:00:00'),
 (2,
  '47770eb9100c2d0c44946d9cf07ec65d',
  '41ce2a54c0b003bf3443c3d931a367089',
  'delivered',
  '2018-08-08 08:38:49',
  '2018-08-08 08:55:23',
  '2018-08-08 13:50:00',
  '2018-08-17 18:06:29',
  '2018-09-04 00:00:00'),
 (3,
  '949d5b44dbf5de918fe9c16f97b45f8a',
  'f88197465ea7920adcdbec7375364d82',
  'delivered',
  '2017-11-18 19:28:06',
  '2017-11-18 19:45:59',
  '2017-11-22 13:39:59',
  '2017-12-02 00:28:42',
  '2017-12-15 00:00:00'),
 (4,
  'ad21c59c0840e6cb83a9ceb5573f8159',
  '8ab97904e6daea8866dbdbc4fb7aad2c',
  'delivered',
  '2018-02-13 21:18:39',
  '2018-02-13 22:20:29',
  '2018-02-14 19:46:34',
  '2018-02-16 18:17:02',
  '2018-02-26 00:00:00)]]
```

## Módulo 18 – Criando um banco de dados para nosso estudo

Agora, podemos utilizar nossa função, criada em módulos anteriores, para executar queries SQL em nosso banco de dados e continuar nossos estudos:

```
1 # Importando o sqlite e o pandas
2 import sqlite3
3 import pandas as pd
```

```
1 # Criando uma conexão
2 con = sqlite3.connect('vendas_db.db')
```

```
1 # Criando o cursor
2 cur = con.cursor()
```

```
1 # Utilizando a função que já criamos
2 def executa_consulta(consulta):
3     resultado = cur.execute(consulta).fetchall()
4     resultado = pd.DataFrame(resultado)
5     colunas = [i[0] for i in cur.description]
6     if resultado.shape[1] > 0:
7         resultado.columns = colunas
8     print(resultado.shape)
9     display(resultado.head(3))
10    return resultado
```

# Módulo 18 – Criando um banco de dados para nosso estudo

Primeiro, vamos conferir que todas as tabelas necessárias estão em nosso banco de dados:

1	# Verificando as tabelas existentes no banco de dados			
2	executa_consulta('SELECT * FROM sqlite_schema WHERE type = "table"')			
(9, 5)				
type	name	tbl_name	rootpage	sql
0	table	customers	customers	2 CREATE TABLE "customers" (\\n"index" INTEGER,\\n...
1	table	product_category_name_translation	product_category_name_translation	2644 CREATE TABLE "product_category_name_translatio...
2	table	order_reviews	order_reviews	2646 CREATE TABLE "order_reviews" (\\n"index" INTEGE...
3	table	geolocation	geolocation	6721 CREATE TABLE "geolocation" (\\n"index" INTEGER,...
4	table	order_items	order_items	21647 CREATE TABLE "order_items" (\\n"index" INTEGER,...
5	table	sellers	sellers	31198 CREATE TABLE "sellers" (\\n"index" INTEGER,\\n ...
6	table	products	products	31255 CREATE TABLE "products" (\\n"index" INTEGER,\\n ...
7	table	order_payments	order_payments	31949 CREATE TABLE "order_payments" (\\n"index" INTEG...
8	table	orders	orders	26182 CREATE TABLE "orders" (\\n"index" INTEGER,\\n ...

## Módulo 18 – Fazendo uma análise

Vamos analisar as avaliações. Podemos juntar as bases de avaliações, de ordens e de itens das ordens:

```
1 # Verificando a base de avaliações
2 avaliacoes = executa_consulta('SELECT * FROM order_reviews')
```

(99224, 8)

	index	review_id	order_id	review_score	review_comment_title	review_comment_message	review_creation_date	review_answer_timestamp
0	0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4	None	None	2018-01-18 00:00:00	2018-01-18 21:46:59
1	1	80e641a11e56f04c1ad469d5645fdfde	a548910a1c6147796b98fdf73dbeba33	5	None	None	2018-03-10 00:00:00	2018-03-11 03:05:13
2	2	228ce5500dc1d8e020d8d1322874b6f0	f9e4b658b201a9f2ecdecbb34bed034b	5	None	None	2018-02-17 00:00:00	2018-02-18 14:36:24

```
1 # Já trazendo a avaliação e o pedido
```

```
2 avaliacoes_pedido = executa_consulta('SELECT * FROM order_reviews ore \
3   LEFT JOIN orders o \
4     ON ore.order_id = o.order_id \
5   LEFT JOIN order_items oi \
6     ON o.order_id = oi.order_id')
```

(113131, 25)

	index	review_id	order_id	review_score	review_comment_title	review_comment_message	review_creation_date	review_answer_timestamp	index	order_id	...	order_delivered_customer
0	0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4	None	None	2018-01-18 00:00:00	2018-01-18 21:46:59	1871	73fc7af87114b39712e6da79b0a377eb	...	2018-01-17
1	0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4	None	None	2018-01-18 00:00:00	2018-01-18 21:46:59	1871	73fc7af87114b39712e6da79b0a377eb	...	2018-01-17
2	1	80e641a11e56f04c1ad469d5645fdfde	a548910a1c6147796b98fdf73dbeba33	5	None	None	2018-03-10 00:00:00	2018-03-11 03:05:13	90488	a548910a1c6147796b98fdf73dbeba33	...	2018-03-09 :

3 rows x 25 columns



## Módulo 18 – Fazendo uma análise

Com o resultado, podemos verificar a média de preço para cada avaliação.

Ao lado, vemos como realizar com Pandas e com SQL, a partir de funções de agregação.

Observe que a média de preço é maior para a menor nota de avaliação.

```
1 # Fazendo a média de venda por avaliação  
2 avaliacoes_pedido.groupby('review_score')['price'].mean()
```

```
review_score  
1 127.350068  
2 115.849138  
3 110.059704  
4 118.602628  
5 121.219825  
Name: price, dtype: float64
```

```
1 # Executando essa consulta agora no SQL  
2 avaliacoes_pedido_mean = executa_consulta('SELECT review_score, AVG(price) as media_preco \  
FROM order_reviews ore \  
LEFT JOIN orders o \  
ON ore.order_id = o.order_id \  
LEFT JOIN order_items oi \  
ON o.order_id = oi.order_id \  
GROUP BY review_score')
```

```
(5, 2)  
review_score  media_preco  
---  
0            1    127.350068  
1            2    115.849138  
2            3    110.059704
```

```
1 avaliacoes_pedido_mean
```

```
review_score  media_preco  
---  
0            1    127.350068  
1            2    115.849138  
2            3    110.059704  
3            4    118.602628  
4            5    121.219825
```

## Módulo 18 – Cenário de estudos

Agora, será que esta métrica é relevante? Quais os problemas que o cliente busca resolver? E quais os problemas que o cliente ainda não conseguiu visualizar em seus próprios dados? Estas são algumas perguntas que nós como cientistas de dados devemos buscar responder.

Além disso, devemos apresentar de forma adequada os resultados.

Pare termos um cenário, vamos considerar que a tarefa é ajudar a área de negócios a **montar uma apresentação para a diretoria** para provar a **necessidade de investimento em uma área de melhoria da experiência do cliente ao ter um atraso na entrega**.

Algumas considerações importantes:

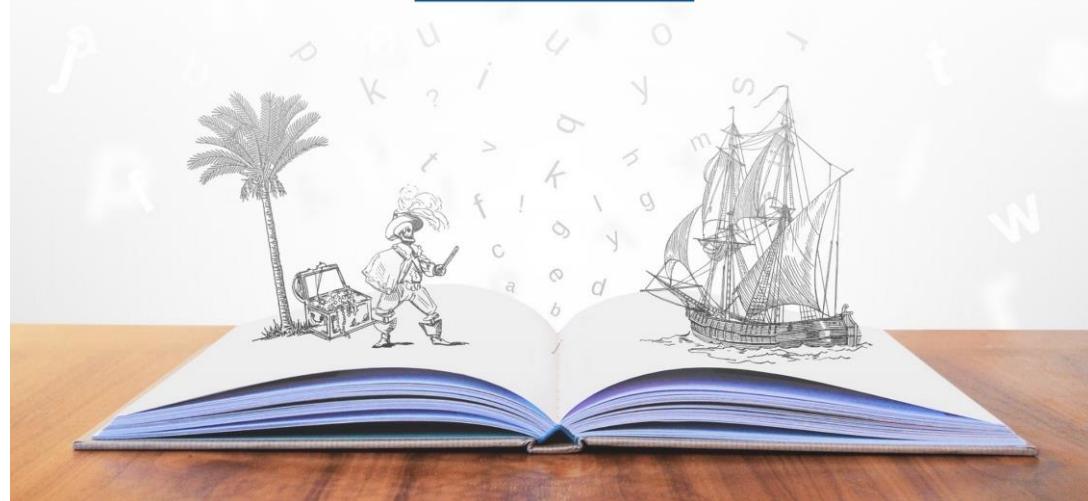
- O *time de logística* não considera que o atraso na entrega é um problema relevante, e falou que, em média, as entregas estão sendo feitas 10 dias antes do prazo combinado;
- Não é desejado a previsão de uma entrega atrasada, apenas a exposição que esse é um problema que pode impactar os clientes;
- Não queremos uma abordagem de “nenhuma entrega pode atrasar”. Vamos seguir a linha de “uma entrega pode atrasar. Como eu posso melhorar a experiência do cliente caso isso aconteça?”

# Módulo 18 – Storytelling com dados

No material desta apostila, vamos focar no código por trás da resolução do problema.

Mas, é **muito importante** que você veja os vídeos e a apostila sobre *Storytelling com dados* disponibilizada na plataforma para aprender a apresentar os resultados de uma forma atraente.

## Criando uma história **com seus dados**



### Como contar uma boa história?



Apresentações geralmente são chatas e, muitas vezes, as pessoas já entram nas reuniões desanimadas por pensarem que vai ser só mais uma reunião que poderia ser um e-mail



Alguns gráficos são tão complexos que o público não consegue entender nem mesmo o que aquele gráfico representa (e muito menos a informação que ele deveria passar)



Você vai estar, o tempo todo, disputando a atenção com notificação no celular, outras demandas do público, novo e-mail e até mesmo pensamentos aleatórios



Algumas vezes recebemos tanta informação e de forma tão desordenada que no final acaba que não absorvemos nada e nem chegamos a nenhuma conclusão

### A estrutura de uma história

Uma história pode ser dividida em 3 atos:



#### PREPARAÇÃO

Você vai mostrar ao público o motivo de valer a pena ficar ali, vai introduzir as informações mais relevantes e explicar o contexto



#### CONFLITO

Qual o problema que queremos resolver ou a oportunidade que temos de melhoria? O que justifica isso que estamos mostrando?



#### SOLUÇÃO

Apresentar as ideias que serão usadas para resolver o conflito e chamar para a ação ou gerar uma discussão

## Módulo 18 – Preparando nosso ambiente

O início já deve estar começando a ficar enraizado neste momento. Importar as bibliotecas, criar a conexão com o banco de dados e o respectivo cursor, e definir a função que irá auxiliar com as queries SQL:

```
[1]: 1 # Importando o sqlite3
      2 import sqlite3

[2]: 1 # Importando o pandas
      2 import pandas as pd

[3]: 1 # Criando uma conexão
      2 con = sqlite3.connect('vendas_db.db')

[4]: 1 # Então criando o cursor
      2 cur = con.cursor()

[5]: 1 # Usando a função python que já havíamos criado
      2 def executa_consulta(consulta):
      3     resultado = cur.execute(consulta).fetchall()
      4     resultado = pd.DataFrame(resultado)
      5     colunas = [i[0] for i in cur.description]
      6     if resultado.shape[1] > 0:
      7         resultado.columns = colunas
      8         print(resultado.shape)
      9         display(resultado.head(3))
     10     return resultado
```

## Módulo 18 – Validando o problema

O primeiro passo é validar que o problema apontado é real. Ou seja, que existem atrasos.

Primeiro, vamos selecionar as colunas que contêm os dados de data e hora:

```
: 1 # Podemos visualizar a tabela de pedidos
 2 pedidos = executa_consulta('SELECT order_id, \
 3                             order_purchase_timestamp, \
 4                             order_delivered_customer_date, \
 5                             order_estimated_delivery_date \
 6                         FROM orders')
(99441, 4)
```

	order_id	order_purchase_timestamp	order_delivered_customer_date	order_estimated_delivery_date
0	e481f51cbdc54678b7cc49136f2d6af7	2017-10-02 10:56:33	2017-10-10 21:25:13	2017-10-18 00:00:00
1	53cdb2fc8bc7dce0b6741e2150273451	2018-07-24 20:41:37	2018-08-07 15:27:45	2018-08-13 00:00:00
2	47770eb9100c2d0c44946d9cf07ec65d	2018-08-08 08:38:49	2018-08-17 18:06:29	2018-09-04 00:00:00

```
: 1 # Verificando as informações dessa base
 2 pedidos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   order_id         99441 non-null   object 
 1   order_purchase_timestamp  99441 non-null   object 
 2   order_delivered_customer_date 96476 non-null   object 
 3   order_estimated_delivery_date 99441 non-null   object 
dtypes: object(4)
memory usage: 3.0+ MB
```

## Módulo 18 – Validando o problema

Como já visto antes no módulo, precisamos deixar as colunas com o tipo e formato adequados para data e hora.

Com os tipos corretos, podemos fazer a diferença em dias das datas (estimada menos entregue, de forma que um valor positivo significa entrega antes do prazo) e calcular a média de atraso.

Como temos um valor positivo, temos uma média de cerca de 11 dias de entrega antes do prazo, compatível com a informação passada pelo time de logística.

```
1 # Ajustando o tipo das colunas de data
2 pedidos['order_purchase_timestamp'] = pd.to_datetime(pedidos['order_purchase_timestamp'],format='%Y-%m-%d %H:%M:%S')

1 lista_colunas = ['order_purchase_timestamp','order_delivered_customer_date','order_estimated_delivery_date']
2
3 for i in lista_colunas:
4     pedidos[i] = pd.to_datetime(pedidos[i],format='%Y-%m-%d %H:%M:%S')
```

```
1 import datetime as dt

1 # Calculando a diferença entre a data estimada e a data entregue
2 pedidos['atraso'] = (pedidos.order_estimated_delivery_date.dt.date - pedidos.order_delivered_customer_date.dt.date).dt.days

1 # Visualizando novamente a base
2 pedidos.head(3)

order_id  order_purchase_timestamp  order_delivered_customer_date  order_estimated_delivery_date  atraso
0  e481f51cbdc54678b7cc49136f2d6af7    2017-10-02 10:56:33          2017-10-10 21:25:13        2017-10-18   8.0
1  53cdb2fc8bc7dce0b6741e2150273451    2018-07-24 20:41:37          2018-08-07 15:27:45        2018-08-13   6.0
2  47770eb9100c2d0c44946d9cf07ec65d    2018-08-08 08:38:49          2018-08-17 18:06:29        2018-09-04  18.0

1 # Verificando a média dos atrasos
2 pedidos.atraso.mean()

11.876881296902857
```

## Módulo 18 – Validando o problema

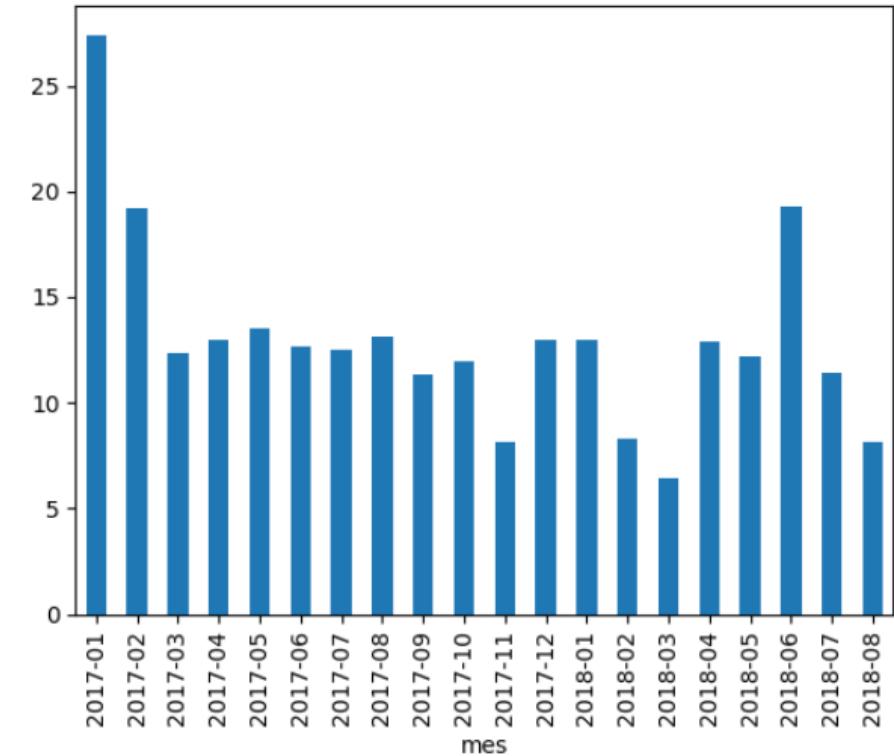
Uma hipótese a ser verificada é se o atraso é sazonal, ou seja, se há alguns meses com mais atraso do que outros.

O método `to_period` permite agrupar dados temporais por períodos. O M é para agrupar por meses.

Com os dados agrupados, podemos visualizar com um gráfico a média de atrasos por mês de compra:

```
1 # Verificando o período (mês/ano) da compra  
2 pedidos['mes'] = pedidos.order_purchase_timestamp.dt.to_period("M")
```

```
1 # Verificando o atraso por mês  
2 pedidos.groupby('mes')['atraso'].mean().plot.bar();
```



## Módulo 18 – Validando o problema

Sempre que se usa média, devemos ter cuidado de verificar se ela é a melhor métrica. Isto porque a média é muito sensível a extremos.

Vamos, então, visualizar a média juntamente com os valores mínimos e máximos por mês. Perceba que há entregas muito adiantadas e entregas muito atrasadas, de forma que a média não necessariamente é uma métrica informativa.

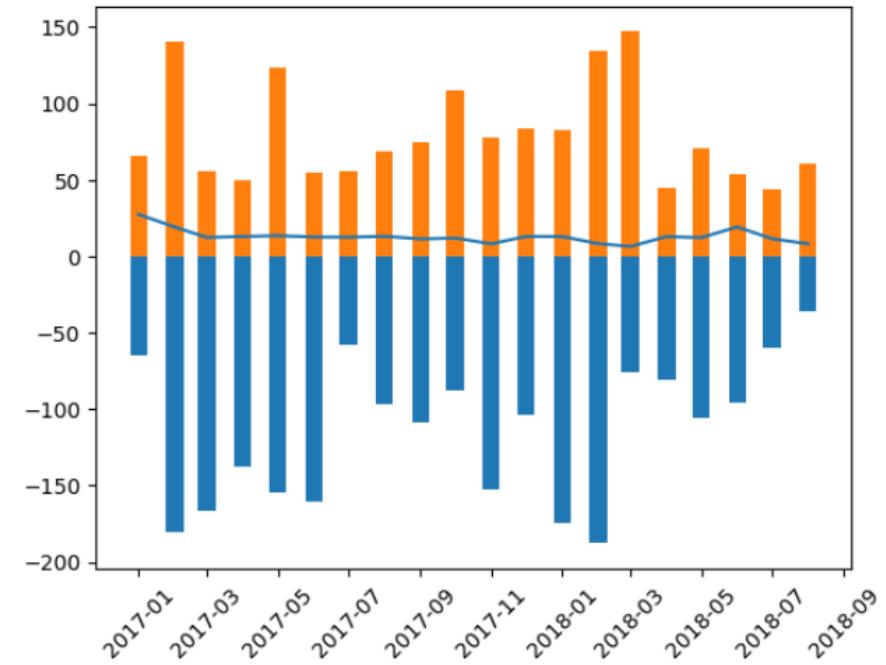
```
1 # Além da média, vamos analisar o máximo e o mínimo
2 media = pedidos.groupby('mes')['atraso'].mean()
3 minimo = pedidos.groupby('mes')['atraso'].min()
4 maximo = pedidos.groupby('mes')['atraso'].max()

1 import matplotlib.pyplot as plt
2 import numpy as np

1 media.values
```

array([27.408 , 19.1954023 , 12.32717989, 12.97177594, 13.49788434,
 12.64433812, 12.47081612, 13.09754352, 11.34819277, 11.92340331,
 8.12349067, 13.0085253 , 12.9343613 , 8.28767541, 6.42767385,
 12.89511621, 12.19099126, 19.24409449, 11.44152047, 8.16217918])

```
1 # Plotando graficamente
2 fig, ax = plt.subplots()
3
4 ax.plot(media.index.to_timestamp(), media.values)
5 ax.bar(minimo.index.to_timestamp(), minimo.values, width=15)
6 ax.bar(maximo.index.to_timestamp(), maximo.values, width=15)
7
8 plt.xticks(rotation=45)
9
10 plt.show()
```



## Módulo 18 – Validando o problema

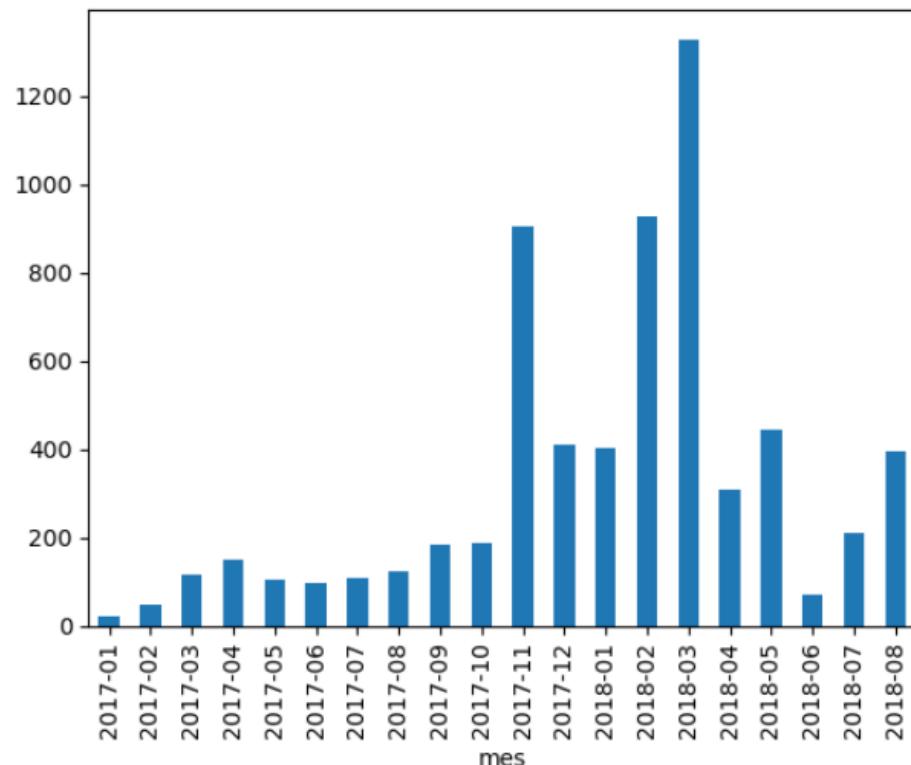
Podemos, também, verificar quantos pedidos efetivamente atrasaram. Perceba que estamos usando uma função anônima, Lambda em Python. Ou seja, uma função não declarada anteriormente, passada diretamente para o método `apply` do Pandas.

Como definimos que atraso recebe valor 1 e no prazo valor 0, uma simples soma fornece o número de pedidos atrasados em um mês, o que está representado no gráfico. Observe que parece haver uma crescente de atrasos, em especial a partir de novembro de 2017, mas que talvez esteja sob resolução a partir de junho de 2018.

E se analisarmos a quantidade de pedidos com atraso?

```
1 # Criando uma marcação para verificar se o pedido atrasou ou não  
2 pedidos['flag_atraso'] = pedidos.atraso.apply(lambda x: 1 if x < 0 else 0)
```

```
1 # E contando, mensalmente, quantos pedidos atrasaram  
2 pedidos.groupby('mes')['flag_atraso'].sum().plot.bar();
```



## Módulo 18 – Validando o problema

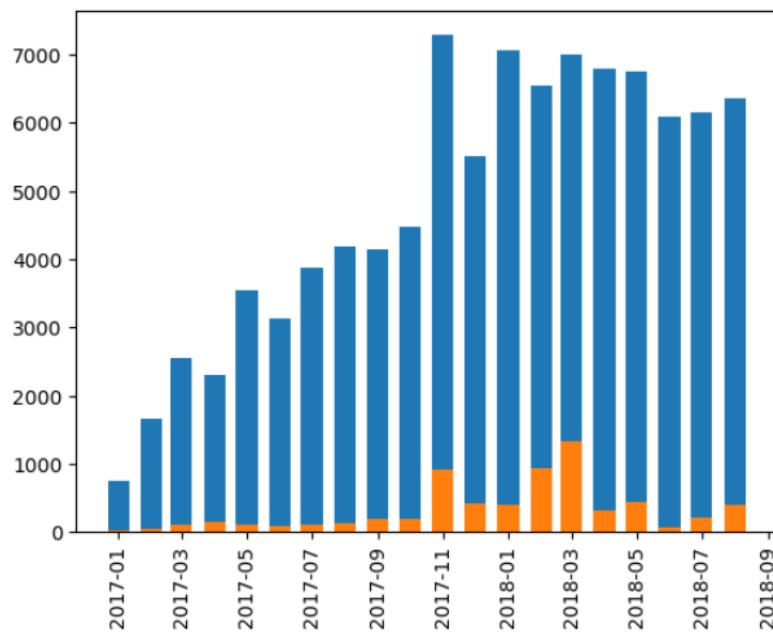
É interessante termos uma noção se a quantidade de atrasos é significativa frente ao total de pedidos.

Perceba que o mês de novembro de 2017 apresenta um grande aumento de pedidos, fazendo sentido que também tenha havido um aumento de atrasos. A questão é se o aumento de atrasos foi proporcional ao aumento de pedidos ou não.

Na próxima página, veremos como calcular a porcentagem em cada mês.

```
1 # Analisando a relação entre o número de pedidos feitos e atrasados
2 pedidos_atrasados = pedidos.groupby('mes')[['flag_atraso']].sum()
3 contagem_pedidos = pedidos.groupby('mes')[['flag_atraso']].count()

1 # Visualizando graficamente
2 fig, ax = plt.subplots()
3
4 ax.bar(contagem_pedidos.index.to_timestamp(), contagem_pedidos.values,width=20)
5 ax.bar(pedidos_atrasados.index.to_timestamp(), pedidos_atrasados.values,width=20)
6
7 plt.xticks(rotation=90)
8
9 plt.show()
```



## Módulo 18 – Validando o problema

Veja como os meses destacados nas páginas anteriores realmente apresentaram uma proporção de atrasos maior que os demais meses. E o mês de junho de 2018 foi o mês com menor proporção de atrasos.

Estas informações podem ajudar a equipe de logística, já que esta deve ter o controle de toda a parte de entrega e pode avaliar o quanto sobrecarregada estava esta parte em cada período e adaptar de acordo com flutuações sazonais.

```
1 (pedidos_atrasados / contagem_pedidos) * 100
```

mes	
2017-01	2.933333
2017-02	2.964307
2017-03	4.556167
2017-04	6.556665
2017-05	2.990127
2017-06	3.030303
2017-07	2.789256
2017-08	2.909611
2017-09	4.385542
2017-10	4.175971
2017-11	12.403952
2017-12	7.455106
2018-01	5.700948
2018-02	14.139719
2018-03	18.963301
2018-04	4.501324
2018-05	6.563935
2018-06	1.164698
2018-07	3.378817
2018-08	6.188002

Freq: M, Name: flag\_atraso, dtype: float64

```
1 pedidos_atraso = pedidos_atrasados
2 contagem = contagem_pedidos
3 percentual = round((pedidos_atrasados / contagem_pedidos) * 100, 1)
```

## Módulo 18 – Validando o problema

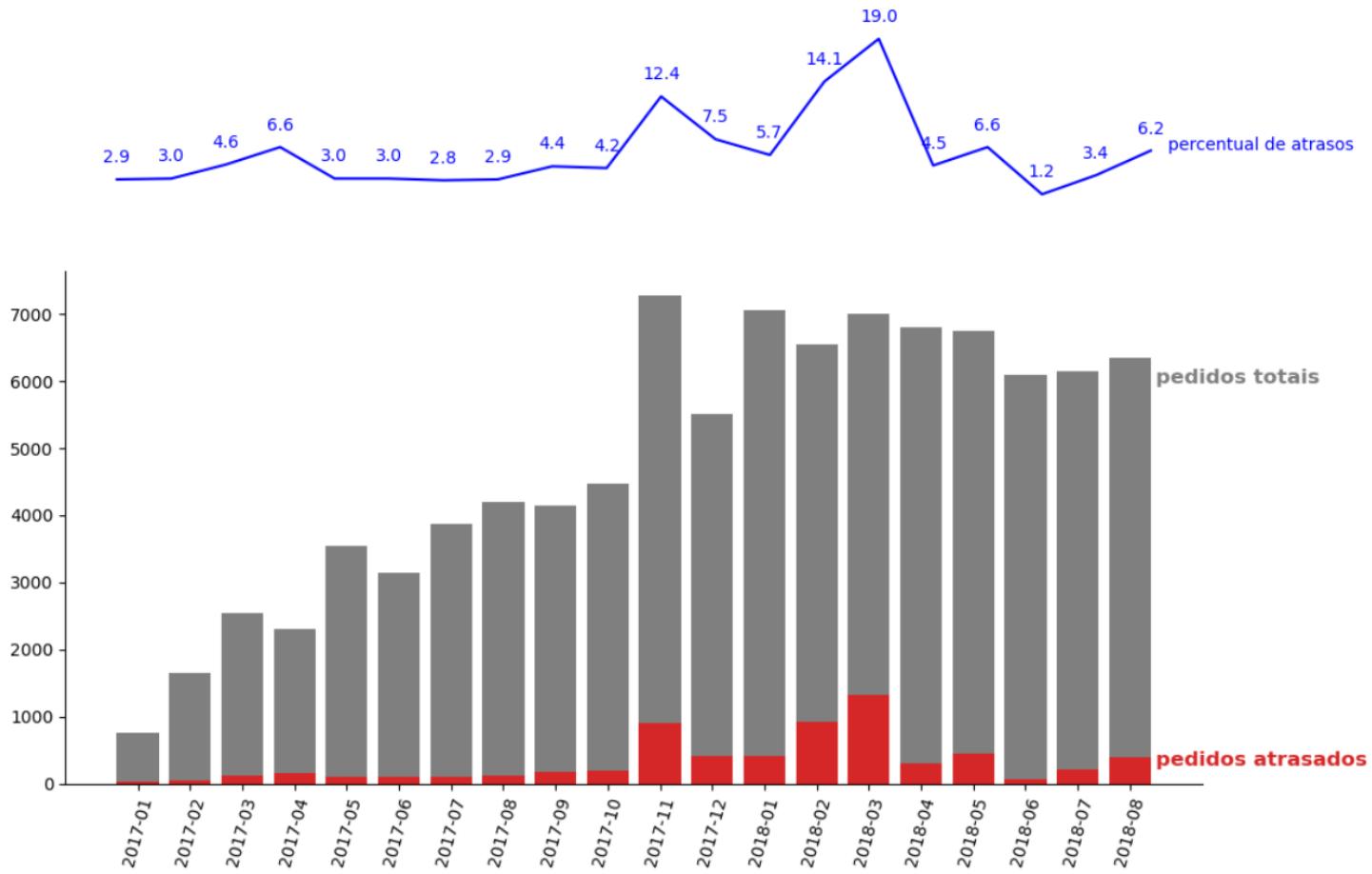
Podemos aglutinar todas estas informações em um único gráfico. Recomendamos que veja o vídeo para entender cada parte, mas em suma:

- Criamos dois eixos com o método `subplots`
- Tornamos invisíveis as linhas de contorno dos eixos.
- Adicionamos anotações para explicar as barras e as linhas.
- Adicionamos os valores de porcentagem em cada ponto do gráfico de linha.

```
1 # Melhorando a visualização
2 fig, ax = plt.subplots(nrows=2,figsize=(12,8),gridspec_kw={'height_ratios': [1,3]})
3
4 x = np.arange(0,len(contagem))
5
6 ax[1].bar(x, contagem.values, linewidth=2.0,color='tab:gray')
7 ax[1].bar(x, pedidos_atraso.values, linewidth=2.0,color='tab:red')
8 ax[0].plot(x,percentual.values,c='b')
9
10 ax[0].xaxis.set_visible(False)
11 ax[0].yaxis.set_visible(False)
12 for i in x:
13     ax[0].annotate(percentual.values[i],(i,percentual.values[i]),ha="center",xytext=(0,10),textcoords="offset points",c='b')
14     ax[0].spines['top'].set_visible(False)
15     ax[0].spines['left'].set_visible(False)
16     ax[0].spines['right'].set_visible(False)
17     ax[0].spines['bottom'].set_visible(False)
18     ax[0].annotate('percentual de atrasos',(x[-1],percentual.values[-1]),ha="left",xytext=(10,0),textcoords="offset points",c='b')
19
20 ax[1].set_xticks(x)
21 ax[1].set_xticklabels(contagem.index,rotation=75)
22 ax[1].spines['top'].set_visible(False)
23 ax[1].spines['right'].set_visible(False)
24 ax[1].annotate('pedidos totais',(x[-1],contagem.values[-1]),ha="left",xytext=(15,-15),
25                 textcoords="offset points",c='tab:gray',fontsize=12,fontweight='bold')
26 ax[1].annotate('pedidos atrasados',(x[-1],pedidos_atraso.values[-1]),ha="left",xytext=(15,-5),
27                 textcoords="offset points",c='tab:red',fontsize=12,fontweight='bold')
28
29 plt.savefig('pedidos_atrasados',transparent=True)
```

## Módulo 18 – Validando o problema

Eis o gráfico resultante. Observe como, em uma apresentação, todas as informações relevantes estão presentes de forma clara.



## Módulo 18 – Relação entre atraso e nota de avaliação

Uma segunda análise que podemos fazer é se há reflexo do atraso nas avaliações dos pedidos. Veja que parte significativa das avaliações é de notas 4 e 5, mas há 11,5 % de notas 1:

```
1 # Vamos começar visualizando a tabela de avaliações
2 avaliacoes = executa_consulta('SELECT * FROM order_reviews')
```

(99224, 8)

	index	review_id	order_id	review_score	review_comment_title	review_comment_message	review_creation_date	review_answer_timestamp
0	0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4	None	None	2018-01-18 00:00:00	2018-01-18 21:46:59
1	1	80e641a11e56f04c1ad469d5645fdfde	a548910a1c6147796b98fdf73dbeba33	5	None	None	2018-03-10 00:00:00	2018-03-11 03:05:13
2	2	228ce5500dc1d8e020d8d1322874b6f0	f9e4b658b201a9f2ecdecbb34bed034b	5	None	None	2018-02-17 00:00:00	2018-02-18 14:36:24

```
1 # Verificando a quantidade de avaliações em cada uma das notas
2 avaliacoes.review_score.value_counts().sort_index()
```

```
1 11424
2 3151
3 8179
4 19142
5 57328
Name: review_score, dtype: int64
```

```
1 # Visualizando em percentual
2 round((avaliacoes.review_score.value_counts().sort_index()/avaliacoes.shape[0])*100,1)
```

```
1 11.5
2 3.2
3 8.2
4 19.3
5 57.8
Name: review_score, dtype: float64
```

## Módulo 18 – Relação entre atraso e nota de avaliação

Vamos fazer um JOIN das tabelas de avaliações e de ordens, pegando as colunas de interesse para a análise.

Observe que há pedidos sem avaliação:

1	# Podemos pegar apenas as colunas que quisermos utilizar
2	avaliacao_ordem = executa_consulta('SELECT \n    o.order_id \n    ,o.order_purchase_timestamp \n    ,o.order_delivered_customer_date \n    ,o.order_estimated_delivery_date \n    ,ore.review_score \n    FROM orders o \n    LEFT JOIN order_reviews ore \n    ON o.order_id = ore.order_id')
3	(99992, 5)
4	order_id order_purchase_timestamp order_delivered_customer_date order_estimated_delivery_date review_score
5	0 e481f51cbdc54678b7cc49136f2d6af7 2017-10-02 10:56:33 2017-10-10 21:25:13 2017-10-18 00:00:00 4.0
6	1 53cdb2fc8bc7dce0b6741e2150273451 2018-07-24 20:41:37 2018-08-07 15:27:45 2018-08-13 00:00:00 4.0
7	2 47770eb9100c2d0c44946d9cf07ec65d 2018-08-08 08:38:49 2018-08-17 18:06:29 2018-09-04 00:00:00 5.0
8	# Todos os pedidos possuem avaliação?
9	avaliacao_ordem[avaliacao_ordem.review_score.isnull()]
10	order_id order_purchase_timestamp order_delivered_customer_date order_estimated_delivery_date review_score
11	16 403b97836b0c04a622354cf531062e5f 2018-01-02 19:00:43 2018-01-20 01:38:59 2018-02-06 00:00:00 NaN
12	154 6942b8da583c2f9957e990d028607019 2018-01-10 11:33:07 None 2018-02-07 00:00:00 NaN
13	311 4906eeadde5f70b308c20c4a8f20be02 2017-12-08 04:45:26 2018-01-09 18:04:58 2018-01-03 00:00:00 NaN
14	382 b7a4a9ecb1cd3ef6a3e36a48e200e3be 2017-05-19 18:13:54 2017-06-08 07:53:42 2017-06-16 00:00:00 NaN
15	390 59b32faedc12322c672e95ec3716d614 2018-06-27 11:10:11 2018-07-06 16:37:36 2018-07-26 00:00:00 NaN
16	... ... ... ... ... ...
17	99242 0c384d67524b5b92aa2fa6c8baa9a983 2017-06-05 19:20:11 2017-06-13 14:09:21 2017-06-27 00:00:00 NaN
18	99327 906a6b0a96d89ee226e4977e99b80b9e 2017-08-28 15:14:21 2017-09-05 19:47:44 2017-09-18 00:00:00 NaN
19	99554 5333db16fe357175d39c82840dd3269d 2018-03-10 18:18:20 2018-04-03 15:32:52 2018-03-29 00:00:00 NaN
20	99684 2f2df159f26ddb73d55ee72372200d3e 2017-07-17 01:19:50 2017-07-26 09:44:00 2017-08-09 00:00:00 NaN
21	99796 dff2b9b8d7cfc595836945e1443789c3 2018-07-16 12:59:02 2018-07-20 20:41:32 2018-08-07 00:00:00 NaN
22	768 rows × 5 columns

# Módulo 18 – Relação entre atraso e nota de avaliação

É sempre importante verificar a consistência dos dados. No caso, verificamos que há ordens que possuem mais de uma avaliação.

Abaixo, vemos um destes casos. Aparentemente, o cliente deu inicialmente uma nota 3 e depois ajustou para 5.

