

Distribuição de Frequência – Parte 02

**Disciplina: Estatística Aplicada a Engenharia de
Software/Computação**

Prof. Me. Max Gabriel Steiner

Distribuição de Frequência - Python

- Por fim, podemos fazer mais um teste:
- Antes disso, vale a pena conferir estes dois links com documentações da biblioteca numpy para visualizarmos outras aplicações matemáticas possíveis de utilizar:

<https://numpy.org/doc/stable/reference/generated/numpy.histogram.html>

https://numpy.org/doc/stable/reference/generated/numpy.histogram_bin_edges.html#numpy.histogram_bin_edges

Voltando ao Python:

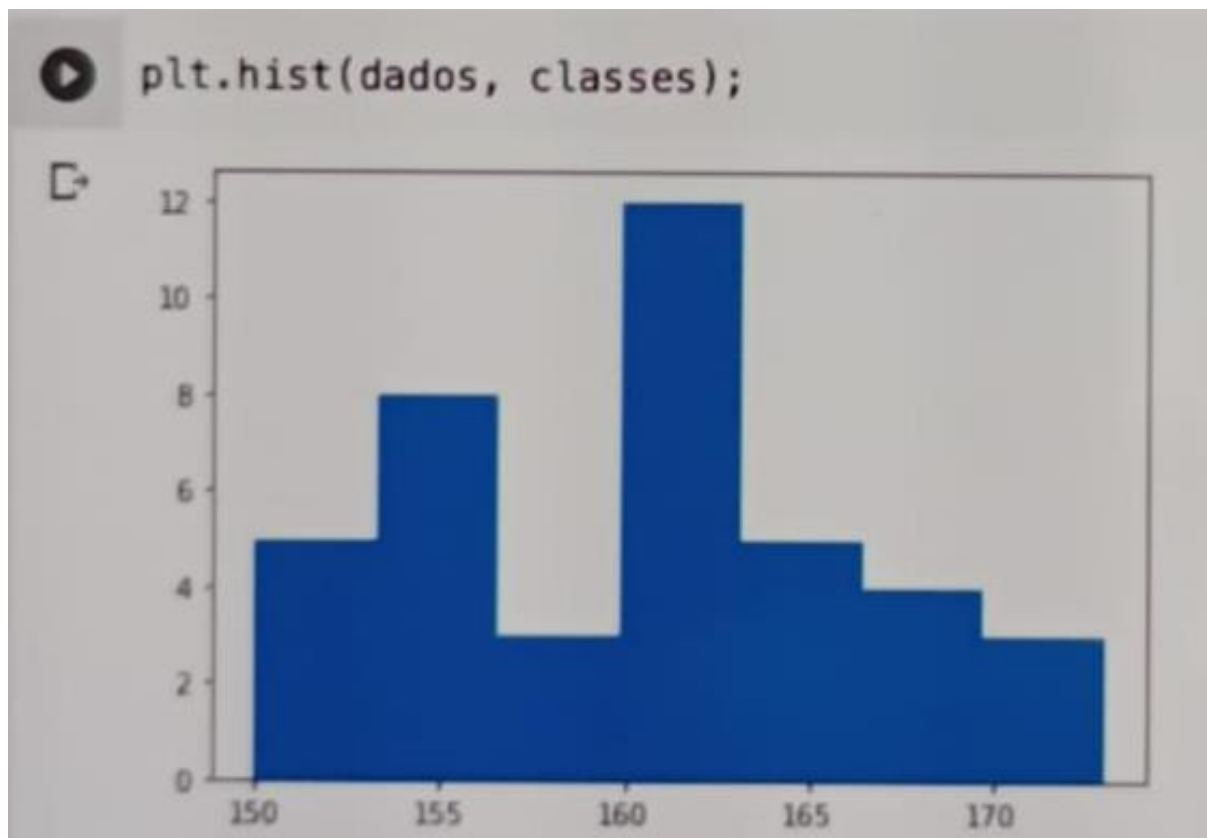
frequencia, classes = np.histogram(dados, bins='sturges') #neste caso quem vai definir o número de divisões é a fórmula de sturges que vimos anteriormente.

print(frequencia, classes)

```
➤ frequencia, classes = np.histogram(dados, bins = 'sturges')  
frequencia, classes  
  
[ (array([ 5,  8,  3, 12,  5,  4,  3]),  
  array([150.          , 153.28571429, 156.57142857, 159.85714286,  
         163.14285714, 166.42857143, 169.71428571, 173.          ]))
```

Distribuição de Frequência - Python

- Podemos perceber que neste caso, a fórmula de sturges fez o arredondamento para cima, portanto definiu 7 classes, diferente de nossos cálculos anteriores, pois arredondamos para baixo.
- Plotando o histograma:



Distribuição de Frequência - Python

- **Podemos agora gerar a distribuição de frequência e o histograma utilizando o pandas e a biblioteca de visualização seaborn.**
- Esse tipo de codificação também é muito necessário, pois, nem sempre teremos os dados no formato do numpy, muitas vezes estaremos utilizando data frames, principalmente no caso da leitura de arquivos csv, por isso é muito importante saber utilizar também a biblioteca pandas.
- Vamos ao Python:
- Podemos perceber que os dados da variável dados estão no tipo/formato numpy

```
type(dados)
```

```
numpy.ndarray
```

Distribuição de Frequência - Python

- Vamos agora criar então um DataFrame chamado de dataset, utilizando o formato de dicionário para criação, definimos portanto o nome da coluna por 'dados' e fazemos a leitura de nossa variável dados que contém nossa tabela primitiva :

```
[17] dataset = pd.DataFrame({'dados': dados})
```

- Se digitarmos:

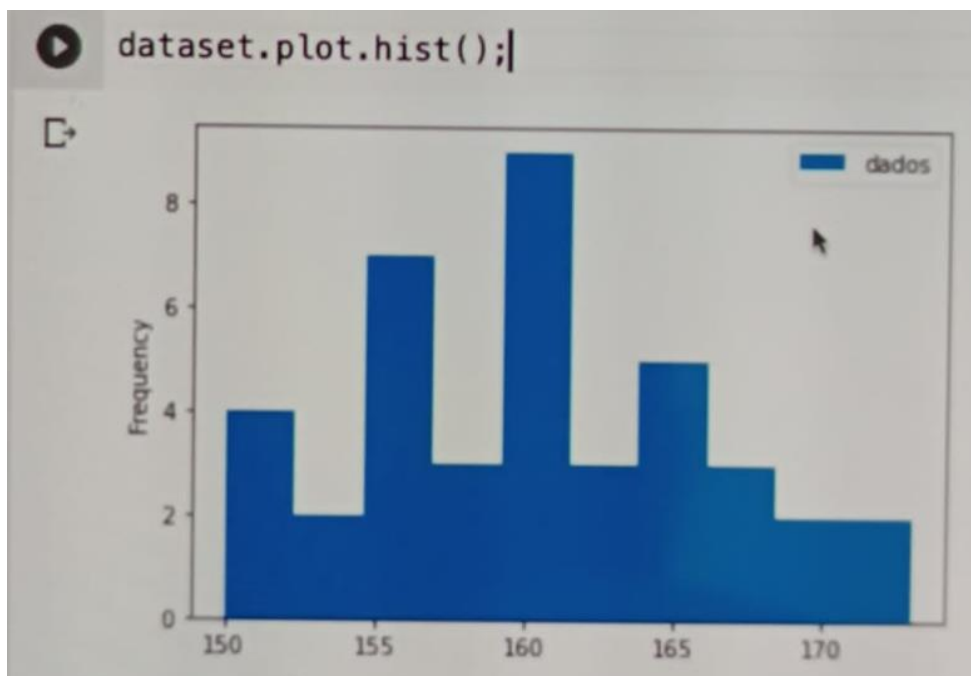
```
dataset.head()
```

- Teremos então a visualização do registro das 5 primeiras pessoas, e vejamos que estes dados não estão ordenados e não há problema por não estarem ordenados.

