

---

## **Python para Usuários de Excel**

Asimov Academy

# ASIMOV

## Conteúdo

<b>1. O que vamos aprender nesse curso?</b>	<b>4</b>
Mas afinal, o que eu vou construir? . . . . .	4
Quais ferramentas serão utilizadas? . . . . .	7
<b>2. Criando uma tabela</b>	<b>8</b>
Breve introdução a Pandas . . . . .	8
Quais tipos de dados trabalhamos em Pandas . . . . .	8
Criando DataFrames . . . . .	9
Criando a partir de listas . . . . .	10
Criando a partir de lista de listas . . . . .	10
Criando a partir de dicionário de listas . . . . .	10
Criando com índice . . . . .	10
Criando a partir de uma lista de dicionários . . . . .	11
Principais Atributos . . . . .	11
columns . . . . .	11
index . . . . .	11
shape . . . . .	11
<b>3. Importando arquivos Excel</b>	<b>13</b>
Lendo arquivos xlsx . . . . .	13
Lendo arquivos csv . . . . .	14
<b>4. Visualizando Tabelas</b>	<b>16</b>
Visualização básica no jupyter . . . . .	16
Visualização os primeiros valores (.head) . . . . .	16
Visualização os últimos valores (.tail) . . . . .	17
Visualizando tabelas maiores no Jupyter (display.max_rows) . . . . .	18
Estilizando a visualização . . . . .	18
Formatando floats . . . . .	18
Estilizando colunas . . . . .	19
Estilizando index . . . . .	20
Adicionando gradiente (formatação condicional) . . . . .	20
Juntando tudo . . . . .	21
<b>5. PROJETO 1 - Visualizando dados em um Web App</b>	<b>23</b>

<b>6. Selecionando colunas e filtrando linhas</b>	<b>24</b>
Selecionando colunas . . . . .	24
Filtrando por indexação (.iloc) . . . . .	25
Filtrando por condições (.loc) . . . . .	26
Filtrando por múltiplas condições (.loc) . . . . .	26
Filtrando por lista de valores (.isin) . . . . .	27
Filtrando por Series . . . . .	27
<b>7. Adicionando novas colunas</b>	<b>28</b>
Criando colunas a partir de listas . . . . .	28
Criando colunas com o método .insert . . . . .	29
Criando colunas a partir de Series . . . . .	29
<b>8. Adicionando novas linhas</b>	<b>31</b>
Com uma lista utilizando .loc . . . . .	31
Com um DataFrame e utilizando o método pd.concat . . . . .	32
<b>9. PROJETO 2 - Seleção de colunas e filtro de linhas</b>	<b>34</b>
<b>10. PROJETO 3 - Adição de linhas à tabela</b>	<b>36</b>
<b>11. Mesclando tabelas (o PROCV no Python)</b>	<b>38</b>
Utilizando o método .merge . . . . .	39
Utilizando o método .map . . . . .	40
Utilizando o método .replace . . . . .	41
<b>12. Realizando cálculos na tabela</b>	<b>42</b>
Vendo as características gerais dos dados . . . . .	42
Contando valores únicos . . . . .	44
Operações por toda a tabela e por coluna . . . . .	44
<b>13. PROJETO 4 - Visualizando volume de vendas e comissões</b>	<b>46</b>
<b>14. Pivotando e agrupando tabelas (as TABELAS DINÂMICAS)</b>	<b>48</b>
Utilizando o método pivot_table . . . . .	48
Utilizando o método groupby . . . . .	49
<b>15. PROJETO 5 - Tabela dinâmica</b>	<b>51</b>
<b>16. Criando gráficos com plotly</b>	<b>52</b>
Introdução a Plotly . . . . .	52

Criando gráficos com Plotly . . . . .	52
Criando gráficos de linha . . . . .	53
Criando gráficos de barra . . . . .	54
Criando gráficos de pizza . . . . .	55
<b>17. PROJETO 6 - Adicionando gráficos</b>	<b>56</b>

## 1. O que vamos aprender nesse curso?

Este curso prático foi projetado especialmente para aqueles que desejam aprender através de exemplos práticos e começar a colocar a mão na massa desde o início.

Frequentemente, ao iniciar a jornada de aprendizado em programação, muitos se sentem desmotivados devido à frustração de investir horas estudando sem conseguir realizar operações simples, comparáveis às que fazemos em uma planilha do Excel. Essa experiência pode ser desanimadora, uma vez que nos mantém distantes das habilidades práticas necessárias no nosso mundo moderno.