```
1 # Verificando essa primeira ordem na tabela de pedidos
2 pedidos[pedidos.order_id == 'c88b1d1b157a9999ce368f218a407141']
```

index	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date	
37179	37179	c88b1d1b157a9999ce368f218a407141	ae0fb7b01c548c4fd30f19f55453ec4a	delivered	2017-07-17 22:44:23	2017-07-18 22:50:12	2017-07-20 15:27:41	2017-07-21 17:06:30	2017-08-01 00:00:00

```
1 # Agora verificando na tabela de avaliações
2 avaliacoes[avaliacoes.order_id == 'c88b1d1b157a9999ce368f218a407141']
```

index	review_id	order_id	review_score	review_comment_title	review_comment_message	review_creation_date	review_answer_timestamp
1985	1985	ffb8cff872a625632ac983eb1f88843c	3	None	None	2017-07-22 00:00:00	2017-07-26 13:41:07
82525	82525	202b5f44d09cd3fc0d6bd12f01b044c	5	None	None	2017-07-22 00:00:00	2017-07-26 13:40:22
89360	89360	fb96ea2ef8cce1c888f4d45c8e22b793	5	None	None	2017-07-21 00:00:00	2017-07-26 13:45:15

```
1 avaliacao_ordem[avaliacao_ordem.order_id == 'c88b1d1b157a9999ce368f218a407141']
```

order_id	order_purchase_timestamp	order_delivered_customer_date	order_estimated_delivery_date	review_score
37365	c88b1d1b157a9999ce368f218a407141	2017-07-17 22:44:23	2017-07-21 17:06:30	2017-08-01 00:00:00
37366	c88b1d1b157a9999ce368f218a407141	2017-07-17 22:44:23	2017-07-21 17:06:30	2017-08-01 00:00:00
37367	c88b1d1b157a9999ce368f218a407141	2017-07-17 22:44:23	2017-07-21 17:06:30	2017-08-01 00:00:00

```
1 # Visualizando se alguma ordem apareceu mais de 1 vez
2 avaliacao_ordem.order_id.value_counts()
```

```
8e17072ec97ce29f0e1f111e598b0c85      3
03c939fd7fd3b38f8485a0f95798f1f6      3
c88b1d1b157a9999ce368f218a407141      3
df56136b8031ecd28e200bb18e6ddb2e      3
95442deb81a5d91c97c0df96b431634a      2
..                                         ..
94a1020970476388adf12c46628499be      1
b266c12d9bececa3942c1bef84f13716      1
e82e0697786dccef38c215ec9d2dc308a      1
e940e33cefca8e48b759474760c2fb41      1
66dea50a8b16d9b4dee7af250b4be1a5      1
Name: order_id, Length: 99441, dtype: int64
```

## Módulo 18 – Relação entre atraso e nota de avaliação

Agora, temos que resolver o que fazer nestes casos. Podemos, por exemplo, avaliar se faz sentido usar a média das avaliações ou a avaliação máxima nestes casos onde há mais de uma nota. Vamos trazer as duas agregações em nossa query:

```
1 # Primeiramente fazendo o join entre as tabelas
2 avaliacao_ordem = executa_consulta('SELECT \
3     o.order_id \
4     ,o.order_purchase_timestamp \
5     ,o.order_delivered_customer_date \
6     ,o.order_estimated_delivery_date \
7     ,AVG(ore.review_score) as media_nota \
8     ,MAX(ore.review_score) as maximo_nota \
9     FROM orders o \
10    LEFT JOIN order_reviews ore \
11      ON o.order_id = ore.order_id \
12    WHERE ore.review_score IS NOT NULL \
13    GROUP BY \
14        o.order_id \
15        ,o.order_purchase_timestamp \
16        ,o.order_delivered_customer_date \
17        ,o.order_estimated_delivery_date')
```

(98673, 6)

	order_id	order_purchase_timestamp	order_delivered_customer_date	order_estimated_delivery_date	media_nota	maximo_nota
0	00010242fe8c5a6d1ba2dd792cb16214	2017-09-13 08:59:02	2017-09-20 23:43:48	2017-09-29 00:00:00	5.0	5
1	00018f77f2f0320c557190d7a144bdd3	2017-04-26 10:53:06	2017-05-12 16:04:24	2017-05-15 00:00:00	4.0	4
2	000229ec398224ef6ca0657da4fc703e	2018-01-14 14:33:31	2018-01-22 13:19:16	2018-02-05 00:00:00	5.0	5

## Módulo 18 – Relação entre atraso e nota de avaliação

Agrupando os dados, vemos que são poucos os registros onde o máximo difere da média. Desta forma, posso considerar o máximo sem muito impacto:

```
1 # Analisando o impacto de usar o máximo ao invés da média
2 avaliacao_ordem.groupby(['media_nota','maximo_nota'])['order_id'].count()
```

media_nota	maximo_nota	count
1.0	1.0	11316
1.5	2.0	8
2.0	2.0	3113
	3.0	12
2.5	3.0	10
	4.0	24
3.0	3.0	8094
	4.0	9
	5.0	33
3.3	4.0	1
3.5	4.0	17
	5.0	8
4.0	4.0	18993
	5.0	25
4.3	5.0	1
4.5	5.0	54
5.0	5.0	56955
		Name: order_id, dtype: int64

## Módulo 18 – Relação entre atraso e nota de avaliação

Assim, temos:

```
1 # Vamos considerar uma única coluna de nota  
2 avaliacao_ordem['nota'] = avaliacao_ordem.maximo_nota
```

```
1 # E apagar as outras colunas para evitar confusão  
2 avaliacao_ordem = avaliacao_ordem.drop(['media_nota','maximo_nota'],axis=1)
```

```
1 avaliacao_ordem.head(3)
```

	order_id	order_purchase_timestamp	order_delivered_customer_date	order_estimated_delivery_date	nota
0	00010242fe8c5a6d1ba2dd792cb16214	2017-09-13 08:59:02	2017-09-20 23:43:48	2017-09-29	5.0
1	00018f77f2f0320c557190d7a144bdd3	2017-04-26 10:53:06	2017-05-12 16:04:24	2017-05-15	4.0
2	000229ec398224ef6ca0657da4fc703e	2018-01-14 14:33:31	2018-01-22 13:19:16	2018-02-05	5.0

## Módulo 18 – Relação entre atraso e nota de avaliação

Agora, podemos finalmente responder nossa pergunta. Observe que a média de notas é bem menor quando há atraso (flag com valor 1):

```
1 import datetime as dt
1 # Calculando o atraso na entrega
2 avaliacao_ordem['atraso'] = (avaliacao_ordem.order_estimated_delivery_date.dt.date - avaliacao_ordem.order_delivered_customer_date.dt.date).dt.days
1 # Marcando entregas que tiveram atraso com uma marcação (flag)
2 avaliacao_ordem['flag_atraso'] = avaliacao_ordem.atraso.apply(lambda x: 1 if x < 0 else 0)
1 # Verificando novamente a base
2 avaliacao_ordem.head(3)
```

	order_id	order_purchase_timestamp	order_delivered_customer_date	order_estimated_delivery_date	nota	atraso	flag_atraso
0	00010242fe8c5a6d1ba2dd792cb16214	2017-09-13 08:59:02	2017-09-20 23:43:48	2017-09-29	5.0	9.0	0
1	00018f77f2f0320c557190d7a144bdd3	2017-04-26 10:53:06	2017-05-12 16:04:24	2017-05-15	4.0	3.0	0
2	000229ec398224ef6ca0657da4fc703e	2018-01-14 14:33:31	2018-01-22 13:19:16	2018-02-05	5.0	14.0	0

```
1 # Verificando a nota quando existe atraso e quando não existe atraso
2 avaliacao_ordem.groupby('flag_atraso')['nota'].mean()
```

```
flag_atraso
0    4.214376
1    2.274052
Name: nota, dtype: float64
```

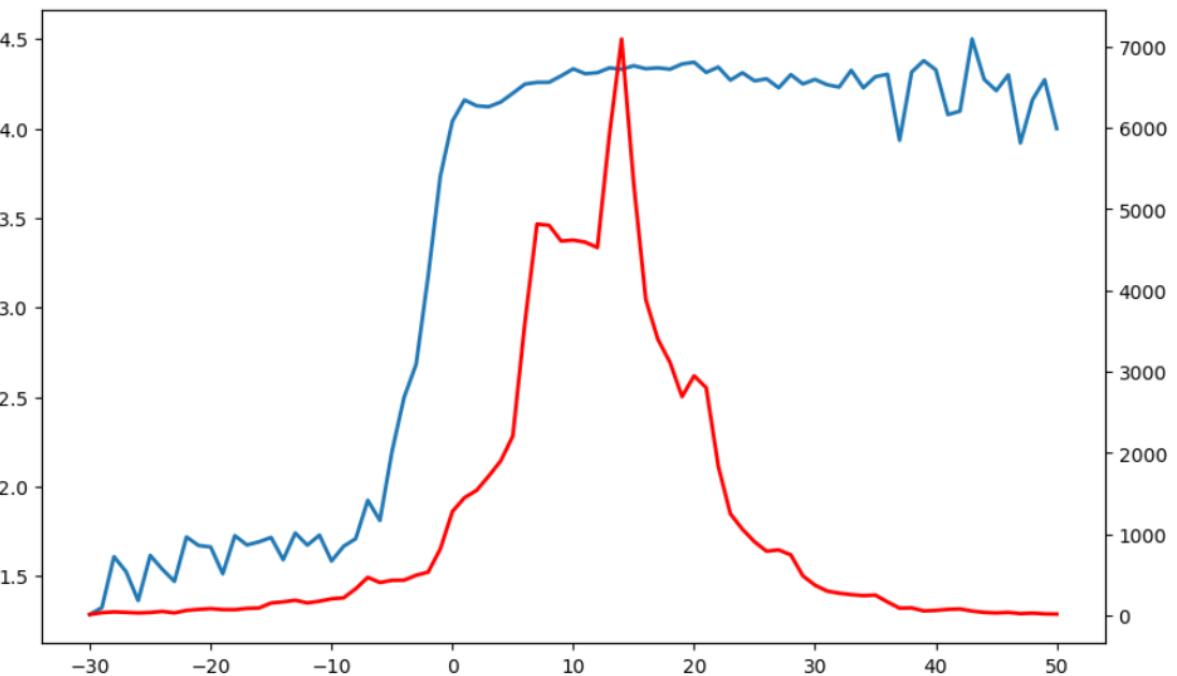
## Módulo 18 – Relação entre atraso e nota de avaliação

Graficamente, podemos mostrar a relação entre contagem de avaliações por dia, em vermelho, e média das notas, em azul.

Veja que quando há poucas avaliações, a média flutua mais, já que uma nota muito alta ou muito baixa terá grande influência no total de avaliações. Quando há muitas avaliações, a curva de média tende a ser mais suave.

Observe, também, que há muito mais avaliações entre 0 e 20 dias de antecedência na entrega

```
1 # Melhorando o gráfico e visualizando a quantidade de registros em cada dia
2 fig, ax = plt.subplots(figsize=(10,6))
3
4 base2 = avaliacao_ordem[(avaliacao_ordem.atraso >= -30) & (avaliacao_ordem.atraso <= 50)]
5
6 media_notas = base2.groupby('atraso')['nota'].mean()
7 qtd_avaliacoes = base2.groupby('atraso')['nota'].count()
8
9 ax2 = ax.twinx()
10
11 ax.plot(media_notas.index, media_notas.values, linewidth=2.0)
12 ax2.plot(qtd_avaliacoes.index, qtd_avaliacoes.values, linewidth=2.0,c='r')
13
14 plt.show()
```



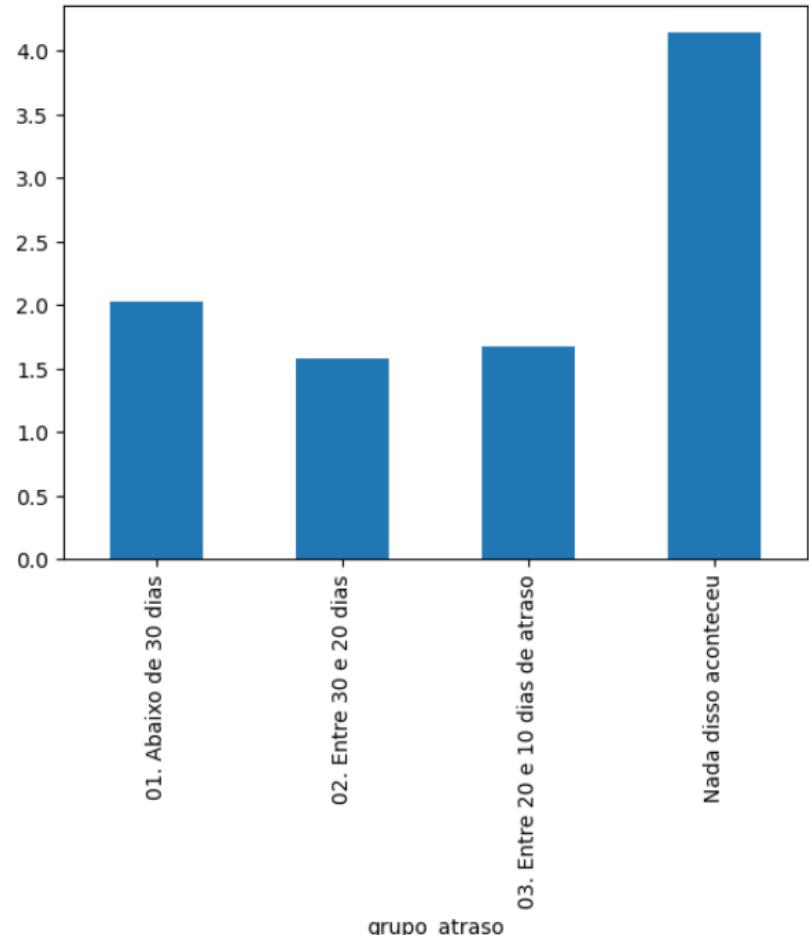
## Módulo 18 – Relação entre atraso e nota de avaliação

Podemos criar uma função para criar faixas de atraso e verificar como isto afeta a média.

```
1 # Será que podemos agrupar os atrasos?  
2 def agrupa_atraso(atraso):  
3     if atraso <= -30:  
4         return '01. Abaixo de 30 dias'  
5     elif atraso <= -20:  
6         return '02. Entre 30 e 20 dias'  
7     elif atraso <= -10:  
8         return '03. Entre 20 e 10 dias de atraso'  
9     else:  
10        return 'Nada disso aconteceu'
```

```
1 # Aplicando a função  
2 avaliacao_ordem['grupo_atraso'] = avaliacao_ordem.atraso.apply(agrupa_atraso)
```

```
1 # Verificando a média das notas por faixa de atraso  
2 avaliacao_ordem.groupby('grupo_atraso')['nota'].mean().plot.bar();
```



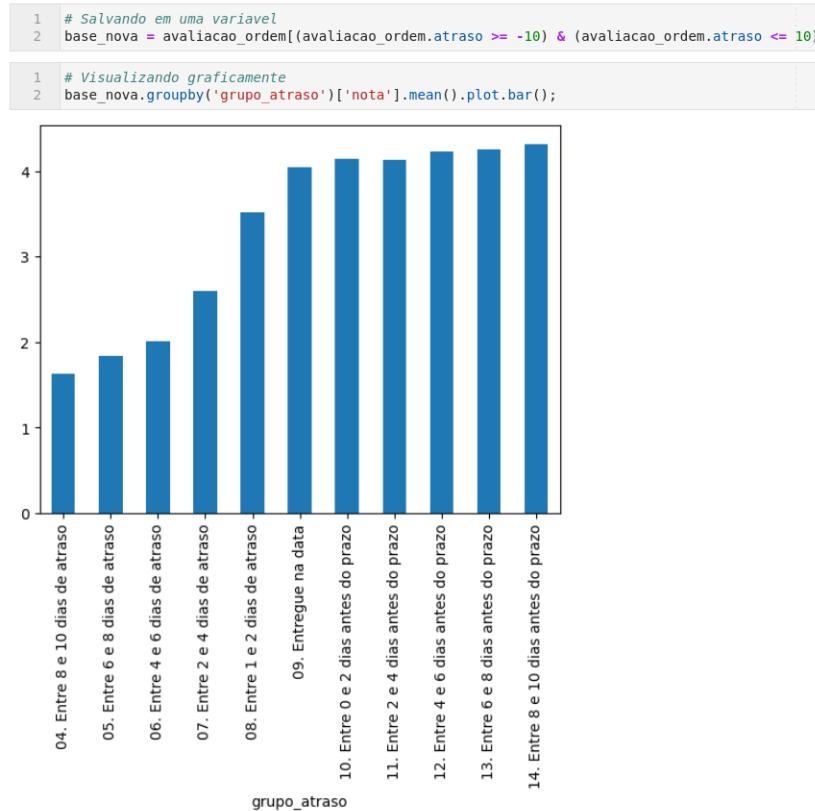
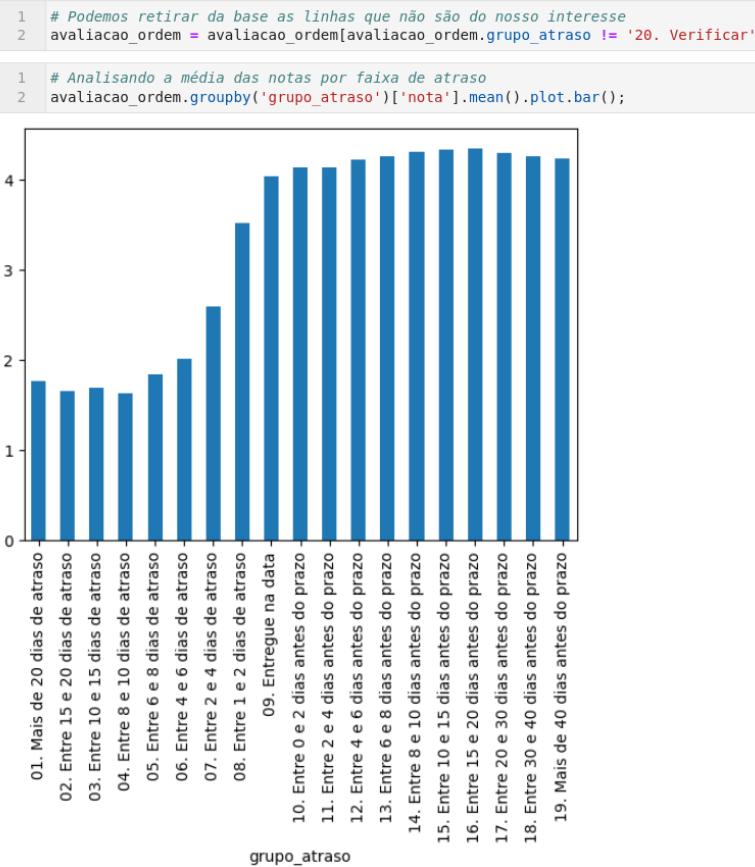
## Módulo 18 – Relação entre atraso e nota de avaliação

Claro que podemos criar quantos grupos quisermos:

```
1 # Usando a função já pronta apenas para adiantar a escrita do código
2 def agrupa_atraso(atraso):
3     if atraso < -20:
4         return '01. Mais de 20 dias de atraso'
5     elif atraso < -15:
6         return '02. Entre 15 e 20 dias de atraso'
7     elif atraso < -10:
8         return '03. Entre 10 e 15 dias de atraso'
9     elif atraso < -8:
10        return '04. Entre 8 e 10 dias de atraso'
11    elif atraso < -6:
12        return '05. Entre 6 e 8 dias de atraso'
13    elif atraso < -4:
14        return '06. Entre 4 e 6 dias de atraso'
15    elif atraso < -2:
16        return '07. Entre 2 e 4 dias de atraso'
17    elif atraso < 0:
18        return '08. Entre 1 e 2 dias de atraso'
19    elif atraso == 0:
20        return '09. Entregue na data'
21    elif atraso <= 2:
22        return '10. Entre 0 e 2 dias antes do prazo'
23    elif atraso <= 4:
24        return '11. Entre 2 e 4 dias antes do prazo'
25    elif atraso <= 6:
26        return '12. Entre 4 e 6 dias antes do prazo'
27    elif atraso <= 8:
28        return '13. Entre 6 e 8 dias antes do prazo'
29    elif atraso <= 10:
30        return '14. Entre 8 e 10 dias antes do prazo'
31    elif atraso <= 15:
32        return '15. Entre 10 e 15 dias antes do prazo'
33    elif atraso <= 20:
34        return '16. Entre 15 e 20 dias antes do prazo'
35    elif atraso <= 30:
36        return '17. Entre 20 e 30 dias antes do prazo'
37    elif atraso <= 40:
38        return '18. Entre 30 e 40 dias antes do prazo'
39    elif atraso > 40:
40        return '19. Mais de 40 dias antes do prazo'
41    else:
42        return '20. Verificar'
```

# Módulo 18 – Relação entre atraso e nota de avaliação

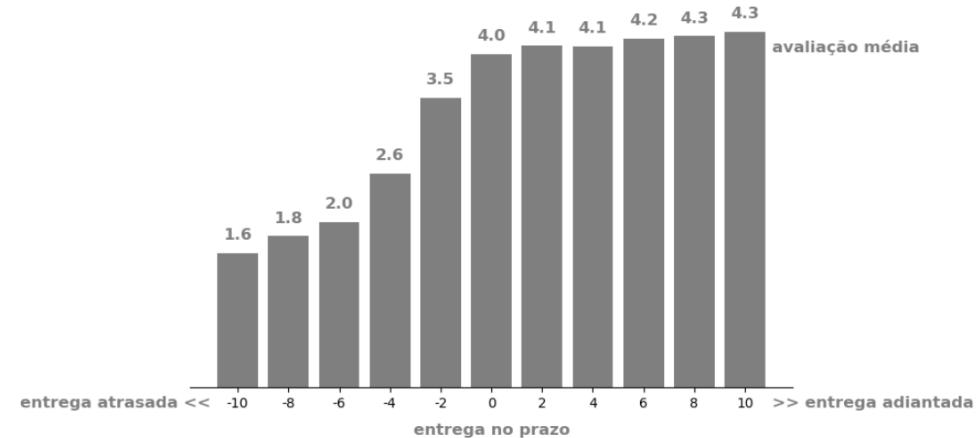
E criar um gráfico de barras. Lembre-se que muita informação pode confundir o usuário da mesma. Assim, podemos posteriormente restringir para um intervalo menor, mostrando menos barras. Observe que qualquer atraso já diminui a média, mas há uma queda significativa a partir da faixa de 2 a 4 dias de atraso:



## Módulo 18 – Relação entre atraso e nota de avaliação

Para uma apresentação, podemos melhorar ainda mais nosso gráfico. Observe como colocamos anotações explicando cada parte de nosso gráfico, em especial o eixo horizontal, já que as pessoas podem ficar confusas com o significado de valores positivos e negativos.

```
1 valores_grafico = base_nova.groupby('grupo_atraso')[['nota']].mean()
2
3 import numpy as np
4
5 # Melhorando a visualização
6 fig, ax = plt.subplots(figsize=(8,5))
7
8 x = np.arange(0,len(valores_grafico))
9
10 ax.bar(x,valores_grafico.values,color='tab:gray')
11
12 for i in range(0,len(valores_grafico)):
13     ax.annotate(round(valores_grafico.values[i],1),(x[i],valores_grafico.values[i]),ha="center",xytext=(0,10),
14                 textcoords="offset points",c='tab:gray',fontsize=12,fontweight='bold')
15
16 ax.set_xticks(x)
17 ax.set_xticklabels([-10,-8,-6,-4,-2,0,2,4,6,8,10])
18 ax.yaxis.set_visible(False)
19
20 ax.spines['top'].set_visible(False)
21 ax.spines['left'].set_visible(False)
22 ax.spines['right'].set_visible(False)
23
24 ax.annotate('avaliação média',(x[-1],valores_grafico.values[-1]),ha="left",xytext=(20,-15),
25             textcoords="offset points",c='tab:gray',fontsize=12,fontweight='bold')
26 ax.annotate('entrega atrasada <<',(x[0],0),ha="right",xytext=(-20,-15),
27             textcoords="offset points",c='tab:gray',fontsize=12,fontweight='bold')
28 ax.annotate('>> entrega adiantada',(x[-1],0),ha="left",xytext=(+20,-15),
29             textcoords="offset points",c='tab:gray',fontsize=12,fontweight='bold')
30 ax.annotate('entrega no prazo',(x[5],0),ha="center",xytext=(0,-35),
31             textcoords="offset points",c='tab:gray',fontsize=12,fontweight='bold')
32
33 plt.show()
```



# Módulo 18 – Avaliando os comentários dos pedidos atrasados

Parece realmente haver uma relação entre atraso e queda nas notas de avaliação. Mas podemos nos certificar ainda mais que a nota baixa é devido ao atraso verificando o texto dos comentários das avaliações. Abaixo, selecionamos os comentários para entregas com 5 dias de atraso ou mais:

```
1 # Pegando os ids das ordens entre 10 e 5 dias de atraso (as 3 primeiras colunas)
2 ids = base_nova.loc[base_nova.atraso <= -5,'order_id']
```

```
1 # Verificando as avaliações desses pedidos
2 avaliacoes.loc[avaliacoes.order_id.isin(ids),'review_comment_message']
```

```
32 Sempre compro pela Internet e a entrega ocorre...
39 Nada de chegar o meu pedido.
45 None
106 None
168 Comprei o produto dia 25 de fevereiro e hoje d...
...
99091 Ainda nao recevi o produto.
99094 Gostaria de saber porque meu produto ainda não...
99098 muito bom! recomendo!
99144 None
99174 None
Name: review_comment_message, Length: 2078, dtype: object
```

```
1 # Visualizando esses comentários
2 for i in avaliacoes.loc[avaliacoes.order_id.isin(ids),'review_comment_message']:
3     print(i)
```

```
Sempre compro pela Internet e a entrega ocorre antes do prazo combinado, que acredito ser o prazo máximo. No stark o prazo máximo já se esgotou e ainda não recebi o produto.
Nada de chegar o meu pedido.
None
None
Comprei o produto dia 25 de fevereiro e hoje dia 29 de março não fora entregue na minha residência. Não sei se os correios desse Brasil e péssimo ou foi a própria loja que demorou postar.
None
Não posso! Estou aguardando a chegada do produto que comprei!Logo que isso aconteça, terei minha opinião formada!
Comprei dois produtos e recebi somente um. Mandei dois emails para a loja e não tive nenhum retorno. Estou insatisfeita.
Até agora não recebi.Compre este aparelho por questao de saude,portanto não recomendo esta loja.
comprei um produto para ser entregue por essa loja
e ainda não recebi
pois assim já se passou a data de entrega
None
Até agora não recebi nada, a colcha estava prevista para anteontem(18/12) e até agora nada
None
None
Até o momento eu n recebir o produto
Não recebi o produto.e agora?
None
Preciso de uma previsão.
Recebi outra boneca, quero devolver e não consigo contato! Jamais compraria novamente dessa loja!
Estou esperando há mais de vinte dias e ainda não recebi meu produto e não consigo nenhuma informação.
Gostei muito do que comprei com vcs muito obrigada breve farei outras compras vanda
None
Produto não foi entregue!
```

## Módulo 18 – Avaliando os comentários dos pedidos atrasados

Com a biblioteca `wordcloud`, podemos criar uma nuvem de palavras com base nesses comentários. Na nuvem, as palavras ou termos maiores serão aquelas que mais aparecem. Observe que “não recebi” aparece claramente em destaque:

```
1 # Instalando o wordcloud  
2 # !pip install wordcloud
```

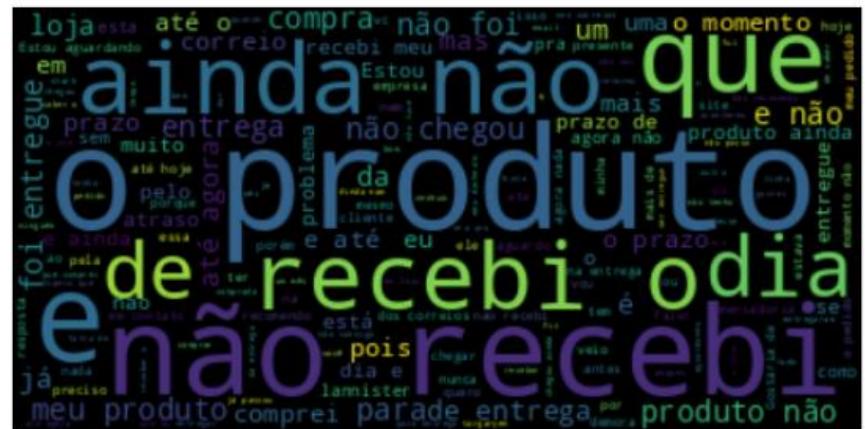
```
1 # Usando a nuvem de palavras  
2 from wordcloud import WordCloud
```

Para usar a biblioteca, precisamos ter um texto, então podemos transformar todos os comentários em um único texto

```
1 # Transformando todos os comentários em um único texto
2 texto = ''
3 for i in avaliacoes.loc[avaliacoes.order_id.isin(ids), 'review_comment_message']:
4     if i:
5         texto += i
6         texto += ' '
```

```
1 # Gerando a nuvem de palavras  
2 wordcloud = WordCloud().generate(texto)
```

```
1 # Exibindo a imagem
2 import matplotlib.pyplot as plt
3 plt.imshow(wordcloud, interpolation='bilinear')
4 plt.axis("off")
5 plt.show()
```



## Módulo 18 – Avaliando os comentários dos pedidos atrasados

Podemos retirar os espaços, pegando frases que mais aparecem. Novamente, temos mais evidências que os comentários são predominantemente reclamando do atraso na entrega:

```
1 # E se a gente tentar agrupar por frases?
2 texto = ''
3 for i in avaliacoes.loc[avaliacoes.order_id.isin(ids), 'review_comment_message']:
4     if i:
5         i = i.replace(' ', '')
6         texto += i
7         texto += ' '
8
9 # Gerando a nuvem de palavras
10 wordcloud = WordCloud(stopwords=['porém', 'mail', 'Bomdia'], colormap='RdGy').generate(texto)
11
12 # Exibindo a imagem
13 import matplotlib.pyplot as plt
14 plt.imshow(wordcloud, interpolation='bilinear')
15 plt.axis("off")
16 plt.show()
```



## Módulo 18 – Storytelling com dados

Com isto, fechamos as análises deste módulo. Lembrando, novamente, para que vejam as aulas sobre storytelling e o material em slides correspondente. No material e nos vídeos, será mostrada a estrutura de uma história, como no slide abaixo:

### A estrutura de uma história

Uma história pode ser dividida em **3 atos**:

- PREPARAÇÃO**  
Você vai mostrar ao público **o motivo de valer a pena ficar ali**, vai introduzir as informações mais relevantes e explicar o **contexto**
- CONFLITO**  
Qual o **problema** que queremos resolver ou a **oportunidade** que temos de melhoria? O que justifica isso que estamos mostrando?
- SOLUÇÃO**  
Apresentar as **ideias** que serão usadas para resolver o conflito e **chamar para a ação** ou gerar uma **discussão**

## Módulo 18 – Storytelling com dados

Também será mostrado como adicionar as figuras geradas durante o módulo com textos que destaque o que realmente importa nas figuras:



## Módulo 18 – Storytelling com dados

E, claro, as figuras podem ser combinadas, agregando ainda mais valor ao estudo apresentado. Então, não passe de módulo antes de conferir o material de storytelling!



## MÓDULO 19

# Criando um modelo de identificação de fraude

## Módulo 19 – Conhecendo nossa base de dados

Neste módulo desenvolveremos um projeto. Este projeto tem como objetivo introduzir os conceitos fundamentais de Ciência de Dados através da análise de um problema real: a detecção de fraudes em cartões de crédito. Utilizaremos técnicas de aprendizado de máquina para identificar padrões suspeitos e prever possíveis fraudes. Sobre a base de dados:

- Contém transações feitas em setembro de 2013 por empresas de cartão de crédito europeias.
- O conjunto de dados é composto por transações que ocorreram em dois dias, com um total de 284.807 transações.
- De todas as transações, 492 foram classificadas como fraudulentas, tornando o conjunto de dados altamente desbalanceado.
- As variáveis do conjunto de dados são todas numéricas e são o resultado de uma transformação PCA (*Principal Component Analysis*). São nomeadas como V1, V2, ... V28.
- Devido a questões de confidencialidade, os recursos originais e mais informações básicas sobre os dados não estão disponíveis.
- As únicas variáveis que não foram transformadas com o PCA são 'Time' e 'Amount'.
- A característica 'Time' representa os segundos decorridos entre cada transação e a primeira transação no conjunto de dados.
- A característica 'Amount' representa a quantidade da transação.
- A característica 'Class' é a variável de resposta e assume o valor 1 em caso de fraude e 0 em caso contrário.

## Módulo 19 – Conhecendo nossa base de dados

Como já fizemos diversas vezes nos módulos anteriores, vamos começar importando a biblioteca Pandas e criando um dataframe a partir do arquivo de nossa base de dados.

Veja que as primeiras entradas correspondem à descrição passada na página anterior:

```
1 # Importando o pandas
2 import pandas as pd
```

```
1 # Importando a base
2 base = pd.read_csv('creditcard.csv')
```

```
1 # Visualizando a base
2 base.head(3)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0

3 rows × 31 columns

## Módulo 19 – Conhecendo nossa base de dados

Com o método `info`, vemos que realmente todas as colunas são numéricas e que aparentemente não há valores nulos na base:

```
1 # Verificando as informações
2 base.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column   Non-Null Count   Dtype  
--- 
 0   Time      284807 non-null    float64
 1   V1        284807 non-null    float64
 2   V2        284807 non-null    float64
 3   V3        284807 non-null    float64
 4   V4        284807 non-null    float64
 5   V5        284807 non-null    float64
 6   V6        284807 non-null    float64
 7   V7        284807 non-null    float64
 8   V8        284807 non-null    float64
 9   V9        284807 non-null    float64
 10  V10       284807 non-null    float64
 11  V11       284807 non-null    float64
 12  V12       284807 non-null    float64
 13  V13       284807 non-null    float64
 14  V14       284807 non-null    float64
 15  V15       284807 non-null    float64
 16  V16       284807 non-null    float64
 17  V17       284807 non-null    float64
 18  V18       284807 non-null    float64
 19  V19       284807 non-null    float64
 20  V20       284807 non-null    float64
 21  V21       284807 non-null    float64
 22  V22       284807 non-null    float64
 23  V23       284807 non-null    float64
 24  V24       284807 non-null    float64
 25  V25       284807 non-null    float64
 26  V26       284807 non-null    float64
 27  V27       284807 non-null    float64
 28  V28       284807 non-null    float64
 29  Amount    284807 non-null    float64
 30  Class     284807 non-null    int64 
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

# Módulo 19 – Conhecendo nossa base de dados

Na descrição da base feita na primeira página deste módulo, foi dito que as 28 variáveis disponíveis são o resultado de um PCA dos dados originais, aos quais não temos acesso por questões de confidencialidade.

PCA, ou Análise de Componentes Principais, é uma técnica que transforma um conjunto de dados com muitas variáveis em um novo conjunto com menos variáveis, chamadas de "componentes principais". Esses componentes mantêm a maior parte das informações úteis dos dados originais, tornando-os mais fáceis de entender e analisar. É como resumir um livro em seus pontos principais.

Com o método `describe`, vemos que estes componentes possuem valores que variam de negativo a positivo, com média próximo de zero:

1	# E também o resumo estatístico															
2	base.describe()															
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25
count	284807.000000	2.848070e+05	...	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05								
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15	...	1.654067e-16	-3.568593e-16	2.578648e-16	4.473266e-15	5.340915e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	...	7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	...	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	...	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	...	-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	...	1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	...	2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00

8 rows x 31 columns

## Módulo 19 – Conhecendo nossa base de dados

Sempre que lidamos com dados numéricos, é importante verificar se o valor zero aparece com alguma frequência que pareça não usual. Isto porque, às vezes, usasse erroneamente zero quando se quer representar ausência de valor.

Em nossa base, há 1825 registros para *Amount* com valor zero. Considerando o tamanho da base, não é um valor elevado. E podem haver razões para transações com valor zero. Pode, por exemplo, ser apenas uma transação para testar que o cartão existe, bem comum em cadastro de cartões em sites de e-commerce:

1	base[base.Amount == 0]																				
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
383	282.0	-0.356466	0.725418	1.971749	0.831343	0.369681	-0.107776	0.751610	-0.120166	-0.420675	...	0.020804	0.424312	-0.015989	0.466754	-0.809962	0.657334	-0.043150	-0.046401	0.0	0
514	380.0	-1.299837	0.881817	1.452842	-1.293698	-0.025105	-1.170103	0.861610	-0.193934	0.592001	...	-0.272563	-0.360853	0.223911	0.598930	-0.397705	0.637141	0.234872	0.021379	0.0	0
534	403.0	1.237413	0.512365	0.687746	1.693872	-0.236323	-0.650232	0.118066	-0.230545	-0.808523	...	-0.077543	-0.178220	0.038722	0.471218	0.289249	0.871803	-0.066884	0.012986	0.0	0
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.537387	1.391657	-2.770089	...	0.517232	-0.035049	-0.465211	0.320198	0.044519	0.177840	0.261145	-0.143276	0.0	1
575	430.0	-1.860258	-0.629859	0.966570	0.844632	0.759983	-1.481173	-0.509681	0.540722	-0.733623	...	0.268028	0.125515	-0.225029	0.586664	-0.031598	0.570168	-0.043007	-0.223739	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
283719	171817.0	-0.750414	0.904175	0.996461	0.427284	1.720336	0.929256	0.794272	0.176719	-1.836261	...	0.050750	0.115532	-0.623995	-0.186896	0.733759	2.558151	-0.188835	0.001654	0.0	0
283782	171870.0	2.083677	-0.065811	-1.442870	0.135416	0.043035	-1.306975	0.335835	-0.371635	0.730560	...	-0.147536	-0.246599	0.194758	-0.082277	0.012887	-0.069278	-0.048995	-0.065482	0.0	0
283949	172027.0	2.132569	-0.057836	-1.724522	-0.030326	0.412146	-0.903088	0.345843	-0.348132	0.722638	...	-0.188739	-0.343876	0.105024	-0.763831	0.117381	-0.027682	-0.047514	-0.071700	0.0	0
284085	172140.0	-2.210521	-1.039425	0.189704	-1.291932	3.742120	-1.665061	3.120388	-2.324089	0.364926	...	-0.286359	1.326003	-0.361764	-0.268117	1.051309	0.334629	-1.930149	-0.899888	0.0	0
284770	172759.0	-0.822731	1.270140	-0.138566	0.479620	1.242101	0.795218	0.454284	0.556038	-1.550610	...	0.138766	0.450908	-0.192146	-0.196218	-0.261664	2.372675	-0.042743	0.109613	0.0	0

1825 rows x 31 columns

## Módulo 19 – Conhecendo nossa base de dados

Já vimos com o `describe` que não há registros de entradas nulas na base. Mas podemos nos certificar novamente com o método `isnull`:

```
1 # Verificando também se existem valores nulos
2 base.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

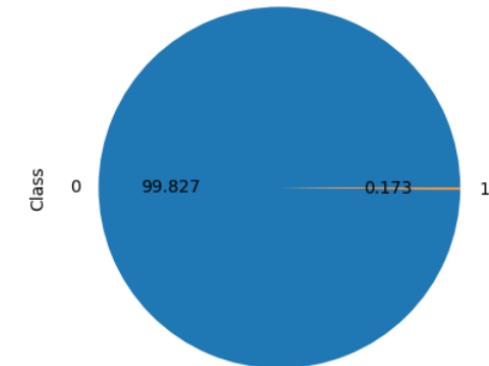
## Módulo 19 – Verificando o desbalanceamento da base

Na descrição da base, já destacamos que há um grande desbalanceamento entre o número de transações não fraudulentas (classe 0) e fraudulentas (classe 1).

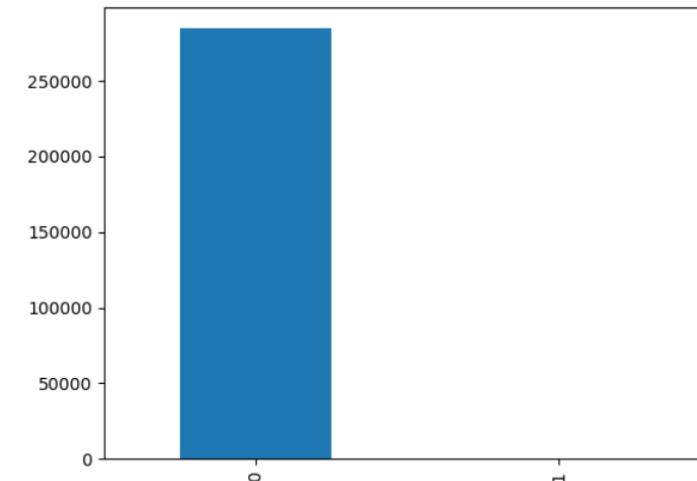
```
1 # Contando os valores em cada classe  
2 y.value_counts()  
  
0    284315  
1     492  
Name: Class, dtype: int64
```

Podemos visualizar a diferença a partir de um gráfico de pizza da porcentagem de cada classe. Perceba como há menos de 0,2 % de transações fraudulentas na base. Em termos absolutos, verifique a diferença de altura em um gráfico de barras, onde nem é possível ver a barra da classe 1:

```
1 # Verificando visualmente o % de transações que são fraude  
2 base.Class.value_counts().plot.pie(autopct='%.3f');
```



```
1 # Visualizando como um gráfico de barras  
2 y.value_counts().plot.bar();
```



## Módulo 19 – Verificando o desbalanceamento da base

Como sabemos, para começar a treinar modelos precisamos separar a base em dados de treino e de teste. No caso de bases muito desbalanceadas, devemos utilizar o parâmetro `stratify`, para garantir que o desbalanceamento permanecerá em cada parte resultante da separação:

```
1 # Separando X e y  
2 X = base.drop('Class',axis=1)  
3 y = base.Class
```

```
1 # Separando em treino e teste  
2 from sklearn.model_selection import train_test_split  
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42,stratify=y)
```

```
1 # Verificando a distribuição das duas classes na base de treino e teste  
2 y_train.value_counts()/y_train.shape[0]
```

```
0    0.998271  
1    0.001729  
Name: Class, dtype: float64
```

```
1 # Para a base de teste  
2 y_test.value_counts()/y_test.shape[0]
```

```
0    0.998276  
1    0.001724  
Name: Class, dtype: float64
```

## Módulo 19 – Verificando o desbalanceamento da base

Agora, podemos treinar um modelo. Por exemplo, uma árvore de decisão, como já feito em módulos anteriores. Observe que as métricas que já conhecemos apresentam valores elevados, em especial a acurácia. No entanto, esse resultado pode não realmente refletir a capacidade do modelo. Afinal, por simples probabilidade, podemos classificar todas as transações como não fraudulentas e teremos uma acurácia de mais de 99 %!

```
1 # Importando a árvore de decisão
2 from sklearn import tree
```

```
1 # Definindo o nosso classificador
2 clf = tree.DecisionTreeClassifier(random_state=42)
```

```
1 # Fazendo o fit para os dados de treino
2 clf = clf.fit(X_train, y_train)
```

```
1 # Fazendo a previsão
2 y_pred = clf.predict(X_test)
```

```
1 # Importando a acurácia
2 from sklearn.metrics import accuracy_score
```

```
1 # Calculando a acurácia
2 accuracy_score(y_test, y_pred)*100
```

99.91594582229457

```
1 # Importando a matriz de confusão
2 from sklearn.metrics import confusion_matrix
3 confusion_matrix(y_test, y_pred)
```

array([[93786, 39],
 [40, 122]])

```
1 # Calculando a precisão
2 from sklearn.metrics import precision_score
3 precision_score(y_test, y_pred)*100
```

75.77639751552795

```
1 (122/(39+122))*100
```

75.77639751552795

```
1 # E o recall
2 from sklearn.metrics import recall_score
3 recall_score(y_test, y_pred)*100
```

75.30864197530865

```
1 (122/(40+122))*100
```

75.30864197530865



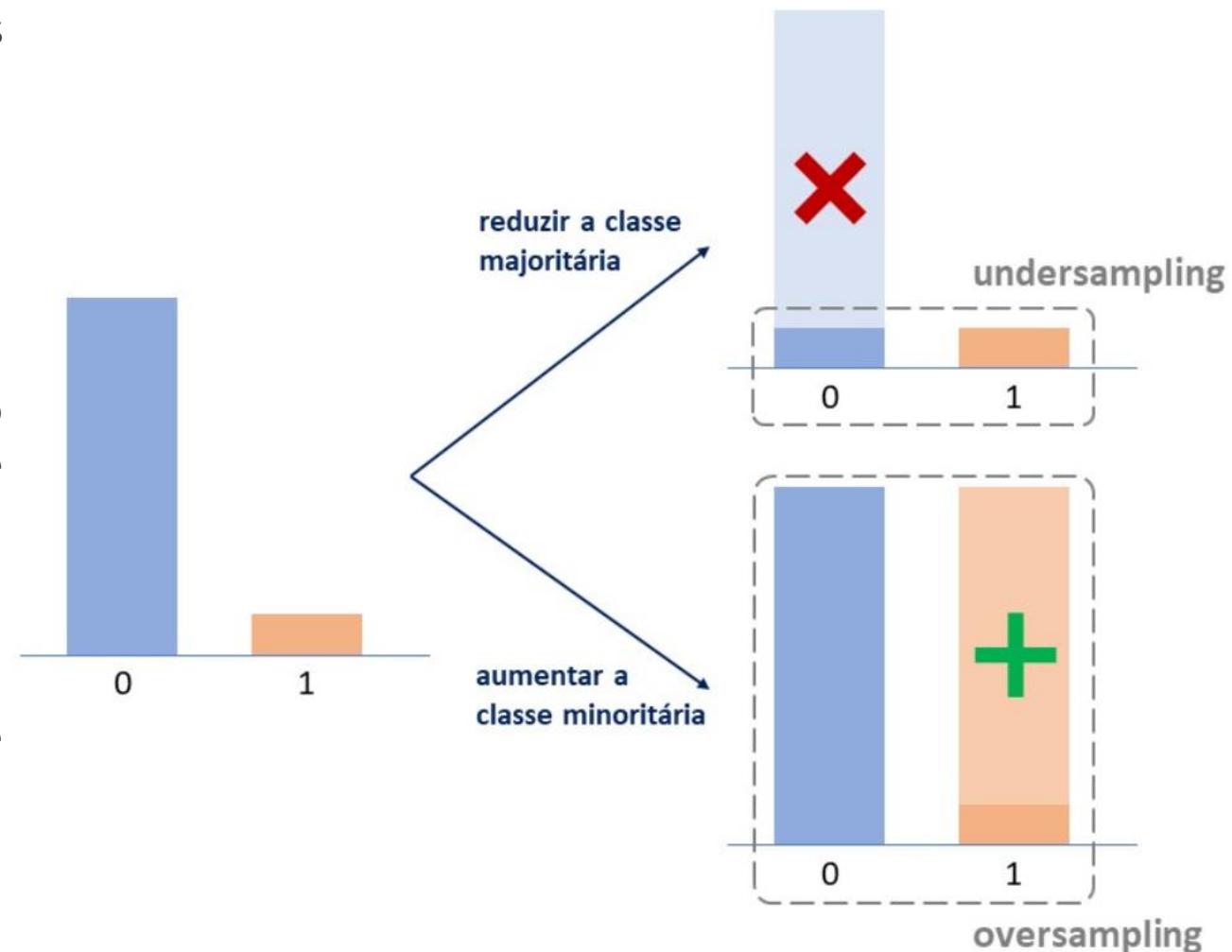
## Módulo 19 – Estratégias para tratar o desbalanceamento

Para resolver o desbalanceamento, há duas estratégias principais de reamostragem:

- Reduzir a classe majoritária (*undersampling*)
- Aumentar a classe minoritária (*oversampling*)

Em ambas as estratégias é necessário cuidado para não haver perda de informação nem viés.

Em Python, há a biblioteca Imbalanced Learn, também conhecida pela forma reduzida *imblearn*, com uma interface similar ao Scikit-Learn, que possui algoritmos para as duas estratégias citadas.



## Módulo 19 – Estratégias para tratar o desbalanceamento

A forma mais simples de *undersampling* é retirar aleatoriamente registros da classe majoritária. Para isto, a *imblearn* possui a classe **RandomUnderSampler**.