	dados
0	160
1	165
2	167
3	164
4	160

Distribuição de Frequência - Python

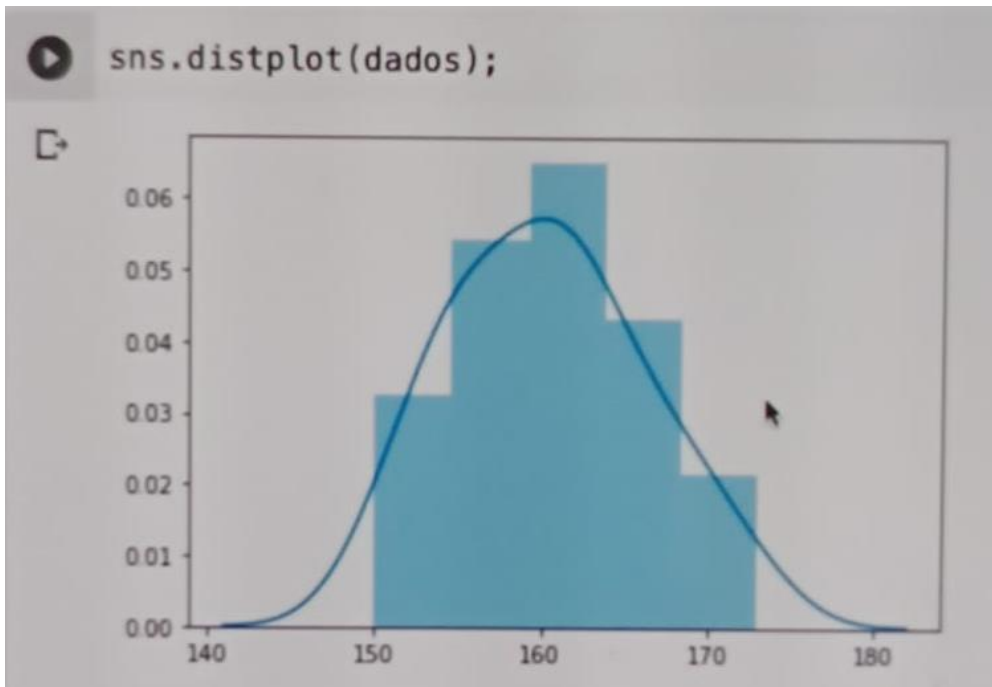
- Para gerar um histograma é muito simples:



- Aqui nos deparamos novamente com o bins para as divisões, por padrão ele gera 10 divisões, porém, podemos digitar bins=5 dentro dos parênteses para mudar para 5 etc.

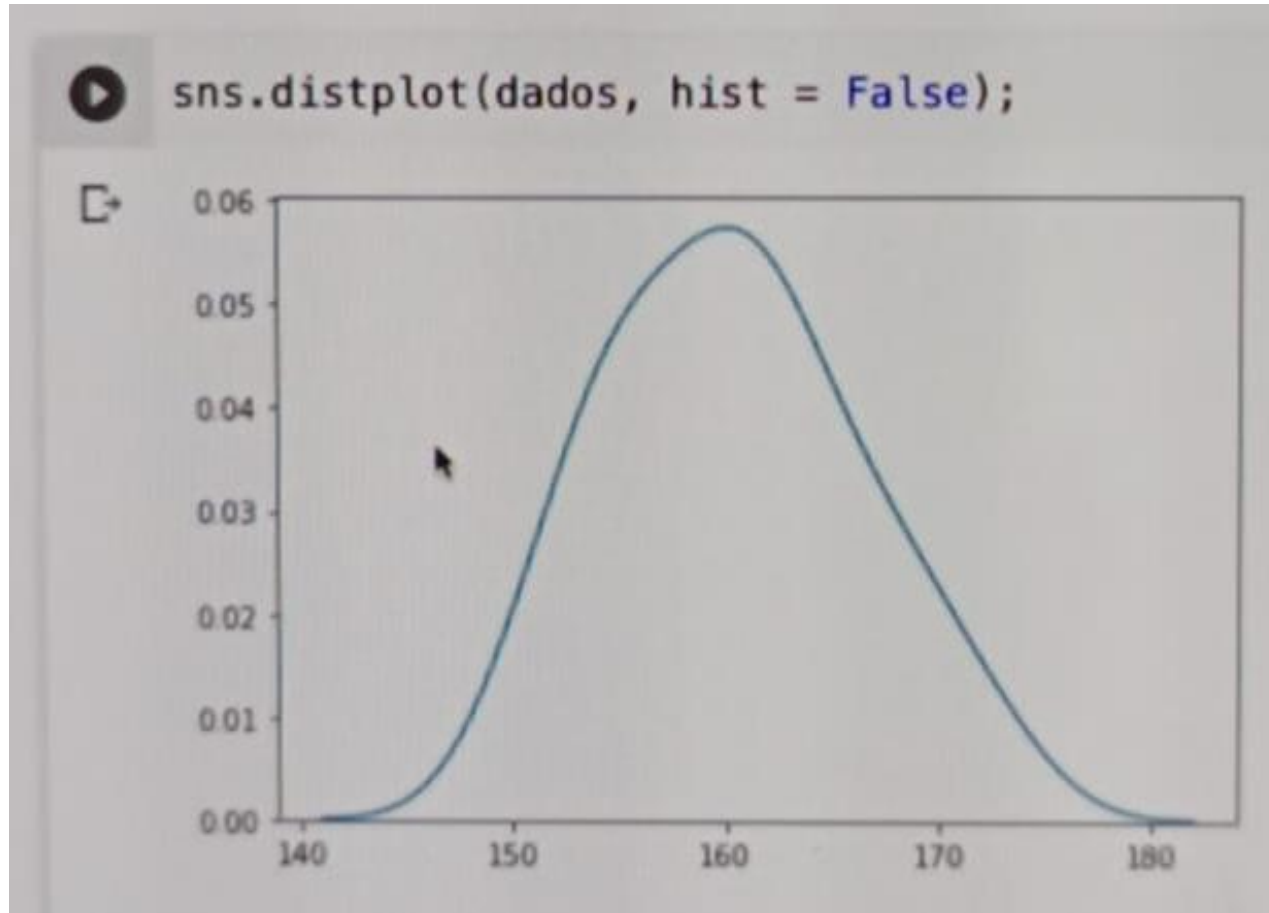
Distribuição de Frequência - Python

- Este mesmo gráfico pode ser gerado através da utilização do Pandas, e podemos fazer gráficos mais interessantes com o seaborn.
- Por exemplo:



Distribuição de Frequência - Python

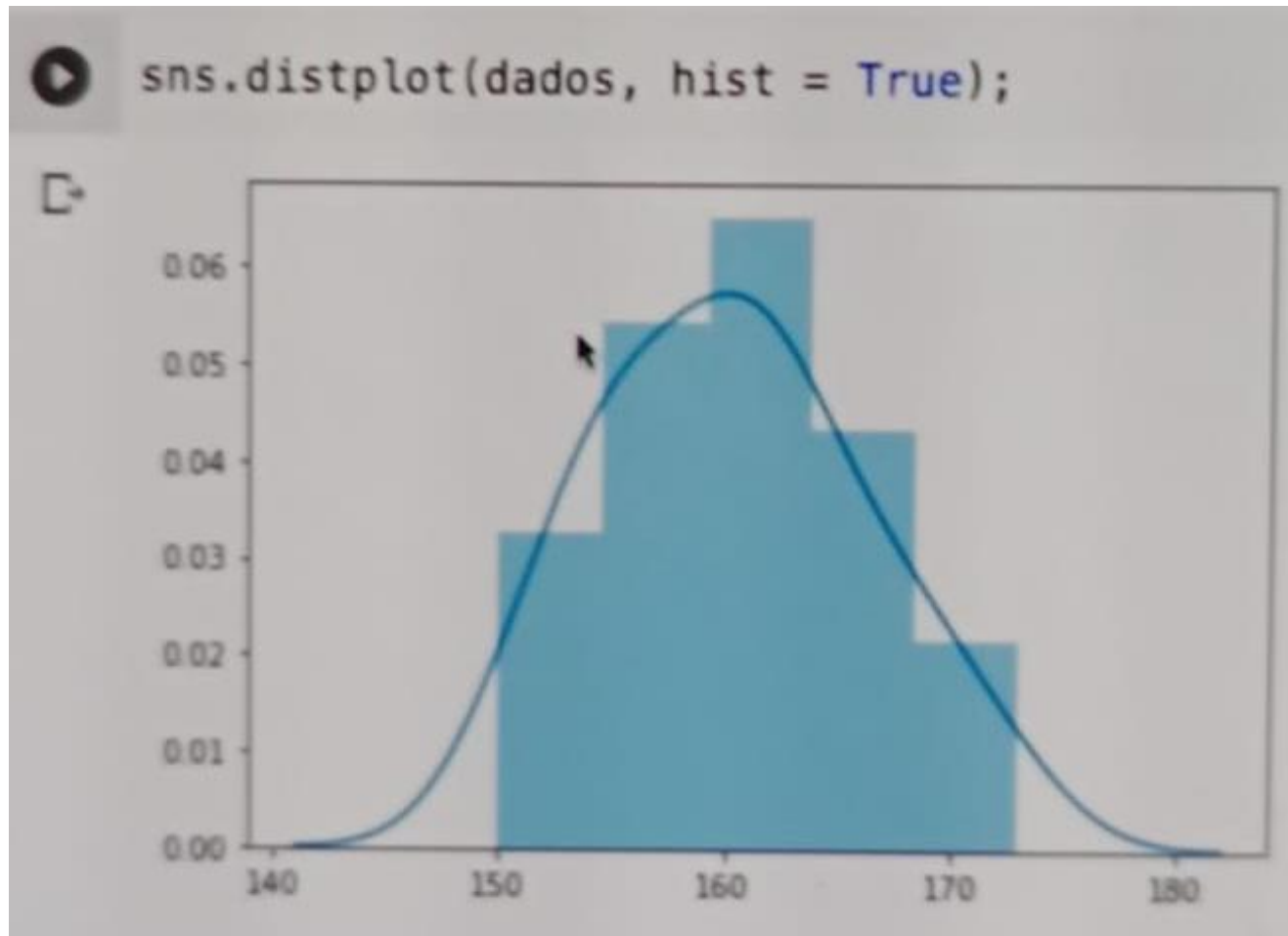
➤ Podemos configurar outras modificações, como por exemplo:



➤ Neste caso, ele não vai gerar o histograma.

Distribuição de Frequência - Python

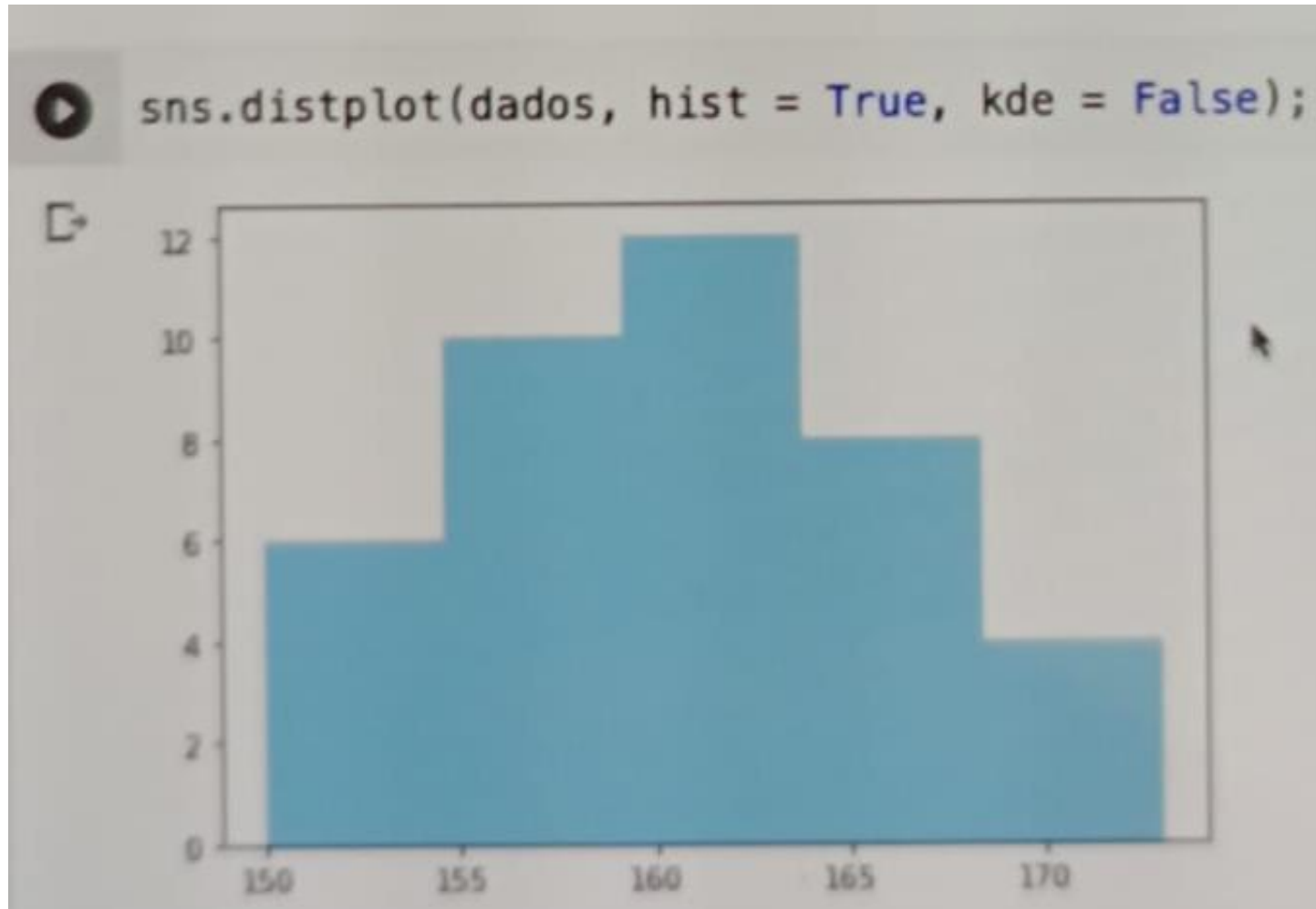
- Podemos configurar outras modificações, como por exemplo:



- Neste caso, ele vai gerar o histograma junto.