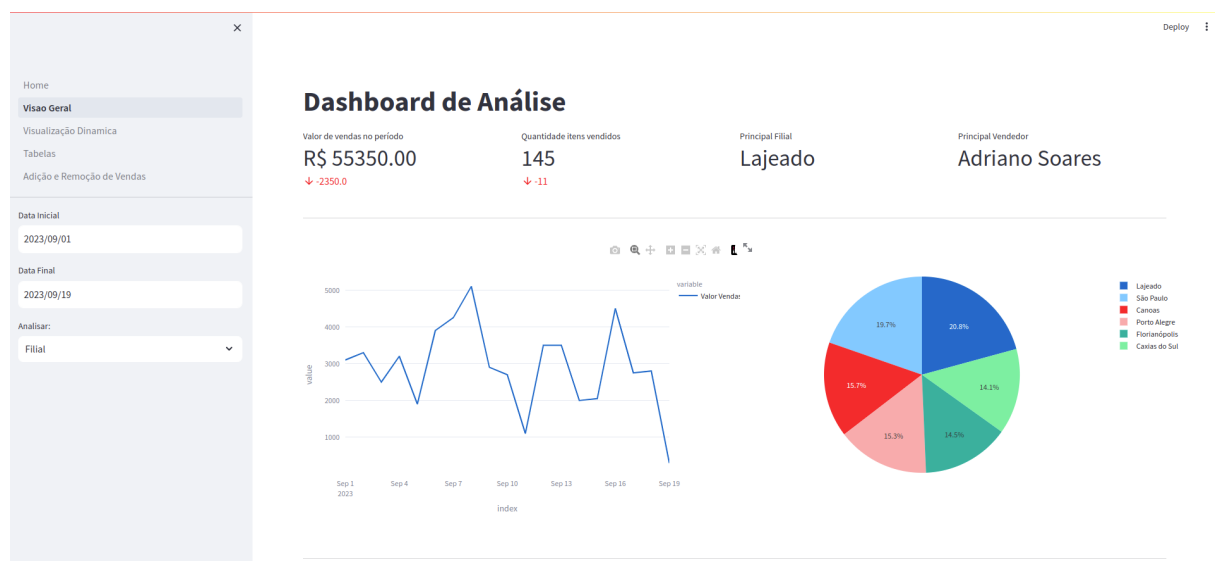
A nossa proposta é mostrar que a programação em Python não é um bicho de sete cabeças. Com facilidade, você estará criando aplicativos web, desenvolvendo dashboards e realizando todas as tarefas que normalmente faria no Excel - tudo isso com uma abordagem prática e direta.

Vamos abordar os conceitos de maneira acessível e introdutória, mas tenha certeza de que, ao ver tudo o que construiremos ao longo deste curso, você terá uma compreensão expandida sobre o potencial da programação e como ela pode ser uma ferramenta poderosa.

Queremos que, ao concluir este curso, você se sinta motivado e se pergunte: “por que demorei tanto pra começar a programar?”. Que pense que aqueles dias de trabalhos manuais e enfadonhos ficaram para trás, pois você é uma nova pessoa, um novo profissional!

### Mas afinal, o que eu vou construir?

No final do curso, você saberá construir um webApp como esse:



Ele possui diversas funcionalidades, desde visualização dos dados que permitem essa análise (são dados de vendas de uma loja com filiais!)

Home

Visão Geral

Visualização Dinâmica

**Tabelas**

Adição e Remoção de Vendas

**Seleção de Tabelas**

Selecione a tabela que você deseja ver:

Vendas

**Filtrar tabela**

Selecione as colunas da tabela:

id\_venda x

filial x

vendedor x

produto x

cliente\_nome x

cliente\_genero x

forma\_pagamento x

Filtrar coluna

Filial

Valor do filtro

Choose an option

Filtrar

Limpar

data	id_venda	filial	vendedor	produto	cliente_nome	cliente_genero	forma_pagamento
2023-01-12 07:42:38	1	Canoas	Luis Fernando	Tenis Nike	Anthony Moon	masculino	credito
2023-01-12 08:21:37	2	Florianópolis	Claudia dos Santos	Tenis Adidas	Bud Loughlin	masculino	pix
2023-01-12 08:27:36	3	São Paulo	Cassia Moraes	Tenis Fila	Gonzalo Kroes	masculino	credito
2023-01-12 09:32:38	4	Florianópolis	Mario Sérgio	Tenis Nike	Lisa Thomas	feminino	credito
2023-01-12 11:01:38	5	Canoas	Luis Fernando	Tenis Fila	Charles Bowlin	masculino	credito
2023-01-12 14:41:39	6	Canoas	Carlos Henrique	Tenis Adidas	Shari Craig	feminino	credito
2023-01-13 06:40:39	7	Florianópolis	Claudia dos Santos	Tenis Adidas	Frank Felker	masculino	credito
2023-01-13 07:21:39	8	São Paulo	Cassia Moraes	Tenis NB	Eric Ryan	masculino	credito
2023-01-13 09:43:37	9	Canoas	Luis Fernando	Tenis Adidas	Myron Wallen	masculino	credito
2023-01-13 09:58:39	10	Caxias do Sul	Rodrigo Vanzeloti	Tenis Nike	Dawn Cobb	feminino	credito
2023-01-13 10:44:38	11	São Paulo	Claudio Bueno	Tenis NB	John Leon	masculino	pix
2023-01-13 12:51:37	12	Florianópolis	Claudia dos Santos	Tenis Nike	Mario Leonard	masculino	credito
2023-01-13 13:45:39	13	Lajeado	Adriano Soares	Tenis NB	Edward Prince	masculino	credito
2023-01-13 14:09:39	14	Caxias do Sul	Rodrigo Vanzeloti	Tenis Adidas	Audrey Hurley	feminino	credito
2023-01-13 15:38:37	15	Lajeado	Adriano Soares	Tenis Adidas	Kenneth Lightfo	masculino	credito
2023-01-14 05:00:40	16	São Paulo	Cassia Moraes	Tenis Nike	Michael Griffin	masculino	credito
2023-01-14 06:59:36	17	Canoas	Carlos Henrique	Tenis Fila	Glen Hesterman	masculino	credito
2023-01-14 07:46:37	18	São Paulo	Cassia Moraes	Tenis Fila	Sue Yang	feminino	credito
2023-01-14 07:47:37	19	Florianópolis	Claudia dos Santos	Tenis NB	Mario Keller	masculino	credito
2023-01-14 10:56:38	20	Caxias do Sul	Rodrigo Vanzeloti	Tenis Fila	Bernice Edwards	feminino	credito
2023-01-14 10:57:38	21	Florianópolis	Mario Sérgio	Tenis Nike	Joan Lawrence	feminino	credito
2023-01-14 14:02:38	22	Lajeado	Juliano Faccioni	Tenis Adidas	Mary Wright	feminino	credito

A visualização dinâmica dos dados (estilo tabela dinâmica do excel):

[Home](#)  
[Visão Geral](#)  
**[Visualização Dinâmica](#)**  
[Tabelas](#)  
[Adição e Remoção de Vendas](#)

Selecione os índices:

vendedor x v

Selecione as colunas:

cliente\_genero x v

Selecione o valor de análise:

preco v

Selecione a métrica:

soma v

vendedor	feminino	masculino	TOTAL GERAL
Adriano Soares	29,650	33,100	62,750
Carlos Henrique	32,750	25,750	58,500
Cassia Moraes	30,800	25,550	56,350
Claudia dos Santos	30,700	34,350	65,050
Claudio Bueno	27,750	35,250	63,000
Juliano Faccioni	30,650	26,650	57,300
Luis Fernando	35,200	40,250	75,450
Luiza Cherobini	27,050	31,050	58,100
Mario Sérgio	36,800	33,600	70,400
Mateus Kienzle	33,400	33,150	66,550

A adição e remoção de dados da tabela:

[Home](#)  
[Visão Geral](#)  
[Visualização Dinâmica](#)  
[Tabelas](#)  
[Adição e Remoção de Vendas](#)

### Adição de Vendas

Selecione a filial:

Porto Alegre/RS

Selecione o vendedor:

Rodrigo Tadewald

Selecione o produto:

Tenis Nike

Nome do cliente:

Gênero do cliente:

feminino

Forma de pagamento

pix

Adicionar venda

data	id_venda	filial	vendedor	produto	cliente_nome	cliente_genero	forma_pagamento
2023-01-12 07:42:38	1	Canoas	Luis Fernando	Tenis Nike	Anthony Moon	masculino	credito
2023-01-12 08:21:37	2	Florianópolis	Claudia dos Santos	Tenis Adidas	Bud Loughlin	masculino	pix
2023-01-12 08:27:36	3	São Paulo	Cassia Moraes	Tenis Fila	Gonzalo Kroes	masculino	credito
2023-01-12 09:32:38	4	Florianópolis	Mario Sérgio	Tenis Nike	Lisa Thomas	feminino	credito
2023-01-12 11:01:38	5	Canoas	Luis Fernando	Tenis Fila	Charles Bowlin	masculino	credito
2023-01-12 14:41:39	6	Canoas	Carlos Henrique	Tenis Adidas	Shari Craig	feminino	credito
2023-01-13 06:40:39	7	Florianópolis	Claudia dos Santos	Tenis Adidas	Frank Felker	masculino	credito
2023-01-13 07:21:39	8	São Paulo	Cassia Moraes	Tenis NB	Eric Ryan	masculino	credito
2023-01-13 09:43:37	9	Canoas	Luis Fernando	Tenis Adidas	Myron Wallen	masculino	credito
2023-01-13 09:58:39	10	Caxias do Sul	Rodrigo Vanzeloti	Tenis Nike	Dawn Cobb	feminino	credito
2023-01-13 10:44:38	11	São Paulo	Claudio Bueno	Tenis NB	John Leon	masculino	pix
2023-01-13 12:51:37	12	Florianópolis	Claudia dos Santos	Tenis Nike	Mario Leonard	masculino	credito
2023-01-13 13:45:39	13	Lajeado	Adriano Soares	Tenis NB	Edward Prince	masculino	credito
2023-01-13 14:09:39	14	Caxias do Sul	Rodrigo Vanzeloti	Tenis Adidas	Audrey Hurley	feminino	credito
2023-01-13 15:38:37	15	Lajeado	Adriano Soares	Tenis Adidas	Kenneth Lightfo	masculino	credito
2023-01-14 05:00:40	16	São Paulo	Cassia Moraes	Tenis Nike	Michael Griffin	masculino	credito
2023-01-14 06:59:36	17	Canoas	Carlos Henrique	Tenis Fila	Glen Hesterman	masculino	credito
2023-01-14 07:46:37	18	São Paulo	Cassia Moraes	Tenis Fila	Sue Yang	feminino	credito
2023-01-14 07:47:37	19	Florianópolis	Claudia dos Santos	Tenis NB	Mario Keller	masculino	credito
2023-01-14 10:56:38	20	Caxias do Sul	Rodrigo Vanzeloti	Tenis Fila	Bernice Edwards	feminino	credito
2023-01-14 10:57:38	21	Florianópolis	Mario Sérgio	Tenis Nike	Joan Lawrence	feminino	credito
2023-01-14 14:02:38	22	Laieado	Juliano Faccioni	Tenis Adidas	Marv Wrieth	feminino	credito