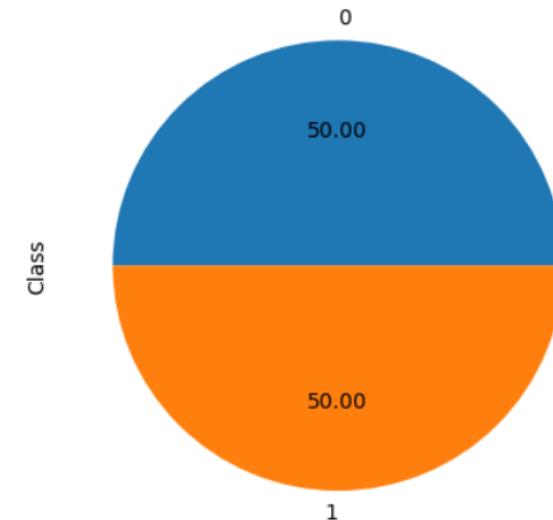
Observe que, após o *resampling* (reamostragem), a coluna alvo *y* possui o mesmo número de valores para cada classe:

```
1 # Importando o RandomUnderSampler do imblearn
2 from imblearn.under_sampling import RandomUnderSampler

1 # Definindo o RandomUnderSampler
2 rus = RandomUnderSampler(random_state=42)

1 # Aplicando para X e y
2 X_res, y_res = rus.fit_resample(X, y)

1 # Visualizando graficamente
2 y_res.value_counts().plot.pie(autopct='%.2f');
```



```
1 # Contando os valores em cada classe
2 y_res.value_counts()
```

Class	Count
0	492
1	492

Name: Class, dtype: int64

## Módulo 19 – Estratégias para tratar o desbalanceamento

Da mesma maneira, a forma mais simples de *oversampling* é aumentar aleatoriamente registros da classe minoritária. Para isto, a *imblearn* possui a classe **RandomOverSampler**.

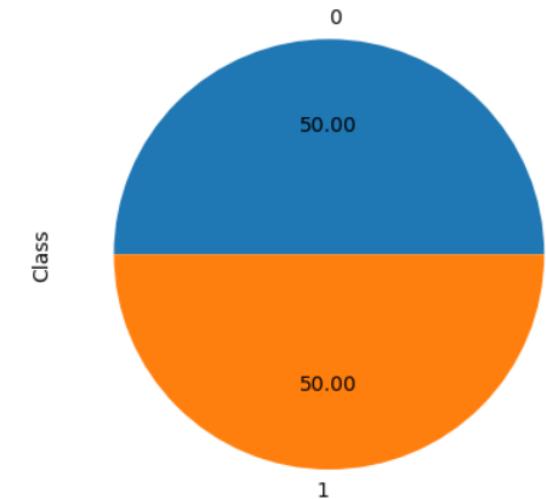
Observe que, após o *resampling* (reamostragem), a coluna alvo *y* possui o mesmo número de valores para cada classe:

```
1 # Importando o RandomOverSampler do imblearn
2 from imblearn.over_sampling import RandomOverSampler
```

```
1 # Instanciando
2 ros = RandomOverSampler(random_state=42)
```

```
1 # Aplicando para X e y
2 X_res, y_res = ros.fit_resample(X, y)
```

```
1 # Visualizando graficamente
2 y_res.value_counts().plot.pie(autopct='%.2f');
```



```
1 # Contando os valores em cada classe
2 y_res.value_counts()
```

```
0    284315
1    284315
Name: Class, dtype: int64
```

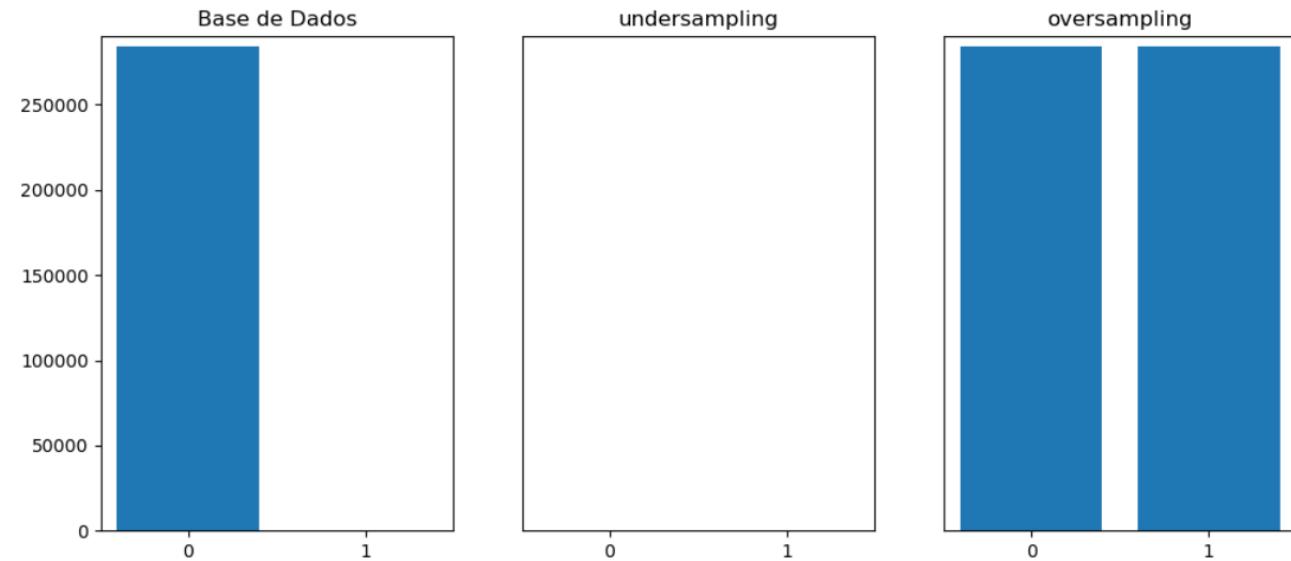
# Módulo 19 – Estratégias para tratar o desbalanceamento

Graficamente, podemos perceber a diferença entre as abordagens. Nossa base original é desbalanceada, de forma que nem é possível enxergar a barra da classe 1.

Com o *undersampling*, não conseguimos enxergar ambas as barras, pois a classe majoritária foi reduzida até o mesmo número da classe minoritária.

Com o *oversampling*, a classe minoritária foi aumentada até atingir o mesmo número da classe majoritária.

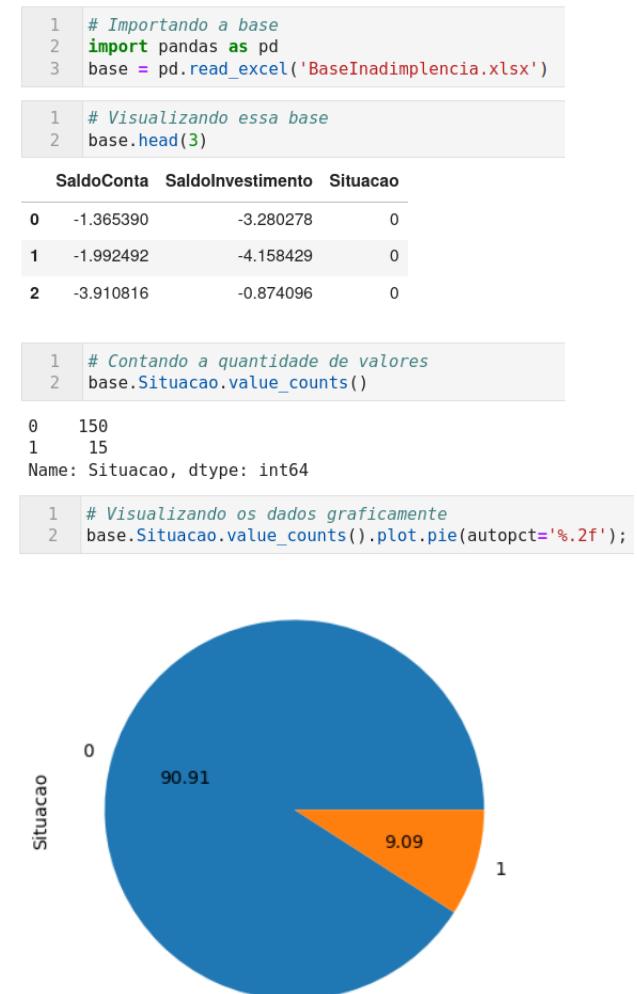
```
1 # Visualizando graficamente
2 import matplotlib.pyplot as plt
3
4
5
6
7
8
9
10
11
12
13
14
15
```



## Módulo 19 – Visualizando as diferentes estratégias

Como é a primeira vez que utilizamos a Imbalanced-Learn em nossos estudos, e a base de dados de fraude é bem grande e com diversas variáveis, vamos pegar uma base de dados mais simples para entender melhor nossa biblioteca. Depois, voltaremos a nossos dados de fraude.

Vamos considerar uma base de dados de inadimplência, já utilizada em módulos anteriores. Nela, temos variáveis para saldo em conta e saldo em investimentos, e a coluna alvo é a situação do cliente, se inadimplente ou não. Veja que também é uma base desbalanceada, mas com bem menos registros e menos variáveis, facilitando a compreensão.



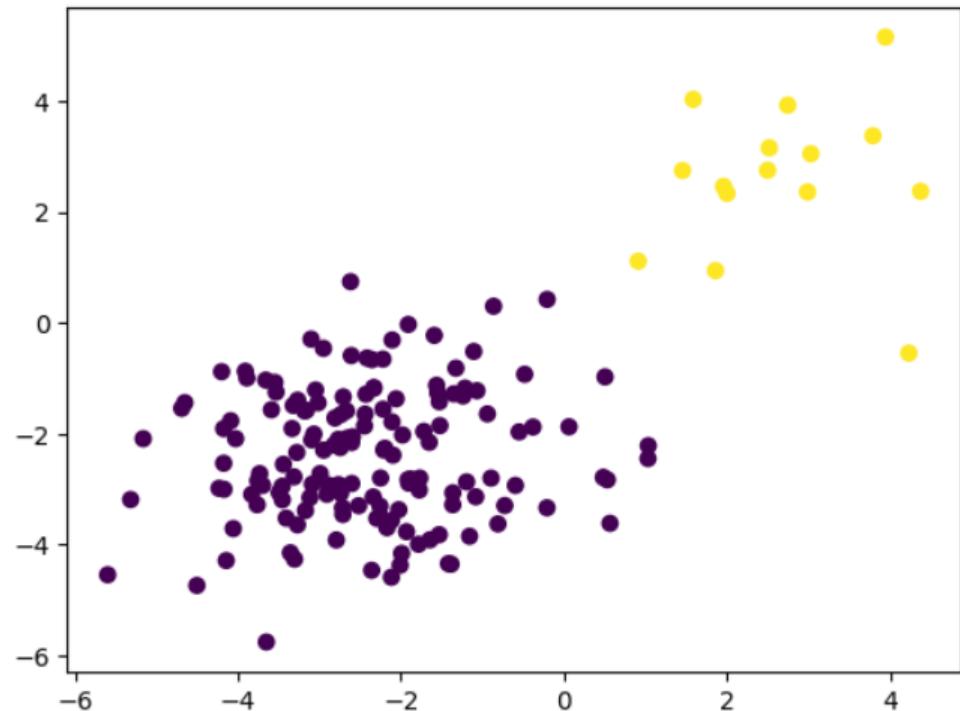
## Módulo 19 – Visualizando as diferentes estratégias

Fazendo um gráfico de dispersão, onde o eixo horizontal representa o saldo em conta e o vertical o saldo em investimentos, vemos que as duas classes são bem separadas.

Os pontos amarelos são da classe de clientes inadimplentes. Conforme esperado, há menos destes pontos quando comparados com os pontos roxos, devido ao desbalanceamento.

Vejamos como cada estratégia de reamostragem afeta este gráfico nas próximas páginas.

```
1 # Visualizando a distribuição dos dados
2 import matplotlib.pyplot as plt
3 fig, ax = plt.subplots()
4
5 ax.scatter(base.SaldoConta, base.SaldoInvestimento, c=base.Situacao)
6
7 plt.show()
```



## Módulo 19 – Visualizando as diferentes estratégias

Começando pelo RandomUnderSampler. Vemos que 15 pontos aleatórios da classe 0, de cor roxa, foram selecionados, o mesmo número presente na classe 1, de cor amarela. Todos os pontos resultantes da reamostragem foram destacados em vermelho no gráfico.

```
1 # Separando X e y
2 X = base.drop('Situacao',axis=1)
3 y = base.Situacao
```

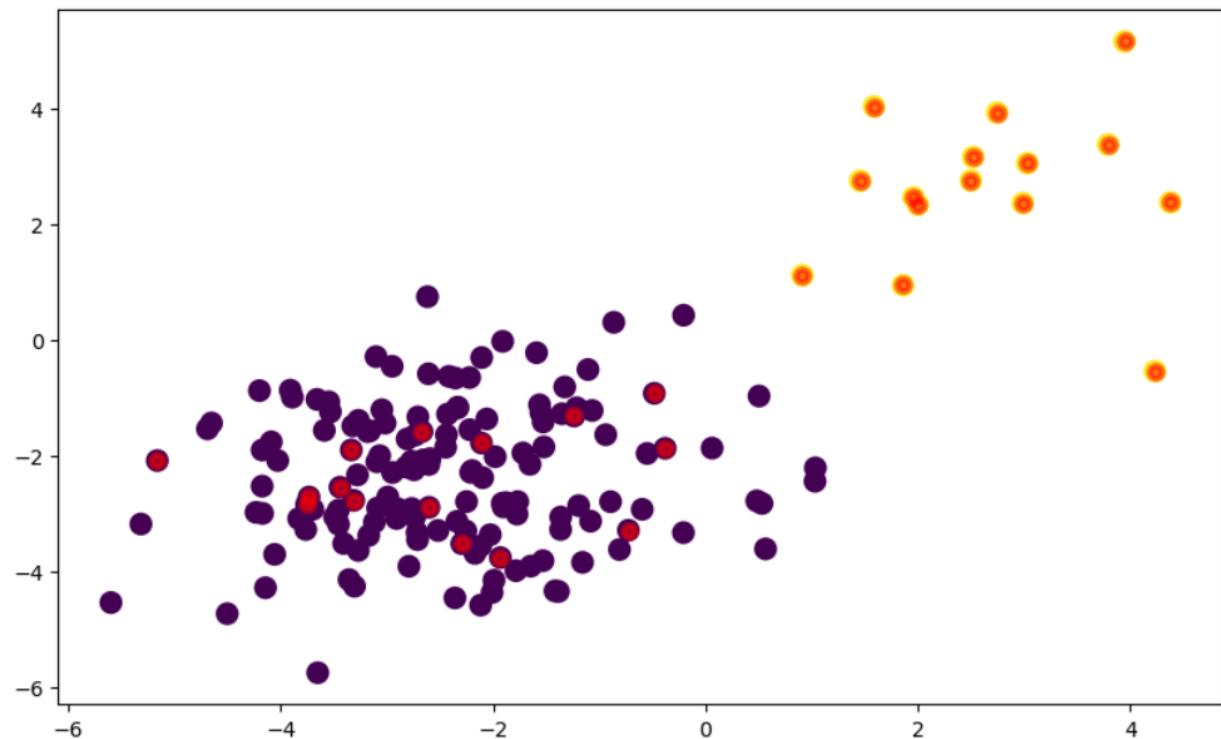
```
1 # Definindo o RandomUnderSampler
2 from imblearn.under_sampling import RandomUnderSampler
```

```
1 # Definindo a nova amostra
2 rus = RandomUnderSampler(random_state=42)
3 X_res, y_res = rus.fit_resample(X, y)
```

```
1 # Contando os valores
2 y_res.value_counts()
```

```
0    15
1    15
Name: Situacao, dtype: int64
```

```
1 # E visualizando graficamente
2 fig, ax = plt.subplots(figsize=(10,6))
3
4 ax.scatter(base.SaldoConta, base.SaldoInvestimento, c=base.Situacao, linewidths=5)
5 ax.scatter(X_res.SaldoConta, X_res.SaldoInvestimento, c='red', linewidths=3, alpha=0.5)
6
7 plt.show()
```



## Módulo 19 – Visualizando as diferentes estratégias

Outra estratégia de *undersampling* é criar pontos a partir de grupos da classe majoritária com `ClusterCentroids`. Tais pontos são chamados de centroides, e são escolhidas via algoritmo de KMeans (que será detalhado mais adiante no curso).

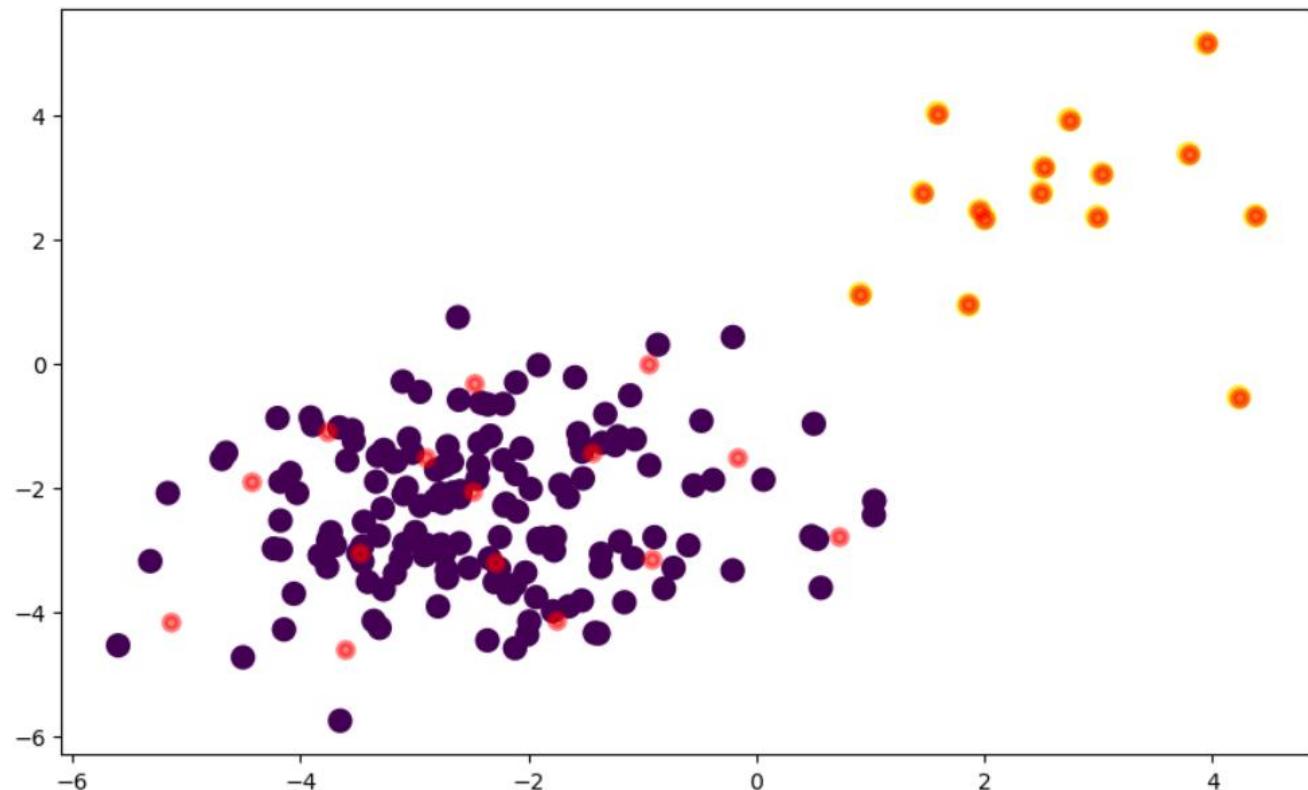
Observe na imagem que a classe majoritária foi dividida em 15 grupos e os centros destes grupos são os pontos que serão considerados.

```
1 # Importando
2 from imblearn.under_sampling import ClusterCentroids

1 # Definindo o ClusterCentroids
2 cc = ClusterCentroids(random_state=42)

1 # Criando a amostra dos dados
2 X_res, y_res = cc.fit_resample(X, y)
```

```
1 # Visualizando graficamente
2 fig, ax = plt.subplots(figsize=(10,6))
3
4 ax.scatter(base.SaldoConta, base.SaldoInvestimento, c=base.Situacao, linewidths=5)
5 ax.scatter(X_res.SaldoConta, X_res.SaldoInvestimento, c='red', linewidths=3, alpha=0.5)
6
7 plt.show()
```



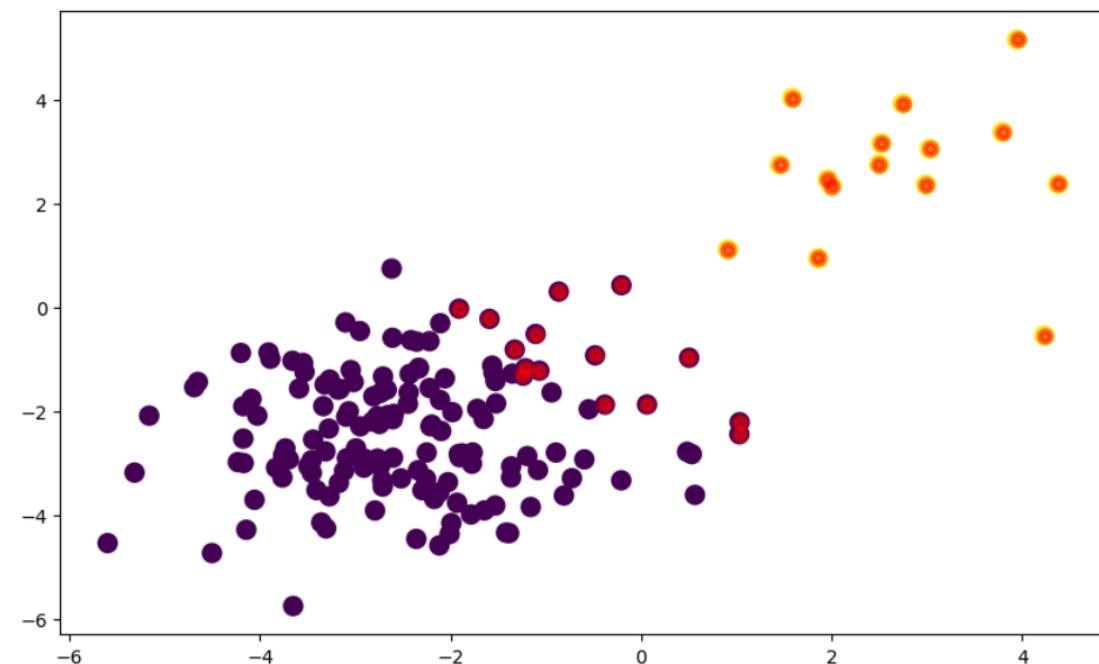
## Módulo 19 – Visualizando as diferentes estratégias

Ainda em estratégias de *undersampling*, temos a **NearMiss**. Esta usa o algoritmo de KNN – KNearestNeighbors para selecionar os pontos da classe majoritária que possuem a menor distância aos pontos da classe minoritária.

Observe no gráfico que foram selecionados 15 pontos da classe majoritária. E tais pontos são aqueles mais próximos da classe minoritária.

```
1 # Fazendo a reamostragem usando o NearMiss
2 from imblearn.under_sampling import NearMiss
3 nm = NearMiss()
4 X_res, y_res = nm.fit_resample(X, y)
```

```
1 # Visualizando graficamente
2 fig, ax = plt.subplots(figsize=(10,6))
3
4 ax.scatter(base.SaldoConta, base.SaldoInvestimento, c=base.Situacao, linewidths=5)
5 ax.scatter(X_res.SaldoConta, X_res.SaldoInvestimento, c='red', linewidths=3, alpha=0.5)
6
7 plt.show()
```



## Módulo 19 – Visualizando as diferentes estratégias

Passando agora para estratégias de *oversampling*. A primeira é utilizando o `RandomOverSampler`, que gera repetições de registros da classe minoritária. Abaixo, veja que um determinado registro foi replicado 11 vezes:

```
1 # Importando
2 from imblearn.over_sampling import RandomOverSampler

1 # Definindo o ros
2 ros = RandomOverSampler(random_state=42)

1 # Refazendo nossa amostra
2 X_res, y_res = ros.fit_resample(X, y)

1 # Contando os valores
2 y_res.value_counts()

0    150
1    150
Name: Situacao, dtype: int64
```

```
1 # Podemos ver quantas vezes essa primeira linha foi duplicada
2 X_res[(X_res.SaldoConta == base.iloc[151,0]) & (X_res.SaldoInvestimento == base.iloc[151,1])]

   SaldoConta  SaldoInvestimento
151      2.498148        2.749414
187      2.498148        2.749414
192      2.498148        2.749414
226      2.498148        2.749414
231      2.498148        2.749414
236      2.498148        2.749414
250      2.498148        2.749414
253      2.498148        2.749414
262      2.498148        2.749414
265      2.498148        2.749414
281      2.498148        2.749414
```

## Módulo 19 – Visualizando as diferentes estratégias

No entanto, por vezes não é interessante ter registros exatamente iguais duplicados, já que pode criar viés nos modelos treinados. Assim, podemos passar valores para o parâmetro `shrinkage`, que geram pequenas variações nos pontos criados, de forma que não ficam exatamente iguais aos pontos de origem.

```
1 # Definindo o ros  
2 ros = RandomOverSampler(random_state=42, shrinkage=0.5)
```

```
1 # Refazendo nossa amostra  
2 X_res, y_res = ros.fit_resample(X, y)
```

```
1 # Contando os valores  
2 y_res.value_counts()
```

```
0    150  
1    150  
Name: Situacao, dtype: int64
```

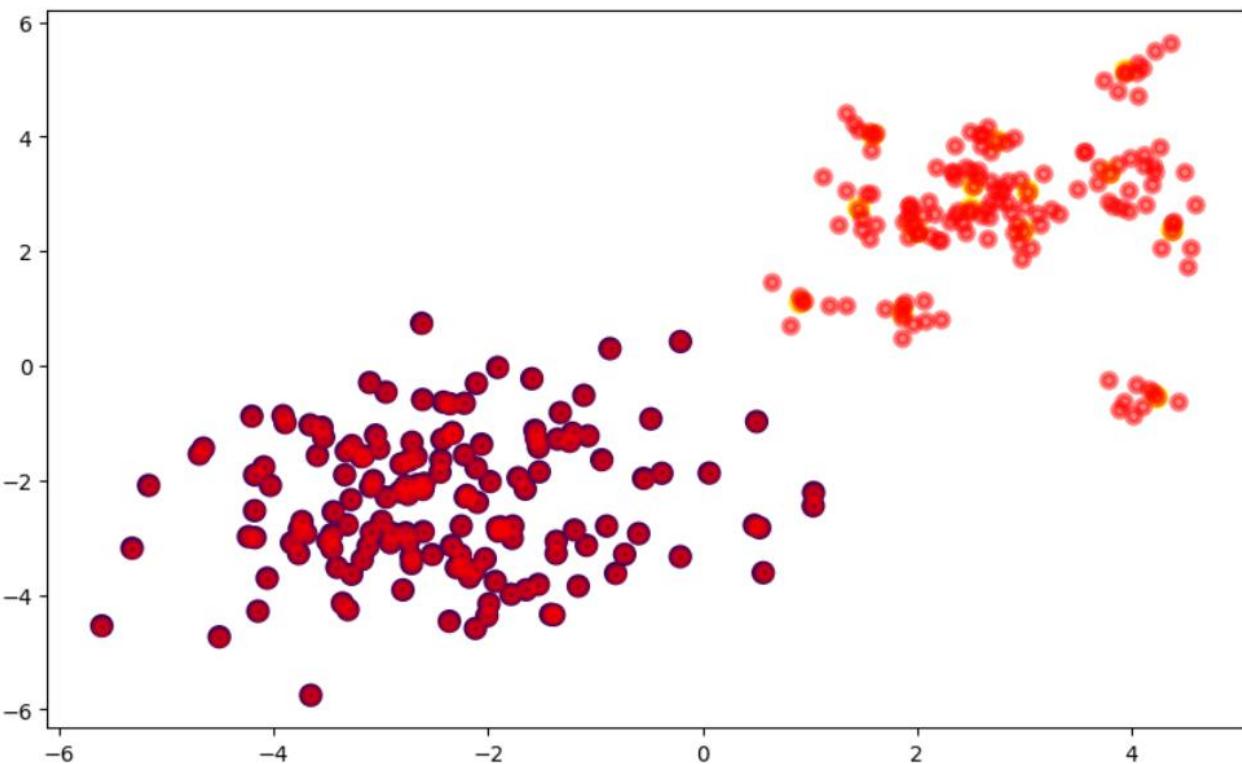
```
1 # Podemos ver quantas vezes essa primeira linha foi duplicada  
2 X_res[(X_res.SaldoConta == base.iloc[151,0]) & (X_res.SaldoInvestimento == base.iloc[151,1])]
```

	SaldoConta	SaldoInvestimento
151	2.498148	2.749414

## Módulo 19 – Visualizando as diferentes estratégias

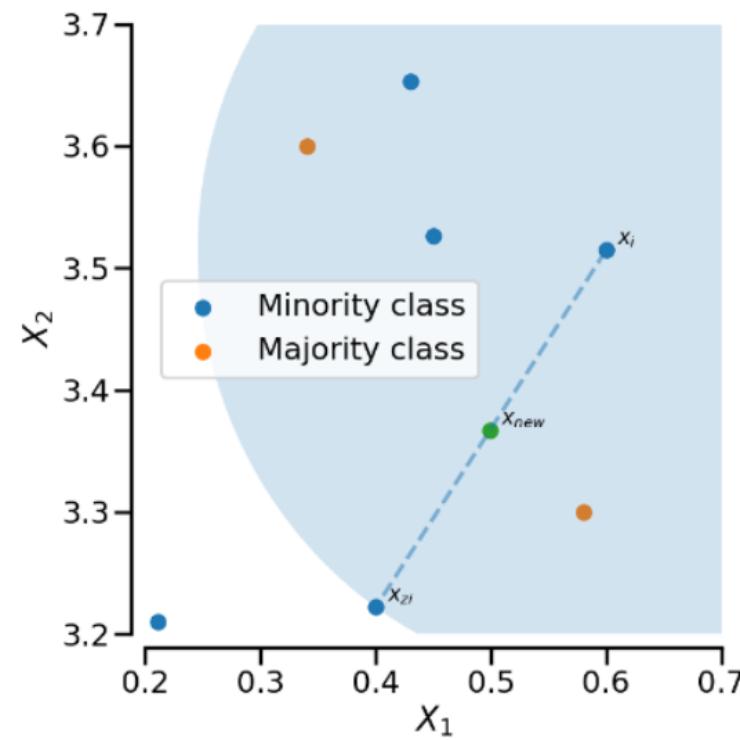
No gráfico, vemos que pontos gerados com o parâmetro `shrinkage=0.5` ainda ficam relativamente próximos aos seus pontos de origem. Observe como é possível distinguir grupos na região que originalmente era da classe minoritária. Se o valor do parâmetro for aumentado, os pontos ficarão mais dispersos.

```
1 # Visualizando graficamente
2 fig, ax = plt.subplots(figsize=(10,6))
3
4 ax.scatter(base.SaldoConta, base.SaldoInvestimento, c=base.Situacao, linewidths=5)
5 ax.scatter(X_res.SaldoConta, X_res.SaldoInvestimento, c='red', linewidths=3, alpha=0.5)
6
7 plt.show()
```



## Módulo 19 – Visualizando as diferentes estratégias

Outra forma de realizar *oversampling* é utilizar a estratégia SMOTE. Nesta, utiliza-se o algoritmo KNN para gerar amostras. Por exemplo, na figura abaixo, um novo ponto, destacado em verde, é gerado a partir de seus vizinhos mais próximos.

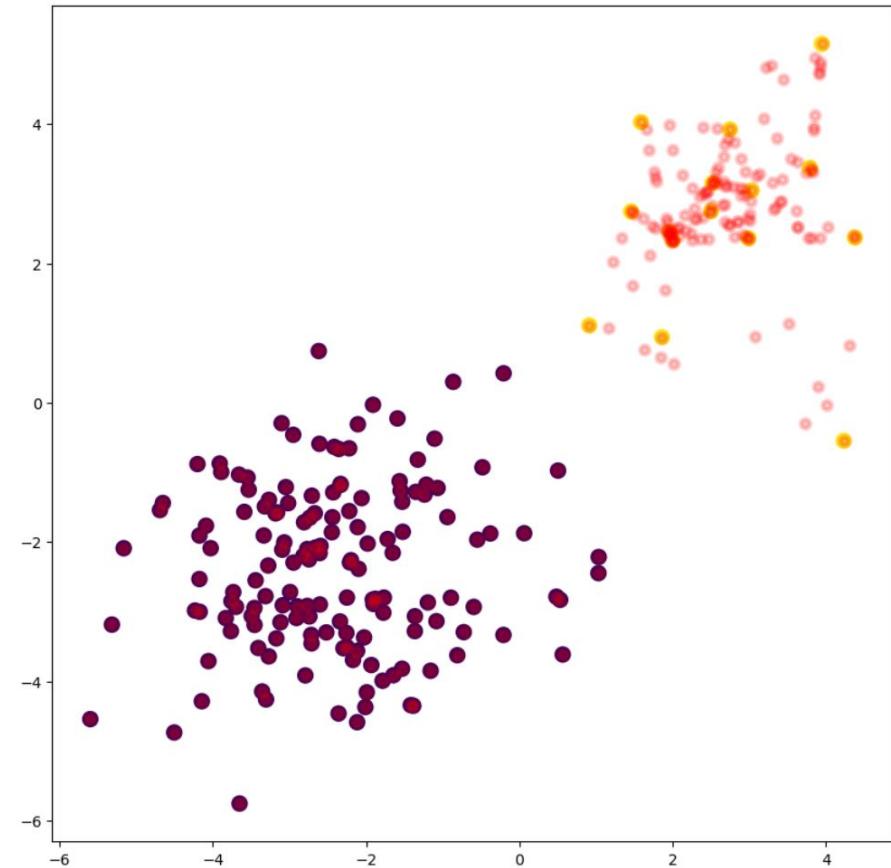


## Módulo 19 – Visualizando as diferentes estratégias

Observe no gráfico ao lado, para nossa base de estudos, como novos pontos foram gerados a diferentes distâncias entre dois pontos quaisquer de nossa classe minoritária original.

```
1 # Utilizando o SMOTE
2 from imblearn.over_sampling import SMOTE
3 sm = SMOTE(random_state=42)
4 X_res, y_res = sm.fit_resample(X, y)
```

```
1 # Visualizando graficamente
2 fig, ax = plt.subplots(figsize=(10,10))
3 ax.scatter(base.SaldoConta, base.SaldoInvestimento, c=base.Situacao, linewidths=5)
4 ax.scatter(X_res.SaldoConta, X_res.SaldoInvestimento, c='red', linewidths=3, alpha=0.2)
5
6
7 plt.show()
```

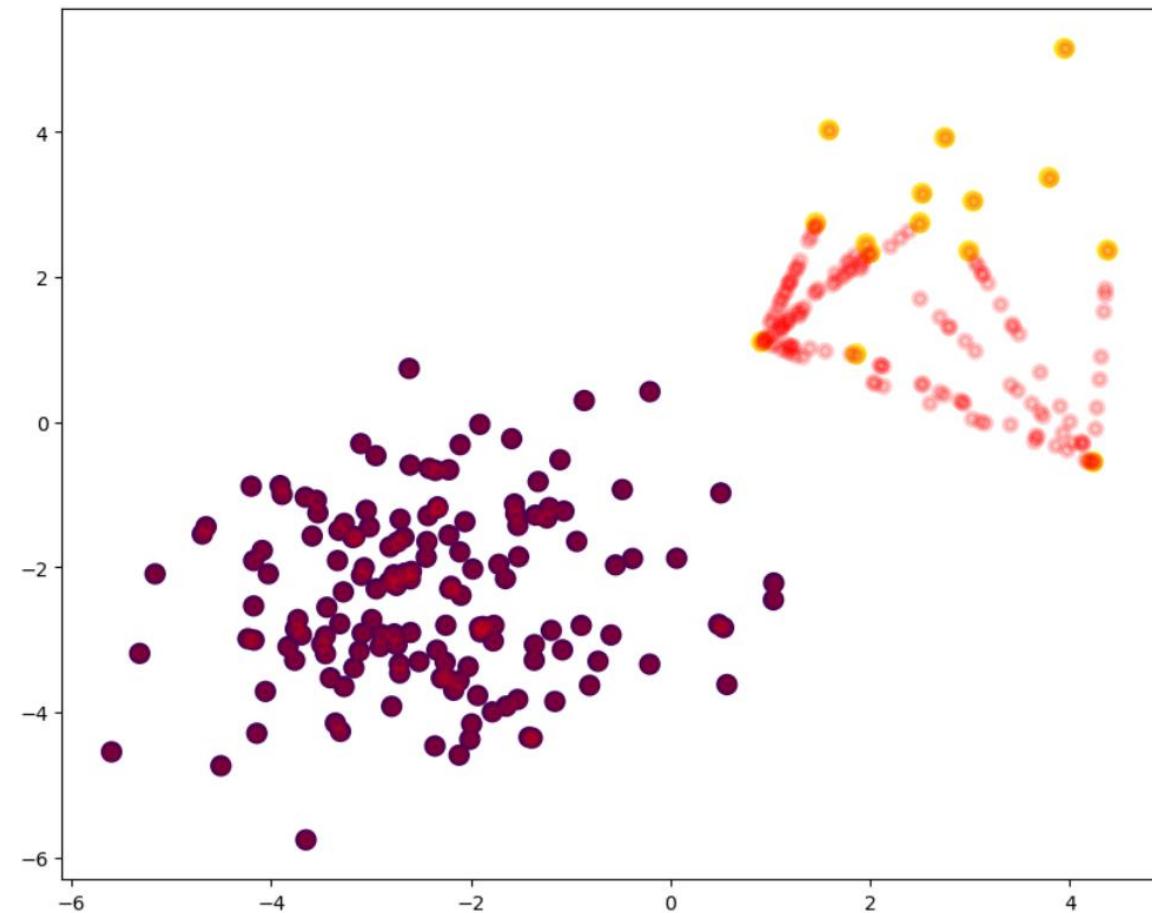


## Módulo 19 – Visualizando as diferentes estratégias

Por fim, a estratégia ADASYN. Esta é similar ao SMOTE, mas irá gerar mais pontos a partir dos pontos originais da classe minoritária que estão mais próximos da classe majoritária. Vemos isto claramente no gráfico, com mais pontos gerados próximo da fronteira entre as duas classes.

```
1 # Utilizando ADASYN
2 from imblearn.over_sampling import ADASYN
3 ada = ADASYN(random_state=42)
4 X_res, y_res = ada.fit_resample(X, y)

1 # Visualizando graficamente
2 fig, ax = plt.subplots(figsize=(10,8))
3
4 ax.scatter(base.SaldoConta, base.SaldoInvestimento, c=base.Situacao, linewidths=5)
5 ax.scatter(X_res.SaldoConta, X_res.SaldoInvestimento, c='red', linewidths=3, alpha=0.2)
6
7 plt.show()
```



## Módulo 19 – Aplicando o RandomUnderSampler em nossa base de dados

Agora que já entendemos o básico de *undersampling* e *oversampling*, vejamos como podem ser aplicados em nossa base de fraudes.

Relembrando, esta é a nossa base:

```
1 # Importando o pandas  
2 import pandas as pd
```

```
1 # Importando a base  
2 transacoes = pd.read_csv('creditcard.csv')
```

```
1 # Visualizando a base  
2 transacoes.head(3)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0

# Módulo 19 – Aplicando o RandomUnderSampler em nossa base de dados

Assim como já feito antes, vamos separar nossa base em treino e teste, mantendo o desbalanceamento com o parâmetro **stratify**:

```
1 # Separando X e y
2 X = transacoes.drop('Class',axis=1)
3 y = transacoes.Class

1 # Separando em treino e teste
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42,stratify=y)

1 # Verificando a distribuição das duas classes na base de treino e teste
2 y_train.value_counts()/y_train.shape[0]

0    0.998271
1    0.001729
Name: Class, dtype: float64

1 # Para a base de teste
2 y_test.value_counts()/y_test.shape[0]

0    0.998276
1    0.001724
Name: Class, dtype: float64

1 y_test.value_counts()

0    93825
1     162
Name: Class, dtype: int64
```

## Módulo 19 – Aplicando o RandomUnderSampler em nossa base de dados

Como discutido anteriormente, o simples fato de ser uma base com muito desbalanceamento gera valores elevados de acurácia e outras métricas:

```
1 # Importando a árvore de decisão
2 from sklearn import tree
```

```
1 # Definindo o nosso classificador
2 clf = tree.DecisionTreeClassifier(random_state=42)
```

```
1 # Fazendo o fit para os dados de treino
2 clf.fit(X_train, y_train)
```

```
1 # Fazendo a previsão
2 y_pred = clf.predict(X_test)
```

```
1 # Importando a acurácia
2 from sklearn.metrics import accuracy_score
```

```
1 # Calculando a acurácia
2 accuracy_score(y_test,y_pred)
```

0.9991594582229457

```
1 # Importando a matriz de confusão
2 from sklearn.metrics import confusion_matrix
```

```
1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test,y_pred)
```

```
array([[93786,      39],
       [    40,     122]])
```

```
1 # Calculando a precisão
2 from sklearn.metrics import precision_score
3 precision_score(y_test,y_pred)
```

0.7577639751552795

```
1 # E o recall
2 from sklearn.metrics import recall_score
3 recall_score(y_test,y_pred)
```

0.7530864197530864

# Módulo 19 – Aplicando o RandomUnderSampler em nossa base de dados

Usando `RandomUnderSampler`, vemos que a acurácia diminui um pouco, a precisão diminui drasticamente, e o recall aumenta significativamente:

```
1 # Importando o RandomUnderSampler do imblearn
2 from imblearn.under_sampling import RandomUnderSampler
```

```
1 # Definindo o RandomUnderSampler
2 rus = RandomUnderSampler(random_state=42)
```

```
1 # Definindo a nova amostra
2 X_res, y_res = rus.fit_resample(X_train, y_train)
```

```
1 # Verificando a quantidade de valores de y
2 y_res.value_counts()
```

```
0    330
1    330
Name: Class, dtype: int64
```

```
1 # Fazendo o fit para os dados de treino já balanceados
2 clfRU = tree.DecisionTreeClassifier(random_state=42)
3 clfRU = clfRU.fit(X_res, y_res)
```

```
1 # Fazendo a previsão para os dados de teste
2 y_predRU = clfRU.predict(X_test)
```

```
1 # Calculando a acurácia
2 accuracy_score(y_test,y_predRU)
```

```
0.9004543181503825
```

```
1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test,y_predRU)
```

```
array([[84484,   9341],
       [   15,   147]])
```

```
1 precision_score(y_test,y_predRU)
```

```
0.015493254637436763
```

```
1 recall_score(y_test,y_predRU)
```

```
0.9074074074074074
```

## Módulo 19 – Aplicando o RandomOverSampler em nossa base de dados

Já com o RandomOverSampler, comparando a matriz de confusão com este método com a do anterior, vemos que há menos acertos dos casos de fraude (137 contra os 147 do RandomUnderSampler).