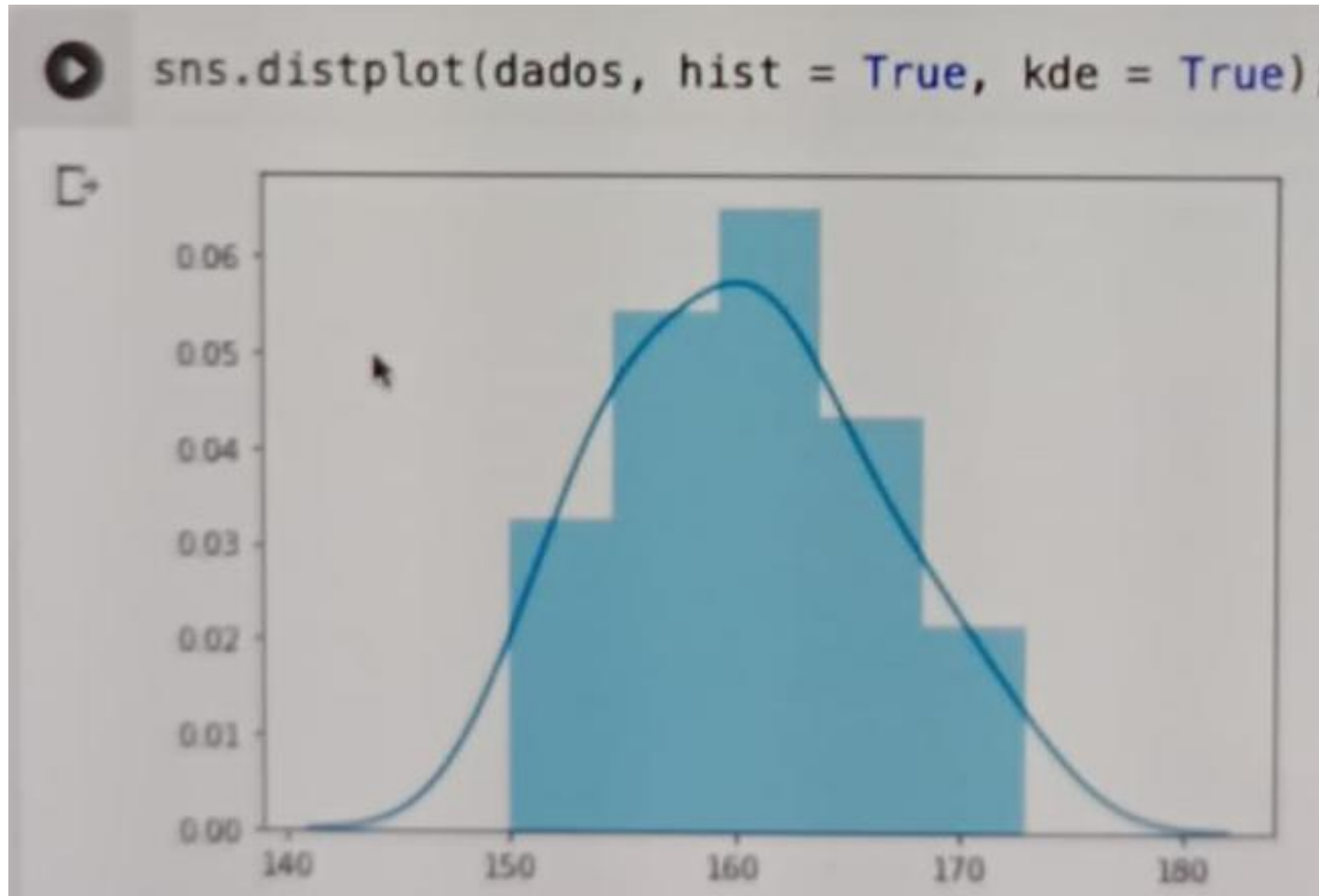
Distribuição de Frequência - Python

- Caso queiramos mostrar o histograma sem a linha, usamos o parâmetro `kde`, aqui ele vai fazer a plotagem da linha de densidade sem a linha:



Distribuição de Frequência - Python

➤ E neste caso, mudando para True ele mostra com a linha:



➤ Todos esses últimos gráficos conseguimos utilizar o bin para selecionar o número de divisões.

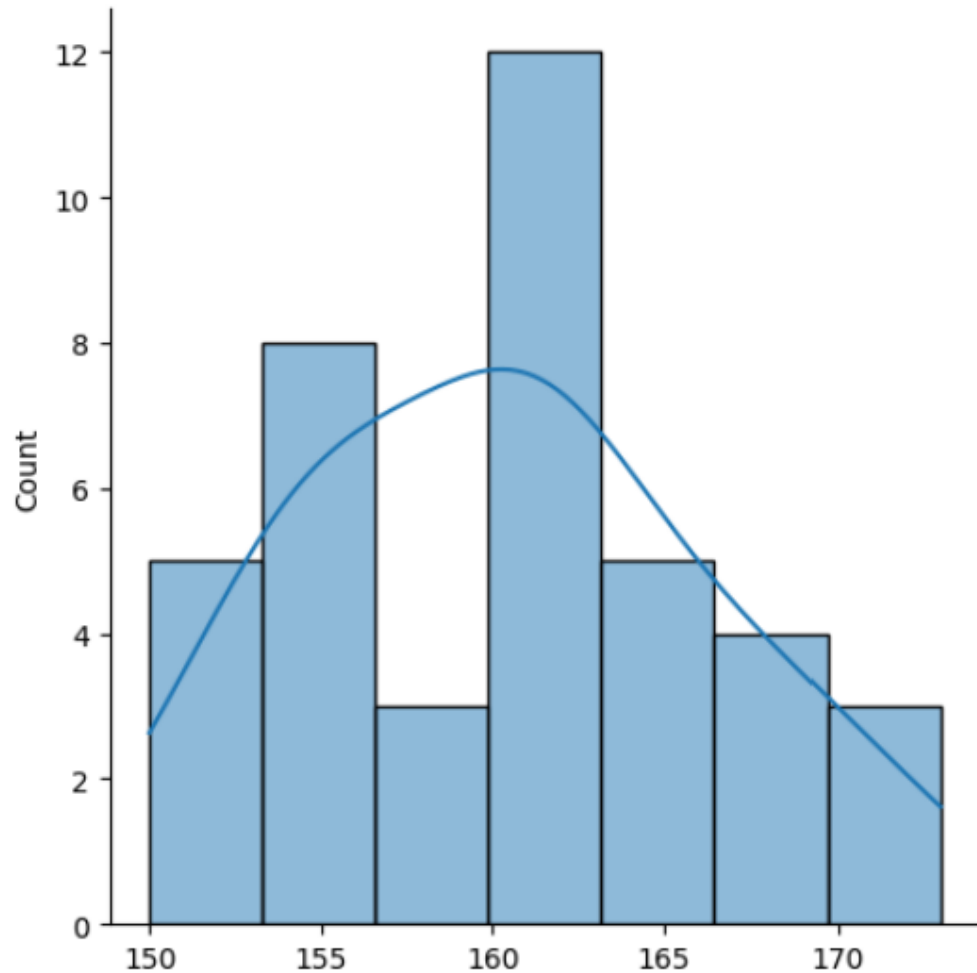
Distribuição de Frequência - Python



```
#Podemos utilizar quando ficar obsoleto  
sns.displot(dados, kde=True)
```