No final você terá a capacidade de construir algo similar em menos de uma hora.

### Quais ferramentas serão utilizadas?

É esperado que agora você já saiba os conceitos básicos de programação Python. Mas apenas isso, nenhum outro pré-requisito é necessário.

Nesse curso usaremos três principais bibliotecas:

- pandas: para a manipulação de dados em tabelas
- plotly: para geração de gráficos
- streamlit: para criação de webApps

Temos cursos específicos dessas três ferramentas, que podem ser assistidos posteriormente caso você queira se aprofundar em alguma delas.

E sem mais delongas, vamos começar o curso!



## 2. Criando uma tabela

Vamos começar apresentando a primeira ferramenta necessário quando queremos trabalhar com tabelas: **pandas**.

### Breve introdução a Pandas

Pandas é uma biblioteca desenvolvida para manipulação de dados tabulares e análise de dados. A ideia, desde sua construção, é de ser rápida e altamente flexível. Com pandas é possível trabalhar com dados vindos de várias origens: csv, excel, json, parquet, SQL, etc. Basicamente, ele se comunica com as mais diferentes formas de armazenamento de dados e consegue extrair informação de forma rápida desses mesmos dados.

Em seu site pandas diz:

“Pandas tem o objetivo mais amplo de se tornar a ferramenta de análise/manipulação de dados de código aberto mais poderosa e flexível disponível em qualquer linguagem”

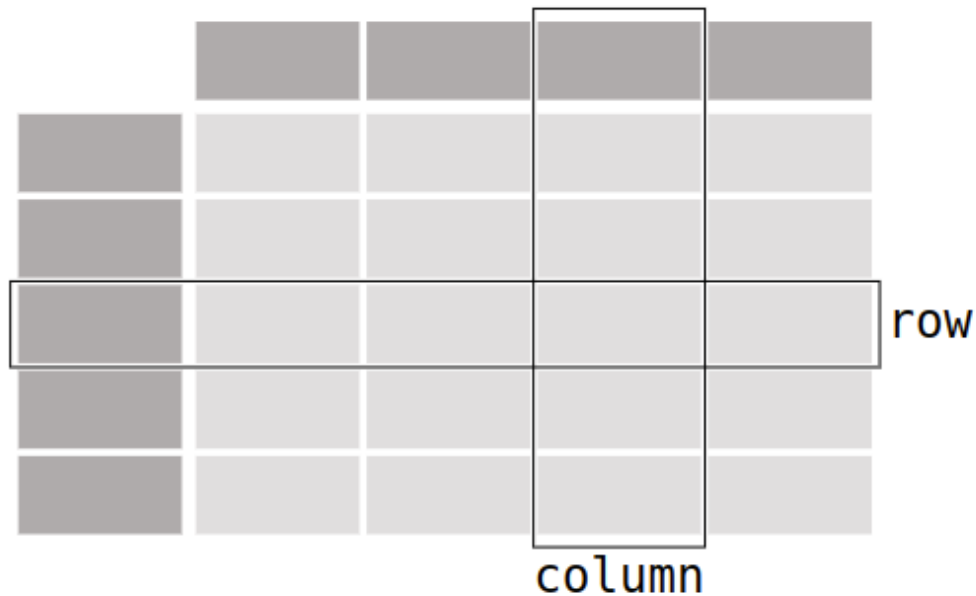
E dado a minha experiência como analista de dados, acredito que eles já tenham chegado lá!