```
1 # Importando
2 from imblearn.over_sampling import RandomOverSampler

1 # Definindo o ros
2 ros = RandomOverSampler(random_state=42, shrinkage=0.5)

1 # Refazendo nossa amostra
2 X_res, y_res = ros.fit_resample(X_train, y_train)

1 # Fazendo o fit para os dados de treino já平衡ados
2 clfR0 = tree.DecisionTreeClassifier(random_state=42)
3 clfR0 = clfR0.fit(X_res, y_res)

1 # Fazendo a previsão para os dados de teste
2 y_predR0 = clfR0.predict(X_test)

1 # Calculando a acurácia
2 accuracy_score(y_test, y_predR0)

0.9921159309266175

1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test, y_predR0)

array([[93109,    716],
       [   25,   137]])
```

## Módulo 19 – Aplicando o ClusterCentroids e NearMiss em nossa base de dados

Agora, podemos aplicar nesta base outras estratégias de *undersampling* e verificar o impacto. Abaixo, vemos que, tanto o ClusterCentroids quanto o NearMiss melhoram o desempenho de detecção de casos de fraude quando comparados com o RandomUnderSampler:

```
1 # Importando o ClusterCentroids
2 from imblearn.under_sampling import ClusterCentroids

1 # Definindo
2 cc = ClusterCentroids(random_state=42)

1 # Criando a amostra dos dados
2 X_res, y_res = cc.fit_resample(X_train, y_train)

1 # Fazendo o fit para os dados de treino já balanceados
2 clfCC = tree.DecisionTreeClassifier(random_state=42)
3 clfCC = clfCC.fit(X_res, y_res)
4
5 # Fazendo a previsão para os dados de teste
6 y_predCC = clfCC.predict(X_test)
7
8 # Visualizando a matriz de confusão
9 confusion_matrix(y_test,y_predCC)

array([[22799, 71026],
       [    2,   160]])
```

```
1 # Fazendo a reamostragem usando o NearMiss
2 from imblearn.under_sampling import NearMiss
3 nm = NearMiss()
4 X_res, y_res = nm.fit_resample(X_train, y_train)

1 # Fazendo o fit para os dados de treino já平衡ados
2 clfNM = tree.DecisionTreeClassifier(random_state=42)
3 clfNM = clfNM.fit(X_res, y_res)
4
5 # Fazendo a previsão para os dados de teste
6 y_predNM = clfNM.predict(X_test)
7
8 # Visualizando a matriz de confusão
9 confusion_matrix(y_test,y_predNM)

array([[50758, 43067],
       [    9,   153]])
```

## Módulo 19 – Aplicando outras estratégias de oversampling em nossa base de dados

Da mesma forma, podemos alterar a estratégia de *oversampling*. Abaixo, alteramos o valor do parâmetro **shrinkage** do `RandomOverSampler`, e ao lado utilizamos **SMOTE** e **ADASYN**. Observe que em nenhum caso tivemos resultados melhores para detectar fraudes do que quando utilizamos estratégias de *undersampling*.

```
1 # Definindo o ros com shrinkage
2 ros = RandomOverSampler(random_state=42, shrinkage=0.9)
3
4 # Refazendo nossa amostra
5 X_res, y_res = ros.fit_resample(X_train, y_train)
```

```
1 # Fazendo o fit para os dados de treino já平衡ados
2 clfR02 = tree.DecisionTreeClassifier(random_state=42)
3 clfR02 = clfR02.fit(X_res, y_res)
4
5 # Fazendo a previsão para os dados de teste
6 y_predR02 = clfR02.predict(X_test)
7
8 # Visualizando a matriz de confusão
9 confusion_matrix(y_test, y_predR02)
```

```
array([[93355,    470],
       [   26,   136]])
```

```
1 # Utilizando o SMOTE
2 from imblearn.over_sampling import SMOTE
3 sm = SMOTE(random_state=42)
4 X_res, y_res = sm.fit_resample(X_train, y_train)
```

```
1 # Fazendo o fit para os dados de treino já平衡ados
2 clfS = tree.DecisionTreeClassifier(random_state=42)
3 clfS = clfS.fit(X_res, y_res)
4
5 # Fazendo a previsão para os dados de teste
6 y_predS = clfS.predict(X_test)
7
8 # Visualizando a matriz de confusão
9 confusion_matrix(y_test, y_predS)
```

```
array([[93650,    175],
       [   42,   120]])
```

```
1 # Utilizando ADASYN
2 from imblearn.over_sampling import ADASYN
3 ada = ADASYN(random_state=42)
4 X_res, y_res = ada.fit_resample(X_train, y_train)
```

```
1 # Fazendo o fit para os dados de treino já平衡ados
2 clfA = tree.DecisionTreeClassifier(random_state=42)
3 clfA = clfA.fit(X_res, y_res)
4
5 # Fazendo a previsão para os dados de teste
6 y_predA = clfA.predict(X_test)
7
8 # Visualizando a matriz de confusão
9 confusion_matrix(y_test, y_predA)
```

```
array([[93662,    163],
       [   38,   124]])
```

## Módulo 19 – Combinando over e undersampling

Também há métodos que combinam *oversampling* e *undersampling*, como o SMOTEENN. Para fraudes, não teve resultado tão bom quanto as estratégias de *undersampling* mostradas anteriormente.

```
1 # Utilizando o SMOTEENN
2 from imblearn.combine import SMOTEENN
3 sme = SMOTEENN(random_state=42)
4 X_res, y_res = sme.fit_resample(X_train, y_train)
```

```
1 # Fazendo o fit para os dados de treino já balanceados
2 clfSE = tree.DecisionTreeClassifier(random_state=42)
3 clfSE = clfSE.fit(X_res, y_res)
4
5 # Fazendo a previsão para os dados de teste
6 y_predSE = clfSE.predict(X_test)
7
8 # Visualizando a matriz de confusão
9 confusion_matrix(y_test,y_predSE)
```

```
array([[93629,    196],
       [   36,   126]])
```

```
1 y_res.value_counts()
```

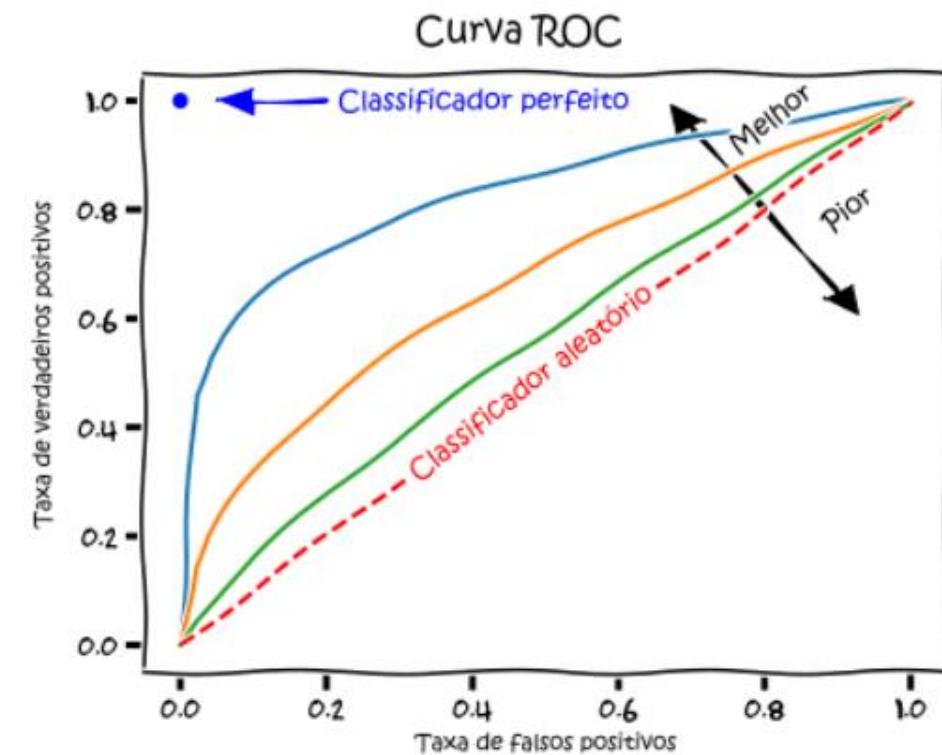
```
1 181952
0 173770
Name: Class, dtype: int64
```

## Módulo 19 – Curva ROC

Quando estávamos estudando, em módulos anteriores, métricas de avaliação, vimos acurácia, precisão e recall. E uma das discussões feitas foi a de que, por vezes, quando se tenta aumentar o recall, há diminuição da precisão, e vice-versa. Assim, seria interessante haver uma métrica que fosse mais objetiva de forma numérica e visual e, ainda, que possibilite comparar diferentes modelos.

Uma dessas métricas é a Curva ROC (*Receiver Operating Characteristic*) ou Característica de Operação do Receptor. Ela ilustra o desempenho de um sistema classificador binário, mostrando a taxa de verdadeiros positivos (o mesmo que recall) versus a taxa de falsos positivos.

No gráfico, vemos que é simples comparar os modelos versus um classificador aleatório, assim como comparar os modelos entre si.



## Módulo 19 – Curva ROC

Para entender melhor, vamos pegar um exemplo. Considere que temos dois modelos que fizeram previsões distintas (`y_pred`, `y_pred2`) para uma base de dados, que possui um conjunto de dados de teste (`y_test`).

Passando cada par (previsão, teste) para o método `roc_curve`, obtemos a taxa de falsos positivos (`fpr`), a taxa de verdadeiros positivos (`tpr`), e os limites (`thresholds`) para cada par. O significado dos limites será mais claro nos exemplos.

```
1 # Importando o metrics do sklearn
2 from sklearn import metrics

1 # Primeiro vamos testar com valores escolhidos por nós de y_test e y_pred
2 y_test = [0,0,1,1]
3 y_pred = [0,1,1,1]
4 y_pred2 = [0,0,0,1]

1 # Calculando os parâmetros da curva ROC
2 fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred)
3 fpr2, tpr2, thresholds2 = metrics.roc_curve(y_test, y_pred2)
```

```
1 # Exibindo o fpr
2 fpr2
array([0., 0., 1.])

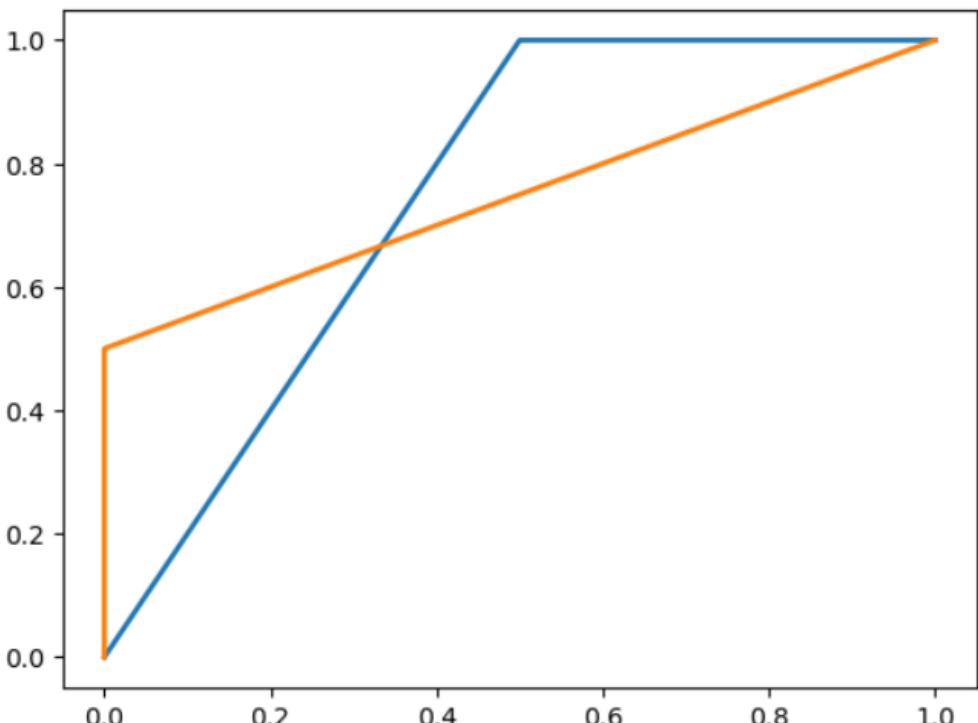
1 # 0 tpr
2 tpr2
array([0., 0.5, 1.])

1 # E os limites
2 thresholds2
array([2, 1, 0])

1 # Calculando a área abaixo dessa curva
2 metrics.roc_auc_score(y_test,y_pred2)
```

0.75

```
1 # Exibindo graficamente
2 import matplotlib.pyplot as plt
3 fig, ax = plt.subplots()
4
5 ax.plot(fpr, tpr, linewidth=2.0)
6 ax.plot(fpr2, tpr2, linewidth=2.0)
7
8 plt.show()
```

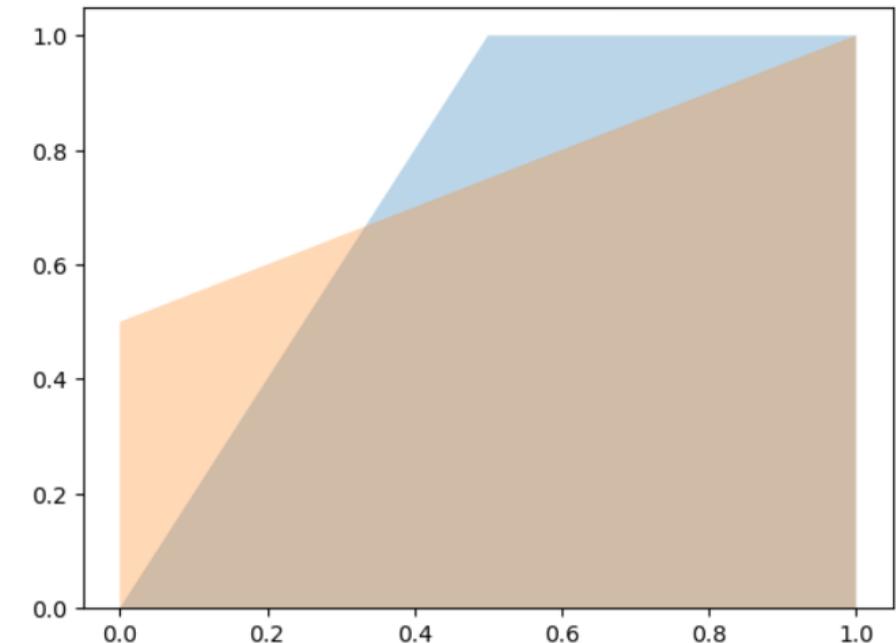


## Módulo 19 – Curva ROC

No gráfico anterior, vemos que as curvas de cruzam. Nestes casos, as áreas sob as curvas servem de comparação entre os modelos. Isto é dado pelo método `roc_auc_score`. AUC é do inglês *area under the curve*, literalmente área sob a curva. No caso, a área dos dois modelos é a mesma, de forma que, por essa métrica, não há diferença entre os modelos.

Com o método `stackplot` do Matplotlib, conseguimos colorir as áreas como no gráfico ao lado.

```
1 # Traçando graficamente
2 fig, ax = plt.subplots()
3
4 ax.stackplot(fpr, tpr, linewidth=2.0,alpha=0.3)
5 ax.stackplot(fpr2, tpr2, linewidth=2.0,alpha=0.3)
6
7 plt.show()
```



## Módulo 19 – Curva ROC

Vamos considerar mais um exemplo. Considere que temos valores que representam um resultado de um exame clínico. Valores iguais ou maiores que um valores específico indicam que o paciente possui uma determinada doença, o que é representado pela classe 1. Do contrário, classe 0.

A base abaixo apresenta valores para alguns pacientes e as classes.

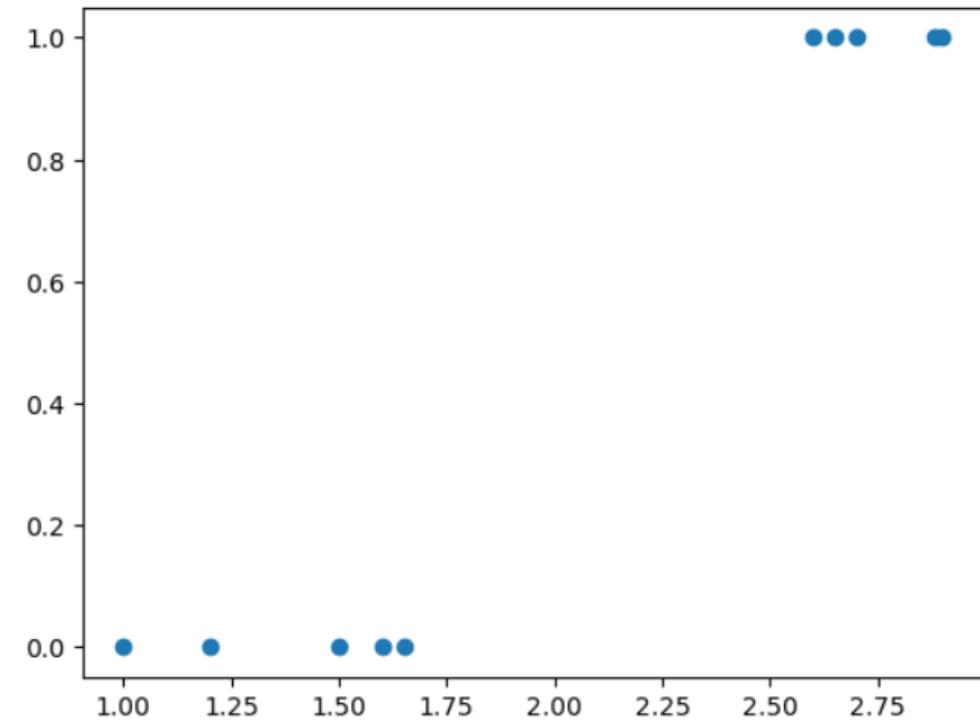
```
1 # Importando o pandas
2 import pandas as pd

1 # Utilizando a base abaixo, vamos transformar em um DataFrame
2 base = {
3     'valores': [1,1.2,1.5,1.6,1.65,2.6,2.65,2.7,2.88,2.9],
4     'classes': [0,0,0,0,0,1,1,1,1,1]
5 }
6
7 base = pd.DataFrame(base)

1 # Separando em X e y
2 X = base.valores
3 y = base.classes

1 # E agora separando em treino e teste
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
1 # Plotando treino e teste
2 fig, ax = plt.subplots()
3
4 ax.scatter(X, y)
5
6 plt.show()
```



## Módulo 19 – Curva ROC

Com uma árvore de decisão, temos que o modelo consegue classificar perfeitamente os casos. Daí o perfil apresentado pela nossa curva ROC. Compare com o gráfico ilustrativo mostrado na primeira página onde se explicou o que era ROC.

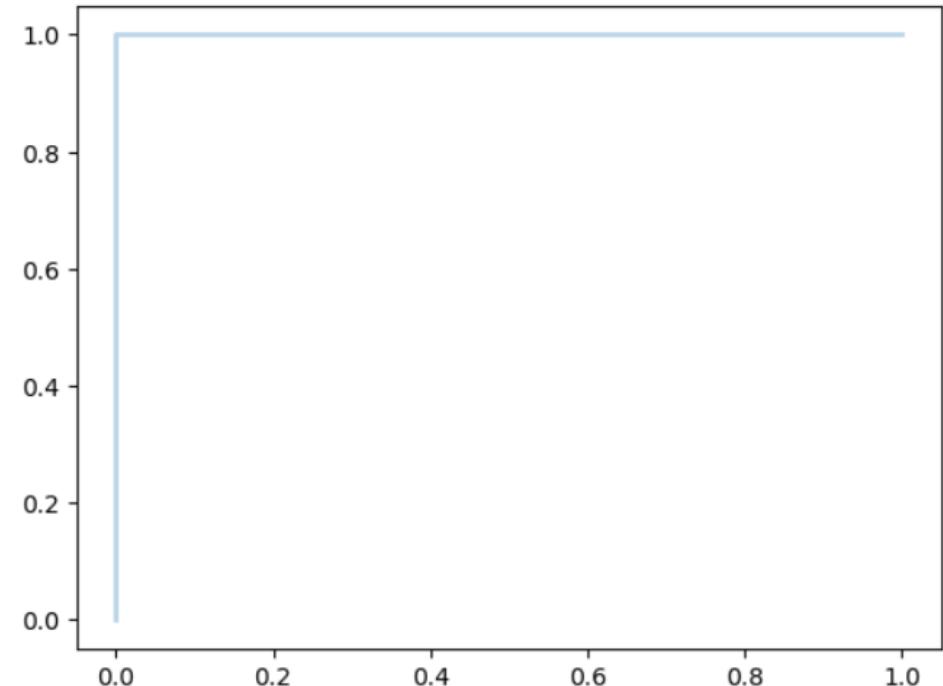
```
1 # Usando a árvore de decisão para fazer a previsão
2 from sklearn import tree
3 clf = tree.DecisionTreeClassifier().fit(X_train.values.reshape(-1, 1), y_train)
```

```
1 # Fazendo a previsão
2 y_pred = clf.predict(X_test.values.reshape(-1, 1))
```

```
1 # Verificando os valores reais
2 y_test
```

```
8 1
1 0
5 1
0 0
Name: classes, dtype: int64
```

```
1 # Traçando a curva
2 # Determinando fpr e tpr
3 fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred)
4
5 # Traçando a curva utilizando os novos valores
6 fig, ax = plt.subplots()
7
8 ax.plot(fpr, tpr, linewidth=2.0, alpha=0.3)
9
10 plt.show()
```



## Módulo 19 – Curva ROC

Considerando agora que há uma mudança na base no que diz respeito a classes, como abaixo. Veja como o gráfico se alterou.

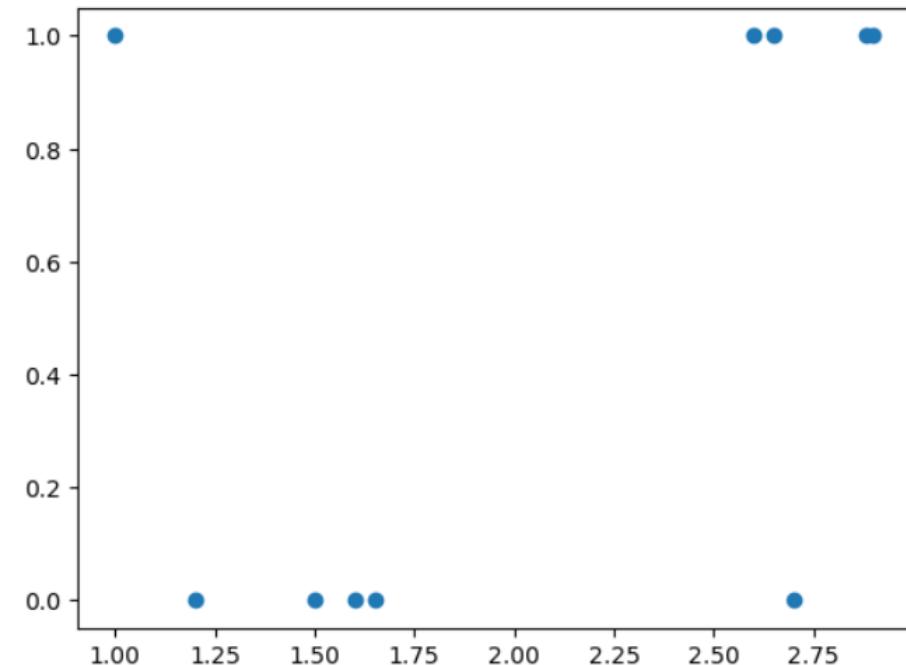
```
1 # Importando o pandas
2 import pandas as pd

1 # Utilizando a base abaixo, vamos transformar em um DataFrame
2 base = {
3     'valores': [1,1.2,1.5,1.6,1.65,2.6,2.65,2.7,2.88,2.9],
4     'classes': [1,0,0,0,1,1,0,1,1]
5 }
6
7 base = pd.DataFrame(base)

1 # Separando em X e y
2 X = base.valores
3 y = base.classes

1 # E agora separando em treino e teste
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
1 # Plotando treino e teste
2 fig, ax = plt.subplots()
3
4 ax.scatter(X, y)
5
6 plt.show()
```



# Módulo 19 – Curva ROC

Agora, vemos que a árvore de decisão não mais acerta todas as previsões, pelo perfil do gráfico. Compare o gráfico com os valores das variáveis *fpr* e *tpr*.

```
1 # Usando a árvore de decisão para fazer a previsão
2 from sklearn import tree
3 clf = tree.DecisionTreeClassifier().fit(X_train.values.reshape(-1, 1),y_train)
```

```
1 # Fazendo a previsão
2 y_pred = clf.predict_proba(X_test.values.reshape(-1, 1))[:,1]
3 y_pred
```

```
array([1., 0., 1., 0.])
```

```
1 # Verificando os valores reais
2 y_test
```

```
8    1
1    0
5    1
0    1
Name: classes, dtype: int64
```

```
1 # Visualizando confusion_matrix
2 metrics.confusion_matrix(y_test,y_pred)
```

```
array([[1, 0],
       [1, 2]])
```

```
1 # Verificando o fpr
2 fpr
```

```
array([0., 0., 1.])
```

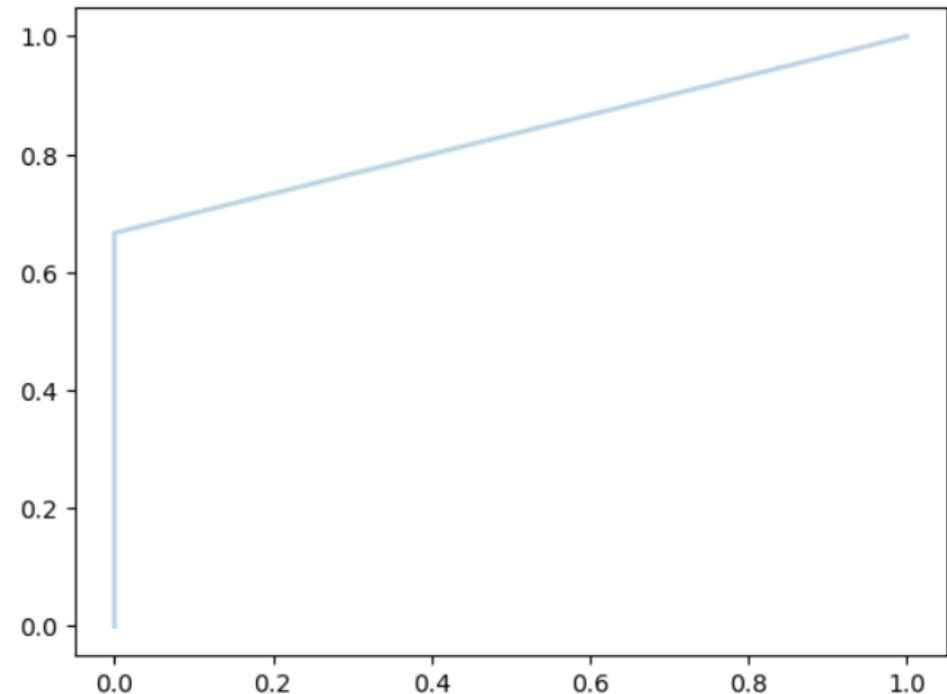
```
1 # O tpr
2 tpr
```

```
array([0.          , 0.66666667, 1.          ])
```

```
1 # E os limites
2 thresholds
```

```
array([2., 1., 0.])
```

```
1 # Traçando a curva
2 # Determinando fpr e tpr
3 fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred)
4
5 # Traçando a curva utilizando os novos valores
6 fig, ax = plt.subplots()
7
8 ax.plot(fpr, tpr, linewidth=2.0,alpha=0.3)
9
10 plt.show()
```



## Módulo 19 – Curva ROC

Vamos utilizar outro modelo para podermos visualizar um exemplo de comparação entre modelos. No caso, vamos usar regressão logística, um modelo já visto em módulos anteriores.

Com este modelo, além da previsão dos rótulos, também podemos obter a probabilidade de cada uma das classes. Temos três métodos de interesse:

- **predict**: previsão do rótulo das classes
- **predict\_proba**: estima a probabilidade de cada uma das classes
- **decision\_function**: prevê pontuações de confiança para as amostras, que é proporcional à distância da amostra ao hiperplano que separa as classes

Mais adiante detalharemos mais este modelo e cada um dos métodos.

```
1 # Utilizando a regressão logística
2 from sklearn.linear_model import LogisticRegression
3 clf2 = LogisticRegression(random_state=0).fit(X_train.values.reshape(-1,1), y_train)

1 # Fazendo o predict
2 y_predRL = clf2.predict(X_test.values.reshape(-1,1))

1 # Agora visualizando o predict_proba
2 clf2.predict_proba(X_test.values.reshape(-1,1))

array([[0.53406853, 0.46593147],
       [0.82401235, 0.17598765],
       [0.59170863, 0.40829137],
       [0.84700557, 0.15299443]])

1 # E então calculando a decision_function
2 y_predRL = clf2.decision_function(X_test.values.reshape(-1,1))

1 # Alterando a decision_function pelo predict_proba
2 y_predRL = clf2.predict_proba(X_test.values.reshape(-1,1))[:,1]
3 y_predRL

array([0.46593147, 0.17598765, 0.40829137, 0.15299443])
```

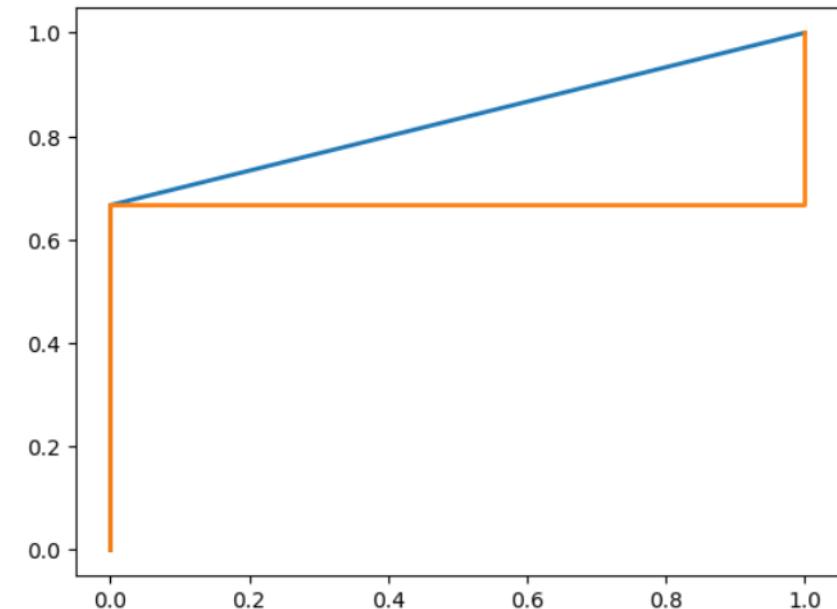
# Módulo 19 – Curva ROC

Agora, podemos comparar os dois modelos:

```
1 # E novamente visualizando os parâmetros
2 fprRL
array([0., 0., 0., 1., 1.])
1 tprRL
array([0.          , 0.33333333, 0.66666667, 0.66666667, 1.        ])
1 thresholdsRL
array([1.46593147, 0.46593147, 0.40829137, 0.17598765, 0.15299443])
```

```
1 # E então calculando a roc_curve para a regressão logística
2 fprRL, tprRL, thresholdsRL = metrics.roc_curve(y_test, y_predRL)
```

```
1 # Traçando a árvore de decisão e a regressão logística
2 fig, ax = plt.subplots()
3
4 ax.plot(fpr, tpr, linewidth=2.0)
5 ax.plot(fprRL, tprRL, linewidth=2.0)
6
7 plt.show()
```

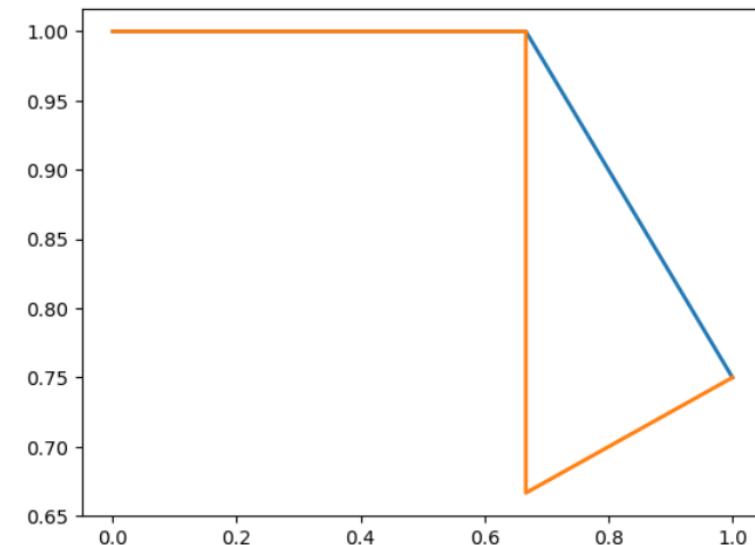


# Módulo 19 – Curva PRC

Além da curva ROC, também existe a curva PRC, das iniciais em inglês de Curva de Precisão Recall. Esta curva tem maior relevância em casos de bases desbalanceadas. A linha de raciocínio do código é bem similar a do ROC, mudando o método para `precision_recall_curve` e calculando a área pelo método `auc`, para o qual se passa os valores de recall e precisão. No nosso exemplo, a árvore de decisão possui maior área, sendo um modelo melhor por esta métrica.

```
1 # Verificando a precisão
2 precisionRL
array([0.75      , 0.66666667, 1.        , 1.        , 1.        ])
1 # 0 recall
2 recallRL
array([1.        , 0.66666667, 0.66666667, 0.33333333, 0.        ])
1 # E os limites
2 thresholdsRL
array([0.15299443, 0.17598765, 0.40829137, 0.46593147])
1 # Arvore de Decisao
2 metrics.auc(recall,precision)
0.958333333333333
1 # Regressão Log.
2 metrics.auc(recallRL,precisionRL)
0.9027777777777777
```

```
1 # Calculando a curva
2 precision, recall, thresholds = metrics.precision_recall_curve(y_test, y_pred)
3 precisionRL, recallRL, thresholdsRL = metrics.precision_recall_curve(y_test, y_predRL)
1 # Traçando a árvore de decisão e a regressão logística
2 fig, ax = plt.subplots()
3
4 ax.plot(recall, precision, linewidth=2.0)
5 ax.plot(recallRL, precisionRL, linewidth=2.0)
6
7 plt.show()
```



# Módulo 19 – Comparando diferentes modelos em nossa base de dados

Agora que vimos métricas e como comparar diferentes modelos, podemos voltar a trabalhar com nossa base de fraudes.

Como de costume, vamos importar bibliotecas, a base, e separar os dados de treino e de teste. Para referência, treinamos um modelo de árvore de decisão:

```
1 # Importando o pandas  
2 import pandas as pd
```

```
1 # Importando a base  
2 transacoes = pd.read_csv('creditcard.csv')
```

```
1 # Separando X e y  
2 X = transacoes.drop('Class',axis=1)  
3 y = transacoes.Class
```

```
1 # Separando em treino e teste  
2 from sklearn.model_selection import train_test_split  
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42,stratify=y)
```

```
1 # Importando a árvore de decisão  
2 from sklearn import tree
```

```
1 # Definindo o nosso classificador  
2 clf = tree.DecisionTreeClassifier(random_state=42)
```

```
1 # Fazendo o fit para os dados de treino  
2 clf = clf.fit(X_train, y_train)
```

```
1 # Prevendo os valores  
2 y_pred = clf.predict(X_test)
```

```
1 from sklearn.metrics import confusion_matrix  
2 confusion_matrix(y_test,y_pred)
```

```
array([[93786,      39],  
       [   40,    122]])
```

## Módulo 19 – Comparando diferentes modelos em nossa base de dados

Durante o desenvolvimento deste módulo, vimos que estratégias de *undersampling* melhoraram a detecção de casos de fraude. Assim, vamos treinar novamente uma árvore de decisão utilizando o RandomUnderSampler antes:

```
1 # Importando o RandomUnderSampler do imblearn
2 from imblearn.under_sampling import RandomUnderSampler

1 # Definindo o RandomUnderSampler
2 rus = RandomUnderSampler(random_state=42)

1 # Definindo a nova amostra
2 X_resRU, y_resRU = rus.fit_resample(X_train, y_train)

1 # Fazendo o fit para os dados de treino já平衡ados
2 clf_AD = tree.DecisionTreeClassifier(random_state=42).fit(X_resRU, y_resRU)

1 # Fazendo a previsão para os dados de teste
2 y_pred_AD = clf_AD.predict(X_test)

1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test,y_pred_AD)

array([[84484,   9341],
       [   15,   147]])
```

# Módulo 19 – Comparando diferentes modelos em nossa base de dados

Para comparação, vamos treinar mais 4 modelos: regressão logística, vetor suporte, KNN e random forest:

```
1 # Fazendo o fit
2 from sklearn.linear_model import LogisticRegression
3 clf_RL = LogisticRegression(random_state=0).fit(X_resRU, y_resRU)
```

```
1 # Fazendo a previsão
2 y_pred_RL = clf_RL.predict(X_test)
```

```
1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test,y_pred_RL)
```

```
array([[89362,  4463],
       [ 19,   143]])
```

```
1 # Fazendo o fit
2 from sklearn.neighbors import KNeighborsClassifier
3 clf_KNN = KNeighborsClassifier(n_neighbors=3).fit(X_resRU, y_resRU)
```

```
1 # Fazendo a previsão
2 y_pred_KNN = clf_KNN.predict(X_test)
```

```
1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test,y_pred_KNN)..
```

```
array([[60332, 33493],
       [ 54,  108]])
```

```
1 # Fazendo o fit
2 from sklearn.svm import SVC
3 clf_SVC = SVC(random_state=0).fit(X_resRU, y_resRU)
```

```
1 # Fazendo a previsão
2 y_pred_SVC = clf_RL.predict(X_test)
```

```
1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test,y_pred_SVC)
```

```
array([[89362,  4463],
       [ 19,   143]])
```

```
1 # Fazendo o fit
2 from sklearn.neighbors import KNeighborsClassifier
3 clf_KNN = KNeighborsClassifier(n_neighbors=3).fit(X_resRU, y_resRU)
```

```
1 # Fazendo a previsão
2 y_pred_KNN = clf_KNN.predict(X_test)
```

```
1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test,y_pred_KNN)..
```

```
array([[60332, 33493],
       [ 54,  108]])
```

```
1 # Fazendo o fit
2 from sklearn.ensemble import RandomForestClassifier
3 clf_RF = RandomForestClassifier(max_depth=2, random_state=0).fit(X_resRU, y_resRU)
```

```
1 # Fazendo a previsão
2 y_pred_RF = clf_RF.predict(X_test)
```

```
1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test,y_pred_RF)
```

```
array([[93168,   657],
       [ 25,  137]])
```

# Módulo 19 – Comparando diferentes modelos em nossa base de dados

Agora, basta comparar as diversas métricas dos modelos. Se formos utilizar a área sob a curva PRC, o melhor modelo é o de regressão logística:

```
1 # Importar o metrics do sklearn
2 from sklearn import metrics
```

```
1 # Visualizando precisão e recall
2 print(metrics.precision_score(y_test,y_pred_AD))
3 print(metrics.recall_score(y_test,y_pred_AD))
4 print(metrics.precision_score(y_test,y_pred_RL))
5 print(metrics.recall_score(y_test,y_pred_RL))
6 print(metrics.precision_score(y_test,y_pred_SVC))
7 print(metrics.recall_score(y_test,y_pred_SVC))
8 print(metrics.precision_score(y_test,y_pred_KNN))
9 print(metrics.recall_score(y_test,y_pred_KNN))
10 print(metrics.precision_score(y_test,y_pred_RF))
11 print(metrics.recall_score(y_test,y_pred_RF))
```

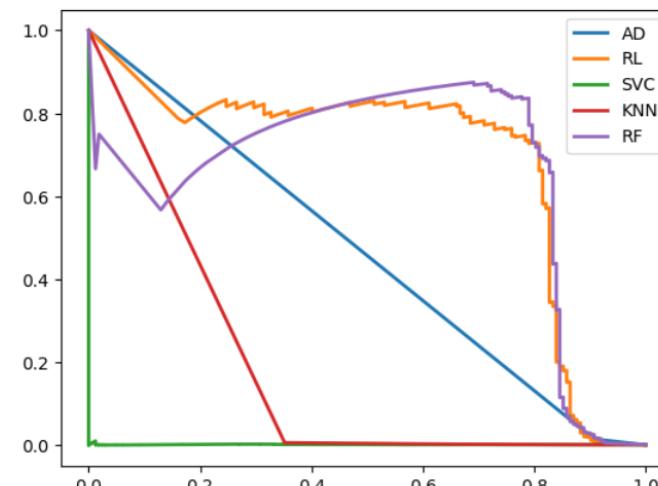
```
0.015493254637436763
0.9074074074074074
0.031046461137646548
0.8827160493827161
0.002222860975376066
0.7160493827160493
0.0032141900538674444
0.6666666666666666
0.172544080604534
0.845679012345679
```

```
1 # Avaliando a área abaixo da curva de cada um dos casos
2 print(metrics.auc(recall_AD, precision_AD))
3 print(metrics.auc(recall_RL, precision_RL))
4 print(metrics.auc(recall_SVC, precision_SVC))
5 print(metrics.auc(recall_KNN, precision_KNN))
6 print(metrics.auc(recall_RF, precision_RF))
```

```
0.46153012929239556
0.6857094208704474
0.0022305388405119003
0.17950296059526705
0.6504912358563952
```

```
1 # Gerando os parâmetros da curva precisão x recall para a base
2 precision_AD,recall_AD,thresholds_AD = metrics.precision_recall_curve(y_test,y_pred_proba_AD)
3 precision_RL,recall_RL,thresholds_RL = metrics.precision_recall_curve(y_test,y_pred_proba_RL)
4 precision_SVC,recall_SVC,thresholds_SVC = metrics.precision_recall_curve(y_test,y_pred_proba_SVC)
5 precision_KNN,recall_KNN,thresholds_KNN = metrics.precision_recall_curve(y_test,y_pred_proba_KNN)
6 precision_RF,recall_RF,thresholds_RF = metrics.precision_recall_curve(y_test,y_pred_proba_RF)
```

```
1 # Visualizando graficamente
2 import matplotlib.pyplot as plt
3 fig, ax = plt.subplots()
4
5 ax.plot(recall_AD, precision_AD, linewidth=2.0,label='AD')
6 ax.plot(recall_RL, precision_RL, linewidth=2.0,label='RL')
7 ax.plot(recall_SVC, precision_SVC, linewidth=2.0,label='SVC')
8 ax.plot(recall_KNN, precision_KNN, linewidth=2.0,label='KNN')
9 ax.plot(recall_RF, precision_RF, linewidth=2.0,label='RF')
10
11 ax.legend()
12
13 plt.show()
```



## Módulo 19 – Normalizando dados

Quando lidamos com dados numéricos, por vezes os modelos performam melhor se a faixa de valores dos dados for normalizada. Em nossa base, as colunas de tempo e valor destoam das demais. Podemos normalizá-las entre 0 e 1 e verificar se os modelos performam melhor:

```
1 # Primeiro para a coluna Time  
2 transacoes.Time = transacoes.Time / transacoes.Time.max()  
  
1 # E então para a coluna Amount  
2 transacoes.Amount = transacoes.Amount / transacoes.Amount.max()  
  
1 # Importar o metrics do sklearn  
2 from sklearn import metrics  
  
1 # Visualizando precisão e recall  
2 print(metrics.precision_score(y_test,y_pred_AD))  
3 print(metrics.recall_score(y_test,y_pred_AD))  
4 print(metrics.precision_score(y_test,y_pred_RL))  
5 print(metrics.recall_score(y_test,y_pred_RL))  
6 print(metrics.precision_score(y_test,y_pred_SVC))  
7 print(metrics.recall_score(y_test,y_pred_SVC))  
8 print(metrics.precision_score(y_test,y_pred_KNN))  
9 print(metrics.recall_score(y_test,y_pred_KNN))  
10 print(metrics.precision_score(y_test,y_pred_RF))  
11 print(metrics.recall_score(y_test,y_pred_RF))  
  
0.017826194602624412  
0.8888888888888888  
0.0552423900789177  
0.9074074074074074  
0.103397341211226  
0.8641975308641975  
0.0525164113785558  
0.8888888888888888  
0.13908629441624365  
0.845679012345679
```

# Módulo 19 – Normalizando dados

Agora, podemos avaliar as métricas. Observe como alguns modelos, em especial o SVC, melhorou significativamente o valor de AUPRC com a normalização dos valores.