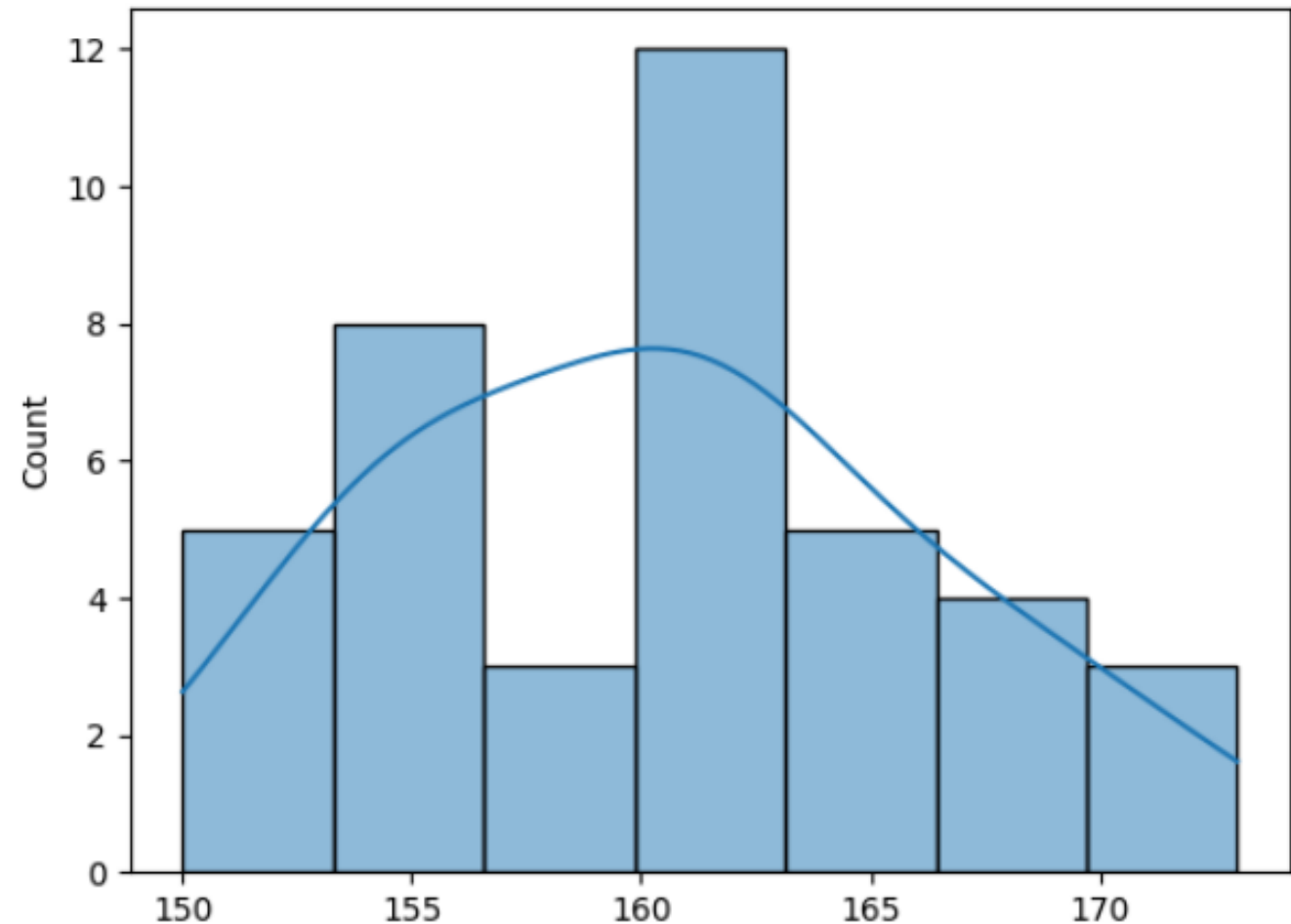


```
<seaborn.axisgrid.FacetGrid at 0x7aa2a4c27280>
```



```
#Caso seja necessário um controle mais fino dos dados  
sns.histplot(dados, kde=True)
```

```
<Axes: ylabel='Count'>
```



Distribuição de Frequência – Python - EXERCÍCIO

- O objetivo desta tarefa é aplicar a distribuição de frequência utilizando o atributo “**age**” da base de dados do censo.
- Carregue a base de dados census.csv
- Faça testes utilizando o parâmetro **bins** para visualizar a distribuição dos dados.

Distribuição de Frequência – Python - EXERCÍCIO

➤ Resolução:

- Primeiramente precisamos fazer o carregamento da base de dados através do arquivo csv.

```
import pandas as pd  
dataset = pd.read_csv('census.csv')
```

```
dataset.head()
```

	age	workclass	final-weight	education
0	39	State-gov	77516	Bachelors
1	50	Self-emp-not-inc	83311	Bachelors
2	38	Private	215646	HS-grad

Distribuição de Frequência – Python - EXERCÍCIO

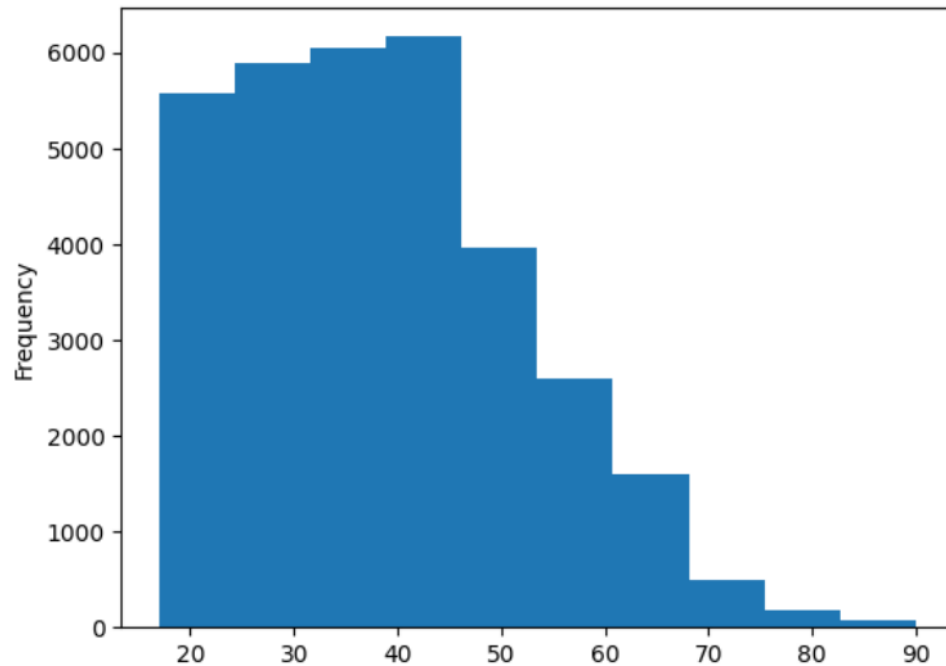
- Com o comando a seguir podemos identificar qual é o maior e o menor valor do conjunto:

```
dataset['age'].max(), dataset['age'].min()
```

(90, 17)

- Utilizando recursos do próprio pandas podemos plotar o histograma em **age**.

```
dataset['age'].plot.hist();
```



Distribuição de Frequência – Python - EXERCÍCIO

- Algo legal que podemos fazer é definir faixas personalizadas, utilizando o recurso **cut** do pandas, utilizado para segmentar dados.
- No caso do código abaixo vamos definir intervalos personalizados, de 0 até 17, de 17 até 25, de 25 até 40, de 40 até 60, de 60 até 90 através da definição em bins.
- Vamos utilizar dos labels para nomear essas faixas.

```
dataset['age'] = pd.cut(dataset['age'], bins=[0, 17, 25, 40, 60, 90],  
                        labels=['Faixa1', "Faixa2", "Faixa3", "Faixa4", "Faixa5"])
```

- O **bins** especifica os intervalos de divisão

Distribuição de Frequência – Python - EXERCÍCIO

- Ao visualizar os registros, podemos ver que ele transformou a idade em faixas de valores de acordo com o critério acima.

```
dataset.head()
```

	age	workclass	final-weight	education	education-num	marital-status	occupation	re:
0	Faixa3	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	↑
1	Faixa4	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	
2	Faixa3	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	↑
3	Faixa4	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	
4	Faixa3	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	

- Podemos ver que não temos mais os dados numéricos, somente as faixas. Dessa maneira podemos fazer **n** transformações em dados utilizando o pandas.

```
dataset['age'].unique()
```

```
['Faixa3', 'Faixa4', 'Faixa2', 'Faixa5', 'Faixa1']
```

```
Categories (5, object): ['Faixa1' < 'Faixa2' < 'Faixa3' < 'Faixa4' < 'Faixa5']
```

Geração de Regras de Associação

Geração de Regras de Associação

➤ **Prateleiras de Mercado:**

- Em que prateleira o biscoito de chocolate deve ser colocado para maximizar suas vendas?
- Suco de uva costuma ser comprado com refrigerante?
- Qual produto pode ser colocado em promoção para uma venda casada com tomates?
- Com a aplicação de regras de associação podemos responder esse tipo de perguntas.
- Podemos fazer promoções com itens que são vendidos em conjunto.
- Planejar catálogos das lojas e folhetos de promoções, como exemplo dos folhetos de supermercado que sempre possuem produtos aleatórios nesses folhetos, podemos otimizar esses catálogos para colocar nos folhetos produtos que costumam ser vendidos em conjunto.
- Podemos também aplicar em universidades para gerar controle de evasão nas universidades, criando algum tipo de regra para identificar previamente o perfil de alunos que possuem tendência maior em evadir, etc.