Pandas é extremamente relevante para quem programa em Python e, se você ainda não se deparou com ele, tenho certeza que em breve se depararia!

### Quais tipos de dados trabalhamos em Pandas

Ao trabalhar com dados tabulares, como dados armazenados em planilhas ou bancos de dados, o pandas é a ferramenta ideal. pandas irá ajudá-lo a explorar, limpar e processar seus dados. No pandas, uma tabela de dados é chamada de **DataFrame**.

# DataFrame



**DataFrame** é a principal forma de representar dados de tabelas em Python! Ele consiste em colunas (columns) e linhas (rows) e é, portanto, uma estrutura de 2 dimensões.

## Criando DataFrames

Existe diversas formas para se criar um DataFrame, passaremos pelas principais. Mas antes de qualquer coisa, vamos importar pandas:

```
import pandas as pd
```

É uma convenção importar pandas como **pd**. E seguiremos essa convenção até para manter a compatibilidade dos códigos que desenvolvemos com o de outros programadores!

A criação se dá sempre chamando a classe DataFrame do pandas, da seguinte forma:

```
import pandas as pd
```

```
df = pd.DataFrame()
```

```
df
```

**Atenção** com as letras maiúsculas. Se não colocarmos o **D** e o **F** em maiúsculos, acabaremos em um erro!

### Criando a partir de listas

```
import pandas as pd

data = [1, 2, 3, 4, 5]
df = pd.DataFrame(data, columns=['Numeros'])

df
```

### Criando a partir de lista de listas

```
import pandas as pd

data = [[0, 1],
        [2, 3],
        [4, 5]]
df = pd.DataFrame(data)

df
```

### Criando a partir de dicionário de listas

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle'],}
df = pd.DataFrame(data)

df
```

### Criando com índice

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle'],}
df = pd.DataFrame(data, index=['p1', 'p2', 'p3', 'p4'])

df
```

## Criando a partir de uma lista de dicionários

```
import pandas as pd

data = [{'Nome': 'Adriano', 'Sobrenome': 'Soares'},
        {'Nome': 'Rodrigo', 'Sobrenome': 'Tadewald'},
        {'Nome': 'Juliano', 'Sobrenome': 'Faccioni'},
        {'Nome': 'Mateus', 'Sobrenome': 'Kienzle'},
        ]
df = pd.DataFrame(data)

df
```

## Principais Atributos

### columns

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle'],}
df = pd.DataFrame(data, index=['p1', 'p2', 'p3', 'p4'])

df.columns
```

Retorna uma lista com o nome das colunas da tabela

### index

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle'],}
df = pd.DataFrame(data, index=['p1', 'p2', 'p3', 'p4'])

df.index
```

Retorna o índice das linhas do DataFrame

### shape

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle'],}
df = pd.DataFrame(data, index=['p1', 'p2', 'p3', 'p4'])

df.shape
```

Retorna o formato do DataFrame (quantas linhas e quantas colunas)

### 3. Importando arquivos Excel

Maior parte das vezes que utilizarmos pandas, nós não criaremos os dados e sim acessaremos dados que já estavam previamente criados. Esses dados podem estar armazenados das mais diferentes formas e pandas consegue acessá-los pelos seus métodos **read** (ler em inglês). Por exemplo, se queremos acessar dados armazenados em csv utilizaríamos o método **pd.from\_csv**, dados armazenado em uma base de dados sql utilizaríamos **pd.from\_sql**, dados xlsx seria **pd.from\_excel**, e assim por diante!

Vamos explorar a leitura de dados csv e xlsx nesse curso, pois são os mais comuns no nosso dia-a-dia!

#### Lendo arquivos xlsx

Para ler arquivos xlsx, utilizaremos o método `.read_excel`:

```
import pandas as pd

tabela_vendas = pd.read_excel('datasets/vendas.xlsx')
tabela_vendas
```

O método recebe diferentes argumentos, vamos explorar aqui os principais:

- `sheet_name` (default 0):
  - Nome da aba. Caso não seja passado, retorna a primeira aba.
  - Caso passe o valor 0, retorna a primeira aba, se passar o valor 1, retorna a segunda e assim por diante
  - Pode ser passado uma string com o nome da aba. Ex.: “nome\_da\_aba”
  - Pode ser passado uma lista, com nome ou números das abas. Ex.: [0, 1, “nome\_da\_aba”]. Isso retornará um dicionário de DataFrames como resultado.
- `index_col` (default None):
  - Define qual coluna será tomada como índice do DataFrame. Deve ser passado como um valor inteiro, sendo que 0 representa a primeira coluna, 1 a segunda e assim por diante
- `decimal` (default ‘.’):
  - Para converter valores para numérico valores decimais da tabela. No caso, se temos na nossa tabelas valores numéricos com vírgula escritos da seguinte forma: 10,800541, teríamos que utilizar **decimal = ‘,’**
- `usecols` (default None):

- Se `None`, retorna todas as colunas da planilha. Se “A:E”, por exemplo, retorna apenas as colunas de A até E. Pode ser passado na forma de uma lista de ints também, por exemplo, `[1, 2, 3, 4, 5]` para pegar das colunas A até E.