```
1 # Gerando os parâmetros da curva precisão x recall para a base
2 precision_AD,recall_AD,thresholds_AD = metrics.precision_recall_curve(y_test,y_pred_proba_AD)
3 precision_RL,recall_RL,thresholds_RL = metrics.precision_recall_curve(y_test,y_pred_proba_RL)
4 precision_SVC,recall_SVC,thresholds_SVC = metrics.precision_recall_curve(y_test,y_pred_proba_SVC)
5 precision_KNN,recall_KNN,thresholds_KNN = metrics.precision_recall_curve(y_test,y_pred_proba_KNN)
6 precision_RF,recall_RF,thresholds_RF = metrics.precision_recall_curve(y_test,y_pred_proba_RF)
```

```
1 # Avaliando a área abaixo da curva de cada um dos casos após a mudança
2 print(metrics.auc(recall_AD, precision_AD))
3 print(metrics.auc(recall_RL, precision_RL))
4 print(metrics.auc(recall_SVC, precision_SVC))
5 print(metrics.auc(recall_KNN, precision_KNN))
6 print(metrics.auc(recall_RF, precision_RF))
```

```
0.4534532996697248
0.6949903066783492
0.6847172261858195
0.5534876929920411
0.6553157871101132
```

```
1 metrics.confusion_matrix(y_test,y_pred_RL)
```

```
array([[91311,  2514],
       [   15,  147]])
```

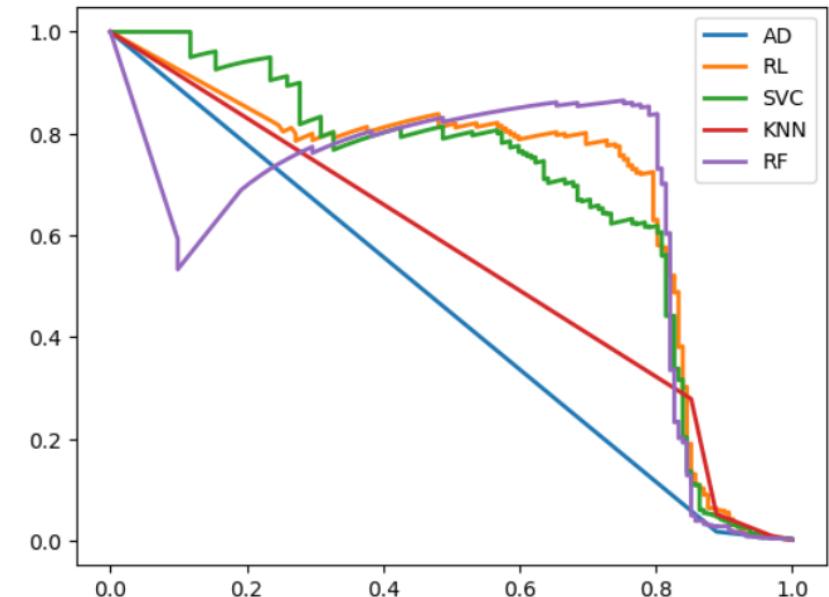
```
1 metrics.confusion_matrix(y_test,y_pred_RF)
```

```
array([[92977,   848],
       [   25,  137]])
```

```
1 metrics.confusion_matrix(y_test,y_pred_SVC)
```

```
array([[92611,  1214],
       [   22,  140]])
```

```
1 # Visualizando graficamente
2 import matplotlib.pyplot as plt
3 fig, ax = plt.subplots()
4
5 ax.plot(recall_AD, precision_AD, linewidth=2.0,label='AD')
6 ax.plot(recall_RL, precision_RL, linewidth=2.0,label='RL')
7 ax.plot(recall_SVC, precision_SVC, linewidth=2.0,label='SVC')
8 ax.plot(recall_KNN, precision_KNN, linewidth=2.0,label='KNN')
9 ax.plot(recall_RF, precision_RF, linewidth=2.0,label='RF')
10
11 ax.legend()
12
13 plt.show()
```



# Módulo 19 – Testando diferentes hiperparâmetros

Do visto até o momento, sabemos que, para nossa base de dados, normalizar os valores de tempo e gasto melhora os modelos, assim como utilizar alguma estratégia de *undersampling*.

No entanto, durante todo o curso incentivamos a leitura das documentações dos métodos. Assim, você já deve ter percebido que os métodos possuem diversos parâmetros que podem ser ajustados.

O Scikit-Learn possui formas práticas de testar múltiplos parâmetros por modelo, de forma que podemos verificar qual combinação melhora uma determinada métrica.

```
1 # Importando o pandas
2 import pandas as pd

1 # Importando a base
2 transacoes = pd.read_csv('creditcard.csv')

1 # Separando em treino e teste
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0, stratify=y)

1 # Primeiro para a coluna Time
2 transacoes.Time = transacoes.Time / transacoes.Time.max()

1 # Então para a coluna Amount
2 transacoes.Amount = transacoes.Amount / transacoes.Amount.max()

1 # Separando X e y
2 X = transacoes.drop('Class', axis=1)
3 y = transacoes.Class

1 # Importando o RandomUnderSampler do imblearn
2 from imblearn.under_sampling import RandomUnderSampler

1 # Definindo o RandomUnderSampler
2 rus = RandomUnderSampler(random_state=42)

1 # Definindo a nova amostra
2 X_resRU, y_resRU = rus.fit_resample(X_train, y_train)
```

## Módulo 19 – Testando diferentes hiperparâmetros

Para referência, vamos treinar um modelo de regressão logística sem modificar os parâmetros disponíveis. Ou seja, deixando-os com seus valores padrão.

Obtivemos uma área sob a curva PRC de 0,69 e um recall de 0,91.

```
1 # Fazendo o fit
2 from sklearn.linear_model import LogisticRegression
3 clf_RL = LogisticRegression(random_state=42).fit(X_resRU, y_resRU)

1 # Fazendo a previsão
2 y_pred_RL = clf_RL.predict(X_test)
3 y_pred_proba_RL = clf_RL.predict_proba(X_test)[:,1]

1 # Visualizando a matriz de confusão
2 from sklearn.metrics import confusion_matrix
3 confusion_matrix(y_test,y_pred_RL)

array([[91311,  2514],
       [  15,   147]])

1 # Traçando a área sobre a curva precisão x recall
2 from sklearn import metrics
3 precision_RL,recall_RL,thresholds_RL = metrics.precision_recall_curve(y_test,y_pred_proba_RL)
4 print(metrics.auc(recall_RL, precision_RL))

0.6949903066783492

1 from sklearn.metrics import recall_score
2 recall_score(y_test,y_pred_RL)

0.9074074074074
```

# Módulo 19 – Testando diferentes hiperparâmetros

Na documentação, vemos que há diversos parâmetros e suas respectivas explicações. Vamos pegar, por exemplo, os parâmetros `solver` e `C` e modificar seus valores com base nos possíveis pela documentação.

Observe que obtivemos valores maiores de AUPRC e de recall.

Mas será que ainda é possível melhorar? E se alterarmos os demais parâmetros?

```
1 # Fazendo o fit
2 from sklearn.linear_model import LogisticRegression
3 clf_RL2 = LogisticRegression(random_state=42,
4                               solver='newton-cg', C=100
5                               ).fit(X_resRU, y_resRU)
```

```
1 # Fazendo a previsão
2 y_pred_RL2 = clf_RL2.predict(X_test)
3 y_pred_proba_RL2 = clf_RL2.predict_proba(X_test)[:,1]
```

```
1 confusion_matrix(y_test,y_pred_RL2)
```

```
array([[90223, 3602],
       [ 14, 148]])
```

```
1 precision_RL2,recall_RL2,thresholds_RL2 = metrics.precision_recall_curve(y_test,y_pred_proba_RL2)
2 print(metrics.auc(recall_RL2, precision_RL2))
```

```
0.74148927073376
```

```
1 recall_score(y_test,y_pred_RL2)
```

```
0.9135802469135802
```

## Módulo 19 – Seleção de hiperparâmetros com GridSearchCV

O Scikit-Learn possui a classe `GridsearchCV` para a qual podemos passar um modelo, um dicionário com os parâmetros e os valores a serem testados, e qual a métrica a ser maximizada.

O que o `GridsearchCV` fará é a combinação de todos os parâmetros possíveis e retornar a combinação que resulta em um maior valor da métrica selecionada.

No exemplo, dentre os parâmetros passados, a combinação que maximiza a métrica recall é a mostrada no dicionário resultante da última célula da imagem ao lado:

```
1 # Importando o GridSearchCV
2 from sklearn.model_selection import GridSearchCV

1 # Definindo os parâmetros que queremos testar
2 parametros = {
3     'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
4     'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
5 }

1 # Selecionando a Regressão Logística
2 LogReg = LogisticRegression(random_state=42)

1 # Criando um novo classificador usando os parâmetros que escolhemos anteriormente
2 clf_GS = GridSearchCV(LogReg, parametros,
3                       scoring='recall'
4                     )

1 # Fazendo o fit dos nossos dados
2 clf_GS = clf_GS.fit(X_resRU,y_resRU)

1 # Visualizando os melhores parâmetros definidos pelo GridSearchCV-
2 clf_GS.best_params_
```

{'C': 0.001, 'solver': 'liblinear'}

## Módulo 19 – Seleção de hiperparâmetros com GridSearchCV

Observe que, realmente, o valor de recall é maior que os dois exibidos anteriormente. No entanto, é preciso ficar atento que, ao escolher maximizar uma métrica, outras podem ser significativamente penalizadas. Veja que a precisão ficou com um valor bem baixo.

```
1 # Usando esse modelo para fazer as previsões  
2 y_pred_GS = clf_GS.predict(X_test)
```

```
1 # Analisando a matriz de confusão  
2 confusion_matrix(y_test,y_pred_GS)
```

```
array([[85217,  8608],  
       [   10,   152]])
```

```
1 # O recall  
2 recall_score(y_test,y_pred_GS)
```

0.9382716049382716

```
1 # E a precisão  
2 metrics.precision_score(y_test,y_pred_GS)
```

0.017351598173515982

# Módulo 19 – Seleção de hiperparâmetros com GridSearchCV

É possível exportar os resultados de todas as combinações feitas caso queira maiores detalhes:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_solver	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
0	0.007727	0.000270	0.002411	0.000134	0.001	newton-cg	{"C": 0.001, "solver": "newton-cg"}	0.757576	0.848485	0.863636	0.803030	0.863636	0.827273	0.041328	33
1	0.004647	0.000369	0.002354	0.000090	0.001	lbfgs	{"C": 0.001, "solver": "lbfgs"}	0.757576	0.848485	0.863636	0.803030	0.863636	0.827273	0.041328	33
2	0.002888	0.000173	0.002224	0.000081	0.001	liblinear	{"C": 0.001, "solver": "liblinear"}	0.909091	0.969697	0.954545	0.924242	0.954545	0.942424	0.022268	1
3	0.022456	0.000090	0.002363	0.000036	0.001	sag	{"C": 0.001, "solver": "sag"}	0.757576	0.848485	0.863636	0.803030	0.863636	0.827273	0.041328	33
4	0.020007	0.003609	0.002016	0.000235	0.001	saga	{"C": 0.001, "solver": "saga"}	0.772727	0.863636	0.863636	0.803030	0.893939	0.839394	0.044536	32
5	0.008661	0.000369	0.002344	0.000072	0.01	newton-cg	{"C": 0.01, "solver": "newton-cg"}	0.833333	0.939394	0.878788	0.848485	0.893939	0.878788	0.037113	30

```
1 # E até exportar para um Excel  
2 pd.DataFrame(clf_GS.cv_results_).to_excel('visualizar.xlsx')
```

# Módulo 19 – Seleção de hiperparâmetros com GridSearchCV

Tudo que vimos pode ser facilmente replicado para outros modelos. Basta passarmos um dicionário com parâmetros que façam sentido para o modelo escolhido. Abaixo, a esquerda, temos o modelo SVC padrão; a direita, o modelo após uso do GridSearchCV buscando otimizar a métrica de recall. Observe como a matriz de confusão resultante é diferente.

```
1 # Importando o RandomUnderSampler do imblearn
2 from imblearn.under_sampling import RandomUnderSampler
```

```
1 # Definindo o RandomUnderSampler
2 rus = RandomUnderSampler(random_state=42)
```

```
1 # Definindo a nova amostra
2 X_resRU, y_resRU = rus.fit_resample(X_train, y_train)
```

```
1 # Fazendo o fit
2 from sklearn.svm import SVC
3 clf_SVC = SVC(random_state=0,probability=True).fit(X_resRU, y_resRU)
```

```
1 # Fazendo a previsão
2 y_pred_SVC = clf_SVC.predict(X_test)
3 y_pred_proba_SVC = clf_SVC.predict_proba(X_test)[:,1]
```

```
1 # Visualizando a matriz de confusão
2 from sklearn.metrics import confusion_matrix
3 confusion_matrix(y_test,y_pred_SVC)
```

```
array([[92611, 1214],
       [ 22, 140]])
```

```
1 # Traçando a área sobre a curva precisão x recall
2 from sklearn import metrics
3 precision_SVC,recall_SVC,thresholds_SVC = metrics.precision_recall_curve(y_test,y_pred_proba_SVC)
4 print(metrics.auc(recall_SVC, precision_SVC))
```

```
0.6847172261858195
```

```
1 # Importando o GridSearchCV
2 from sklearn.model_selection import GridSearchCV
```

```
1 # Utilizando os parâmetros do SVC
2 parametros = {
3     'C': [0.001,0.01,0.1,1,10,100,1000],
4     'kernel': ['linear','poly','rbf','sigmoid'],
5     'gamma': ['scale','auto']
6 }
```

```
1 # Selecionando a SVC
2 svc_GS = SVC(random_state=0,probability=True)
```

```
1 # Criando um novo classificador usando os parâmetros que escolhemos anteriormente
2 clf_GS = GridSearchCV(svc_GS,parametros,
3                       scoring='recall'
4 )
```

```
1 # Fazendo o fit dos nossos dados
2 clf_GS.fit(X_resRU, y_resRU)
```

```
1 # Visualizando os melhores parâmetros definidos pelo GridSearchCV
2 clf_GS.best_params_
{'C': 100, 'gamma': 'auto', 'kernel': 'rbf'}
```

```
1 # Usando esse modelo para fazer as previsões
2 y_pred_GS = clf_GS.predict(X_test)
```

```
1 # Analisando a matriz de confusão
2 confusion_matrix(y_test,y_pred_GS)
```

```
array([[85791, 8034],
       [ 12, 150]])
```

# Módulo 19 – Seleção de hiperparâmetros com GridSearchCV

O mesmo agora sendo feito para o modelo Random Forest:

```
1 # Fazendo o fit
2 from sklearn.ensemble import RandomForestClassifier
3 clf_RF = RandomForestClassifier(max_depth=2, random_state=0).fit(X_resRU, y_resRU)

1 # Fazendo a previsão
2 y_pred_RF = clf_RF.predict(X_test)
3 y_pred_proba_RF = clf_RF.predict_proba(X_test)[:,1]

1 # Visualizando a matriz de confusão
2 confusion_matrix(y_test,y_pred_RF)

array([[92977,    848],
       [   25,   137]])
```

```
1 # Utilizando os parâmetros do Random Forest
2 parametros = {
3     'n_estimators': [10,20,30,40,50,100,200],
4     'criterion': ['gini','entropy','log_loss'],
5     'max_depth': [2,3,4,5,6,7,8,9]
6 }
```

```
1 # Selecionando o Random Forest
2 RF_GS = RandomForestClassifier(random_state=0)
```

```
1 # Criando um novo classificador usando os parâmetros que escolhemos anteriormente
2 clf_GS2 = GridSearchCV(RF_GS,parametros,
3                         scoring='recall'
4 )
```

```
1 # Fazendo o fit dos nossos dados
2 clf_GS2 = clf_GS2.fit(X_resRU,y_resRU)
```

```
1 # Visualizando os melhores parâmetros definidos pelo GridSearchCV
2 clf_GS2.best_params_
```

```
{'criterion': 'gini', 'max_depth': 9, 'n_estimators': 30}
```

```
1 # Usando esse modelo para fazer as previsões
2 y_pred_RF = clf_GS2.predict(X_test)
```

```
1 # Analisando a matriz de confusão
2 confusion_matrix(y_test,y_pred_RF)
```

```
array([[91476,    2349],
       [   19,   143]])
```

```
1 metrics.recall_score(y_test,y_pred_RF)

0.8827160493827161
```

# Módulo 19 – Seleção de hiperparâmetros com GridSearchCV

No caso do KNN, vemos que o resultado da busca pelos melhores parâmetros acabou chegando no mesmo que o modelo com os valores padrão:

```
1 # Fazendo o fit
2 from sklearn.neighbors import KNeighborsClassifier
3 clf_KNN = KNeighborsClassifier(n_neighbors=3).fit(X_resRU, y_resRU)
```

```
1 # Fazendo a previsão
2 y_pred_KNN = clf_KNN.predict(X_test)
3 y_pred_proba_KNN = clf_KNN.predict_proba(X_test)[:,1]
```

```
1 # Visualizando a matriz de confusão
2 from sklearn.metrics import confusion_matrix
3 confusion_matrix(y_test,y_pred_KNN)**
```

```
array([[91227,  2598],
       [   18,   144]])
```

```
1 # Traçando a área sobre a curva precisão x recall
2 from sklearn import metrics
3 precision_KNN,recall_KNN,thresholds_KNN = metrics.precision_recall_curve(y_test,y_pred_proba_KNN)
4 print(metrics.auc(recall_KNN, precision_KNN))
```

```
0.5534876929920411
```

```
1 # Importando o GridSearchCV
2 from sklearn.model_selection import GridSearchCV
```

```
1 # Utilizando os parâmetros do KNN
2 parametros = {
3     'n_neighbors': [3,5,7,9,11],
4     'algorithm': ['auto','ball_tree','kd_tree','brute'],
5     'weights': ['uniform','distance']
6 }
```

```
1 # Selecionando o KNN
2 KNN_GS = KNeighborsClassifier()
```

```
1 # Criando um novo classificador usando os parâmetros que escolhemos anteriormente
2 clf_GS = GridSearchCV(KNN_GS, parametros,
3                       scoring='recall'
4 )
```

```
1 # Fazendo o fit dos nossos dados
2 clf_GS = clf_GS.fit(X_resRU,y_resRU)
```

```
1 # Visualizando os melhores parâmetros definidos pelo GridSearchCV*
2 clf_GS.best_params_
{'algorithm': 'auto', 'n_neighbors': 3, 'weights': 'uniform'}
```

```
1 # Usando esse modelo para fazer as previsões
2 y_pred_GS = clf_GS.predict(X_test)
```

```
1 # Analisando a matriz de confusão
2 confusion_matrix(y_test,y_pred_GS)
```

```
array([[91227,  2598],
       [   18,   144]])
```

# Módulo 19 – Comparando modelos e avaliando novos hiperparâmetros

Agora que temos estes resultados, podemos reiniciar nosso estudo e comparar os modelos com os melhores parâmetros.

Ao lado, fazemos a normalização dos valores, a separação em treino e teste, e *undersampling*.

```
1 # Primeiro para a coluna Time  
2 transacoes.Time = transacoes.Time / transacoes.Time.max()  
  
1 # E então para a coluna Amount  
2 transacoes.Amount = transacoes.Amount / transacoes.Amount.max()  
  
1 # Separando X e y  
2 X = transacoes.drop('Class',axis=1)  
3 y = transacoes.Class  
  
1 # Separando em treino e teste  
2 from sklearn.model_selection import train_test_split  
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0,stratify=y)  
  
1 y_train.value_counts()  
  
0    190490  
1      330  
Name: Class, dtype: int64  
  
1 # Importando o RandomUnderSampler do imblearn  
2 from imblearn.under_sampling import RandomUnderSampler  
  
1 # Definindo o RandomUnderSampler  
2 rus = RandomUnderSampler(random_state=42)  
  
1 # Definindo a nova amostra  
2 X_resRU, y_resRU = rus.fit_resample(X_train, y_train)  
  
1 y_resRU.value_counts()  
  
0    330  
1    330  
Name: Class, dtype: int64
```

# Módulo 19 – Comparando modelos e avaliando novos hiperparâmetros

Agora, podemos efetivamente comparar os modelos com os parâmetros selecionados anteriormente. Veja que, considerando a métrica recall, o modelo de regressão logística é o que possui melhor performance.

```
1 # Importando a matriz de confusão
2 from sklearn.metrics import confusion_matrix
```

```
1 # Regressão Logística
2 from sklearn.linear_model import LogisticRegression
3
4 clf_RL = LogisticRegression(random_state=0,
5                             C=0.001,
6                             solver='liblinear').fit(X_resRU, y_resRU)
7
8 y_pred_RL = clf_RL.predict(X_test)
9
10 confusion_matrix(y_test,y_pred_RL)
```

```
array([[85217,  8608],
       [   10,   152]])
```

```
1 # O recall
2 recall_score(y_test,y_pred_RL)
```

```
0.9382716049382716
```

```
1 # SVC
2 from sklearn.svm import SVC
3
4 clf_SVC = SVC(random_state=0,
5                 probability=True,
6                 C=100,gamma='auto',
7                 kernel='rbf').fit(X_resRU, y_resRU)
8
9 y_pred_SVC = clf_SVC.predict(X_test)
10
11 confusion_matrix(y_test,y_pred_SVC)
```

```
array([[85791,  8034],
       [   12,   150]])
```

```
1 recall_score(y_test,y_pred_SVC)
```

```
0.9259259259259259
```

```
1 # Random Forest
2 from sklearn.ensemble import RandomForestClassifier
3
4 clf_RF = RandomForestClassifier(max_depth=9,
5                                 random_state=0,
6                                 criterion='gini',
7                                 n_estimators=30).fit(X_resRU, y_resRU)
8
9 y_pred_RF = clf_RF.predict(X_test)
10
11 confusion_matrix(y_test,y_pred_RF)
```

```
array([[91476,  2349],
       [   19,   143]])
```

```
1 recall_score(y_test,y_pred_RF)
```

```
0.8827160493827161
```

## Módulo 19 – Comparando modelos e avaliando novos hiperparâmetros

Obviamente, podemos continuar adicionando parâmetros para serem analisados com o GridSearchCV. Quanto mais parâmetros, mais tempo irá demorar e não necessariamente um ganho significativo será obtido.

Veja que, mesmo colocando mais parâmetros, o valor de recall é praticamente igual ao visto anteriormente para o modelo de regressão logística, e já é um valor elevado. Assim, provavelmente não compensa mais continuar buscando melhorar os parâmetros para este modelo. Análise de custo-benefício faz parte do papel de um cientista de dados.

```
1 # Adicionando novos parâmetros
2 parametros = {
3     'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
4     'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
5     'penalty': ['l1', 'l2', 'elasticnet', 'none'],
6     'max_iter': [100, 500, 1000]
7 }
```

```
1 # Selecionando a Regressão Logística
2 LogReg = LogisticRegression(random_state=42)
```

```
1 # Criando o classificador
2 clf_RL2 = GridSearchCV(LogReg, parametros,
3                         scoring='recall'
4                         ).fit(X_resRU,y_resRU)
```

```
1 # E avaliando o modelo
2 y_pred_RL2 = clf_RL2.predict(X_test)
3
4 confusion_matrix(y_test,y_pred_RL2)
```

```
array([[78529, 15296],
       [   11,    151]])
```

```
1 # Recall com todos os parâmetros
2 recall_score(y_test,y_pred_RL2)
```

0.9320987654320988

## Módulo 19 – Comparando modelos e avaliando novos hiperparâmetros

Além disso, todos os últimos estudos utilizaram o `RandomUnderSampler` para *undersampling*. É possível testar outras estratégias, como as vistas anteriormente (`NearMiss` e `ClusterCentroids`).

Também é possível passar mais de uma métrica para o parâmetro `scoring`, de forma que será buscado um compromisso entre as métricas. Nestes casos, também é necessário passar o parâmetro `refit` com uma das métricas, a métrica que se considera mais importante, para ser utilizada no `fit`.

```
1 # Criando o classificador
2 clf_RL3 = GridSearchCV(LogReg, parametros,
3                         scoring=['recall','precision'],
4                         refit='recall'
5 )
```

```
1 clf_RL3 = clf_RL3.fit(X_resRU,y_resRU)
```

```
1 # Avaliando o novo modelo
2 y_pred_RL3 = clf_RL3.predict(X_test)
3
4 confusion_matrix(y_test,y_pred_RL3)
```

```
array([[85217,  8608],
       [ 10,   152]])
```

## Módulo 19 – Comparando modelos e avaliando novos hiperparâmetros

E, claro, podemos utilizar os métodos de *oversampling* vistos anteriormente. Só tenha em mente que pode demorar um tempo considerável.

```
[28]: 1 # Utilizando o ADASYN para o oversampling
2 from imblearn.over_sampling import ADASYN
3 X_resA, y_resA = ADASYN(random_state=42).fit_resample(X_train, y_train)

[29]: 1 # Vamos usar o datetime só para nós dar uma ideia do tempo de processamento
2 import datetime as dt

[*]: 1 # Copiando exatamente o mesmo código e adicionando apenas o print da hora atual
2 print(dt.datetime.now())
3
4 clf_RL4 = GridSearchCV(LogReg, parametros,
5                         scoring=['recall','precision'],
6                         refit='recall'
7                         )
8
9 clf_RL4 = clf_RL4.fit(X_resA, y_resA)
10
11 print(dt.datetime.now())
```

2023-05-29 16:05:53.178665

Depois de escolhido o modelo, está na hora de colocar em produção! Assunto do próximo módulo.

## MÓDULO 20

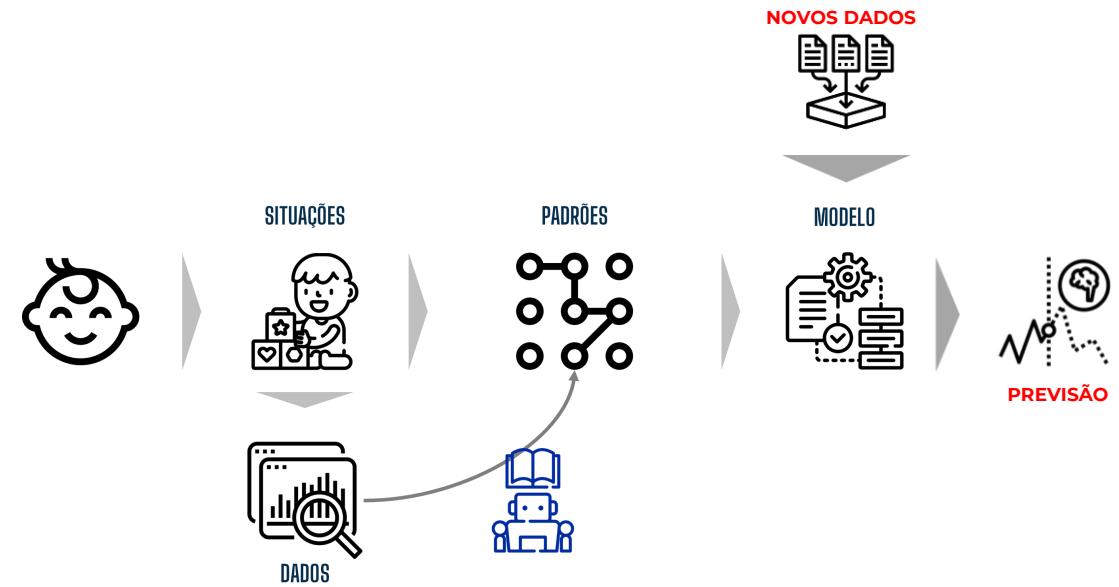
# Subindo seu modelo para produção (deploy)

## Módulo 20 – O que faremos neste módulo?

No módulo passado, fizemos um projeto completo, desde a análise até o treinamento de modelos, e seleção do mais adequado segundo métricas adequadas para o caso. Agora, iremos ver como persistir o nosso modelo para que outras pessoas possam usar. Além disso, veremos como utilizar um modelo com novos dados em produção.

Neste módulo:

- Vamos treinar um modelo de regressão linear com a base *dadosVenda.xlsx* e persistir tal modelo
- Depois de exportar o modelo, vamos utilizar os dados em *dadosVenda\_producao.xlsx*, simulando novos dados após o nosso modelo estar em produção
- Assim, fechamos o ciclo visto nos módulos anteriores, de treinamento com dados históricos e previsão com novos dados



# Módulo 20 – Criando um modelo de regressão linear

Para começar, os procedimentos já corriqueiros de:

- Importar bibliotecas
- Importar nossa base
- Visualizar a base
- Tratar a base
  - No nosso caso, os valores nulos de desconto foram considerados zero
- Separar a base em X e y, isolando nossa variável alvo

Continua na próxima página...

```
1 # Importando o pandas
2 import pandas as pd
```

```
1 # Importando nossa base
2 base = pd.read_excel('dadosVenda.xlsx')
```

```
1 # Visualizando a base
2 base.head()
```

	IDRegistro	PrecoOriginal	Desconto	VendaQtd
0	1	23	1.15	81
1	2	5	0.70	3361
2	3	15	1.35	1551
3	4	7	2.03	3036
4	5	10	0.40	2436

```
1 # Verificando se existem valores vazios
2 base.isnull().sum()
```

```
IDRegistro      0
PrecoOriginal   0
Desconto        27
VendaQtd        0
dtype: int64
```

```
1 # Entendendo e tratando os valores nulos
2 base.loc[base.Desconto.isnull(), 'Desconto'] = 0
```

```
1 # Separando X e y
2 X = base[['PrecoOriginal', 'Desconto']]
3 y = base.VendaQtd
```

# Módulo 20 – Criando um modelo de regressão linear

Continuando:

- Separar a base em treino e teste
- Treinar o modelo
- Avaliar as métricas de treinamento

Estas etapas já foram exaustivamente abordadas em projetos prévios, de forma que já devem ser naturais a você.

Observe os valores baixos das métricas de erro.

```
1 # Separando os dados
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
1 # Criando o nosso modelo
2 from sklearn.linear_model import LinearRegression
3 reg = LinearRegression().fit(X_train, y_train)
```

```
1 # Fazendo a previsão
2 y_pred = reg.predict(X_test)
```

```
1 # Importando as métricas do sklearn
2 from sklearn.metrics import mean_absolute_error
3 from sklearn.metrics import mean_squared_error
```

```
1 # Avaliando o erro médio absoluto
2 mean_absolute_error(y_test,y_pred)
```

0.25078643242464616

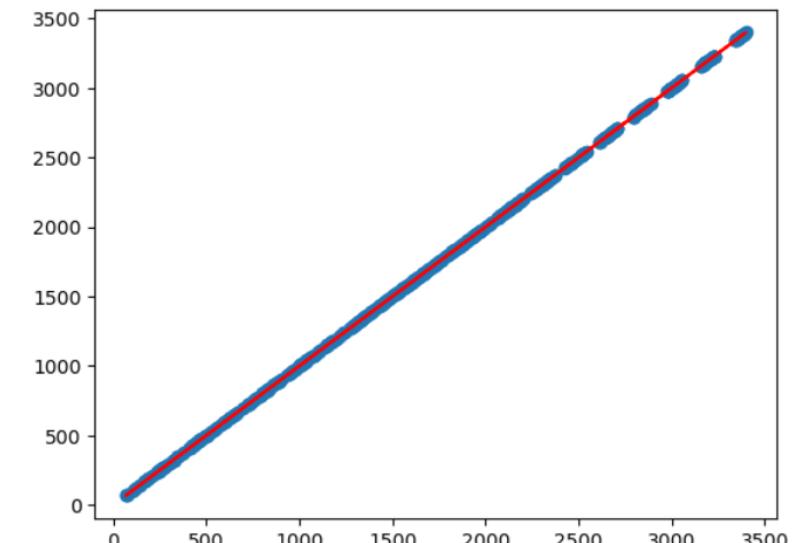
```
1 # Avaliando o erro médio quadrático
2 mean_squared_error(y_test,y_pred)
```

0.08547903210839182

## Módulo 20 – Criando um modelo de regressão linear

Podemos visualizar o ajuste graficamente, assim como obter os coeficientes angular e linear da reta:

```
1 # Traçando o gráfico de y_test x y_pred
2 import matplotlib.pyplot as plt
3
4 fig, ax = plt.subplots()
5
6 ax.scatter(y_test,y_pred)
7 ax.plot(y_test,y_test,'r')
8
9 plt.show()
```



```
1 # Verificando o coef_
2 reg.coef_
array([-182.99973759,   31.00151165])
```

```
1 # E então o intercept_
2 reg.intercept_
4253.9971316726005
```

## Módulo 20 – Persistindo o modelo com joblib

Agora, uma das novidades deste módulo, a persistência do modelo treinado.

Com a biblioteca `joblib`, podemos salvar nosso modelo para um arquivo com o método `dump`:

```
1 # Importando o dump do joblib
2 from joblib import dump, load

1 # Então fazendo o dump do nosso modelo
2 dump(reg, 'regressor.joblib')*
['regressor.joblib']
```

Para verificar que realmente foi salvo o modelo recém criado, podemos, em um outro Jupyter Notebook, recuperar este modelo com o método `load` e verificar que os coeficientes são os mesmos obtidos durante o treinamento:

```
1 # Importando o load do joblib
2 from joblib import load

1 # Carregando o modelo que havíamos exportado
2 reg = load('regressor.joblib')

1 # Verificando o coef_
2 reg.coef_

array([-182.99973759,   31.00151165])

1 # Então o intercept_
2 reg.intercept_

4253.9971316726005
```

## Módulo 20 – Utilizando o modelo criado em dados de produção

Podemos utilizar este modelo em dados novos, presentes no arquivo *dadosVenda\_producao.xlsx*:

```
1 # Importando a base  
2 baseNova = pd.read_excel('dadosVenda_producao.xlsx')
```

```
1 # Visualizando a base  
2 baseNova.head()
```

	IDRegistro	PrecoOriginal	Desconto	VendaQtd
0	1605	13	2.86	NaN
1	1606	21	2.52	NaN
2	1607	5	1.95	NaN
3	1608	15	0.75	NaN
4	1609	11	0.99	NaN

## Módulo 20 – Utilizando o modelo criado em dados de produção

Assim como antes, precisamos tratar os valores nulos da coluna *Desconto*. A coluna *VendaQtd* irá receber posteriormente os valores previstos pelo nosso modelo.

Após o tratamento, passamos para a previsão.

```
1 # Verificando valores vazios  
2 baseNova.isnull().sum()
```

```
IDRegistro      0  
PrecoOriginal   0  
Desconto        1  
VendaQtd        30  
dtype: int64
```

```
1 # Tratando os valores vazios do Desconto  
2 baseNova.loc[baseNova.Desconto.isnull(), 'Desconto'] = 0
```

```
1 # Separando X e y  
2 X = baseNova[['PrecoOriginal', 'Desconto']]
```

```
1 # Fazendo essa previsão  
2 y_pred = reg.predict(X)
```

## Módulo 20 – Utilizando o modelo criado em dados de produção

Com auxílio do NumPy, podemos deixar os valores mais condizentes com o esperado, com uma única casa decimal. Então, salvar os valores na coluna *VendaQtd*.

Os dados novos juntamente com as previsões podem ser salvos em um arquivo Excel, formato mais amigável para pessoas de uma equipe de negócios avaliar:

```
1 # Podemos deixar com 1 única casa decimal
2 import numpy
3 numpy.around(y_pred,1)

array([1963.7, 489.1, 3399.5, 1532.3, 2271.7, 262.1, 2008. , 2517. ,
       941.9, 1643.9, 593.3, 2809.8, 515.2, 1583.4, 137.7, 818.2,
      3223. , 2473.6, 2529.4, 777. , 1770.1, 2080.3, 3217.4, 3384. ,
     2979.5, 1907.2, 2504.6, 1597.4, 1405.4, 3710.6])
```

```
1 # Salvando a previsão em uma variável
2 y_pred = numpy.around(y_pred,1)
```

```
1 # Podemos substituir essa previsão na base inicial
2 baseNova['VendaQtd'] = y_pred
```

```
1 # Visualizando novamente a base
2 baseNova.head()
```

	IDRegistro	PrecoOriginal	Desconto	VendaQtd
0	1605	13	2.86	1963.7
1	1606	21	2.52	489.1
2	1607	5	1.95	3399.5
3	1608	15	0.75	1532.3
4	1609	11	0.99	2271.7

```
1 # Exportando para o Excel
2 baseNova.to_excel('resultado.xlsx')
```

## Módulo 20 – Modelo em produção com Jupyter Notebook

Agora, veremos algumas maneiras de compartilhar o projeto.

Primeiro, vamos melhorar a estrutura de arquivos de nosso projeto. Vamos criar 3 pastas:

- *entradas*: onde ficarão nossos arquivos de dados de entrada
- *saidas*: onde ficarão as saídas do nosso modelo, no caso a planilha com previsão
- *modelo*: onde ficará o arquivo do nosso modelo gerado pelo joblib

Esta é uma estrutura padrão, muito utilizada. Pode ser replicada em outros projetos, especialmente se forem mais complexos, onde mais arquivos podem ser utilizados em cada pasta.

Observe que adaptamos nosso código para esta nova estrutura de pastas. Agora, qualquer pessoa com o arquivo Notebook, Python, Jupyter, e as bibliotecas necessárias pode usufruir do modelo, que criará uma planilha na pasta de saídas.

entradas  
saidas  
modelo

```
1 # Importando o load do joblib
2 from joblib import load

1 # Carregando o modelo que havíamos exportado
2 reg = load('modelo/regressor.joblib')

1 # Importando a base
2 import pandas as pd
3 baseNova = pd.read_excel('entradas/dadosVenda_producao.xlsx')

1 # Tratando os valores vazios do Desconto
2 baseNova.loc[baseNova.Desconto.isnull(),'Desconto'] = 0

1 # Separando X e y
2 X = baseNova[['PrecoOriginal','Desconto']]

1 # Fazendo essa previsão
2 y_pred = reg.predict(X)

1 # Podemos deixar com 1 única
2 import numpy
3 y_pred = numpy.around(y_pred,1)

1 # Podemos substituir essa previsão na base inicial
2 baseNova['VendaQtd'] = y_pred

1 # Exportando para o Excel
2 baseNova.to_excel('saidas/resultado.xlsx',index=False)
```

# Módulo 20 – Modelo em produção com arquivo Python

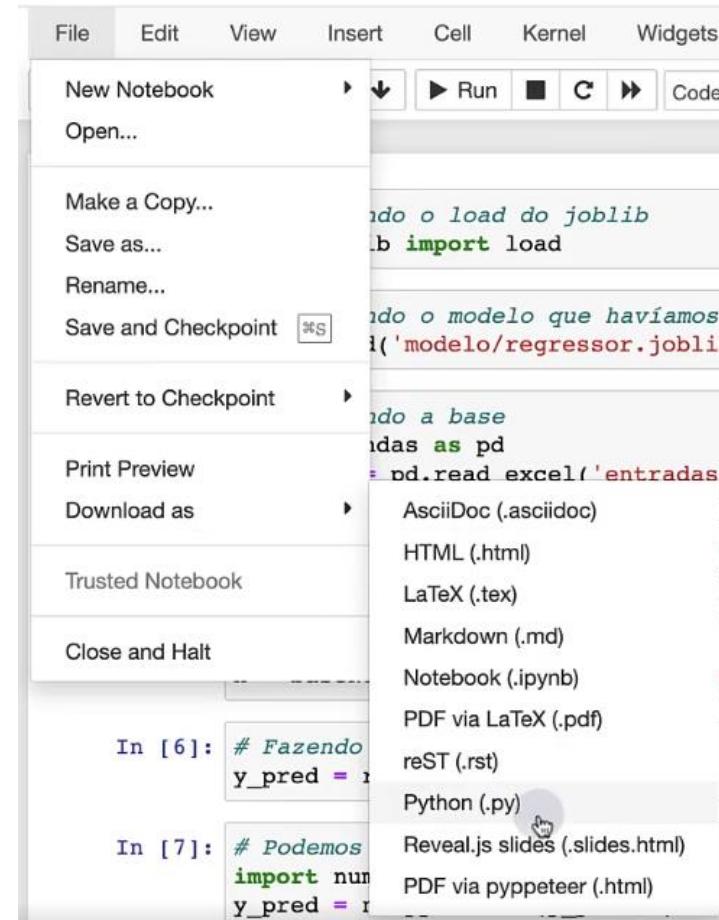
No entanto, nem todos, mesmo com conhecimento de programação, tem familiaridade com Jupyter Notebook. Assim, a segunda forma de compartilhar é exportar o código para um arquivo Python (extensão .py). A exportação pode ser feita pelo menu do Jupyter mostrado na figura ao lado.

Desta forma, qualquer pessoa com Python em sua máquina pode utilizar o modelo para previsões. Supondo o arquivo ter nome *modelo\_python.py*, basta executar no terminal, com os arquivos de entrada e de modelo nas pastas adequadas:

```
python modelo_python.py
```

A desvantagem é que a pessoa deve ser minimamente treinada em programação. E, em equipes multidisciplinares e diversas, é normal haver pessoas sem conhecimento técnico. Assim, devemos buscar outras formas de compartilhamento com a equipe.

Exportando em formato py:



Arquivo py gerado:

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:
# Importando o load do joblib
from joblib import load

# In[2]:
# Carregando o modelo que havíamos exportado
reg = load('modelo/regressor.joblib')

# In[3]:
# Importando a base
import pandas as pd
baseNova = pd.read_excel('entradas/dadosVenda_producao.xlsx')

# In[4]:
# Tratando os valores vazios do Desconto
baseNova.loc[baseNova.Desconto.isnull(),'Desconto'] = 0

# In[5]:
# Separando X e y
X = baseNova[['PrecoOriginal','Desconto']]

# In[6]:
# Fazendo essa previsão
y_pred = reg.predict(X)

# In[7]:
# Podemos deixar com 1 única
import numpy
y_pred = numpy.around(y_pred,1)

# In[8]:
# Podemos substituir essa previsão na base inicial
baseNova['VendaQtd'] = y_pred

# In[9]:
# Exportando para o Excel
baseNova.to_excel('saídas/resultado.xlsx',index=False)
```

Execução no terminal:

```
python modelo_python.py
```

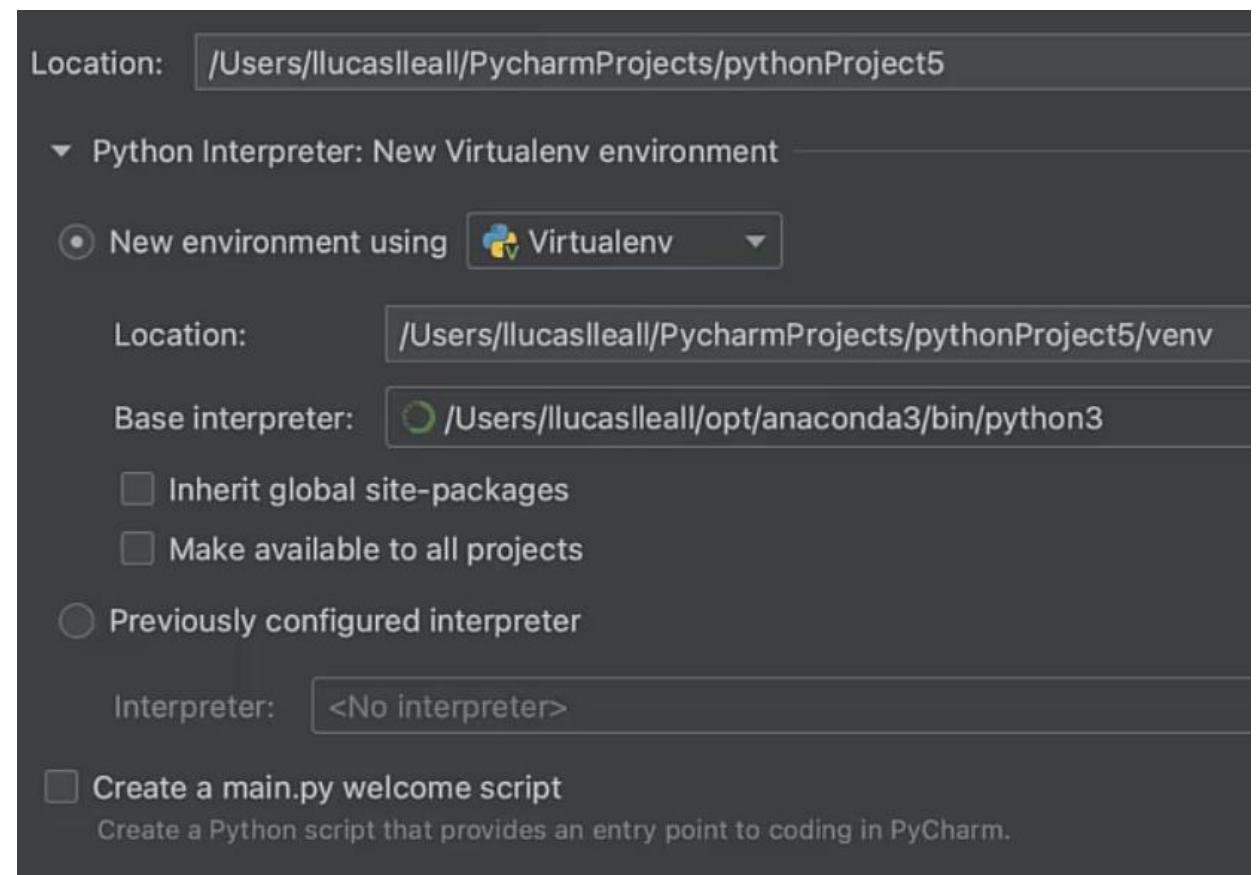
## Módulo 20 – Modelo em produção com arquivo executável

Uma terceira forma de compartilhar é criar um arquivo executável, extensão exe. Esta é uma forma mais amigável, já que provavelmente as pessoas da equipe já tiveram a experiência de clicar 2 vezes em um arquivo para executar um programa.