Geração de Regras de Associação – Algoritmo Apriori

- **Algoritmo Apriori:** veremos agora a base da teoria do algoritmo apriori que é o principal algoritmo utilizado para geração de regras de associação.
- Nessa tabela clássica, cada linha representa uma transação em um mercado e vai indicar se determinados produtos foram ou não foram comprados.
- Por exemplo, citar itens que foram comprados ou não na linha 1 e 2.
- O objetivo desse algoritmo é o de fazer uma análise nesses dados, nessas transações, para fazer a geração de regras de associação.
- Por exemplo, se compra pão e leite, então compra manteiga, se compra arroz então compra feijão.

Nº	Leite	Café	Cerveja	Pão	Manteiga	Arroz	Feijão
1	Não	Sim	Não	Sim	Sim	Não	Não
2	Sim	Não	Sim	Sim	Sim	Não	Não
3	Não	Sim	Não	Sim	Sim	Não	Não
4	Sim	Sim	Não	Sim	Sim	Não	Não
5	Não	Não	Sim	Não	Não	Não	Não
6	Não	Não	Não	Não	Sim	Não	Não
7	Não	Não	Não	Sim	Não	Não	Não
8	Não	Não	Não	Não	Não	Não	Sim
9	Não	Não	Não	Não	Não	Sim	Sim
10	Não	Não	Não	Não	Não	Sim	Não

- O objetivo principal é gerar regras do tipo “se, então...”

Geração de Regras de Associação – Algoritmo Apriori

- Para cumprir com os objetivos, podemos utilizar de dois cálculos básicos:

- O primeiro:

Suporte = Número de registros com X e Y / Número total de registros

- Notem que em nossa base de dados temos somente 10 registros, ou seja, somente 10 compras que foram realizadas.
- Sempre que vamos gerar regras de associação precisamos definir “o suporte mínimo”, ou seja, qual a quantidade mínima de vezes que um produto vai aparecer na base de dados para gerarmos uma regra.
- Por exemplo, se considerarmos:
- Suporte $\geq 0,3$

Nº	Leite	Café	Cerveja	Pão	Manteiga	Arroz	Feijão
1	Não	Sim	Não	Sim	Sim	Não	Não
2	Sim	Não	Sim	Sim	Sim	Não	Não
3	Não	Sim	Não	Sim	Sim	Não	Não
4	Sim	Sim	Não	Sim	Sim	Não	Não
5	Não	Não	Sim	Não	Não	Não	Não
6	Não	Não	Não	Não	Sim	Não	Não
7	Não	Não	Não	Sim	Não	Não	Não
8	Não	Não	Não	Não	Não	Não	Sim
9	Não	Não	Não	Não	Não	Sim	Sim
10	Não	Não	Não	Não	Não	Sim	Não

- Como temos somente 10 registros, o 0,3 significa que nós queremos os produtos que aparecem em pelo menos 30% das transações.

Geração de Regras de Associação – Algoritmo Apriori

- Se observarmos o café ele aparece 3 vezes, ou seja, o suporte do café é 30%.
- O pão, ele aparece 5 vezes, ou seja, em 50% das transações é vendido pão.
- Com relação a manteiga, aparece 5 vezes, 50% também.
- Já o leite aparece 2 vezes somente, ou seja, não satisfaz a condição de ser maior ou igual a 30% de suporte.
- A cerveja apenas 20%. O arroz apenas 20%. O feijão apenas 20%. Ou seja:
- Vamos gerar regras considerando somente o café, pão e a manteiga, pois apenas estes satisfazem os 30% de suporte.

Nº	Leite	Café	Cerveja	Pão	Manteiga	Arroz	Feijão
1	Não	Sim	Não	Sim	Sim	Não	Não
2	Sim	Não	Sim	Sim	Sim	Não	Não
3	Não	Sim	Não	Sim	Sim	Não	Não
4	Sim	Sim	Não	Sim	Sim	Não	Não
5	Não	Não	Sim	Não	Não	Não	Não
6	Não	Não	Não	Não	Sim	Não	Não
7	Não	Não	Não	Sim	Não	Não	Não
8	Não	Não	Não	Não	Não	Não	Sim
9	Não	Não	Não	Não	Não	Sim	Sim
10	Não	Não	Não	Não	Não	Sim	Não

Geração de Regras de Associação – Algoritmo Apriori

- O segundo conceito de cálculo:

**Confiança = Número de registros com X e Y /
Número total de registros com X**

- Aqui você pode observar que nas duas regras temos o X e Y que você pode calcular o suporte do café e do pão sendo vendidos em conjunto.
- Por exemplo, podemos calcular o suporte do: café e do pão vendidos em conjuntos.
- Observamos na tabela que em 3 situações encontramos café e pão sendo vendidos em conjunto, portanto o suporte desses dois produtos é de 0,3, ou seja, 30%.

Nº	Leite	Café	Cerveja	Pão	Manteiga	Arroz	Feijão
1	Não	Sim	Não	Sim	Sim	Não	Não
2	Sim	Não	Sim	Sim	Sim	Não	Não
3	Não	Sim	Não	Sim	Sim	Não	Não
4	Sim	Sim	Não	Sim	Sim	Não	Não
5	Não	Não	Sim	Não	Não	Não	Não
6	Não	Não	Não	Não	Sim	Não	Não
7	Não	Não	Não	Sim	Não	Não	Não
8	Não	Não	Não	Não	Não	Não	Sim
9	Não	Não	Não	Não	Não	Sim	Sim
10	Não	Não	Não	Não	Não	Sim	Não

Geração de Regras de Associação – Algoritmo Apriori

- Precisamos definir também um valor de confiança mínima, por exemplo:
- Vamos supor que queremos extrair regras com confiança mínima de 80%:
- Confiança $\geq 0,8$
- Se nós considerarmos {café, pão}, ou seja, se considerarmos somente café e pão, podemos gerar essas duas regras:

SE café ENTÃO pão – confiança = $3 / 3 = 1,0$
SE pão ENTÃO café – confiança = $3 / 5 = 0,6$

- Parece a mesma regras, mas não é!
- Essa 1ª parte da regra, nº de registros com X e Y, precisamos verificar quantas vezes aparece tanto o café quanto o pão, podemos observar que são 3 vezes.

- O nº total de registros X é a 1ª parte que está na condição:

SE café ENTÃO pão – confiança = $3 / 3 = 1,0$

- Portanto, “café”, aparece quantas vezes? Vide na tabela são 3 vezes.
- Portanto, na fórmula:

Confiança = $\frac{\text{Número de registros com X e Y}}{\text{Número total de registros com X}}$

- Temos: Confiança = $3/3 = 1,0$
- Isso indica que em 100% das transações quando você compra café, então você também compra pão.

Geração de Regras de Associação – Algoritmo Apriori

- No caso da 2ª parte da regra:

SE pão ENTÃO café – confiança = $3 / 5 = 0,6$

- O nº de registros com X e Y, é o mesmo, portanto segue sendo 3 vezes.
- O nº total de registros X é a 1ª parte que está na condição, portanto agora trata-se do “pão”.
- Portanto, “pão”, aparece quantas vezes? Vide na tabela são 5 vezes.
- Portanto, na fórmula:

Confiança = $\frac{\text{Número de registros com X e Y}}{\text{Número total de registros com X}}$

- Temos: Confiança = $3/5 = 0,6$

- Isso indica que em 60% das transações, quando você compra pão, então você também compra café.