```
import pandas as pd

tabela_vendas = pd.read_excel('datasets/vendas.xlsx',
                              index_col=0,
                              decimal=',',
                              usecols=[0, 1, 2, 3],
                              )

print(tabela_vendas)
```

### Lendo arquivos csv

Para ler arquivos csv, utilizaremos o método `.read_csv`. O funcionamento é bem similar a leitura de dados xlsx. Reforço apenas a necessidade de utilizar o parâmetro `sep= ';'``, pois o padrão de csv brasileiro é diferente do americano e, ao não utilizarmos esse parâmetro, podemos ter alguns problemas na leitura dos nossos dados:

```
import pandas as pd

tabela_vendas = pd.read_csv('datasets/vendas.csv')
tabela_vendas
```

O método recebe diferentes argumentos, vamos explorar aqui os principais:

- `index_col` (default `None`):
  - Define qual coluna será tomada como índice do DataFrame. Deve ser passado como um valor inteiro, sendo que 0 representa a primeira coluna, 1 a segunda e assim por diante
- `sep` (default `','`):
  - Caractere utilizado para delimitar as colunas de dados. Para o padrão brasileiro deve ser usado com  `';'``!
- `decimal` (default  `'.'``):
  - Para converter valores para numérico valores decimais da tabela. No caso, se temos na nossa tabelas valores numéricos com vírgula escritos da seguinte forma: 10,800541, teríamos que utilizar **`decimal = ';'``**
- `usecols` (default `None`):
  - Se `None`, retorna todas as colunas da planilha. Se “A:E”, por exemplo, retorna apenas as colunas de A até E. Pode ser passado na forma de uma lista de ints também, por exemplo, `[1, 2, 3, 4, 5]` para pegar das colunas A até E.

```
import pandas as pd

tabela_vendas = pd.read_csv('datasets/vendas.csv',
                             index_col=0,
                             sep=';',
                             decimal=',',
                             )

tabela_vendas
```



## 4. Visualizando Tabelas

Vamos utilizar jupyter notebook nesse curso, pois é IDE mais apropriada para visualização e manipulação de tabelas.

### Visualização básica no jupyter

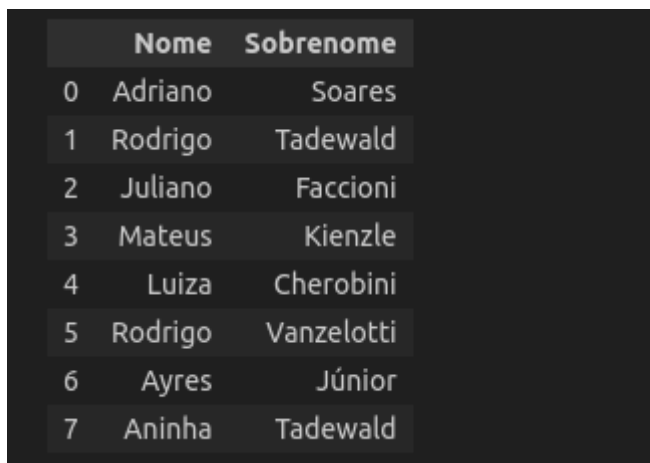
A primeira forma de visualizar dados no jupyter é simplesmente escrevendo o nome do DataFrame que contém os dados no final da célula:

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus', 'Luiza', 'Rodrigo', 'Ayres',
               ↪ 'Aninha'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle', 'Cherobini', 'Vanzelotti',
                      ↪ 'Júnior', 'Tadewald'],}

df = pd.DataFrame(data)

df
```



	Nome	Sobrenome
0	Adriano	Soares
1	Rodrigo	Tadewald
2	Juliano	Faccioni
3	Mateus	Kienzle
4	Luiza	Cherobini
5	Rodrigo	Vanzelotti
6	Ayres	Júnior
7	Aninha	Tadewald

### Visualização os primeiros valores (.head)

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus', 'Luiza', 'Rodrigo', 'Ayres',
               ↪ 'Aninha'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle', 'Cherobini', 'Vanzelotti',
                      ↪ 'Júnior', 'Tadewald'],}

df = pd.DataFrame(data)

df.head()
```

	Nome	Sobrenome
0	Adriano	Soares
1	Rodrigo	Tadewald
2	Juliano	Faccioni
3	Mateus	Kienzle
4	Luiza	Cherobini

Dessa forma visualizamos apenas os primeiros dados do DataFrame. O valor default é 5, portanto visualizamos as primeiras 5 linhas. Mas podemos passar qualquer valor inteiro desejado:

```
df.head(2)
```

	Nome	Sobrenome
0	Adriano	Soares
1	Rodrigo	Tadewald

## Visualização os últimos valores (.tail)

```
import pandas as pd
```

```
data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus', 'Luiza', 'Rodrigo', 'Ayres',  
↪ 'Aninha'],  
'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle', 'Cherobini', 'Vanzelotti', 'Júnior',  
↪ 'Tadewald'],}
```

```
df = pd.DataFrame(data)
```

```
df.tail()
```

	Nome	Sobrenome
3	Mateus	Kienzle
4	Luiza	Cherobini
5	Rodrigo	Vanzelotti
6	Ayres	Júnior
7	Aninha	Tadewald

Assim visualizamos os 5 últimos dados do DataFrame. da mesma forma, qualquer valor inteiro pode ser passado:

```
df.tail(3)
```

	Nome	Sobrenome
5	Rodrigo	Vanzelotti
6	Ayres	Júnior
7	Aninha	Tadewald

### Visualizando tabelas maiores no Jupyter (display.max\_rows)

Por padrão, quando pedimos para visualizar um DataFrame ele só mostrará algumas linhas, não todas. Para aumentarmos esse limite, podemos modificar as configurações padrão do pandas com o seguinte comando

```
import pandas as pd

pd.set_option('display.max_rows', 100)
tabela_vendas = pd.read_csv('datasets/vendas.csv', sep=';', decimal=',')

tabela_vendas.head(100)
```

Dessa forma, visualizaremos 100 linhas. Podemos modificar para o número que desejarmos dessa forma!

### Estilizando a visualização

Mostraremos algumas estilizações simples na visualização de tabelas que podem ser úteis.

#### Formatando floats

```
import pandas as pd
import numpy as np

df_tempo = pd.DataFrame(np.random.rand(10,2)*5,
                        index=pd.date_range(start="2023-01-01", periods=10),
                        columns=["Porto Alegre", "São Paulo"])
df_tempo.style.format(precision=3, decimal=",")
```

	Porto Alegre	São Paulo
2023-01-01 00:00:00	2,405	2,117
2023-01-02 00:00:00	3,279	4,353
2023-01-03 00:00:00	2,094	0,664
2023-01-04 00:00:00	1,200	1,443
2023-01-05 00:00:00	3,180	4,453
2023-01-06 00:00:00	1,678	3,229
2023-01-07 00:00:00	4,478	1,328
2023-01-08 00:00:00	0,814	4,365
2023-01-09 00:00:00	1,420	3,476
2023-01-10 00:00:00	2,020	4,716

Modificamos o decimal para ‘,’ na visualização e pedimos para ele mostrar apenas três casas após a vírgula

### Estilizando colunas

```
df_tempo.style.format_index(str.upper, axis=1)
```

	PORTO ALEGRE	SÃO PAULO
2023-01-01 00:00:00	2.405119	2.116671
2023-01-02 00:00:00	3.278992	4.353374
2023-01-03 00:00:00	2.094192	0.663706
2023-01-04 00:00:00	1.200185	1.442539
2023-01-05 00:00:00	3.179657	4.452899
2023-01-06 00:00:00	1.677692	3.229483
2023-01-07 00:00:00	4.477642	1.327793
2023-01-08 00:00:00	0.813513	4.364713
2023-01-09 00:00:00	1.419958	3.475957
2023-01-10 00:00:00	2.019972	4.715559

Os nomes das colunas se tornarão maiúsculas

## Estilizando index

```
df_tempo.style.format_index(lambda v: v.strftime("%d/%m/%Y"))
```

	Porto Alegre	São Paulo
01/01/2023	2.405119	2.116671
02/01/2023	3.278992	4.353374
03/01/2023	2.094192	0.663706
04/01/2023	1.200185	1.442539
05/01/2023	3.179657	4.452899
06/01/2023	1.677692	3.229483
07/01/2023	4.477642	1.327793
08/01/2023	0.813513	4.364713
09/01/2023	1.419958	3.475957
10/01/2023	2.019972	4.715559

Formatamos a visualização da data no índice ### Adicionando título

```
df_tempo.style.set_caption('Condição do tempo')
```

Condição do tempo		
	Porto Alegre	São Paulo
2023-01-01 00:00:00	2.405119	2.116671
2023-01-02 00:00:00	3.278992	4.353374
2023-01-03 00:00:00	2.094192	0.663706
2023-01-04 00:00:00	1.200185	1.442539
2023-01-05 00:00:00	3.179657	4.452899
2023-01-06 00:00:00	1.677692	3.229483
2023-01-07 00:00:00	4.477642	1.327793
2023-01-08 00:00:00	0.813513	4.364713
2023-01-09 00:00:00	1.419958	3.475957
2023-01-10 00:00:00	2.019972	4.715559

## Adicionando gradiente (formatação condicional)