Nesta etapa, recomendamos usar o PyCharm, por este proporcionar uma interface simples para as etapas necessárias.

Ao criar um projeto no PyCharm, uma janela é exibida onde se pode criar um ambiente virtual para o projeto. No nosso caso, marque a opção de criar um novo ambiente virtual. Quando terminar, abra, no próprio PyCharm, um terminal com o ambiente criado e instale as bibliotecas necessárias com

```
pip install <nome do pacote>
```



## Módulo 20 – Modelo em produção com arquivo executável

No código, precisamos explicitamente importar cada biblioteca. Observe que adicionamos a importação do Scikit-Learn (`import sklearn`) e de mais uma biblioteca, a `openpyxl` (`import openpyxl`), para exportação em formato Excel.

Para criar o arquivo executável em si, utilizaremos o pacote `pyinstaller`. Com a opção `onefile`, um único arquivo será criado dentro de uma pasta chamada `dist`, que o próprio pacote irá criar:

Arquivo py com as importações de todas as bibliotecas:

```
# Importando o load do joblib
from joblib import load

# Carregando o modelo que havíamos exportado
import sklearn
reg = load('/Users/llucaslleall/Desktop/Aula 29/Producao - Executavel/modelo/regressor.joblib')

# Importando a base
import pandas as pd
baseNova = pd.read_excel('/Users/llucaslleall/Desktop/Aula 29/Producao - Executavel/entradas/dadosVenda_producao.xlsx')

# Tratando os valores vazios do Desconto
baseNova.loc[baseNova.Desconto.isnull(),'Desconto'] = 0

# Separando X e y
X = baseNova[['PrecoOriginal','Desconto']]

# Fazendo essa previsão
y_pred = reg.predict(X)

# Podemos deixar com 1 única
import numpy
y_pred = numpy.around(y_pred,1)

# Podemos substituir essa previsão na base inicial
baseNova['VendaQtd'] = y_pred

# Exportando para o Excel
import openpyxl
baseNova.to_excel('/Users/llucaslleall/Desktop/Aula 29/Producao - Executavel/saidas/resultado.xlsx',index=False)
```

Execução no terminal:

```
pyinstaller --onefile CriarExecutavel.py
```

Executável aparecerá numa pasta chamada `dist`:



## Módulo 20 – Modelo em produção com arquivo executável

Ao clicar duas vezes no arquivo, o usuário verá uma janela de terminal, que irá mostrar que o modelo está em execução e avisará quando estiver concluído. Após o término, o usuário poderá acessar a planilha na pasta saída, conforme estrutura de pastas criada anteriormente:

```
Last login: Mon Oct 17 23:02:19 on ttys002
/Users/llucaslleall/Desktop/Aula\ 29/Producao\ -\ Executavel/CriarExecutavel ; exit;
(base) llucaslleall@MacBook-Pro-de-Lucas ~ % /Users/llucaslleall/Desktop/Aula\ 29/Producao\ -\ Executavel/CriarExecutavel ; exit;

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Processo concluído]
```

## Módulo 20 – Modelo em produção com Streamlit

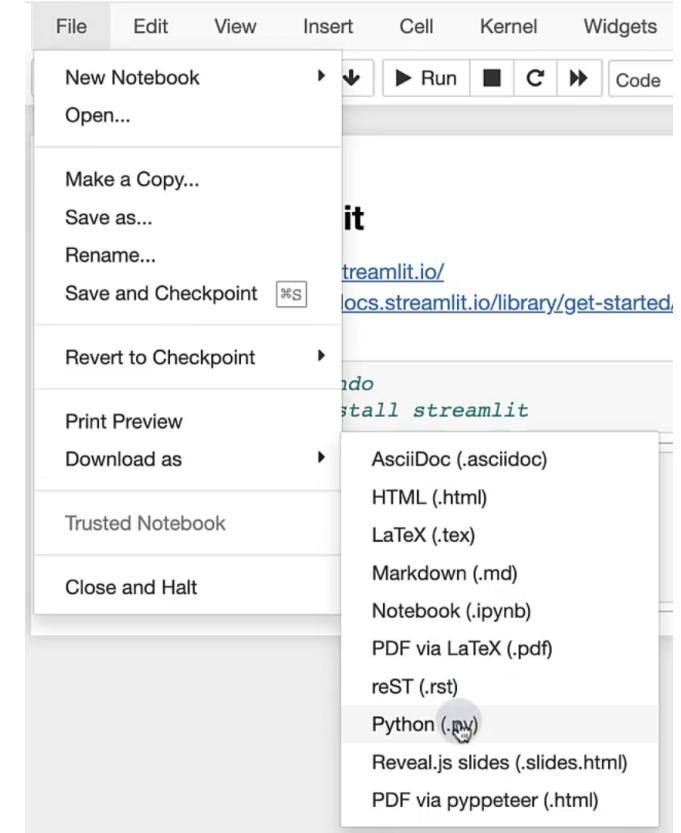
Por fim, a forma que talvez seja a mais amigável, com uma página criada pelo Streamlit.

O Streamlit é um pacote Python que permite criar facilmente interfaces web. Como qualquer usuário de computador atualmente já deve ter interagido com uma página web, esta é a maneira mais amigável de compartilhar o projeto. Além disso, evita eventuais problemas de segurança no compartilhamento de arquivos executáveis.

Para conhecer a biblioteca, vamos criar uma página simples, que exibe apenas o texto *Olá mundo!* no navegador. O método `write` recebe uma string e a exibe na página.

O Streamlit precisa ser executado via terminal a partir de um arquivo Python. Assim, vamos salvar o código em formato py, da mesma forma já vista anteriormente:

```
1 # Importando o streamlit
2 import streamlit as st
3
4 # Escrevendo Olá mundo!
5 st.write('Olá mundo!')
```



## Módulo 20 – Modelo em produção com Streamlit

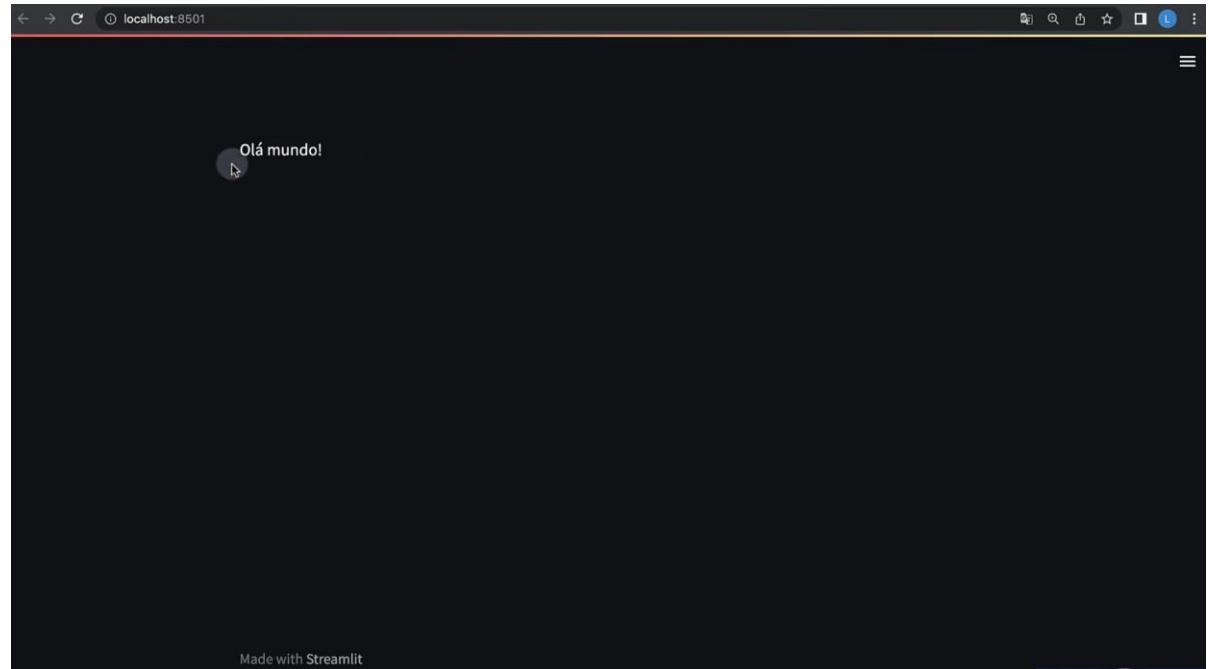
Com o arquivo gerado, executamos no terminal, na pasta onde o arquivo está, o comando

```
streamlit run <arquivo py>
```

No terminal, aparecerá o link para o projeto que pode ser acessado via navegador.

Observe que aparece exatamente o texto que colocamos, em uma interface web amigável:

```
streamlit run streamlit_1.py
```



## Módulo 20 – Modelo em produção com Streamlit

No nosso caso, o código precisa carregar o modelo, receber um preço e um desconto, e realizar a previsão.

Veja que, de forma intuitiva, criamos um título para a página (método `title`) e duas entradas numéricas (método `number_input`).

Com o método `button`, criamos um botão que, quando pressionado (condição `if` no código), executa a previsão e apresenta (método `write`) o resultado na página:

```
1 # Importando o streamlit
2 import streamlit as st
3 from joblib import load
4
5 reg = load('regressor.joblib')
6
7 st.title('Prevendo a venda quantidade')
8
9 preco = st.number_input('Preço de Venda')
10 desconto = st.number_input('Desconto')
11
12 botao = st.button('PREVER')
13
14 if botao:
15     texto = round(reg.predict([[preco,desconto]])[0],1)
16     st.write(texto)
```

## Módulo 20 – Modelo em produção com Streamlit

Executando o código com o Streamlit, teremos uma interface web amigável, onde o usuário pode inserir valores, clicar no botão e verificar o valor previsto com base nos valores de entrada.

Em termos de usabilidade, provavelmente este é o formato mais amigável aos usuários.

Da forma implementada, estará rodando localmente em sua máquina. Conversando com a equipe de TI de sua empresa, você pode colocar em uma página na intranet da empresa, de forma que outras pessoas da equipe possam utilizar, ou mesmo em alguma interface que os clientes interessados possam utilizar.

### Prevendo a venda quantidade

Preço de Venda  
5,00 - +

Desconto  
1,95 - +

**PREVER**

3399.5

## MÓDULO 21

# Ajustando os dados para o modelo (Data Cleaning)

## Módulo 21 – Explicando a importância da limpeza dos dados e importando a base

Ao longo dos módulos anteriores, utilizamos várias técnicas de limpeza de dados, mas dada a importância desse tema, teremos um módulo inteiro para falar sobre esse assunto. Com essa etapa, vamos garantir que os nossos modelos estão trabalhando com os dados corretos. Lembre-se que sempre que, **lixo entra, lixo sai!** Ou seja, se entregarmos dados sujos, teremos um resultado ruim e que pode nos levar a tomar decisões erradas, gerando custos inestimáveis para o negócio.

Neste módulo:

- Inicialmente, utilizaremos o arquivo **alunos.xlsx**, incluindo suas diversas abas para treinar várias técnicas de limpeza de dados. Usaremos essa base sintética, porque nela conseguimos inserir todos os tipos de problemas, enquanto que em um real, em geral, teríamos apenas alguns casos isolados
- Veremos técnicas de preenchimento de valores vazios, tratamento de outliers, etc
- No fim, vamos utilizar a base real do **titanic** para testar nossos conhecimentos em limpeza de dados
- O principal objetivo é desenvolver o seu senso crítico de como tratar e limpar uma base de dados, já que é impossível abordar todos tipos de caso

## Módulo 21 – Explicando a importância da limpeza dos dados e importando a base

Vamos iniciar com a aba de **pagamento** do nosso arquivo **alunos.xlsx** que está disponível para download na parte de cima do notebook inicial da aula.

Podemos visualizar essa base com o **head(2)**. Inclusive, observe que já temos valores vazios.

E ao visualizar o **info()**, inicialmente já observamos, que além, de valores vazios em mais de uma coluna, temos que a coluna **Mensalidade** está com o tipo **object**.

```
1 # Importando o pandas
2 import pandas as pd

1 # Importando a aba "pagamento" da base de alunos
2 base = pd.read_excel('alunos.xlsx',sheet_name='pagamento')

1 # Visualizando essa base
2 base.head(2)

ID_aluno Mensalidade Situacao Meses atraso
0 1 1660 Pago NaN
1 2 1610 Atrasado 2.0

1 # E as informações (número de valores não nulos, tipo dos dados, etc)
2 base.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17 entries, 0 to 16
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  -- 
 0   ID_aluno    17 non-null     int64  
 1   Mensalidade 16 non-null     object 
 2   Situacao    17 non-null     object 
 3   Meses atraso 6  non-null     float64
dtypes: float64(1), int64(1), object(2)
memory usage: 672.0+ bytes
```

## Módulo 21 – Buscando na base por valores nulos e linhas duplicadas

Podemos iniciar o nosso tratamento filtrando as linhas vazias da coluna **Mensalidade** e em seguida da coluna **Meses atraso**.

Note que no caso da coluna **Meses atraso**, já visualizamos um padrão, em que sempre que a situação está como “**Pago**”, o a quantidade de meses em atraso está vazia. O que dá a entender que simplesmente, quando essa coluna está vazia, não há atraso algum.

```
# Verificando se temos valores vazios na coluna Mensalidade  
base[base.Mensalidade.isnull()]
```

ID_aluno	Mensalidade	Situacao	Meses atraso
7	8	NaN	Pago

```
# E verificando os valores vazios na coluna Meses atraso  
base[base['Meses atraso'].isnull()]
```

ID_aluno	Mensalidade	Situacao	Meses atraso
0	1	1660	Pago
2	3	1,650.00	Pago
5	6	1760	Pago
7	8	NaN	Pago
8	9	1650	Pago
10	11	1710	Pago
11	12	1630	Pago
13	14	1700	Pago
14	15	1710	Pago
15	14	1700	Pago
16	15	1710	Pago

## Módulo 21 – Buscando na base por valores nulos e linhas duplicadas

Para validar melhor essa informação, podemos visualizar os casos em que a coluna não está vazia.

E aqui temos a opção de usar, por exemplo, a condição **notnull()** ou simplesmente adicionar um ~ a nossa **isnull()**. Nesse caso o ~ representa uma negação da condição que vem em seguida.

```
# E verificando os valores vazios na coluna Meses atraso  
base[base['Meses atraso'].notnull()]
```

ID_aluno	Mensalidade	Situacao	Meses atraso
1	2	1610	Atrasado
3	4	1770	Atrasado
4	5	1790	Atrasado
6	7	1620	Atrasdo
9	10	1640	Atrasado
12	13	1630	Atrasado

```
# E verificando os valores vazios na coluna Meses atraso  
base[~base['Meses atraso'].isnull()]
```

ID_aluno	Mensalidade	Situacao	Meses atraso
1	2	1610	Atrasado
3	4	1770	Atrasado
4	5	1790	Atrasado
6	7	1620	Atrasdo
9	10	1640	Atrasado
12	13	1630	Atrasado

## Módulo 21 – Buscando na base por valores nulos e linhas duplicadas

Vamos agora verificar linhas duplicadas. E para isso, podemos usar o método **duplicated()** do pandas. Que vai nos retornar **False**, quando a linha não é duplicada, e **True**, caso a linha seja duplicada.

```
# Verificando se existe valores duplicados
base.duplicated()

0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
10   False
11   False
12   False
13   False
14   False
15   True
16   True
dtype: bool
```

## Módulo 21 – Buscando na base por valores nulos e linhas duplicadas

Visualizando as últimas linhas da base, veja que as duas últimas estão exatamente iguais as duas anteriores. Mas apenas as duas últimas foram marcadas como duplicadas.

base.tail()				
ID_aluno	Mensalidade	Situacao	Meses atraso	
12	13	1630	Atrasado	1.0
13	14	1700	Pago	NaN
14	15	1710	Pago	NaN
15	14	1700	Pago	NaN
16	15	1710	Pago	NaN

Isso acontece, porque, por padrão, o parâmetro **keep** do método **duplicated()**, faz com que a primeira aparição daquela linha duplicada seja marcada como **False**, ou seja, ela é a escolhida para ser mantida ([Documentação](#)).

**keep : {'first', 'last', False}, default 'first'**

Determines which duplicates (if any) to mark.

- **first** : Mark duplicates as **True** except for the first occurrence.

## Módulo 21 – Buscando na base por valores nulos e linhas duplicadas

Com isso, podemos visualizar diretamente todos as linhas duplicas, usando o parâmetro **keep=False**, e já colocando como filtro de linhas da nossa base.

```
# Verificando se existe valores duplicados  
base[base.duplicated(keep=False)]
```

ID_aluno	Mensalidade	Situacao	Meses atraso
13	14	1700	Pago
14	15	1710	Pago
15	14	1700	Pago
16	15	1710	Pago

Só que na verdade o que queremos é, filtrar nossa base para remover os valores duplicados, porém mantendo um dos valores.

## Módulo 21 – Buscando na base por valores nulos e linhas duplicadas

Nosso objetivo é atingindo ao usarmos o **keep=“first”** ou **keep=“last”**, para manter um dos valores, e adicionando o `~` antes da condição, já que queremos todo mundo que não é duplicado, ou seja, queremos todo mundo que é **False** no **duplicated()**.

```
# Verificando se existe valores duplicados  
base[~base.duplicated(keep='first')]
```

	ID_aluno	Mensalidade	Situacao	Meses atraso
0	1	1660	Pago	NaN
1	2	1610	Atrasado	2.0
2	3	1,650.00	Pago	NaN
3	4	1770	Atrasado	1.0
4	5	1790	Atrasado	1.0
5	6	1760	Pago	NaN
6	7	1620	Atrasado	5.0
7	8	NaN	Pago	NaN
8	9	1650	Pago	NaN
9	10	1640	Atrasado	1.0
10	11	1710	Pago	NaN
11	12	1630	Pago	NaN
12	13	1630	Atrasado	1.0
13	14	1700	Pago	NaN
14	15	1710	Pago	NaN

## Módulo 21 – Procurando na base alguns problemas que podem ter sido gerados por erros humanos

Até agora já visualizamos valores vazios e duplicados na nossa base. Porém, em geral, a maioria dos problemas em uma base, são causados por erros humanos. Podem ser, por exemplo, erros de digitação, enviar um formulário duas vezes ou preencher algo fora da maneira padrão.

É nosso dever, como cientista de dados, garantir que os dados estão corretos. No caso do nosso desafio, podemos verificar, por exemplo, se na coluna **Situação**, tem algum valor fora do padrão. Para isso podemos verificar se há algum valor nessa coluna que não é “**Pago**” ou “**Atrasado**”.

```
# Será que alguma situação foi escrito errado?  
valores_aceitos = ['Pago', 'Atrasado']  
base[~base.Situacao.isin(valores_aceitos)]
```

ID_aluno	Mensalidade	Situacao	Meses atraso
6	7	1620	Atrasdo

## Módulo 21 – Procurando na base alguns problemas que podem ter sido gerados por erros humanos

E sim, note que, tem uma situação com um erro de digitação, “**Atrasdo**” em vez de “**Atrasado**”.

Uma outra forma de verificar esse problema, é usando o método **value\_counts()**.

```
# Contando valores em cada situação
base.Situacao.value_counts()

Pago      11
Atrasado   5
Atrasodo   1
Name: Situacao, dtype: int64
```

## Módulo 21 – Procurando na base alguns problemas que podem ter sido gerados por erros humanos

Partindo para outro problema, lembra que a coluna **Mensalidade** está como **object?** Vamos tentar transformá-la em uma coluna numérica.

```
# Tentando transformar o valor da mensalidade para número
pd.to_numeric(base.Mensalidade)

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-14-6124b5e8b2b2> in <module>
      1 # Tentando transformar o valor da mensalidade para número
----> 2 pd.to_numeric(base.Mensalidade)

~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/tools/numeric.py in to_numeric(arg, errors, downcast)
    150     coerce_numeric = errors not in ("ignore", "raise")
    151     try:
--> 152         values = lib.maybe_convert_numeric(
    153             values, set(), coerce_numeric=coerce_numeric
    154         )

pandas/_libs/lib.pyx in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "1,650.00" at position 2
```

## Módulo 21 – Procurando na base alguns problemas que podem ter sido gerados por erros humanos

Acontece um erro, porque um dos valores é uma **string**, e justamente por ter essa troca entre vírgula e ponto, o pandas não consegue transformar esse valor em um número.

Em vez de deixar esse erro acontecer, podemos informar o que deve acontecer quando ele não conseguir transformar, por exemplo, ele pode deixar vazio. O que nos permite filtrar esse valor vazio, e comparar com a coluna original, assim sendo possível analisar porque ele não conseguiu transformar. Isso é possível, passando o valor **“coerce”** para o parâmetro de erros da **pd.to\_numeric()**. Com isso, podemos guardar essa transformação em uma nova coluna de checagem.

```
1 # Criando uma nova coluna onde o erro do to_numeric resulte em NaN
2 base['Mensalidade_Check'] = pd.to_numeric(base.Mensalidade, errors='coerce')
```

## Módulo 21 – Procurando na base alguns problemas que podem ter sido gerados por erros humanos

Agora podemos filtrar as linhas em que a nova coluna, **Mensalidade\_Check**, é vazia (o pandas não conseguiu transformar esse valor).

```
1 # Visualizando a base
2 base[(base.Mensalidade_Check.isnull())]
```

ID_aluno	Mensalidade	Situacao	Meses atraso	Mensalidade_Check
2	3	1,650.00	Pago	NaN
7	8	NaN	Pago	NaN

E para facilitar a visualização, podemos filtrar também linhas e que a mensalidade não é vazia.

```
1 # Visualizando a base
2 base[(base.Mensalidade_Check.isnull()) & (base.Mensalidade.notnull())]
```

ID_aluno	Mensalidade	Situacao	Meses atraso	Mensalidade_Check
2	3	1,650.00	Pago	NaN

Com isso, fica bem mais fácil de visualizar que nessa linha, temos um erro de digitação.

## Módulo 21 – Procurando na base alguns problemas que podem ter sido gerados por erros humanos

A principal lição que fica até agora, é:

**"Por mais que você confie muito em quem te entregou sua base de dados sempre confira se nenhuma limpeza de dados será realmente necessária"**

E por mais que, para outras áreas a base esteja "limpa", nem sempre ela vai estar "limpa" para a sua aplicação. De forma geral, é mais seguro assumir que nenhuma base vai ser perfeita.

## Módulo 21 – Tratando valores vazios e linhas duplicadas

Até agora buscamos e visualizamos vários problemas, mas agora é hora de efetivamente tratar os dados e realizar as limpezas necessárias.

Lembrando que a forma como devemos resolver esses problemas, depende de nós mesmos, do negócio e do que queremos considerar.

Então no caso de um dado vazio, podemos simplesmente excluir essa linha, ou buscar esse dado com alguém da empresa. E qual decisão tomar, no geral, depende do custo para conseguir essa informação.

## Módulo 21 – Tratando valores vazios e linhas duplicadas

Iniciando com a coluna **Mensalidade**, no nosso caso, vamos supor que o time de dados informou que o aluno de **ID=8**, que está vazio, na verdade paga **1670** reais. Então, é possível preencher com esse valor. Lembrando que, se não tivéssemos esse valor, poderíamos, por exemplo, preencher com a média da coluna.

```
# Podemos substituir o valor faltante pelo valor informado  
base.loc[base.Mensalidade.isnull(),'Mensalidade'] = 1670
```

Já no caso da coluna **Meses atraso** temos algo diferente, em que o valor vazio, significa simplesmente que os alunos estão em dia com a mensalidade, ou seja, podemos substituir os valores vazios por 0.

```
# E o meses atraso vazio igual a 0 para alunos em dia com o pagamento  
base.loc[base['Meses atraso'].isnull(),'Meses atraso'] = 0
```

Por fim, no caso dos valores duplicados, podemos reutilizar o código que usamos para visualizar a tabela sem duplicados anteriormente, e guardar essa tabela já sem linhas duplicadas, dentro da própria variável **base**.

```
# Ajustando os valores duplicados  
base = base[~base.duplicated(keep='first')]
```

## Módulo 21 – Tratando valores vazios e linhas duplicadas

E agora já conseguimos visualizar a nossa tabela bem mais limpa, por exemplo, já sem valores nulos e linhas duplicadas.

Lembrando que a coluna **Mensalidade\_Check** é somente uma coluna auxiliar, como ainda estamos fazendo os tratamentos, vamos mantê-la, mas ao final devemos removê-la.

base.shape					
(15, 5)					
base					
0	1	1660	Pago	0.0	1660.0
1	2	1610	Atrasado	2.0	1610.0
2	3	1,650.00	Pago	0.0	NaN
3	4	1770	Atrasado	1.0	1770.0
4	5	1790	Atrasado	1.0	1790.0
5	6	1760	Pago	0.0	1760.0
6	7	1620	Atrasado	5.0	1620.0
7	8	1670	Pago	0.0	NaN
8	9	1650	Pago	0.0	1650.0
9	10	1640	Atrasado	1.0	1640.0
10	11	1710	Pago	0.0	1710.0
11	12	1630	Pago	0.0	1630.0
12	13	1630	Atrasado	1.0	1630.0
13	14	1700	Pago	0.0	1700.0
14	15	1710	Pago	0.0	1710.0

## Módulo 21 – Tratando valores digitados errados (erros humanos)

Agora vamos tratar os outros problemas que visualizamos anteriormente.

Iniciaremos tratando o texto digitado incorretamente na coluna **Situacao**. De “**atrasdo**” para “**atrasado**”.

```
# Tratando o texto digitado errado na Situação  
base.loc[base.Situacao == 'Atrasdo','Situacao'] = 'Atrasado'  
  
# Verificando os valores  
base.Situacao.value_counts()  
  
Pago      9  
Atrasado   6  
Name: Situacao, dtype: int64
```

Lembrando que fizemos de maneira bem direta, mas poderíamos fazer de maneiras mais robustas, prevendo outros tipos de erro de digitação, etc.

## Módulo 21 – Tratando valores digitados errados (erros humanos)

No caso da coluna **Mensalidade** temos também um texto, onde deveria ser um número. E novamente há várias formas de se fazer essa alteração, mas faremos de uma maneira bem direta.

```
# E o valor da mensalidade digitado errado  
base.loc[base.Mensalidade == '1,650.00', 'Mensalidade'] = 1650
```

Só que ainda temos um problema. A coluna **Mensalidade** ainda vai continuar com o tipo **object**.

```
# Também vamos verificar o tipo de dado de cada coluna  
base.info()
```

#	Column	Non-Null Count	Dtype
0	ID_aluno	15 non-null	int64
1	Mensalidade	15 non-null	object
2	Situacao	15 non-null	object
3	Mezes atraso	15 non-null	float64
4	Mensalidade_Check	13 non-null	float64

dtypes: float64(2), int64(1), object(2)  
memory usage: 720.0+ bytes

## Módulo 21 – Tratando valores digitados errados (erros humanos)

Para corrigir isso, é só usar a **pd.to\_numeric()**, que não vai dar erros, já que tratamos corretamente todos os valores da coluna.

```
# E podemos ajustar a coluna mensalidade com o tipo numérico  
base['Mensalidade'] = pd.to_numeric(base.Mensalidade)
```

Por fim, podemos eliminar a coluna **Mensalidade\_Check**, já que ela só estava sendo usada para validação.

```
# Depois dos ajustes, podemos eliminar as colunas que usamos para as validações  
base = base.drop('Mensalidade_Check', axis=1)
```

# Módulo 21 – Tratando valores digitados errados (erros humanos)

Pronto, agora temos a nossa base limpa. Sem valores vazios, sem linhas duplicadas, sem erros de digitação, e com os tipos corretos.

```
# Então verificando as informações
base.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 15 entries, 0 to 14
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   ID_aluno    15 non-null     int64  
 1   Mensalidade 15 non-null     int64  
 2   Situacao    15 non-null     object  
 3   Meses atraso 15 non-null     float64 
dtypes: float64(1), int64(2), object(1)
memory usage: 600.0+ bytes
```

# Visualizando a base base				
ID_aluno	Mensalidade	Situacao	Meses atraso	
0	1	1660	Pago	0.0
1	2	1610	Atrasado	2.0
2	3	1650	Pago	0.0
3	4	1770	Atrasado	1.0
4	5	1790	Atrasado	1.0
5	6	1760	Pago	0.0
6	7	1620	Atrasado	5.0
7	8	1670	Pago	0.0
8	9	1650	Pago	0.0
9	10	1640	Atrasado	1.0
10	11	1710	Pago	0.0
11	12	1630	Pago	0.0
12	13	1630	Atrasado	1.0
13	14	1700	Pago	0.0
14	15	1710	Pago	0.0

## Módulo 21 – Limpeza de Dados - Exercício

A ideia aqui é que vocês realizem um exercício sobre limpeza de dados. As instruções para esse desafio, são as seguintes:

### **Limpeza de Dados - Exercício**

- Foi feito um formulário com 15 alunos de uma escola
- As respostas do formulário estão disponíveis na aba (**questionario**) da base de alunos disponível no link abaixo:
  - [alunos.xlsx](#)
- Com essa base, responda:
  - 1. Quantos alunos responderam ao formulário?
  - 2. Qual a soma da mensalidade dos alunos que responderam? (Obs: a mensalidade de um aluno não pode ser contada em duplicidade)
  - 3. Qual o tamanho da blusa do aluno de ID\_aluno = 10?
  - 4. Qual blusa você escolheria para o aluno de ID\_aluno = 9?
  - 5. Quantos alunos vão participar da formatura? (Obs: no formulário, a questão pedia para o aluno responder S ou N)
  - 6. Qual a altura estimada do aluno de ID\_aluno = 7?

Lembrando que essas instruções também estão disponíveis no notebook abaixo da aula.

E inicialmente, tente fazer sozinho. Não tem resposta certa ou errada, faça suposições e adote premissas, sempre tentando deixar explicado o porquê fez determinado tratamento.

Quando finalizar sua solução, siga para os slides seguintes, onde iremos resolver o desafio.

## Módulo 21 – Limpeza de Dados - Exercício

Podemos iniciar importando o pandas e carregando a nossa base.

E possível visualizar que temos as colunas:

- **ID\_questionário**
- **ID\_aluno**
- **altura**
- **tamanho\_camisa**
- **mensalidade**
- **participa\_formatura**

# Importando o pandas	import pandas as pd				
# Importando a base	base = pd.read_excel('alunos.xlsx',sheet_name='questionario')				
# Visualizando a base completa	base				
# Visualizando a base completa					
ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
0	1	1	1.68	M	1.66
1	2	2	1.66	M	1610.00
2	3	3	1.69	M	1650.00
3	4	4	1.71	G	1770.00
4	5	6	1.66	M	1760.00
5	6	7	NaN	M	1620.00
6	7	8	1.70	G	1670.00
7	8	9	1.79	NaN	1650.00
8	9	10	1.80	NaN	1640.00
9	10	11	1.78	GG	1710.00
10	11	4	1.71	G	1770.00
11	12	12	1.78	GG	1630.00
12	13	Lucas	1.65	M	1630.00
13	14	14	1.74	G	1700.00
14	15	15	1.71	G	1710.00
15	16	10	1.80	GG	1640.00

## Módulo 21 – Entendendo a base e respondendo as perguntas sem fazer o tratamento dos dados

Antes de iniciar, lembre-se que as respostas aqui podem estar diferentes das suas, e as duas estarem corretas, já que simplesmente podemos ter assumido premissas diferentes.

Iniciaremos verificando a nossa base com o método **info()**.

Note que já temos vários pontos de atenção, por exemplo, temos **16 linhas, mas temos apenas 15 alunos**, a coluna **ID\_aluno** está com o tipo **object**, mas deveriam ser apenas números. Além disso, temos valores vazios em algumas colunas.

# Visualizando a base  
base.head(3)

	ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
0	1	1	1.68	M	1.66	S
1	2	2	1.66	M	1610.00	N
2	3	3	1.69	M	1650.00	S

# Verificando as informações da base  
base.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   ID_questionario    16 non-null      int64  
 1   ID_aluno           16 non-null      object  
 2   altura             15 non-null      float64 
 3   tamanho_camisa     14 non-null      object  
 4   mensalidade        16 non-null      float64 
 5   participa_formatura 16 non-null      object  
dtypes: float64(2), int64(1), object(3)
memory usage: 896.0+ bytes
```

## Módulo 21 – Entendendo a base e respondendo as perguntas sem fazer o tratamento dos dados

Antes de iniciar efetivamente a limpeza dos dados, vamos tentar simplesmente responder as perguntas sem nenhum tipo de tratamento. Já iniciando com a **pergunta 1** e a **pergunta 2**.

```
# 1. Número de respostas  
res_1 = base.shape[0]  
res_1
```

16

```
# 2. Soma das mensalidades  
res_2 = base.mensalidade.sum()  
res_2
```

25161.66

Já no caso da **pergunta 3**, temos **respostas duplicadas** com o aluno de ID=10.

```
# 3. Tamanho da blusa do aluno de ID = 10  
base[base.ID_aluno == 10]
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
8	9	10	1.8	NaN	S
15	16	10	1.8	GG	S

Então, podemos assumir que ele enviou o formulário duas vezes e a resposta seria “**GG**”.

```
# 3. Tamanho da blusa do aluno de ID = 10  
res_3 = 'GG'
```

## Módulo 21 – Entendendo a base e respondendo as perguntas sem fazer o tratamento dos dados

Para a **pergunta 4**, o tamanho da camisa está vazio, então momentaneamente é uma incógnita.

```
# 4. Tamanho da blusa do aluno de ID = 9  
base[base.ID_aluno == 9]
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
7	8	9	1.79	NaN	1650.0

```
# 4. Tamanho da blusa do aluno de ID = 9  
res_4 = '?'
```

Na **pergunta 5**, podemos filtrar as linhas com “S” na participação da formatura.

```
# 5. Quantidade de alunos na formatura  
res_5 = base[base.participa_formatura == "S"].shape[0]  
res_5
```

11

Por fim, no caso da **pergunta 6**, novamente não temos valor preenchido.

```
# 6. Altura do aluno de ID = 7  
base[base.ID_aluno == 7]
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
5	6	7	NaN	M	1620.0

```
# 6. Altura do aluno de ID = 7  
res_6 = '?'
```

## Módulo 21 – Contando a quantidade de alunos que responderam o questionário

Agora que respondemos as perguntas sem nenhum tratamento, vamos limpar os dados, para garantir que estamos respondendo de forma correta.

### 1. Quantos alunos responderam ao formulário?

A primeira opção realmente é ver quantas linhas temos na nossa base.

```
# Verificando o número de Linhas  
base.shape[0]
```

16

```
# Filtrando valores duplicados  
base.duplicated()
```

0	False
1	False
2	False
3	False
4	False
5	False
6	False
7	False
8	False
9	False
10	False
11	False
12	False
13	False
14	False
15	False

dtype: bool

E nesse caso, a resposta seria 16. Mas vamos analisar melhor essa resposta, por exemplo, verificando se temos linhas duplicadas (linhas totalmente iguais).

E a princípio, não temos linhas duplicadas.

## Módulo 21 – Contando a quantidade de alunos que responderam o questionário

Vamos agora visualizar se algum aluno respondeu mais de uma vez o formulário, mais especificamente, se o ID de um aluno apareceu mais de um vez na nossa base.

E note que, os alunos com o **ID=10** e **ID=4**, enviaram o formulário mais de uma vez.

Para outros times, pode ser que não tenha problema algum o aluno responder o formulário duas vezes. Mas, no nosso caso, vamos levar em consideração que isso é um problema.

# Contando a quantidade de registros na coluna ID_aluno base.ID_aluno.value_counts()	
10	2
4	2
Lucas	1
15	1
14	1
12	1
11	1
9	1
8	1
7	1
6	1
3	1
2	1
1	1
Name: ID_aluno, dtype: int64	

## Módulo 21 – Contando a quantidade de alunos que responderam o questionário

Vamos agora visualizar se algum aluno respondeu mais de uma vez o formulário, mais especificamente, se o ID de um aluno apareceu mais de um vez na nossa base.

E note que, os alunos com o **ID=10** e **ID=4**, enviaram o formulário mais de uma vez.

Para outros times, pode ser que não tenha problema algum o aluno responder o formulário duas vezes. Mas, no nosso caso, vamos levar em consideração que isso é um problema.

# Contando a quantidade de registros na coluna ID_aluno base.ID_aluno.value_counts()	
10	2
4	2
Lucas	1
15	1
14	1
12	1
11	1
9	1
8	1
7	1
6	1
3	1
2	1
1	1
Name: ID_aluno, dtype: int64	

## Módulo 21 – Contando a quantidade de alunos que responderam o questionário

Podemos também contar a quantidade de ID's únicos com o método **nunique()**.

```
# Usando o .nunique() para contar registros distintos  
base.ID_aluno.nunique()
```

14

E agora, confirmamos que **só temos 14 alunos, dos 15 que deveriam ter respondido** o formulário na nossa base.

Além desses problemas, temos um aluno com o **ID=“Lucas”**. E ele pode ser inclusive, um aluno que respondeu duas vezes, uma com ID correto, e outra com o nome dele. Como podemos descobrir isso?

```
# Contando a quantidade de registros na coluna ID_aluno  
base.ID_aluno.value_counts()
```

10	2
4	2
Lucas	?
15	1
14	1
12	1
11	1
9	1
8	1
7	1
6	1
3	1
2	1
1	1

Name: ID\_aluno, dtype: int64

## Módulo 21 – Contando a quantidade de alunos que responderam o questionário

E aqui a solução poderia ser, por exemplo, excluir essa linha. Mas, no nosso caso vamos buscar outras bases, já que nomes e ID's são informações que deveríamos ter.

Para encontrar esse dado, vamos importar o mesmo arquivo alunos.xlsx, porém agora na aba **nomes**.

Visualizando, temos uma coluna com o **ID**, outra com o **Nome** e a última com o **E-mail**.

```
# Importando a base nomes
nomes = pd.read_excel('alunos.xlsx',sheet_name='nomes')

# Visualizando essa base
nomes.head(3)
```

	ID_aluno	Nome	Unnamed: 2
0	1	Gustavo	gustavo@gmail.com
1	2	Miguel	miguel@gmail.com
2	3	Fernanda	fernanda@gmail.com

## Módulo 21 – Contando a quantidade de alunos que responderam o questionário

Ao filtrar os alunos com o nome “Lucas”, temos o aluno com **ID=5**. E com isso, já conseguimos a informação que estava faltando.

```
# Filtrando pelo nome Lucas  
nomes[nomes.Nome == 'Lucas']
```

ID_aluno	Nome	Unnamed: 2
3	5	Lucas lucas@gmail.com

Agora basta alterar o ID na nossa base original **de “Lucas” para 5**.

```
# Trocando na base o nome "Lucas" pelo ID_aluno 5  
base.loc[base.ID_aluno == 'Lucas', 'ID_aluno'] = 5
```

Por fim, é possível verificar novamente a quantidade de ID's únicos, que se mantém em **14**.

```
# Nesse caso o nunique continua do mesmo tamanho  
base.ID_aluno.nunique()
```

14

## Módulo 21 – Eliminando valores duplicados e discutindo sobre o tratamento do ID\_aluno

Neste momento, vamos nos aprofundar nos alunos com ID's duplicados.

Inicialmente, podemos visualizar os ID's, com o **value\_counts()**, que aparecem mais de uma vez.

```
# Contando a quantidade de registros na coluna ID_aluno
ids = base.ID_aluno.value_counts() > 1
ids
```

ID	Valor
4	True
10	True
1	False
2	False
3	False
6	False
7	False
8	False
9	False
11	False
12	False
5	False
14	False
15	False

Name: ID\_aluno, dtype: bool

Logo em seguida, podemos selecionar apenas os ID's que estão duplicados. Que no nosso caso, são os **ID's 4 e 10**.

```
# Selecionando os ID's que queremos visualizar
filtra_ids = ids[ids].index
filtra_ids
```

Int64Index([4, 10], dtype='int64')

## Módulo 21 – Eliminando valores duplicados e discutindo sobre o tratamento do ID\_aluno

Com isso, ao filtrar temos as linhas com esses ID's de aluno, pode visualizar esses envios duplicados diretamente na nossa base.

```
# Filtrando os ID's
base[base.ID_aluno.isin(filtrar_ids)]
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
3	4	4	1.71	G	1770.0
8	9	10	1.80	Nan	1640.0
10	11	4	1.71	G	1770.0
15	16	10	1.80	GG	1640.0

Já que, por exemplo, o aluno de **ID=10** não respondeu o tamanho da camisa no primeiro envio, podemos considerar que o primeiro envio foi um erro em ambos os casos. Então, uma decisão que podemos tomar, é remover o primeiro envio de cada um desses alunos.

```
# Retirando essas duas Linhas da base
base = base.drop([3,8],axis=0)
```

## Módulo 21 – Eliminando valores duplicados e discutindo sobre o tratamento do ID\_aluno

Agora, ao visualizar o **value\_counts()** da coluna “**ID\_aluno**”, temos que nenhum ID está duplicado.

Além disso, note que, o aluno de **ID=13** não respondeu esse formulário.

```
# Podemos novamente contar a quantidade de registros
base.ID_aluno.value_counts()

1    1
2    1
3    1
6    1
7    1
8    1
9    1
11   1
4    1
12   1
5    1
14   1
15   1
10   1
Name: ID_aluno, dtype: int64
```

E com todos esses tratamentos realizados, temos a confiança de responder, que **14 alunos responderam ao formulário**.

```
# Contando a quantidade de Linhas
base.shape[0]
```

14

```
# Usando o nunique
base.ID_aluno.nunique()
```

14

## Módulo 21 – Eliminando valores duplicados e discutindo sobre o tratamento do ID\_aluno

Lembra que o ID do “Lucas” é **5**? Para fins didáticos, vamos agora refazer o processo considerando que o ID desse aluno **será igual a 3**, ou seja, um ID que já existe na base.

Para isso, vamos executar todas as células desde o início, até chegar a célula em que corrigimos o ID do “Lucas”. Ao chegar nessa célula, em vez de 5, colocaremos o 3.

```
# Trocando na base o nome "Lucas" pelo ID_aluno 5  
base.loc[base.ID_aluno == 'Lucas', 'ID_aluno'] = 3
```

E de cara, com o resultado do **nunique()**, já percebemos que temos diferenças nos resultados.