Geração de Regras de Associação – Algoritmo Apriori

➤ **Portanto:**

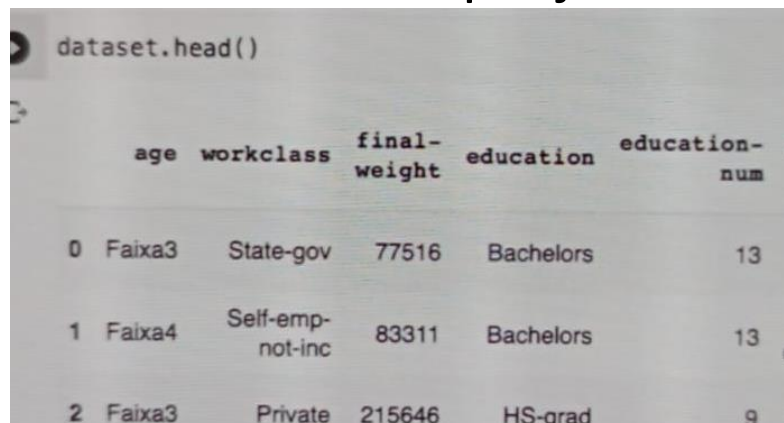
- Podemos compreender que o objetivo do Algoritmo Apriori é analisar toda a base de dados e por meio dos parâmetros: suporte e confiança, gerar uma sequência de regras e essas regras devem ser analisadas pelo cientista de dados.
- Ele deverá identificar se essas regras fazem sentido para a empresa, se faz sentido fazer um remanejamento das prateleiras por exemplo.

Geração de Regras de Associação – Algoritmo Apriori

- **Nessa parte do conteúdo:**
- Vamos trabalhar com a aplicação de regras de associação na base de dados do Census.
- A ideia é você compreender como a distribuição de frequência, a divisão em intervalos de classes, pode ser útil, quando nós trabalhamos com algoritmos de machine learning.

Geração de Regras de Associação – Algoritmo Apriori

- Vamos criar uma nova célula com os dados que já temos em dataset:



```
dataset.head()
```

	age	workclass	final-weight	education	education-num
0	Faixa3	State-gov	77516	Bachelors	13
1	Faixa4	Self-emp-not-inc	83311	Bachelors	13
2	Faixa3	Private	215646	HS-grad	9

- Anteriormente fizemos a alteração de valores numéricos para faixas em **age**.
- É importante fazer essa transformação, pois quando vamos trabalhar com regras de associação, no sentido “se compra pão, então compra café, manteiga” e assim por diante, **não podemos utilizar dados numéricos! Os dados aqui precisam estar como strings!!!**
- Se tivermos por exemplo um dado numérico como por exemplo em **education-num**, na hora de passar para o algoritmo de associação ele vai gerar um erro.

Geração de Regras de Associação – Algoritmo Apriori

- Vamos agora criar um outro dataset chamado de `dataset_apriori` para receber os atributos categóricos: `age`, `workclass`, `education`, `marital-status`, `occupation`, `relationship`, `sex`, `native-country`, `income` (lembrando que aqui selecionamos apenas aqueles que não são numéricos).

```
dataset_apriori = dataset[['age', 'workclass', 'education', 'marital-status', 'relationship', 'occupation',  
                           'sex', 'native-country', 'income']]
```

```
dataset_apriori.head()
```

	age	workclass	education	marital-status	relationship	occupation	sex	native-country	income
0	Faixa3	State-gov	Bachelors	Never-married	Not-in-family	Adm-clerical	Male	United-States	<=50K
1	Faixa4	Self-emp-not-inc	Bachelors	Married-civ-spouse	Husband	Exec-managerial	Male	United-States	<=50K
2	Faixa3	Private	HS-grad	Divorced	Not-in-family	Handlers-cleaners	Male	United-States	<=50K
3	Faixa4	Private	11th	Married-civ-spouse	Husband	Handlers-cleaners	Male	United-States	<=50K
4	Faixa3	Private	Bachelors	Married-civ-spouse	Wife	Prof-specialty	Female	Cuba	<=50K

Geração de Regras de Associação – Algoritmo Apriori

- Se nós observarmos o tamanho deste dataset:

```
dataset.shape
```

```
(32561, 15)
```

- São 32561 elementos, ou seja, se a gente rodar as regras dentro disso tudo vai demorar bastante, portanto, vamos separar uma amostra somente:
- Vamos então, recriar o dataset_apriori utilizando da função sample (que faz uma amostragem simples) e verificar que agora sim, reduzimos para 1000.

```
dataset_apriori = dataset_apriori.sample(n = 1000)  
dataset_apriori.shape
```

```
(1000, 9)
```

Geração de Regras de Associação – Algoritmo Apriori

- Lembrando que o que acabamos de fazer é muito comum!
- Na prática, na maioria das vezes selecionamos apenas uma amostra, uma parte dos dados para trabalhar neles, visando poupar tempo de execução.
- Vamos utilizar a biblioteca: apyori, ela que é uma das bibliotecas mais utilizadas para associação no python.
- Como essa biblioteca python requer que os dados estejam em formato de lista invés de formato de dataframe, precisamos então, transformar esses dados.

Geração de Regras de Associação – Algoritmo Apriori

- Vamos então criar a variável **transacoes** para receber esses dados.

```
transacoes = []
```

- Vamos fazer um for i in range(dataset_apriori.shape[0]): onde a posição zero indica os 1000 registros que contemos em dataset_apriori, e ele vai executar esse for que criamos 1000 vezes.

```
for i in range(dataset_apriori.shape[0]):
```

- Então, vamos adicionar nossa lista de transações em transacoes.append, colocamos abre e fecha colchetes para colocarmos no formato de lista, str para convertermos para string, dataset_apriori.values[i, j], onde o i é a variável do for que varia de 0 até 999 e essa variável j vai controlar o número de colunas que equivale a cada um dos campos da base de dados.

```
transacoes.append([str(dataset_apriori.values[i, j])
```


Geração de Regras de Associação – Algoritmo Apriori

- Vamos agora definir essa variável j, for j in range, chamamos novamente o dataset_apriori, shape na posição 1, quer dizer que nós estamos acessando cada uma das colunas desse nosso dataframe.

```
transacoes.append([str(dataset_apriori.values[i, j]) for j in range(dataset_apriori.shape[1])])
```

- Vamos executar:

```
len(transacoes)
```

```
1000
```

- Podemos observar o tamanho dessa variável transações, 1000 transações.

Geração de Regras de Associação – Algoritmo Apriori

- E para ver efetivamente o que fizemos:
- Vamos digitar **transacoes**, porém vamos pedir apenas pelos primeiros 2 registros.
- Podemos ver que ele trouxe o 1º registro em formato de lista e o 2º também:

```
transacoes[:2]
```

```
[[ 'Faixa4',  
   ' Private',  
   ' 10th',  
   ' Married-civ-spouse',  
   ' Husband',  
   ' Transport-moving',  
   ' Male',  
   ' United-States',  
   ' >50K'],  
 [ 'Faixa3',  
   ' Private',  
   ' HS-grad',  
   ' Married-civ-spouse',  
   ' Husband',  
   ' Craft-repair',  
   ' Male',  
   ' United-States',  
   ' <=50K']]
```

Geração de Regras de Associação – Algoritmo Apriori

- Vamos precisar agora realizar a instalação da biblioteca:

!pip install apyori

- Após finalizar a instalação vamos solicitar um:

```
from apyori import apriori
```

- Apenas para verificar o funcionamento dela.

- Agora vamos fazer a geração das regras.

- Então, eu crio uma variável chamada de regras que recebe a função apriori, passamos as **transações** que é a nossa base de dados e precisamos definir aqueles valores que nós vimos na aula teórica: o suporte mínimo= vamos fazer um teste com 0.3 (registros que aparecem 30% das vezes), vamos definir também a confiança mínima=0.2, ou seja, 20%.

```
from apyori import apriori
```

```
regras = apriori(transacoes, min_support = 0.3, min_confidence = 0.2)  
resultados = list(regras)
```

Geração de Regras de Associação – Algoritmo Apriori

- Criamos a variável resultados e precisamos fazer a transformação dessas regras no formato de lista.

```
regras = apriori(transacoes, min_support = 0.3, min_confidence = 0.2)
resultados = list(regras)
```

- Vamos agora visualizar quantas regras foram geradas:

```
len(resultados)
```

38

- Um total de 38 regras.