```
# Nesse caso o nunique continua do mesmo tamanho  
base.ID_aluno.nunique()
```

13

## Módulo 21 – Eliminando valores duplicados e discutindo sobre o tratamento do ID\_aluno

Seguindo a execução do código, vamos ter duas novas linhas, ao visualizar as linhas duplicadas.

```
# Filtrando os ID's  
base[base.ID_aluno.isin(filtrar_ids)]
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
2	3	3	1.69	M	1650.0
3	4	4	1.71	G	1770.0
8	9	10	1.80	NaN	1640.0
10	11	4	1.71	G	1770.0
12	13	3	1.65	M	1630.0
15	16	10	1.80	GG	1640.0

E exatamente por isso, teríamos que eliminar também a linha de índice 2, visando remover a duplicidade do aluno de **ID=3**.

```
# Retirando essas duas Linhas da base  
base = base.drop([2, 3, 8],axis=0)
```

## Módulo 21 – Eliminando valores duplicados e discutindo sobre o tratamento do ID\_aluno

E agora, ao observar os resultados, nosso resposta seria de **13 alunos**.

Isso acontece, justamente, porque nessa exemplificação estamos considerando que o “Lucas” já tinha respondido uma vez. Ou seja, ele teria respondido uma vez com o ID correto, e outra com o **ID=“Lucas”**.

Para casos como esse, é **especialmente importante, realizar o tratamento antes da contagem**. Já que não teríamos a certeza se o **ID=“Lucas”** é ou não uma linha duplicada.

```
# Contando a quantidade de Linhas  
base.shape[0]
```

13

```
# Usando o nunique  
base.ID_aluno.nunique()
```

13

## Módulo 21 – Eliminando valores duplicados e discutindo sobre o tratamento do ID\_aluno

Agora que já exemplificamos essa importância, lembre-se de reexecutar as células desde o início, substituindo o **3 por 5** e **sem remover a linha de índice 2**.

```
# Trocando na base o nome "Lucas" pelo ID_aluno 5  
base.loc[base.ID_aluno == 'Lucas', 'ID_aluno'] = 5
```

```
# Retirando essas duas Linhas da base  
base = base.drop([3, 8], axis=0)
```

Com isso, vamos obter novamente o resultado esperado, que é de **14 alunos**.

```
# Contando a quantidade de Linhas  
base.shape[0]
```

14

```
# Usando o nunique  
base.ID_aluno.nunique()
```

14

Que já é bem diferente da nossa resposta sem tratamentos realizados, de 16 alunos, que não fazia sentido, já que na turma só temos 15.

```
res_1
```

16

### 2. Qual a soma da mensalidade dos alunos que responderam?

Partiremos agora, para a segunda pergunta.

Já tratamos duplicidades de linhas já foi tratada anteriormente, mas é importante verificar outro fatores, como o tipo dos dados, e os valores em si.

Inicialmente, visualizando o **describe()** da coluna de mensalidade, temos o tipo correto (**float64**).

O desvio padrão, relativamente alto, já é um indicativo de que algo possa estar errado. E ao olhar para o valor mínimo temos um valor bem fora do contexto, já que a média e mediana estão na casa dos **1600**, e esse valor mínimo é de **1.66**.

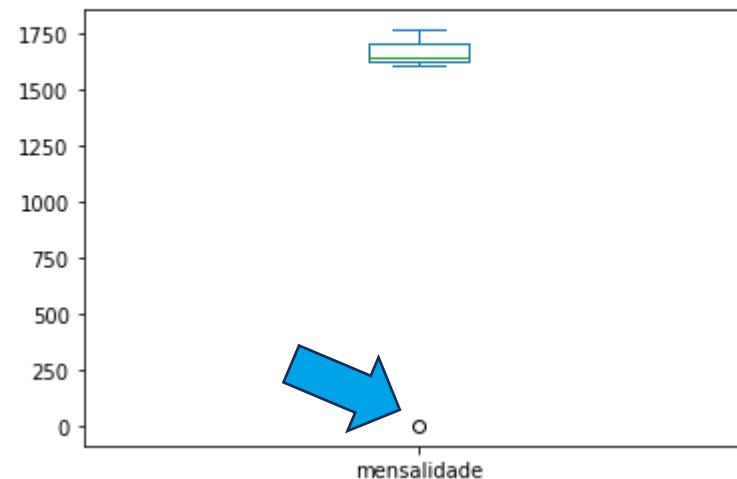
# Podemos verificar as informações estatísticas dessa base	
count	14.000000
mean	1553.690000
std	449.543883
min	1.660000
25%	1630.000000
50%	1650.000000
75%	1707.500000
max	1770.000000
Name:	mensalidade, dtype: float64

## Módulo 21 – Somando a matrícula dos alunos que responderam (visualizando e tratando outliers)

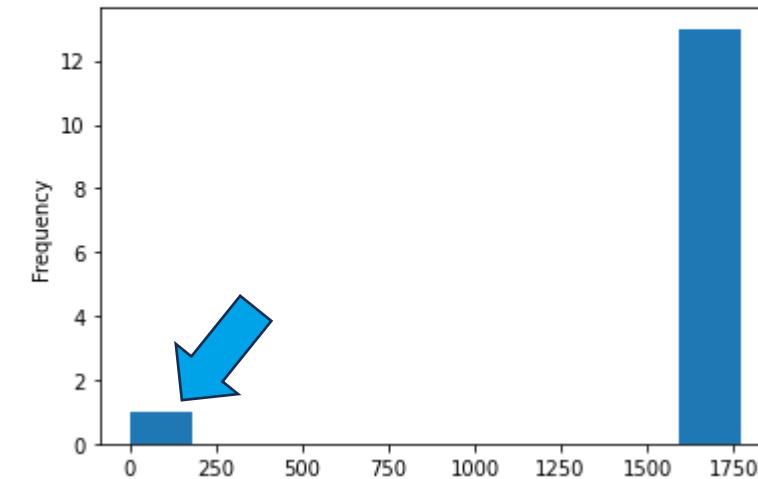
Já poderíamos conversar com o negócio, e provavelmente eles informariam que não existe mensalidade nesse valor, corroborando que é realmente algum tipo de problema.

Para entendermos melhor, podemos também visualizar graficamente essa coluna.

# Visualizando o boxplot  
base.mensalidade.plot.box();



# Visualizando o histograma  
base.mensalidade.plot.hist();



No dois gráficos, visualizamos claramente que há um ponto bem destoante dos demais, esse ponto, é considerado um **outlier**.

## Módulo 21 – Somando a matrícula dos alunos que responderam (visualizando e tratando outliers)

E para visualizar diretamente a linha que têm esse valor, podemos filtrar o valor mínimo da nossa mensalidade.

```
# Selecionando o mínimo da mensalidade  
base[base.mensalidade == base.mensalidade.min()]
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
0	1	1	1.68	M	1.66

Muito provavelmente, esse valor deve ter sido com algum tipo de erro de digitação, em que o **usuário iria inserir 1660 e acabou inserindo 1.66**. Podemos assumir essa premissa, e **substituir esse valor de 1.66, por 1660**.

```
# Substituindo por 1.660  
base.loc[base.mensalidade == 1.66, 'mensalidade'] = 1660
```

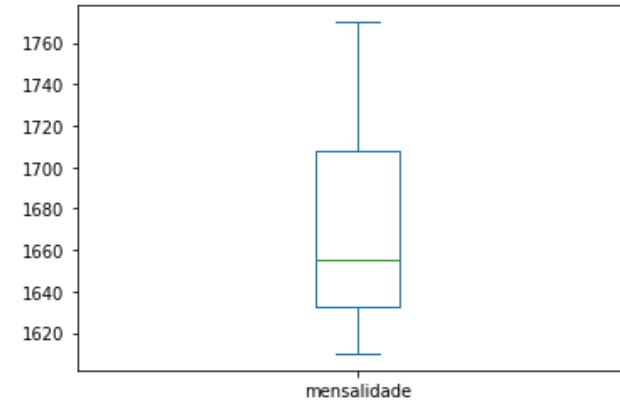
## Módulo 21 – Somando a matrícula dos alunos que responderam (visualizando e tratando outliers)

Ao executar a célula anterior, e visualizar tanto o **describe()**, como os gráficos, agora sim veremos valores que fazem bem mais sentido no contexto do nosso negócio.

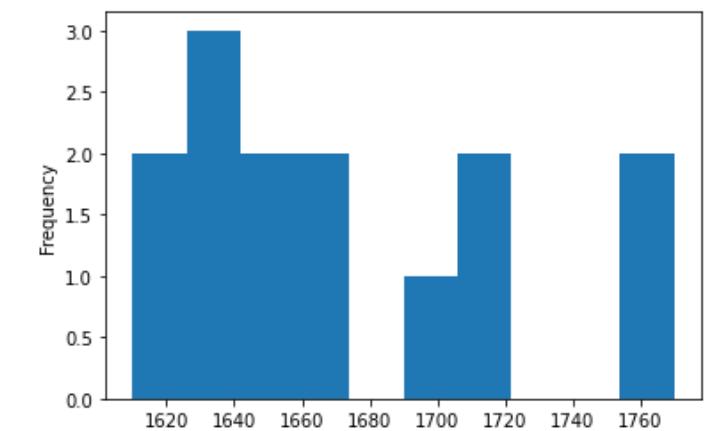
```
# Podemos verificar as informações estatísticas dessa base  
base.mensalidade.describe()
```

```
count      14.000000  
mean     1672.142857  
std       50.562767  
min     1610.000000  
25%    1632.500000  
50%    1655.000000  
75%    1707.500000  
max     1770.000000  
Name: mensalidade, dtype: float64
```

```
# Visualizando o boxplot  
base.mensalidade.plot.box();
```



```
# Visualizando o histograma  
base.mensalidade.plot.hist();
```



## Módulo 21 – Somando a matrícula dos alunos que responderam (visualizando e tratando outliers)

E agora sim podemos calcular, com confiança, a soma das mensalidades.

```
# Somando as mensalidades  
base.mensalidade.sum()  
  
23410.0
```

De forma que, a resposta para a pergunta, de qual é a soma das mensalidades, é **23410.0**. Com uma diferença considerável se comparada a resposta sem tratamento, que era de **25161.66**.

res_2	# diferença percentual (res_2 / base.mensalidade.sum() -1) * 100
25161.66	7.482528833831692

Note que a duplicidade de linhas, e a presença de outliers (que não fazem parte do contexto do negócio), podem trazer prejuízos a informações essenciais ao negócio, pois reportar um valor de faturamento **7% maior do que o real**, pode gerar, por exemplo, **um investimento maior do que o negócio pode pagar**.

## Módulo 21 – Verificando o tamanho da blusa para todos os alunos

### 3. Qual o tamanho da blusa do aluno de ID\_aluno = 10?

Lembra que o aluno de ID=10 tinha respostas duplicadas?

O interessante é que, os tratamentos anteriores já corrigiram os problemas que existiam para responder essa pergunta. Com isso, podemos apenas filtrar a linha do aluno com **ID=10**, e veremos que o tamanho da blusa é “**GG**”.

# Filtrando a base pra esse aluno base[base.ID_aluno == 10]						
ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura	
15	16	10	1.8	GG	1640.0	S

## Módulo 21 – Verificando o tamanho da blusa para todos os alunos

### 3. Qual o tamanho da blusa do aluno de ID\_aluno = 9?

Diferente da pergunta anterior, o aluno com **ID=9** realmente tem seu tamanho de camisa vazio. Com isso, será necessário pensarmos em uma estratégia para preencher esse valor.

# Filtrando a base pra esse aluno  
base[base.ID\_aluno == 9]

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
7	8	9	1.79	NaN	1650.0

## Módulo 21 – Verificando o tamanho da blusa para todos os alunos

Inicialmente, podemos assumir que vamos entregar o tamanho de camisa mais frequente. Mais especificamente, podemos usar o **value\_counts()** na coluna de tamanhos.

E nessa situação, o tamanho de camisa escolhido para o aluno de **ID=9** seria o “**M**”. Mas será que isso faz sentido? Será que pegar o tamanho de camisa mais frequente vai fazer sentido para o aluno em questão?

Como temos a altura de cada um dos alunos, podemos pensar em uma estratégia, que leve em consideração esse valor, já que é esperado que, minimamente, a altura tenha relação com o tamanho de roupa que uma pessoa usa.

```
# Contando a quantidade de camisa em cada tamanho  
base.tamanho_camisa.value_counts()  
  
M    6  
G    5  
GG   3  
Name: tamanho_camisa, dtype: int64
```

## Módulo 21 – Verificando o tamanho da blusa para todos os alunos

Vamos iniciar essa estratégia agrupando a altura por tamanho de camisa. De forma que desse agrupamento, podemos extrair o mínimo o máximo para extrair uma faixa de alturas para cada um dos tamanhos de camisa, e a contagem para verificar a validade de cada uma das faixas (quantidade satisfatória de registros).

```
# Selecionando o mínimo  
minimo = base.groupby('tamanho_camisa')['altura'].min()  
minimo
```

```
tamanho_camisa  
G    1.70  
GG   1.78  
M    1.65  
Name: altura, dtype: float64
```

```
# Selecionando o máximo  
maximo = base.groupby('tamanho_camisa')['altura'].max()  
maximo
```

```
tamanho_camisa  
G    1.74  
GG   1.80  
M    1.69  
Name: altura, dtype: float64
```

```
# Selecionando o número de registros  
contagem = base.groupby('tamanho_camisa')['altura'].count()  
contagem
```

```
tamanho_camisa  
G     5  
GG    3  
M     5  
Name: altura, dtype: int64
```

Em seguida, podemos usar a função **pd.concat()** para unir os resultados.

```
# Unindo tudo  
pd.concat([minimo,maximo,contagem],axis=1)
```

	altura	altura	altura	
tamanho_camisa	G	1.70	1.74	5
	GG	1.78	1.80	3
	M	1.65	1.69	5

## Módulo 21 – Verificando o tamanho da blusa para todos os alunos

Uma outra forma, de realizar essa tarefa de forma mais direta, é usando o método **agg()** logo após o **groupby**, nele podemos passar, de uma só vez, várias operações que queremos de retorno.

```
# Filtrando a base pra esse aluno  
base[base.ID_aluno == 9]
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
7	8	9	1.79	NaN	1650.0

```
# Ou podemos fazer de uma única vez utilizando o agg junto ao groupby  
base.groupby('tamanho_camisa').agg({'altura': ['min', 'max', 'count']})
```

tamanho_camisa	altura		
	min	max	count
G	1.70	1.74	4
GG	1.78	1.80	3
M	1.65	1.69	5

Com isso, temos que o aluno de **ID=9**, que tem **1.79 de altura**, está dentro da faixa de alunos que usam a camisa de tamanho “**GG**”. Então, é uma estimativa bem mais satisfatória dizer que esse aluno usará a camisa de tamanho “**GG**”.

## Módulo 21 – Verificando o tamanho da blusa para todos os alunos

Então, iniciamos sem nenhuma estimativa para o tamanho de camisa desse aluno, cogitamos usar “M” porque era o mais frequente, porém, preferimos usar o “GG” como estimativa, porque estamos relacionando a altura com o tamanho, algo que deve ser bem mais assertivo.

```
# Podemos considerar que esse aluno irá utilizar a camisa de tamanho GG  
base.loc[base.tamanho_camisa.isnull(),'tamanho_camisa'] = 'GG'
```

Após atribuir o tamanho de camisa para esse aluno, podemos verificar que não há mais valores nulos na coluna de tamanho da camisa.

```
# Verificando as informações da base  
base[base.tamanho_camisa.isnull()]
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
-----------------	----------	--------	----------------	-------------	---------------------

# Módulo 21 – Descobrindo quantos alunos vão participar da formatura

## 5. Quantos alunos vão participar da formatura?

Podemos iniciar visualizando novamente a base.

```
# ReLembrando a base  
base.head(3)
```

	ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
0	1	1	1.68	M	1660.0	S
1	2	2	1.66	M	1610.0	N
2	3	3	1.69	M	1650.0	S

Com os tratamentos já realizados anteriormente, se simplesmente contarmos quantos alunos responderam “S”, vamos obter a resposta **10**. Que já é diferente da nossa primeira resposta sem tratamentos que era **11**. Mas será que essa resposta já está correta?

```
base[base.participa_formatura == "S"].shape[0]
```

10

```
res_5
```

11

## Módulo 21 – Descobrindo quantos alunos vão participar da formatura

Conforme a orientação geral, a coluna **participa\_formatura** deve ter apenas “S” ou “N”. Vamos usar o método **value\_counts()** para verificar se está tudo conforme o esperado.

```
# Verificando os valores em participa_formatura
base.participa_formatura.value_counts()

S      10
Sim     2
N       1
Não     1
Name: participa_formatura, dtype: int64
```

E note que, além de “S” e “N”, os alunos também responderam “Sim” e “Não”. E no caso, como o formulário permitiu o usuário digitar, poderíamos ter, ainda mais formas diferentes de resposta.

Uma abordagem que tomar para corrigir esse problema, é pegar apenas a primeira letra da resposta.

## Módulo 21 – Descobrindo quantos alunos vão participar da formatura

Para isso, podemos criar uma coluna de checagem com essa operação.

Para extrair o primeiro caractere podemos usar o **.str.get(0)** ou o **.str[0]**. Lembrando que o 0 é a posição do caractere que queremos na **string**.

```
# Pegando apenas a primeira letra de cada valor
# Poderia ser base['participa_formatura_check'] = base.participa_formatura.str.get(0)
base['participa_formatura_check'] = base.participa_formatura.str[0]
base
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura	participa_formatura_check
0	1	1.68	M	1660.0	S	S
1	2	1.66	M	1610.0	N	N
2	3	1.69	M	1650.0	S	S
4	5	1.66	M	1760.0	Sim	S
5	6	NaN	M	1620.0	S	S
6	7	1.70	G	1670.0	S	S
7	8	1.79	GG	1650.0	S	S
9	10	1.78	GG	1710.0	S	S
10	11	1.71	G	1770.0	Sim	S
11	12	1.78	GG	1630.0	S	S
12	13	1.65	M	1630.0	S	S
13	14	1.74	G	1700.0	Não	N
14	15	1.71	G	1710.0	S	S
15	16	1.80	GG	1640.0	S	S

## Módulo 21 – Descobrindo quantos alunos vão participar da formatura

Após visualizar que tudo parece correto, podemos atribuir os valores a nossa coluna original.

```
# Pegando apenas a primeira letra de cada valor  
base['participa_formatura_check'] = base.participa_formatura.str[0]  
base['participa_formatura'] = base.participa_formatura.str[0]
```

E agora, podemos visualizar novamente o resultado do método value\_counts().

```
# Contando novamente  
base.participa_formatura.value_counts()  
  
S    12  
N     2  
Name: participa_formatura, dtype: int64
```

Agora sim temos uma resposta consistente para a pergunta 5, de que **12 pessoas vão participar da formatura**, que também é diferente da nossa resposta sem tratamentos realizados. Por fim, é interessante eliminar a coluna **participa\_formatura\_check**, já que ela foi criada apenas para validar a nossa operação.

```
# E eliminando a coluna que criamos para visualização  
base = base.drop('participa_formatura_check', axis=1)
```

## Módulo 21 – Estimando a altura de um aluno usando média e mediana dos dados

### 6. Qual a altura estimada do aluno de ID\_aluno = 7?

Vamos iniciar filtrando a linha desse aluno.

```
# Filtrando a base pra esse aluno
base[base.ID_aluno == 7]
```

ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura
5	6	7	NaN	M	1620.0

Note que temos um valor de altura vazio. E para estimar esse valor, temos várias opções, podendo usar, por exemplo a média ou a mediana das alturas.

```
# Poderia usar a média da altura de todos os alunos
base.altura.mean()
```

1.7192307692307691

```
# Também poderia usar a mediana
base.altura.median()
```

1.71

## Módulo 21 – Estimando a altura de um aluno usando média e mediana dos dados

Para melhorar nossa estimativa, podemos também, usar uma premissa que já utilizamos anteriormente, que é, o tamanho da blusa tem relação com a altura. Como o aluno de **ID=7** têm o tamanho da camisa “**M**”, podemos usar a **média** ou a **mediana** filtrando apenas os alunos que têm o mesmo tamanho de camisa.

```
# Ou, simplesmente, usar a média de altura dos alunos que vestem camisa M  
base.loc[base.tamanho_camisa == 'M','altura'].mean()
```

1.668

```
# Ou a median  
valor_altura = base.loc[base.tamanho_camisa == 'M','altura'].median()  
valor_altura
```

1.66

Os dois valores deram bem próximos, mas por questão de escolha, vamos usar a mediana, que é **1.66**, como estimativa para preencher a altura desse aluno.

```
# Por fim, eu escolhi usar a mediana dos alunos com altura M  
base.loc[base.altura.isnull(),'altura'] = valor_altura
```

## Módulo 21 – Estimando a altura de um aluno usando média e mediana dos dados

Com isso, respondemos todas as perguntas e podemos visualizar a base final totalmente tratada.

# Visualiza a base final base						
ID_questionario	ID_aluno	altura	tamanho_camisa	mensalidade	participa_formatura	
0	1	1	1.68	M	1660.0	S
1	2	2	1.66	M	1610.0	N
2	3	3	1.69	M	1650.0	S
4	5	6	1.66	M	1760.0	S
5	6	7	1.66	M	1620.0	S
6	7	8	1.70	G	1670.0	S
7	8	9	1.79	GG	1650.0	S
9	10	11	1.78	GG	1710.0	S
10	11	4	1.71	G	1770.0	S
11	12	12	1.78	GG	1630.0	S
12	13	5	1.65	M	1630.0	S
13	14	14	1.74	G	1700.0	N
14	15	15	1.71	G	1710.0	S
15	16	10	1.80	GG	1640.0	S

## Módulo 21 – Estimando a altura de um aluno usando média e mediana dos dados

Vamos agora realizar algumas simulações.

Inicialmente vamos ver o que faríamos se o aluno de **ID=15** tivesse preenchido **171** em vez de **1.71** na altura e se o aluno de **ID=7** não tivesse preenchido sua altura.

Para não “sujar” a nossa base original, podemos criar uma cópia.

```
# Criando uma segunda base apenas para visualizarmos  
base2 = base.copy()
```

E nessa cópia, podemos então filtrar o aluno de **ID=7** e substituir a sua altura por nulo. Para isso, podemos importar o **numpy**, pois ele nos entrega uma forma de colocar esse valor nulo.

```
# Apagando a altura do aluno de ID = 7  
import numpy as np  
base2.loc[base2.ID_aluno == 7, 'altura'] = np.nan
```

## Módulo 21 – Estimando a altura de um aluno usando média e mediana dos dados

Em seguida, podemos filtrar o aluno de **ID=15**, e preencher sua altura com o valor **171**.

```
# E usando 171 para altura do aluno de ID = 15  
base2.loc[base2.ID_aluno == 15, 'altura'] = 171
```

Agora ao calcular a média, teremos um valor bem absurdo.

```
# Calculando novamente a média da base  
base2.altura.mean()  
  
14.360769230769233
```

Porém, como a mediana é robusta a outliers, teremos um resultado dentro do normal.

```
# E a mediana  
base2.altura.median()  
  
1.71
```

## Módulo 21 – Estimando a altura de um aluno usando média e mediana dos dados

Podemos seguir novamente a estratégia do tamanho das camisas, agrupando e calculando mínimo e máximo.

```
# Podemos fazer o groupby novamente  
base2.groupby('tamanho_camisa').agg({'altura': ['min','max']})
```

tamanho_camisa	altura	
	min	max
G	1.70	171.00
GG	1.78	1.80
M	1.65	1.69

Como o aluno de **ID=7** usa a camisa de tamanho “M”, basicamente não teríamos grandes diferenças em estimar a sua altura, usando média ou mediana, já que o outlier não está nesse filtro.

```
# Calculando a média dos alunos que vestem M  
base2.loc[base.tamanho_camisa == 'M','altura'].mean()
```

1.668

```
# E a mediana desses alunos  
base2.loc[base.tamanho_camisa == 'M','altura'].median()
```

1.66

## Módulo 21 – Estimando a altura de um aluno usando média e mediana dos dados

Mas se continuássemos com um outlier, porém agora, sendo o aluno de **ID=6**, que em vez de preencher **1.66**, preencheu **166**. Para isso vamos executar novamente as células desde da criação da cópia da base original.

```
# Criando uma segunda base apenas para visualizarmos  
base2 = base.copy()  
  
# Apagando a altura do aluno de ID = 7  
import numpy as np  
base2.loc[base2.ID_aluno == 7,'altura'] = np.nan
```

A seguir em vez de modificar o aluno de **ID=15**, vamos modificar o de **ID=6**.

```
# E usando 166 para altura do aluno de ID = 6  
base2.loc[base2.ID_aluno == 6,'altura'] = 166
```

E depois vamos calcular a média e mediana gerais, obtendo valores relativamente próximos aos da situação anterior.

```
# Calculando novamente a média da base  
base2.altura.mean()
```

14.360769230769233

```
# E a mediana  
base2.altura.median()
```

1.71

## Módulo 21 – Estimando a altura de um aluno usando média e mediana dos dados

A grande diferença vem agora, já que dessa vez, o outlier está inserido dentro do grupo de alunos com tamanho de camisa “M”.

```
# Podemos fazer o groupby novamente  
base2.groupby('tamanho_camisa').agg({'altura': ['min', 'max']})
```

tamanho_camisa	altura	
	min	max
G	1.70	1.74
GG	1.78	1.80
M	1.65	166.00

Então, nesse caso vai fazer diferença a escolha de usar média ou mediana.

```
# Calculando a média dos alunos que vestem M  
base2.loc[base.tamanho_camisa == 'M', 'altura'].mean()
```

34.536

```
# E a mediana desses alunos  
base2.loc[base.tamanho_camisa == 'M', 'altura'].median()
```

1.68

## Módulo 21 – Estimando a altura de um aluno usando média e mediana dos dados

De forma, que ao calcular a média dos alunos com tamanho de camisa “M”, temos o valor de **34.536**, enquanto que a mediana é de **1.68**.

Nesse contexto, para estimar o valor de altura do aluno de **ID=7**, faz muito mais sentido usar a mediana, ou seja, o valor de **1.68**.

Então, de forma geral, se a temos um número considerável de outliers na nossa coluna, pode ser mais interessante olhar para a mediana, em vez da média.

**Obs:** Claro que poderíamos tratar o outlier, mas a ideia dessa simulação foi demonstrar as diferenças entre média e mediana, no contexto de preenchimento de valores nulos.

## Módulo 21 – Aprofundando no tratamento de dados: Entendendo a base de notas de português

Vamos partir para outro exemplo, utilizando o mesmo arquivo **alunos.xlsx**, porém agora nas abas **notas\_port** e **notas\_mat**. Nosso objetivo será calcular a média final de cada aluno.

Iniciaremos analisando as notas de português.

```
# Importando o pandas
import pandas as pd

# Importando a base de notas de português
port = pd.read_excel('alunos.xlsx',sheet_name='notas_port')

# Visualizando a base
port.head(3)
```

ID_aluno	Prova	Notas	
0	1	1	10
1	2	1	9
2	3	1	9

## Módulo 21 – Aprofundando no tratamento de dados: Entendendo a base de notas de português

Agora podemos partir para uma rotina de passos que normalmente vamos executar. Iniciando com a visualização do **describe()**.

	ID_aluno	Prova	Notas
count	48.000000	48.000000	48.000000
mean	7.916667	1.979167	8.541667
std	4.241805	0.837666	9.394336
min	1.000000	1.000000	2.000000
25%	4.750000	1.000000	5.000000
50%	7.500000	2.000000	8.000000
75%	11.250000	3.000000	9.000000
max	15.000000	3.000000	70.000000

E já conseguimos visualizar que, por exemplo, a maior nota da prova foi **70**, algo que parece incorreto. Imagine que perguntamos a secretaria da escola, e descobrimos que as provas valem entre **0 e 10**.

## Módulo 21 – Aprofundando no tratamento de dados: Entendendo a base de notas de português

Podemos visualizar também o método `info()`.

Visualizando essas informações, observamos que a base não tem nulos e os tipos parecem estar corretos.

Vamos ver também a cardinalidade de cada uma das colunas.

E aqui já parece haver um problema, pois se temos **15 alunos** únicos e **3 provas** únicas, deveríamos ter no máximo **45 registros**, porém temos **48 registros** na nossa base.

**Obs:** A partir desse contexto, sugerimos que você tente fazer a limpeza da base de português e matemática, para então comparar com a solução a seguir.

# Verificando as informações  
port.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48 entries, 0 to 47
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   ID_aluno    48 non-null    int64  
 1   Prova       48 non-null    int64  
 2   Notas       48 non-null    int64  
dtypes: int64(3)
memory usage: 1.2 KB
```

# E a cardinalidade  
port.nunique()

ID_aluno	15
Prova	3
Notas	10

dtype: int64

## Módulo 21 – Usando o drop\_duplicates para retirar valores duplicados da base

Diante disso, podemos buscar por valores duplicados na nossa base.

Inicialmente vamos verificar novamente a quantidade de alunos únicos com o **nunique()**, e também quantos alunos fizeram cada uma das provas, com o **value\_counts()**.

```
# Verificando o número de alunos que temos na base
port.ID_aluno.nunique()

15

# E quantos alunos fizeram cada uma das provas
port.Prova.value_counts()

1    17
3    16
2    15
Name: Prova, dtype: int64
```

Com isso, temos realmente provas com mais de **15 registros**, e mesmo na prova 2, em que temos os 15 registros, podem haver duplicados.

## Módulo 21 – Usando o drop\_duplicates para retirar valores duplicados da base

Para visualizar melhor, podemos usar o **value\_counts()** nas colunas **ID\_aluno** e **Prova**.

```
# Podemos usar o value_counts considerando essas duas colunas para verificar os valores duplicados
port[['ID_aluno','Prova']].value_counts().head(3)
```

ID aluno	Prova	
7	1	3
6	3	2
1	1	1

dtype: int64

E assim temos que o aluno de **ID=7** está com **3 registros para a prova 1**, enquanto o aluno de **ID=6** está com **2 registros para a prova 3**.

Ao filtrar, por exemplo, o aluno de **ID=7**, teremos exatamente isso, de forma que as nota da **prova 1 está triplicada**.

```
# Filtrando o primeiro aluno dos registros
port[port.ID_aluno == 7]
```

ID_aluno	Prova	Notas	
6	7	1	10
7	7	1	10
8	7	1	10
23	7	2	4
39	7	3	3

## Módulo 21 – Usando o drop\_duplicates para retirar valores duplicados da base

E esse fato, vai causar um incoerência quando calcularmos a média.

```
# Verificando a média da nota do aluno de ID = 7  
port.loc[port.ID_aluno == 7, 'Notas'].mean()
```

7.4

Quando calculamos a média desse aluno, dá forma que a base está agora, obtemos o valor de **7.4**, que poderia dar a aprovação ao aluno.

Agora que já visualizamos a inconsistência de resultados que esses duplicados podem causar, vamos utilizar o método **drop\_duplicates()**([Documentação](#)), que nos ajuda, de forma muito simples, a remover linhas que estejam repetidas.

```
# Utilizando o drop_duplicates  
port = port.drop_duplicates()
```

E com a remoção dessas linhas duplicadas, podemos refazer o cálculo da média, em que vamos obter o valor correto de **5.667**.

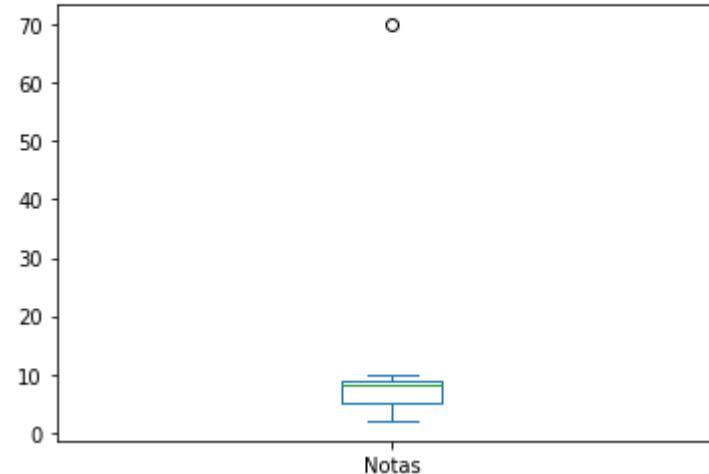
```
# Verificando novamente a média da nota desse aluno  
port.loc[port.ID_aluno == 7, 'Notas'].mean()
```

5.666666666666667

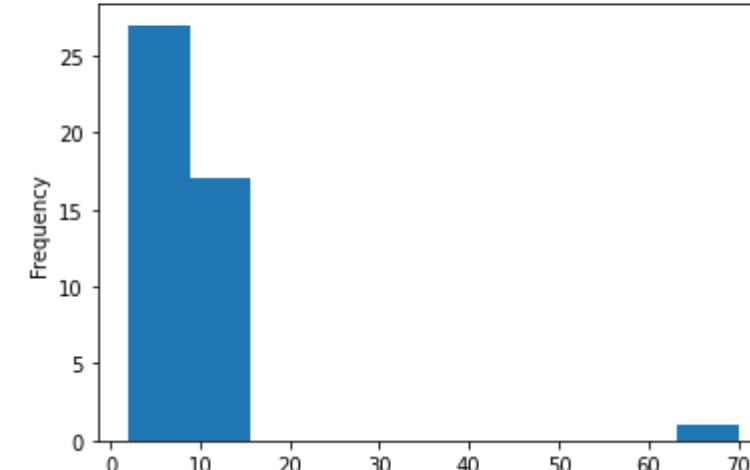
## Módulo 21 – Analisando o describe e o boxplot e tratando outliers nos dados

Agora que já tratamos os duplicados, podemos seguir para o outro problema que visualizamos, um valor **outlier**. Podemos visualizar facilmente esse **outlier**, com o **boxplot** ou com o **histograma**.

```
# Verificando o outlier da coluna nota  
port.Notas.plot.box();
```



```
# Verificando o outlier da coluna nota  
port.Notas.plot.hist();
```



Com esses gráficos, vemos claramente um ponto totalmente fora do contexto do problema.

## Módulo 21 – Analisando o describe e o boxplot e tratando outliers nos dados

Lembrando que nem todo outlier é necessariamente um problema, já que muitas vezes ele faz parte do negócio. Mas, no nosso caso, ele está fora do intervalo de notas que aceitamos. Com isso, podemos assumir que houve erro de digitação, e na verdade a nota é **7**, em vez de **70**.

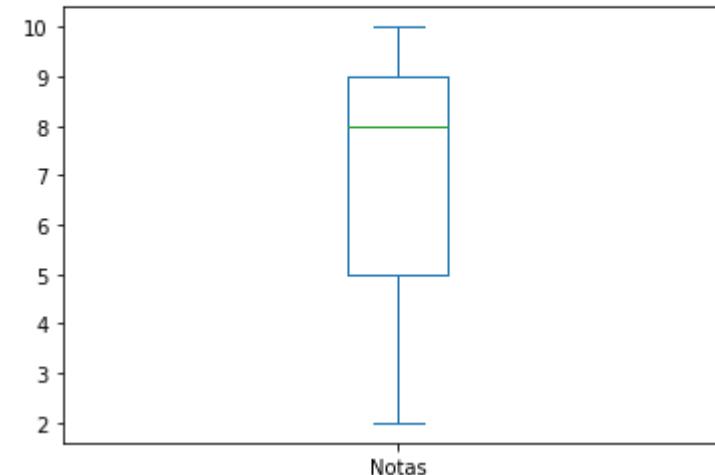
```
# Tratando esse valor  
port.loc[port.Notas == 70, 'Notas'] = 7
```

Após essa alteração, ao visualizarmos, a **Descrição estatística**, o **boxplot** e o **histograma**, teremos resultados dentro do esperado.

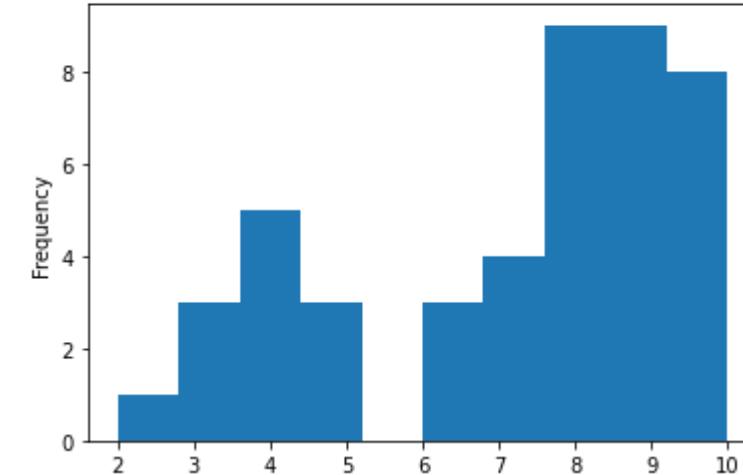
```
# Verificando novamente o describe  
port.describe()
```

	ID_aluno	Prova	Notas
count	45.000000	45.000000	45.000000
mean	8.000000	2.000000	7.222222
std	4.369314	0.825723	2.373039
min	1.000000	1.000000	2.000000
25%	4.000000	1.000000	5.000000
50%	8.000000	2.000000	8.000000
75%	12.000000	3.000000	9.000000
max	15.000000	3.000000	10.000000

```
# Verificando o outlier da coluna nota  
port.Notas.plot.box();
```



```
# Verificando o outlier da coluna nota  
port.Notas.plot.hist();
```



## Módulo 21 – Analisando o describe e o boxplot e tratando outliers nos dados

Por fim, agora que já realizamos os tratamentos necessários na base, podemos usar o **groupby**, para calcular a média de cada um dos alunos.

```
# Fazendo a média por aluno
media_port = port.groupby('ID_aluno')['Notas'].mean()
media_port
```

ID_aluno	Notas
1	9.000000
2	8.000000
3	8.666667
4	8.000000
5	5.666667
6	7.333333
7	5.666667
8	7.666667
9	8.000000
10	6.666667
11	8.666667
12	6.000000
13	4.333333
14	6.666667
15	8.000000

Name: Notas, dtype: float64

Vamos partir agora para a aba “**notas\_mat**”, no mesmo arquivo **alunos.xlsx**.

```
# Importando a base  
mat = pd.read_excel('alunos.xlsx',sheet_name='notas_mat')  
  
# head  
mat.head(3)
```

ID_aluno	Prova	Notas
0	1	1
1	2	1
2	3	1

Note que, as notas não estão em formato numérico, e sim em formato de conceito, com algo muito próximo ao modelo estadunidense, indo de **D até A+**.

## Módulo 21 – Criando a função para transformar as notas dadas em conceitos (textos) em números de 1 a 10

Continuando, podemos seguir o nosso passo a passo, e visualizar outras informações da base de notas de matemática.

```
# describe  
mat.describe()
```

	ID_aluno	Prova
count	45.000000	45.000000
mean	8.000000	2.000000
std	4.369314	0.825723
min	1.000000	1.000000
25%	4.000000	1.000000
50%	8.000000	2.000000
75%	12.000000	3.000000
max	15.000000	3.000000

```
# info  
mat.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 45 entries, 0 to 44  
Data columns (total 3 columns):  
 #   Column      Non-Null Count  Dtype     
---    
 0   ID_aluno    45 non-null    int64    
 1   Prova       45 non-null    int64    
 2   Notas       45 non-null    object   
dtypes: int64(2), object(1)  
memory usage: 1.2+ KB
```

```
# nunique  
mat.nunique()
```

ID_aluno	15
Prova	3
Notas	10
	dtype: int64

E exceto as notas serem textos, não visualizamos indícios de problemas na base.

E podemos visualizar também o método **value\_counts()** para a coluna de notas.

```
# Visualizando a coluna de notas
mat.Notas.value_counts()

A      12
A-      7
A+      7
B-      5
B       4
C+      3
B+      3
C       2
D       1
C-      1
Name: Notas, dtype: int64
```

Para calcular efetivamente a média do aluno, precisamos que essas notas estejam em formato numérico. E nesse caso, para entender o que cada um desses conceitos representam em formato numérico, é necessário perguntar ao negócio .

- Ao perguntar, a escola informou que **A+ equivale a 10**, **A equivale a 9**, e assim por diante, até **C- equivaler a 2** e **D a 1**.

Com esse entendimento, temos várias formas de resolver esse problema.

Podemos, por exemplo, construir uma função, que receba a nota em formato de texto, e tenha várias condicionais, para verificar e retornar qual é a sua equivalência numérica daquela nota.

```
# Criando uma função para trocar texto pela nota
def ajusta_nota(nota):
    if nota == 'A+':
        return 10
    elif nota == 'A':
        return 9
    elif nota == 'A-':
        return 8
    elif nota == 'B+':
        return 7
    elif nota == 'B':
        return 6
    elif nota == 'B-':
        return 5
    elif nota == 'C+':
        return 4
    elif nota == 'C':
        return 3
    elif nota == 'C-':
        return 2
    elif nota == 'D':
        return 1
    else:
        return nota
```

## Módulo 21 – Criando a função para transformar as notas dadas em conceitos (textos) em números de 1 a 10

Em seguida, basta aplicar essa função a coluna “Notas” dentro do método **apply()**, e salvar esse resultado em uma nova coluna.

```
# Aplicando essa função  
mat['NotasNr'] = mat.Notas.apply(ajusta_nota)
```

E assim já teremos todas as notas transformadas corretamente.

```
# Visualizando a base  
mat.head(10)
```

ID_aluno	Prova	Notas	NotasNr
0	1	B-	5
1	2	A	9
2	3	B	6
3	4	A	9
4	5	C+	4
5	6	B+	7
6	7	B	6
7	8	A	9
8	9	A-	8
9	10	A	9

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Podemos otimizar a função anterior, de várias formas. Por exemplo, criando uma lista com todas as possibilidades de notas conceito, e outra lista com as correspondências em número.

```
# Simplificando a escrita dessa função
lista_notas = ['A+', 'A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', 'C-', 'D']
lista_valor = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Então, em vez da nossa função ter que verificar a nota com várias condicionais, podemos pegar o índice que aquele nota têm na lista de notas, e então pegar esse mesmo índice na lista de valores.

```
def verifica_nota(nota):
    return lista_valor[lista_notas.index(nota)]
```

Teríamos que aplicar essa função novamente, e para diferenciar, podemos guardar o resultado em uma outra coluna.

```
# Aplicando essa função
mat['NotarNr2'] = mat.Notas.apply(verifica_nota)
```

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Se não tiver ficado claro, aqui vamos destrinchar o funcionamento da função anterior. Para isso, pegaremos a primeira nota como exemplo.

```
mat.head(1)
```

ID_aluno	Prova	Notas	NotasNr
0	1	1	B-

```
nota = mat.Notas[0]  
nota
```

```
'B-'
```

Podemos pegar o índice dessa nota com o método **index()** na **lista\_notas**.

```
indice = lista_notas.index(nota)  
indice
```

```
5
```

Em seguida extraímos o valor que está nesse mesmo índice, porém na **lista\_valor**, que coincidentemente também é **5**. Ou seja **B-** será substituído por **5**.

```
lista_valor[indice]
```

```
5
```

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Com isso, teríamos o mesmo resultado, porém com um função um pouco mais otimizada.

# Visualizando a base  
mat.head(3)

	ID_aluno	Prova	Notas	NotasNr	NotarNr2
0	1	1	B-	5	5
1	2	1	A	9	9
2	3	1	B	6	6

Agora que já validamos o nosso resultado, vamos substituir a coluna Notas com uma das colunas que acabamos de criar. Além disso, depois dessa substituição, podemos eliminar as duas colunas auxiliares que criamos.

# Substituindo a coluna Notas por qualquer uma dessas colunas  
mat['Notas'] = mat.NotasNr

# Eliminando as colunas nota 1 e nota 2  
mat = mat.drop(['NotasNr', 'NotarNr2'], axis=1)

# Visualizando a base  
mat.head(3)

	ID_aluno	Prova	Notas
0	1	1	5
1	2	1	9
2	3	1	6

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Por fim, podemos calcular a média de cada um dos alunos para as notas de matemática.

```
# E calculando a média de matemática
media_mat = mat.groupby('ID_aluno')['Notas'].mean()
media_mat
```

ID_aluno	Notas
1	6.333333
2	7.333333
3	6.666667
4	7.666667
5	6.333333
6	7.333333
7	6.333333
8	9.666667
9	6.333333
10	8.333333
11	5.000000
12	8.000000
13	6.333333
14	7.333333
15	9.333333

Name: Notas, dtype: float64

Agora precisamos, de alguma forma, unir e fazer a média das duas disciplinas.

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Vamos iniciar visualizando novamente nossas médias de cada disciplina.

```
# Sendo a média de português  
media_port.head(3)
```

```
ID_aluno  
1 9.000000  
2 8.000000  
3 8.666667  
Name: Notas, dtype: float64
```

```
# E a média de matemática  
media_mat.head(3)
```

```
ID_aluno  
1 6.333333  
2 7.333333  
3 6.666667  
Name: Notas, dtype: float64
```

Nesse caso, podemos usar a função **pd.concat()** para unir os resultados, e o **reset\_index()** para que o “**ID\_aluno**” deixe de ser o índice, e vire efetivamente uma coluna.

```
# Unindo as bases  
media_final = pd.concat([media_port,media_mat],axis=1).reset_index()  
media_final.head()
```

	ID_aluno	Notas	Notas
0	1	9.000000	6.333333
1	2	8.000000	7.333333
2	3	8.666667	6.666667
3	4	8.000000	7.666667
4	5	5.666667	6.333333

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Note que o nome de coluna ficou “**Notas**” para as duas disciplinas, então podemos renomear todas as colunas, para que fique mais entendível.

```
# Ajustando as colunas  
media_final.columns = ['ID_aluno','media_port','media_mat']
```

E por fim, podemos calcular efetivamente a média geral de cada um dos alunos, que é dada por **(media\_port + media\_mat) / 2**.

```
# Calculando a média final  
media_final['final'] = (media_final.media_port + media_final.media_mat) / 2
```

```
# Verificando a base  
media_final.head(5)
```

	ID_aluno	media_port	media_mat	final
0	1	9.000000	6.333333	7.666667
1	2	8.000000	7.333333	7.666667
2	3	8.666667	6.666667	7.666667
3	4	8.000000	7.666667	7.833333
4	5	5.666667	6.333333	6.000000

## Módulo 21 – Apresentando a base de cadastro dos alunos e tratando e-mails escritos errados

Agora vamos partir para uma novo exemplo, usando a aba “**cadastro**”, porém ainda no arquivo **alunos.xlsx**.

Nesse exemplo, devemos levar em consideração as seguintes observações:

- O **TokenAluno** é um código interno composto por:
  - **Gênero** do aluno (**M**=masculino e **F**=feminino)
  - 1 ou 0 para aluno **mensalista** (**1**=mensalista e **0**=bolsista)
  - **Atividade** do aluno (**A**=ativo e **I**=inativo)
  - Aluno **reprovado** (**R**=reprovado em anos anteriores)
- Em atividades, é possível verificar se esse aluno realiza **atividades esportivas, culturais ou ambas**

# Importando o pandas import pandas as pd					
# Importando a base de cadastro cadastro = pd.read_excel('alunos.xlsx',sheet_name='cadastro')					
# Visualizando cadastro.head(5)					
ID_aluno		email	Aniversario	TokenAluno	Atividades
0	1	gustavo@gmail.com	21.11.2005	M1A	Esporte
1	2	miguel@gmail.com	20.06.2005	M1I	NaN
2	3	fernanda@gmail.com	26.11.2005	F1A	Cultural
3	5	lucas@gmail.com	26.08.2006	M1AR	Esporte
4	4	vitor@gmail.com	02.12.2006	M1A	Esporte / Cultural

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Ao olhar para a coluna **Aniversario**, percebemos que ela não está em um formato muito habitual. Então podemos verificar o método **info()**, e realmente vemos que a coluna não está com o tipo de data.

```
cadastro.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   ID_aluno    15 non-null    int64  
 1   email       15 non-null    object  
 2   Aniversario 15 non-null    object  object
 3   TokenAluno  15 non-null    object  
 4   Atividades   11 non-null    object  
dtypes: int64(1), object(4)
memory usage: 728.0+ bytes
```

E na coluna **TokenAluno**, temos muita informação agregada em uma única coluna. E da forma que está atualmente, nosso modelo de **machine learning** não iria conseguir utilizar essa coluna com efetividade.

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Vamos iniciar validando a coluna **email**.

Visualmente já podemos observar alguns problemas, mas podemos, a princípio, verificar se todos os e-mails tem “@”.

ID_aluno	email	Aniversario	TokenAluno	Atividades
13	14 roberta%gmail.com	09.05.2006	F1A	Esporte / Cultural

E note que um dos e-mails está com “%” em vez de “@”. Então, podemos ajudar esse e-mail substituindo o caractere.

```
# Ajustando esse e-mail  
cadastro.loc[cadastro.ID_aluno == 14, 'email'] = 'roberta@gmail.com'
```

	cadastro.email
0	gustavo@gmail.com
1	miguel@gmail.com
2	fernanda@gmail.com
3	lucas@gmail.com
4	vitor@gmail.com
5	aline@gmail.com
6	pedro@gmail.com
7	Lara@gmail.com
8	antonio@gmail.com
9	lua@gmail.com
10	thiago@gmail.com
11	priscila@gmail.com
12	vitoria@gmail.com
13	roberta%gmail.com
14	carlos@gmail.com

Name: email, dtype: object

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Partindo para o próximo ponto, temos e-mails com caracteres maiúsculos, e isso não necessariamente é um problema, mas o ideal é que nossa base esteja padronizada. Por isso, vamos alterar tudo para minúsculo aplicando o método **lower()** na coluna **email**.

```
# Colocando todos os valores para lowercase
cadastro['email'] = cadastro.email.str.lower()
```

Outro ponto é a existência de espaços, antes ou depois do e-mail. Como um e-mail, por padrão, não aceita espaços, podemos substituir qualquer espaço por nada. Para isso, podemos aplicar o método **replace(" ", "")** na coluna **email**.

```
# Atualizando a base
cadastro['email'] = cadastro.email.str.replace(' ', '')
```

	cadastro.email
0	gustavo@gmail.com
1	miguel@gmail.com
2	fernanda@gmail.com
3	lucas@gmail.com
4	vitor@gmail.com
5	aline@gmail.com
6	pedro@gmail.com
7	Lara@gmail.com
8	antonio@gmail.com
9	lua@gmail.com
10	thiago@gmail.com
11	priscila@gmail.com
12	vitoria@gmail.com
13	roberta@gmail.com
14	carlos@gmail.com

Name: email, dtype: object

## Módulo 21 – Otimizando a função criada, unindo duas bases e calculando a média final dos alunos

Após essas etapas, o resultado é a coluna **email** totalmente tratada e padronizada.

```
cadastro.email  
0      gustavo@gmail.com  
1      miguel@gmail.com  
2      fernanda@gmail.com  
3      lucas@gmail.com  
4      vitor@gmail.com  
5      aline@gmail.com  
6      pedro@gmail.com  
7      lara@gmail.com  
8      antonio@gmail.com  
9      lua@gmail.com  
10     thiago@gmail.com  
11     priscila@gmail.com  
12     vitoria@gmail.com  
13     roberta@gmail.com  
14     carlos@gmail.com  
Name: email, dtype: object
```

**Obs:** Poderíamos ter realizados esses tratamentos de diversas formas, fique à vontade para buscar novas formas de fazer essas limpezas.

## Módulo 21 – Tratando a data e ajustando as colunas de texto no cadastro dos alunos

Partindo agora para a coluna **Aniversario**. Note que ela está como um texto, e podemos então, usar a função **pd.to\_datetime()** para transformar esse texto em data.

```
# Verificando as informações  
cadastro.Aniversario
```

```
0    21.11.2005  
1    20.06.2005  
2    26.11.2005  
3    26.08.2006  
4    02.12.2006  
5    20.12.2005  
6    09.09.2006  
7    05.07.2006  
8    07.08.2005  
9    04.06.2006  
10   27.11.2006  
11   06.04.2006  
12   18.03.2005  
13   09.05.2006  
14   24.07.2006  
  
Name: Aniversario, dtype: object
```

```
# Transformando em data  
cadastro['Aniversario'] = pd.to_datetime(cadastro.Aniversario, format='%d.%m.%Y')
```

```
# Verificando as informações  
cadastro.Aniversario
```

```
0    2005-11-21  
1    2005-06-20  
2    2005-11-26  
3    2006-08-26  
4    2006-12-02  
5    2005-12-20  
6    2006-09-09  
7    2006-07-05  
8    2005-08-07  
9    2006-06-04  
10   2006-11-27  
11   2006-04-06  
12   2005-03-18  
13   2006-05-09  
14   2006-07-24  
  
Name: Aniversario, dtype: datetime64[ns]
```

**Obs:** No **format** passamos o formato em que aquele texto está. Para saber mais de como representar um formato específico, olhe a [documentação](#).

## Módulo 21 – Tratando a data e ajustando as colunas de texto no cadastro dos alunos

A seguir, vamos tratar a coluna **TokenAluno**.

Basicamente, podemos extrair cada um dos caracteres por posição. Em que, o primeiro caractere representa o gênero, o segundo representa ser mensalista ou não, o terceiro representa se está ativo ou não, e o último, se existir, representa que o aluno já foi reprovado em anos anteriores.

```
▼ # O primeiro elemento é o gênero
cadastro['genero'] = cadastro.TokenAluno.str[0]

▼ # Marcando se o aluno é mensalista
cadastro['mensalista'] = cadastro.TokenAluno.str.get(1)

▼ # Fazendo o mesmo para a atividade
cadastro['atividade'] = cadastro.TokenAluno.str[2]

▼ # E para o reprovado
cadastro['reprovado'] = cadastro.TokenAluno.str.get(3)
```

▼ # Verificando a coluna  
cadastro.TokenAluno

```
0      M1A
1      M1I
2      F1A
3      M1AR
4      M1A
5      F1A
6      M1A
7      F0A
8      M1A
9      M1A
10     M1A
11     F0I
12     F1AR
13     F1A
14     M1A
Name: TokenAluno, dtype: object
```

## Módulo 21 – Tratando a data e ajustando as colunas de texto no cadastro dos alunos

Vamos visualizar novamente a base.

Note que, na coluna **reprovado**, temos vários valores nulos, justamente porque não temos o quarto caractere da coluna **TokenAluno** nos casos de não reprovação. Ou seja, o valor nulo, nesse caso, representa que o aluno nunca foi reprovado.

# Visualizando a base cadastro										
ID_aluno		email	Aniversario	TokenAluno	Atividades	genero	mensalista	atividade	reprovado	
0	1	gustavo@gmail.com	2005-11-21	M1A	Esporte	M	1	A	NaN	
1	2	miguel@gmail.com	2005-06-20	M1I	Nan	M	1	I	NaN	
2	3	fernanda@gmail.com	2005-11-26	F1A	Cultural	F	1	A	NaN	
3	5	lucas@gmail.com	2006-08-26	M1AR	Esporte	M	1	A	R	
4	4	vitor@gmail.com	2006-12-02	M1A	Esporte / Cultural	M	1	A	NaN	
5	6	aline@gmail.com	2005-12-20	F1A	Nan	F	1	A	NaN	
6	7	pedro@gmail.com	2006-09-09	M1A	Esporte	M	1	A	NaN	
7	8	lara@gmail.com	2006-07-05	F0A	Cultural	F	0	A	NaN	
8	9	antonio@gmail.com	2005-08-07	M1A	Cultural	M	1	A	NaN	
9	10	lua@gmail.com	2006-06-04	M1A	Nan	M	1	A	NaN	
10	11	thiago@gmail.com	2006-11-27	M1A	Esporte / Cultural	M	1	A	NaN	
11	12	priscila@gmail.com	2006-04-06	F0I	Cultural	F	0	I	NaN	
12	13	vitoria@gmail.com	2005-03-18	F1AR	Esporte	F	1	A	R	
13	14	roberta@gmail.com	2006-05-09	F1A	Esporte / Cultural	F	1	A	NaN	
14	15	carlos@gmail.com	2006-07-24	M1A	Nan	M	1	A	NaN	

## Módulo 21 – Tratando a data e ajustando as colunas de texto no cadastro dos alunos

Então, podemos preencher esse valores nulos por “N” representando não reprovado.

```
# Podemos trocar os valores vazios da coluna reprovado por N  
cadastro['reprovado'] = cadastro.reprovado.fillna('N')
```

E com isso, teremos essa coluna tratada.

```
cadastro['reprovado']  
0    N  
1    N  
2    N  
3    R  
4    N  
5    N  
6    N  
7    N  
8    N  
9    N  
10   N  
11   N  
12   R  
13   N  
14   N  
Name: reprovado, dtype: object
```

## Módulo 21 – Tratando a data e ajustando as colunas de texto no cadastro dos alunos

Vamos tratar nossa última coluna, que é a **Atividades**. Nosso objetivo aqui, será criar duas colunas, uma para os alunos com “**Esporte**” e outra para os alunos com “**Cultural**”.

```
cadastro.Atividades  
0           Esporte  
1             NaN  
2        Cultural  
3           Esporte  
4  Esporte / Cultural  
5             NaN  
6           Esporte  
7        Cultural  
8        Cultural  
9             NaN  
10     Esporte / Cultural  
11        Cultural  
12           Esporte  
13     Esporte / Cultural  
14             NaN  
Name: Atividades, dtype: object
```

Para verificar que linhas têm cada uma dessas atividades podemos usar o método **contains**. E um ponto interessante, é que, ao usar esse método, podemos passar o **na=False** como parâmetro, o que fará com que ele considere o **NaN** como **False**.

## Módulo 21 – Tratando a data e ajustando as colunas de texto no cadastro dos alunos

Vamos tratar nossa última coluna, que é a **Atividades**. Nosso objetivo aqui, será criar duas colunas, uma para os alunos com “**Esporte**” e outra para os alunos com “**Cultural**”.

Para verificar que linhas têm cada uma dessas atividades podemos usar o método **contains**. E um ponto interessante, é que, ao usar esse método, podemos passar o **na=False** como parâmetro, o que fará com que ele considere o **NaN** como **False**.

```
# Selecionando alunos que praticam esporte
cadastro['esporte'] = cadastro.Atividades.str.contains('Esporte',na=False)

# E alunos que praticam atividades culturais
cadastro['culturais'] = cadastro.Atividades.str.contains('Cultural',na=False)
```

cadastro.Atividades	
0	Esporte
1	NaN
2	Cultural
3	Esporte
4	Esporte / Cultural
5	NaN
6	Esporte
7	Cultural
8	Cultural
9	NaN
10	Esporte / Cultural
11	Cultural
12	Esporte
13	Esporte / Cultural
14	NaN

Name: Atividades, dtype: object

## Módulo 21 – Tratando a data e ajustando as colunas de texto no cadastro dos alunos

Visualizando novamente a base, temos duas coluna com **True**, se a coluna **Atividades** tem a respectiva palavra, e **False** caso contrário.

# Visualizando a base cadastro												
ID_aluno		email	Aniversario	TokenAluno	Atividades	genero	mensalista	atividade	reprovado	esporte	culturais	
0	1	gustavo@gmail.com	2005-11-21	M1A	Esporte	M	1	A	N	True	False	
1	2	miguel@gmail.com	2005-06-20	M1I	Nan	M	1	I	N	False	False	
2	3	fernanda@gmail.com	2005-11-26	F1A	Cultural	F	1	A	N	False	True	
3	5	lucas@gmail.com	2006-08-26	M1AR	Esporte	M	1	A	R	True	False	
4	4	vitor@gmail.com	2006-12-02	M1A	Esporte / Cultural	M	1	A	N	True	True	
5	6	aline@gmail.com	2005-12-20	F1A	Nan	F	1	A	N	False	False	
6	7	pedro@gmail.com	2006-09-09	M1A	Esporte	M	1	A	N	True	False	
7	8	lara@gmail.com	2006-07-05	F0A	Cultural	F	0	A	N	False	True	
8	9	antonio@gmail.com	2005-08-07	M1A	Cultural	M	1	A	N	False	True	
9	10	lua@gmail.com	2006-06-04	M1A	Nan	M	1	A	N	False	False	
10	11	thiago@gmail.com	2006-11-27	M1A	Esporte / Cultural	M	1	A	N	True	True	
11	12	priscila@gmail.com	2006-04-06	F0I	Cultural	F	0	I	N	False	True	
12	13	vitoria@gmail.com	2005-03-18	F1AR	Esporte	F	1	A	R	True	False	
13	14	roberta@gmail.com	2006-05-09	F1A	Esporte / Cultural	F	1	A	N	True	True	
14	15	carlos@gmail.com	2006-07-24	M1A	Nan	M	1	A	N	False	False	

## Módulo 21 – Tratando a data e ajustando as colunas de texto no cadastro dos alunos

E podemos, por exemplo, em vez de ter **True** ou **False**, usar **1 para True** e **0 para False**. E para isso, podemos aplicar uma função **lambda**, que verifica se é **True** ou **False**.

```
# Podemos também trocas True e False por 1 e 0
cadastro['esporte'] = cadastro.esporte.apply(lambda x:1 if x else 0)
cadastro['culturais'] = cadastro.culturais.apply(lambda x:1 if x else 0)
```

Além disso, de forma geral, já traduzimos as colunas **TokenAluno** e **Atividades** em outras bem mais explicativas. E por isso, é interessante eliminar essas colunas, já que **“tudo que não ajuda, atrapalha”**.

```
# Apagando as colunas
cadastro = cadastro.drop(['TokenAluno', 'Atividades'], axis=1)
```

## Módulo 21 – Tratando a data e ajustando as colunas de texto no cadastro dos alunos

Por fim, temos a base de cadastros totalmente tratada.

# Visualizando a base final cadastro								
	ID_aluno	email	Aniversario	genero	mensalista	atividade	reprovado	
0	1	gustavo@gmail.com	2005-11-21	M	1	A	N	
1	2	miguel@gmail.com	2005-06-20	M	1	I	N	
2	3	fernanda@gmail.com	2005-11-26	F	1	A	N	
3	5	lucas@gmail.com	2006-08-26	M	1	A	R	
4	4	vitor@gmail.com	2006-12-02	M	1	A	N	
5	6	aline@gmail.com	2005-12-20	F	1	A	N	
6	7	pedro@gmail.com	2006-09-09	M	1	A	N	
7	8	lara@gmail.com	2006-07-05	F	0	A	N	
8	9	antonio@gmail.com	2005-08-07	M	1	A	N	
9	10	lua@gmail.com	2006-06-04	M	1	A	N	
10	11	thiago@gmail.com	2006-11-27	M	1	A	N	
11	12	priscila@gmail.com	2006-04-06	F	0	I	N	
12	13	vitoria@gmail.com	2005-03-18	F	1	A	R	
13	14	roberta@gmail.com	2006-05-09	F	1	A	N	
14	15	carlos@gmail.com	2006-07-24	M	1	A	N	

## Módulo 21 – Exercício: limpeza dos dados no dataset do titanic

Agora vamos entrar efetivamente na limpeza de dados de uma base real. Usaremos a base do **titanic**, pois como já temos familiaridade, será mais rápido entender os seus problemas. O arquivo com esses dados está disponível abaixo da respectiva aula, ou diretamente no [Kaggle](#). Vamos importar e visualizar essa base.

```
▼ # Importando o pandas  
import pandas as pd
```

```
▼ # Importando a base do titanic  
titanic = pd.read_csv('train.csv')
```

```
▼ # Visualizando a base  
titanic.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Allen, Mr. William Henry	male	35.0	1	0	113803	53.1000	C123	S
4	5	0	3				0	0	373450	8.0500	NaN	S

## Módulo 21 – Exercício: limpeza dos dados no dataset do titanic

Podemos verificar o shape dessa base. Que tem 891 linhas e 12 colunas.

Além disso, aplicando os método **info()**, observamos que temos valores vazios na coluna **Age** e na coluna **Cabin**. Lembrando que também poderíamos visualizar esse fato diretamente somando o resultado do método **isnan()**.

```
# Contando valores nulos
titanic.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked        2
dtype: int64
```

```
# E seu formato
titanic.shape
```

```
(891, 12)
```

```
# info
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count Dtype  
 --- 
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object  
 4   Sex          891 non-null    object  
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object  
 9   Fare         891 non-null    float64 
 10  Cabin        204 non-null    object  
 11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

## Módulo 21 – Exercício: limpeza dos dados no dataset do titanic

Além disso, podemos visualizar a descrição estatística.

Em que conseguimos notar, por exemplo, valores estranhos na coluna **Fare**, com o **mínimo de 0** (o passageiro foi de graça?), um **máximo de 512**, que é **bem distante da média de 32 e da mediana 14**.

```
# Describe  
titanic.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

## Módulo 21 – Exercício: limpeza dos dados no dataset do titanic

Outro ponto interessante é ver a cardinalidade dos dados. Em que veremos que temos apenas valores únicos nas colunas **PassengerId** e **Nome**.

```
# Cardinalidade
titanic.nunique()
```

PassengerId	891
Survived	2
Pclass	3
Name	891
Sex	2
Age	88
SibSp	7
Parch	7
Ticket	681
Fare	248
Cabin	147
Embarked	3
dtype:	int64

Com isso, vimos algumas análises iniciais. E a ideia a partir daqui, é que você faça todos os tratamentos de limpeza que achar necessário, e só após, siga para os próximos slides e visualize como escolhemos tratar esses problemas.

## Módulo 21 – Tratando as informações de embarque vazias e usando a mediana para as idades

Agora que já visualizamos alguns problemas iniciais, partiremos para a limpeza.

Iniciando com a coluna **Embarked**, vamos verificar os registros que estão vazios.

# Verificando os registros com embarque vazio titanic[titanic.Embarked.isnull()]												
PassengerId	Survived	Pclass		Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0	B28	NaN
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	113572	80.0	B28	NaN

São apenas duas linhas, então caberia perguntar ao negócio se temos essa informação em algum local. Mais especificamente, pôr a tragedia do titanic ser um fato tão famoso, conseguimos encontrar várias informações desses passageiros ao pesquisar no google.

**Obs:** Como são apenas duas linhas, poderíamos até pensar em excluir, já que o impacto não deve ser tão grande.

## Módulo 21 – Tratando as informações de embarque vazias e usando a mediana para as idades

Ao pesquisar, encontramos as seguintes informações:

**Name:** Miss Rose Amélie Icard

Titanic Survivor

**Born:** Thursday 31st October 1872 in

**Age:** 39 years 5 months and 15 days (Female)

**Nationality:** French

**Marital Status:** Single

**Occupation:** Personal Maid to Mrs Martha Evelyn Stone

1st Class Passengers

**Embarked:** Southampton on Wednesday 10th April 1912

Fonte: [encyclopedia-titanica](#)

**Name:** Mrs Martha Evelyn Stone

Titanic Survivor

**Born:** Wednesday 29th January 1851 in Charlestown, Massachusetts, United States ♂

**Age:** 61 years 2 months and 17 days (Female)

**Nationality:** American

**Marital Status:** Widowed

**Last Residence:** in New York City, New York, United States

1st Class Passengers

**Embarked:** Southampton on Wednesday 10th April 1912

Fonte: [encyclopedia-titanica](#)

E como subir no porto de “**Southampton**” está representado na coluna **Embarked** através da letra “**S**”, podemos preencher esse valores vazios dessa forma.

```
# Atualizando o valor para as 2 Linhas
titanic.loc[titanic.Embarked.isnull(),'Embarked'] = 'S'
```

## Módulo 21 – Tratando as informações de embarque vazias e usando a mediana para as idades

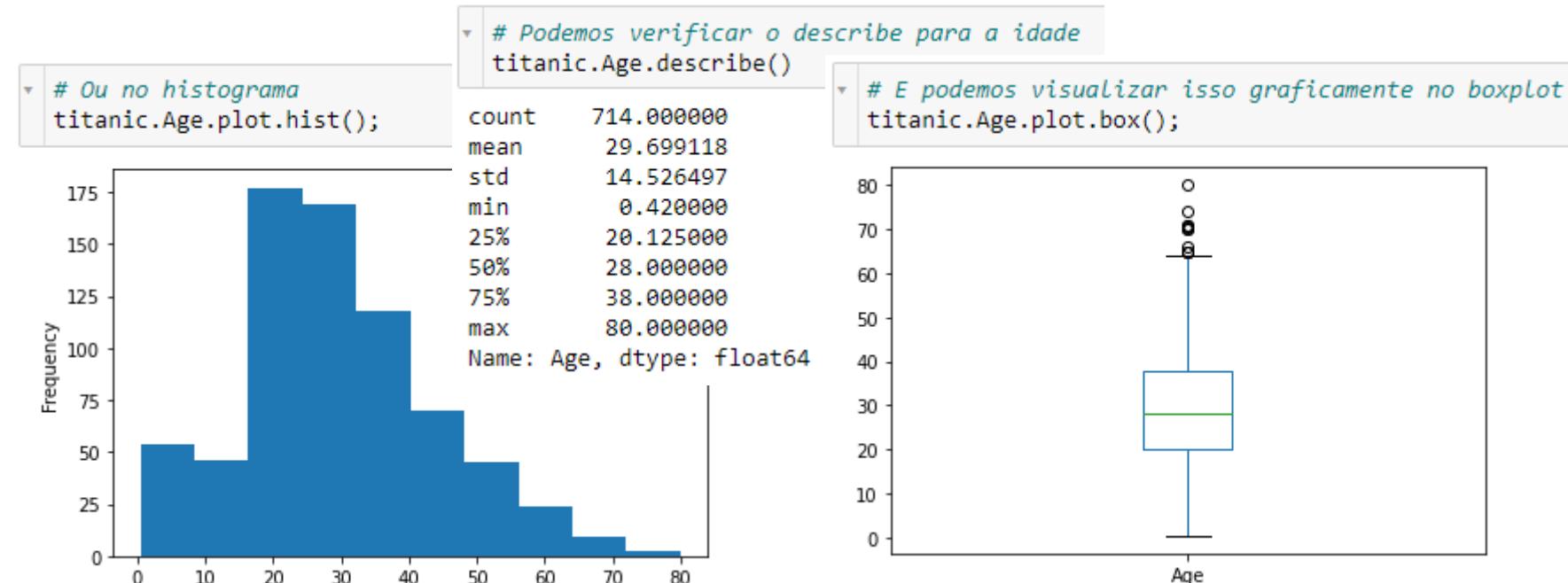
Em seguida, vamos tratar a coluna **Age**. Nessa coluna, temos **177 valores vazios**, e podemos iniciar com a visualização de alguns desses casos.

# Verificando valores nulos na idade titanic[titanic.Age.isnull()].head(3)													
PassengerId	Survived	Pclass	Name			Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
5	6	0	3	Moran, Mr. James		male	NaN	0	0	330877	8.4583	NaN	Q
17	18	1	2	Williams, Mr. Charles Eugene		male	NaN	0	0	244373	13.0000	NaN	S
19	20	1	3	Masselmani, Mrs. Fatima		female	NaN	0	0	2649	7.2250	NaN	C

Temos muito valores, então poderíamos sim pesquisar todos esses nomes manualmente, ou até criar uma automação para buscar essas idades. Porém, com intuito de sermos rápidos, simplesmente estimaremos esses valores de idade da melhor forma possível.

## Módulo 21 – Tratando as informações de embarque vazias e usando a mediana para as idades

E para começar a entender como podemos estimar esses valores, vamos visualizar a descrição estatística dessa coluna, além de visualizar graficamente com, por exemplo, **boxplot** e **histograma**.



E algo que percebemos é que a idade está, relativamente, bem distribuída. Então poderíamos pensar em preencher com, por exemplo, a média ou até a mediana.

```
# Uma opção é usar a mediana, ou a média das idades
titanic.Age.median()
```

28.0

## Módulo 21 – Analisando a média das idades pela classe, gênero e pelo título extraído do nome

Mais podemos ser mais específicos ainda, não apenas calculado a média ou mediana de forma geral, mas sim de forma agrupada de acordo com diferentes categorias. Inicialmente, podemos testar essa estratégia com a coluna **Pclass**.

```
# Ou podemos, por exemplo, tirar a média de idade por alguns grupos como o Pclass  
titanic.groupby('Pclass')['Age'].mean()
```

```
Pclass  
1    38.233441  
2    29.877630  
3    25.140620  
Name: Age, dtype: float64
```

Note que existe um diferença considerável, de forma que pessoas de classes mais caras, têm maior idade. Em seguida, podemos adicionar a coluna **Sex** nesse agrupamento.

```
# Podemos adicionar o gênero  
titanic.groupby(['Pclass','Sex'])['Age'].mean()
```

```
Pclass  Sex  
1      female   34.611765  
        male     41.281386  
2      female   28.722973  
        male     30.740707  
3      female   21.750000  
        male     26.507589  
Name: Age, dtype: float64
```

## Módulo 21 – Analisando a média das idades pela classe, gênero e pelo título extraído do nome

E novamente temos um diferença, em que as melhores parecem ter menor idade média. Claro que não podemos ir adicionando todas as colunas nesse agrupamento, pois poderíamos chegar em grupos muito pequenos que não teriam grande validade estatística, por isso, cabe a nós, cientista de dados, analisar qual é o melhor agrupamento.

Tendo isso em mente, um outro fator interessante que podemos adicionar é o título de cada pessoa. Mas como vamos extrair essa informação?

# E se considerarmos os títulos nos nomes?			
titanic.Name			
0		Braund, Mr.	Owen Harris
1	Cumings, Mrs.	John Bradley (Florence Briggs Th...	
2		Heikkinen, Miss.	Laina
3	Futrelle, Mrs.	Jacques Heath (Lily May Peel)	
4		Allen, Mr.	William Henry
		...	
886		Montvila, Rev.	Juozas
887		Graham, Miss.	Margaret Edith
888	Johnston, Miss.	Catherine Helen "Carrie"	
889		Behr, Mr.	Karl Howell
890		Dooley, Mr.	Patrick
Name: Name, Length: 891, dtype: object			

## Módulo 21 – Analisando a média das idades pela classe, gênero e pelo título extraído do nome

Note que o título está sempre delimitado entre os caracteres “,” e “.”. Com isso, podemos buscar esses caracteres em cada um dos valores da coluna.

```
# E se pegarmos a posição da vírgula?  
titanic.Name.str.find(',')
```

```
0      6  
1      7  
2      9  
3      8  
4      5  
..  
886    8  
887    6  
888    8  
889    4  
890    6  
Name: Name, Length: 891, dtype: int64
```

```
# E a posição do ponto?  
titanic.Name.str.find('.')
```

```
0      10  
1     12  
2      15  
3      13  
4       9  
..  
886    13  
887    12  
888    14  
889     8  
890    10  
Name: Name, Length: 891, dtype: int64
```

Para testar, podemos pegar o primeiro elemento dessa coluna dos índices **7 ao 12**. Porém, como não queremos pegar a “,” vamos adicionar 1, e como não queremos o espaço que vem depois da vírgula, somaremos mais 1.

```
# Podemos pegar o texto entre essas posições  
titanic.Name[1][7+2:12]
```

```
'Mrs'
```

## Módulo 21 – Analisando a média das idades pela classe, gênero e pelo título extraído do nome

Para simplificar, podemos criar e aplicar uma função, de forma que criaremos uma nova coluna chamada **Titulos**.

Porém, ao visualizar o **value\_counts()** dessa coluna, observamos vários títulos com baixa frequência de aparição.

E algo que podemos fazer é manter apenas os mais frequentes, e considerar os demais como um título “**Outros**”.

```
# Criando uma função para buscar o título em cada nome
def extrai_titulo(texto):
    return texto[texto.find(',')+2:texto.find('.')]
```

```
# Usando essa função na base
titanic['Titulos'] = titanic.Name.apply(extrai_titulo)
```

```
# Contando a quantidade de registros em cada título
titanic.Titulos.value_counts()
```

Mr	517
Miss	182
Mrs	125
Master	40
Dr	7

Rev	6
Mlle	2
Major	2
Col	2
the Countess	1

Capt	1
Ms	1
Sir	1
Lady	1
Mme	1

Don	1
Jonkheer	1

```
Name: Titulos, dtype: int64
```

## Módulo 21 – Analisando a média das idades pela classe, gênero e pelo título extraído do nome

Para executar essa ação, podemos alterar a nossa função, para que qualquer título **diferente de “Mr”, “Mrs”, “Miss” ou “Master”** seja considerado **“Outros”**.

```
# Criando uma função para buscar o título em cada nome
def extrai_titulo(texto):
    titulo = texto[texto.find(',')+2:texto.find('.')]
    
    if titulo in ['Mr', 'Mrs', 'Miss', 'Master']:
        return titulo
    else:
        return 'Outros'

# Usando essa função na base
titanic['Titulos'] = titanic.Name.apply(extrai_titulo)
```

E agora, ao visualizar o **value\_counts()** temos um resultado bem mais valido estatisticamente, já que cada grupo tem uma quantidade satisfatória de representantes.

```
# Contando a quantidade de registros em cada título
titanic.Titulos.value_counts()
```

Mr	517
Miss	182
Mrs	125
Master	40
Outros	27

Name: Titulos, dtype: int64

## Módulo 21 – Analisando a média das idades pela classe, gênero e pelo título extraído do nome

E agora podemos visualizar a média de idade para cada um dos grupos utilizando as colunas **Pclass**, **Sex** e **Titulos**.

# Utilizando o título no groupby			
titanic.groupby(['Pclass', 'Sex', 'Titulos'])['Age'].mean()			
Pclass	Sex	Titulos	
1	female	Miss	30.000000
		Mrs	40.882353
		Outros	33.666667
	male	Master	5.306667
		Mr	41.580460
		Outros	48.727273
2	female	Miss	22.390625
		Mrs	33.682927
		Outros	28.000000
	male	Master	2.258889
		Mr	32.768293
		Outros	42.000000
3	female	Miss	16.123188
		Mrs	33.515152
		Master	5.350833
	male	Mr	28.724891
		Name: Age, dtype: float64	

Inclusive, temos diferenças bem discrepantes entre alguns dos títulos, como em “**Master**”, que é usado para menores de idade.

Nosso objetivo será preencher os valores vazios com os valores de média apresentados anteriormente. E para isso o método **transform()** pode nos ajudar.

```
# Criando uma nova coluna das idades
titanic.groupby(['Pclass', 'Sex', 'Titulos'])['Age'].transform('mean')

0      28.724891
1      40.882353
2      16.123188
3      40.882353
4      28.724891
...
886    42.000000
887    30.000000
888    16.123188
889    41.580460
890    28.724891
Name: Age, Length: 891, dtype: float64
```

E podemos salvar essa transformação, para entender melhor o que está acontecendo.

```
# Criando uma nova coluna das idades
titanic['Age1'] = titanic.groupby(['Pclass', 'Sex', 'Titulos'])['Age'].transform('mean')
```

E agora visualizar como ficou a base.

# Visualizando a base	titanic.head(3)												
PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Titulos	Age1
0	1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Nan	S	Mr	28.724891
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C	Mrs	40.882353
2	3	1	Alvarez, Homeless Person	female	26.0	0	0	STON/O2. 3101282	7.9250	Nan	S	Miss	16.123188

Basicamente, estamos pegando a média para aquele grupo específico, por exemplo, a média de idade, dos passageiros de **Pclass=3, Sex=mais e Titulos=Mr** é de **28.72891**.

Note que, ainda não preenchemos os valores vazios, mas podemos usar essa mesma lógica com o método **fillna()**. Ou seja, ele só vai usar esse valor de média, se a coluna **Age** estiver vazia, caso ela tenha valor, esse valor vai ser repetido.

```
# Podemos diretamente utilizar esse valor no fillna  
titanic['Age2'] = titanic.Age.fillna(titanic.groupby(['Pclass', 'Sex', 'Titulos'])['Age'].transform('mean'))
```

E como criamos a coluna auxiliar **Age2**, podemos filtrar valores vazios da coluna **Age** e comparar com a nova coluna.

```
# Visualizando novamente a base  
titanic[titanic.Age.isnull()].head(3)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Titulos	Age1	Age2	
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q	Mr	28.724891	28.724891
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.0000	NaN	S	Mr	32.768293	32.768293
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.2250	NaN	C	Mrs	33.515152	33.515152

Agora que já validamos, podemos realizar essa operação diretamente na coluna Age.

```
# Podemos então usar esse valor na idade  
titanic['Age'] = titanic.Age.fillna(titanic.groupby(['Pclass','Sex','Titulos'])['Age'].transform('mean'))
```

Além disso, devemos eliminar as colunas **Age1** e **Age2**, que foram utilizadas apenas para validação.

```
# E apagar essas duas colunas  
titanic = titanic.drop(['Age1','Age2'],axis=1)
```

E agora, não temos mais valores vazios na coluna **Age**.

```
# Verificando novamente valores nulos
titanic.isnull().sum()

PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked        0
Titulos          0
dtype: int64
```

Porém, ainda continuamos com muitos valores vazios na coluna **Cabin**. Então, essa será a próxima coluna que vamos analisar.

Vamos visualizar o resultado do **value\_counts()**.

```
# Verificando as informações em cabine
titanic.Cabin.value_counts(dropna=False)
```

NaN	687
C23 C25 C27	4
G6	4
B96 B98	4
C22 C26	3
...	
E34	1
C7	1
C54	1
E36	1
C148	1

Name: Cabin, Length: 148, dtype: int64

Note que, além de ter mais de **70% dos valores vazios**, a coluna está com vários valores distintos e de difícil compreensão do significado. Então, para simplificar, podemos remover essa coluna.

```
# Eliminando essa coluna
titanic = titanic.drop('Cabin',axis=1)
```

## Módulo 21 – Analisando outliers, cardinalidade e eliminando colunas desnecessárias

Agora que não temos mais valores vazios na nossa base, vamos partir para o tratamento de outliers.

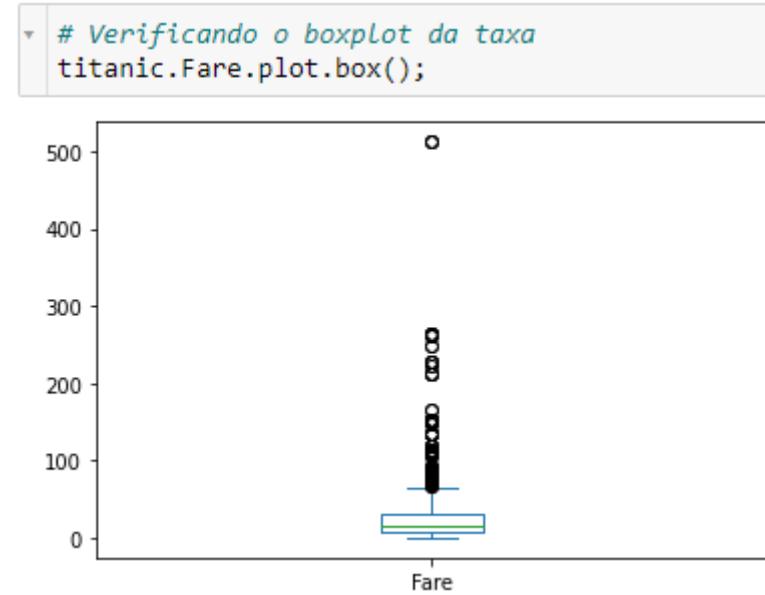
Vamos verificar novamente as informações estatísticas da base.

# Utilizando o describe titanic.describe()							
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.430535	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	13.551396	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	21.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.724891	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	36.750000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Será que realmente existiu essa cobrança de **512** na coluna **Fare**, ou é algum erro?

## Módulo 21 – Analisando outliers, cardinalidade e eliminando colunas desnecessárias

Além disso, podemos visualizar o **boxplot**. Que apresentará realmente alguns pontos acima do normal, e o **512** já bem acima.



E o que podemos fazer nesse caso é recorrer ao negócio, pesquisar se realmente esses valores de tarifa são pagos por passageiros.

## Módulo 21 – Analisando outliers, cardinalidade e eliminando colunas desnecessárias

Para descobrir quais são esses passageiros, podemos filtrar as pessoas que pagaram o equivalente ao máximo da coluna **Fare**.

# Filtando esse valor

```
titanic[titanic.Fare == titanic.Fare.max()]
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Titulos
258	259	1	Ward, Miss. Anna	female	35.0	0	0	PC 17755	512.3292	C	Miss
679	680	1	Cardeza, Mr. Thomas Drake Martinez	male	36.0	0	1	PC 17755	512.3292	C	Mr
737	738	1	Lesurer, Mr. Gustave J	male	35.0	0	0	PC 17755	512.3292	C	Mr

Ao pesquisar no google, vamos encontrar que essas pessoas realmente pagaram esses valores, porque eram cabines especiais.

Então, sabendo que esses valores fazem sentido para a nossa base, não é necessário fazer nenhum tipo de tratamento nesse momento. Porém, é necessário ficarmos cientes que esses outliers existem, pois em passos futuros, podemos tratar essa coluna de forma diferente, talvez agrupando por faixas (**barato, médio, caro**), ou algo parecido.

## Módulo 21 – Analisando outliers, cardinalidade e eliminando colunas desnecessárias

Com isso, vamos visualizar novamente a base, em que parece estar tudo correto.

```
# Base
titanic.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Titulos
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	Mr
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C	Mrs Miss
2	3	1	3		female	26.0	0	0	STON/O2. 3101282	7.9250	S	

Além, do método info(), em que não visualizamos mais valores vazios, e os tipos parecem todos fazer sentido.

```
# Base
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId 891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare          891 non-null    float64
 10  Embarked     891 non-null    object 
 11  Titulos      891 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

## Módulo 21 – Analisando outliers, cardinalidade e eliminando colunas desnecessárias

Outro ponto interessante, é voltar a cardinalidade dos dados.

```
# Voltando na cardinalidade
titanic.nunique()
```

PassengerId	891
Survived	2
Pclass	3
Name	891
Sex	2
Age	97
SibSp	7
Parch	7
Ticket	681
Fare	248
Embarked	3
Titulos	5
dtype:	int64

Em que, ainda temos colunas com **cardinalidade muito alta**, com basicamente, valores únicos para cada pessoa, e que no contexto de **machine learning**, não nos ajudam a generalizar. No caso, a coluna **Name**, até tinha informação, mas já extraímos essa informação de título, então podemos remove-la.

```
# Retirando a coluna Name
titanic = titanic.drop('Name',axis=1)
```

## Módulo 21 – Analisando outliers, cardinalidade e eliminando colunas desnecessárias

Já as colunas **PassengerId** e **Ticket**, não ajudam em nada na identificação do passageiro que irá ou não sobreviver, então também podemos remove-las.

```
# Retirando as colunas PassengerId e Ticket  
titanic = titanic.drop(['PassengerId', 'Ticket'], axis=1)
```

Após essa operação, temos a base final com todos os tratamentos realizados. E lembrando que aproveitaremos esse aprendizado de limpeza de dados nos módulos seguintes.

```
# Visualizando nossa base final  
titanic.head(3)
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Titulos
0	0	3	male	22.0	1	0	7.2500	S	Mr
1	1	1	female	38.0	1	0	71.2833	C	Mrs
2	1	3	female	26.0	0	0	7.9250	S	Miss

```
# Visualizando as informações  
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 9 columns):  
 #   Column      Non-Null Count  Dtype     
---    
 0   Survived    891 non-null   int64    
 1   Pclass      891 non-null   int64    
 2   Sex         891 non-null   object    
 3   Age         891 non-null   float64  
 4   SibSp       891 non-null   int64    
 5   Parch       891 non-null   int64    
 6   Fare        891 non-null   float64  
 7   Embarked    891 non-null   object    
 8   Titulos     891 non-null   object    
dtypes: float64(2), int64(4), object(3)  
memory usage: 62.8+ KB
```