Geração de Regras de Associação – Algoritmo Apriori

- Caso queiramos analisar as regras podemos digitar **resultados** e temos portanto cada uma das regras de associação que foram geradas. Lembrando que o objetivo dessa aula é que você possa ver que é possível trabalhar com regras de associação quando você faz a transformação de atributos numéricos em categóricos.
- Podemos observar que nós temos as faixas da idade que nós definimos anteriormente:

resultados

```
[RelationRecord(items=frozenset({' <=50K'}), support=0.752, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' <=50K'}), confidence=0.752, lift=1.0)]),
 RelationRecord(items=frozenset({' Female'}), support=0.341, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Female'}), confidence=0.341, lift=1.0)]),
 RelationRecord(items=frozenset({' HS-grad'}), support=0.33, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' HS-grad'}), confidence=0.33, lift=1.0)]),
 RelationRecord(items=frozenset({' Husband'}), support=0.396, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Husband'}), confidence=0.396, lift=1.0)]),
 RelationRecord(items=frozenset({' Male'}), support=0.659, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Male'}), confidence=0.659, lift=1.0)]),
 RelationRecord(items=frozenset({' Married-civ-spouse'}), support=0.453, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Married-civ-spouse'}), confidence=0.453, lift=1.0)]),
 RelationRecord(items=frozenset({' Never-married'}), support=0.34, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Never-married'}), confidence=0.34, lift=1.0)]),
 RelationRecord(items=frozenset({' Private'}), support=0.703, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Private'}), confidence=0.703, lift=1.0)]),
 RelationRecord(items=frozenset({' United-States'}), support=0.909, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' United-States'}), confidence=0.909, lift=1.0)]),
 RelationRecord(items=frozenset({' Faixa3'}), support=0.373, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Faixa3'}), confidence=0.373, lift=1.0)]),
 RelationRecord(items=frozenset({' Faixa4'}), support=0.338, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Faixa4'}), confidence=0.338, lift=1.0)]),
 RelationRecord(items=frozenset({' Female', ' <=50K'}), support=0.303, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Female', ' <=50K'}), confidence=0.303, lift=1.0), OrderedStatistic(items_base=frozenset({' <=50K'}), items_add=frozenset({' Female'}), confidence=0.4029255319148936, lift=1.1815998003369312), OrderedStatistic(items_base=frozenset({' Female'}), items_add=frozenset({' <=50K'}), confidence=0.8885630498533723,
```


Geração de Regras de Associação – Algoritmo Apriori

- Vamos analisar a regra abaixo que une os atributos 50000 dólares por ano e sexo masculino, pra isso vamos criar um índice dessa regra. Fazendo a contagem de linhas percebemos que esse índice é o da linha doze.

resultados

```
[RelationRecord(items=frozenset({' <=50K'}), support=0.763, ordered_statistics=[OrderedStatistic(items_base=frozenset()  
RelationRecord(items=frozenset({' Female'}), support=0.341, ordered_statistics=[OrderedStatistic(items_base=frozenset()  
RelationRecord(items=frozenset({' HS-grad'}), support=0.31, ordered_statistics=[OrderedStatistic(items_base=frozenset()  
RelationRecord(items=frozenset({' Husband'}), support=0.385, ordered_statistics=[OrderedStatistic(items_base=frozenset()  
RelationRecord(items=frozenset({' Male'}), support=0.659, ordered_statistics=[OrderedStatistic(items_base=frozenset()  
RelationRecord(items=frozenset({' Married-civ-spouse'}), support=0.441, ordered_statistics=[OrderedStatistic(items_base=fro  
RelationRecord(items=frozenset({' Never-married'}), support=0.345, ordered_statistics=[OrderedStatistic(items_base=fro  
RelationRecord(items=frozenset({' Private'}), support=0.668, ordered_statistics=[OrderedStatistic(items_base=frozenset()  
RelationRecord(items=frozenset({' United-States'}), support=0.893, ordered_statistics=[OrderedStatistic(items_base=fro  
RelationRecord(items=frozenset({' Faixa3'}), support=0.4, ordered_statistics=[OrderedStatistic(items_base=frozenset()  
RelationRecord(items=frozenset({' Faixa4'}), support=0.322, ordered_statistics=[OrderedStatistic(items_base=frozenset()  
RelationRecord(items=frozenset({' <=50K', ' Female'}), support=0.307, ordered_statistics=[OrderedStatistic(items_base=  
RelationRecord(items=frozenset({' <=50K', ' Male'}), support=0.456, ordered_statistics=[OrderedStatistic(items_base=fr  
RelationRecord(items=frozenset({' <=50K', ' Never-married'}), support=0.33, ordered_statistics=[OrderedStatistic(items  
RelationRecord(items=frozenset({' <=50K', ' Private'}), support=0.54, ordered_statistics=[OrderedStatistic(items_base=  
RelationRecord(items=frozenset({' <=50K', ' United-States'}), support=0.679, ordered_statistics=[OrderedStatistic(item  
RelationRecord(items=frozenset({' <=50K', ' Faixa3'}), support=0.309, ordered_statistics=[OrderedStatistic(items_base=f  
RelationRecord(items=frozenset({' Male', ' Husband'}), support=0.385, ordered_statistics=[OrderedStatistic(items_base=  
RelationRecord(items=frozenset({' Married-civ-spouse', ' Husband'}), support=0.385, ordered_statistics=[OrderedStatist  
RelationRecord(items=frozenset({' United-States', ' Husband'}), support=0.346, ordered_statistics=[OrderedStatistic(it  
RelationRecord(items=frozenset({' Male', ' Married-civ-spouse'}), support=0.387, ordered_statistics=[OrderedStatistic(  
RelationRecord(items=frozenset({' Male', ' Private'}), support=0.428, ordered_statistics=[OrderedStatistic(items_base=
```

Geração de Regras de Associação – Algoritmo Apriori

➤ Vamos digitar:

resultados[12]:

```
resultados[12]  
RelationRecord(items=frozenset({' Male', ' <=50K'}), support=0.449, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' Male', ' <=50K'}), confidence=0.449, lift=1.0), OrderedStatistic(items_base=frozenset({' <=50K'}), items_add=frozenset({' Male'}), confidence=0.5970744680851064, lift=0.9060310593097214), OrderedStatistic(items_base=frozenset({' Male'}), items_add=frozenset({' <=50K'}), confidence=0.6813353566009105, lift=0.9060310593097214)])
```

- Essa 1ª parte indica que ele está combinando esses dois atributos e em seguida indica que o suporte desses dois é 0.449, ou seja, em 44,9% da base de dados temos o sexo masculino e a renda igual ou superior aos USD 50.000.
- Seguindo a sequência, podemos observar na parte chamada de OrderedStatistic, ela indica a regra, se por exemplo a pessoa ganha menos do que USD 50.000 por ano, então é do sexo masculino a confiança é 59%.

```
OrderedStatistic(items_base=frozenset({' <=50K'}), items_add=frozenset({' Male'}), confidence=0.5970744680851064,  
stic(items_base=frozenset({' Male'}), items_add=frozenset({' <=50K'}), confidence=0.6813353566009105, lift=0.9060310593097214)])
```

Geração de Regras de Associação – Algoritmo Apriori

- Indo a frente veremos o contrário: se é masculino, então ganha menos de USD 50.000 por ano com uma confiança maior, de quase 70%:

```
OrderedStatistic(items_base=frozenset({' Male'}), items_add=frozenset({' <=50K'}), confidence=0.6813353566009105,
```

- Por fim, pudemos compreender nessa etapa, como podemos utilizar essa técnica de distribuição de frequência para fazer ajustes na base de dados para fazer aplicação de regras de associação que requer somente atributos categóricos.