```
df_tempo.style.background_gradient(axis=None, vmin=1, vmax=5, cmap="YlGnBu")
```

	Porto Alegre	São Paulo
2023-01-01 00:00:00	2.405119	2.116671
2023-01-02 00:00:00	3.278992	4.353374
2023-01-03 00:00:00	2.094192	0.663706
2023-01-04 00:00:00	1.200185	1.442539
2023-01-05 00:00:00	3.179657	4.452899
2023-01-06 00:00:00	1.677692	3.229483
2023-01-07 00:00:00	4.477642	1.327793
2023-01-08 00:00:00	0.813513	4.364713
2023-01-09 00:00:00	1.419958	3.475957
2023-01-10 00:00:00	2.019972	4.715559

### Juntando tudo

```
def condição_chuva(v):  
    if v < 1.75:  
        return "Seco"  
    elif v < 2.75:  
        return "Chuva"  
    return "Chuva Forte"  
  
def embelezar(styler):  
    styler.set_caption("Condição do tempo")  
    styler.format(condição_chuva)  
    styler.format_index(lambda v: v.strftime("%d/%m/%Y"))  
    styler.background_gradient(axis=None, vmin=1, vmax=5, cmap="YlGnBu")  
    return styler  
  
df_tempo.style.pipe(embelezar)
```

Condição do tempo		
	Porto Alegre	São Paulo
01/01/2023	Chuva	Chuva
02/01/2023	Chuva Forte	Chuva Forte
03/01/2023	Chuva	Seco
04/01/2023	Seco	Seco
05/01/2023	Chuva Forte	Chuva Forte
06/01/2023	Seco	Chuva Forte
07/01/2023	Chuva Forte	Seco
08/01/2023	Seco	Chuva Forte
09/01/2023	Seco	Chuva Forte
10/01/2023	Chuva	Chuva Forte

## 5. PROJETO 1 - Visualizando dados em um Web App

Começamos aqui nossa primeira etapa do projeto com Streamlit. Esse é o código final que obteremos nessa aula:

```
from pathlib import Path

import streamlit as st
import pandas as pd

pasta_datasets = Path(__file__).parent.parent / 'datasets'
df_vendas = pd.read_csv(pasta_datasets / 'vendas.csv', decimal=',', sep=';')

st.dataframe(df_vendas, height=800)
```



## 6. Selecionando colunas e filtrando linhas

### Selecionando colunas

Existe duas formas principais de selecionar colunas. Uma é utilizando o .

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus', 'Luiza', 'Rodrigo', 'Ayes',
↪ 'Aninha'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle', 'Cherobini', 'Vanzelotti',
↪ 'Júnior', 'Tadewald'],
        'Idade Atual': [31, 31, 25, 35, 22, 21, 28, 20],
        }

df = pd.DataFrame(data)
df.Sobrenome
```

```
0      Soares
1    Tadewald
2    Faccioni
3    Kienzle
4    Cherobini
5  Vanzelotti
6      Júnior
7    Tadewald
Name: Sobrenome, dtype: object
```

Essa forma não é recomendada por dois motivos: - Não funciona caso o nome da coluna tenha um espaço, portanto, não funcionaria com a coluna “Idade Atual” - Permite apenas a seleção de uma coluna por vez

A forma recomendada que contorna os problemas é utilizando [].

```
df['Sobrenome']
```

```
0      Soares
1    Tadewald
2    Faccioni
3    Kienzle
4    Cherobini
5  Vanzelotti
6      Júnior
7    Tadewald
Name: Sobrenome, dtype: object
```

```
df[['Sobrenome', 'Idade Atual']]
```

	Sobrenome	Idade Atual
0	Soares	31
1	Tadewald	31
2	Faccioni	25
3	Kienzle	35
4	Cherobini	22
5	Vanzelotti	21
6	Júnior	28
7	Tadewald	20

```
df[[col for col in df.columns if not col in ['Sobrenome']]]
```

	Nome	Idade Atual
0	Adriano	31
1	Rodrigo	31
2	Juliano	25
3	Mateus	35
4	Luiza	22
5	Rodrigo	21
6	Ayres	28
7	Aninha	20

### Filtrando por indexação (.iloc)

Da mesma forma que filtramos uma lista, podemos fazer em um DataFrame para filtrar por indexação utilizando iloc:

```
df.iloc[1:3]
```

	Nome	Sobrenome	Idade Atual
1	Rodrigo	Tadewald	31
2	Juliano	Faccioni	25

```
df.iloc[1:3, :1]
```

	Nome	Sobrenome
1	Rodrigo	Tadewald
2	Juliano	Faccioni

### Filtrando por condições (.loc)

```
df[df['Nome'] == 'Rodrigo']
```

	Nome	Sobrenome	Idade Atual
1	Rodrigo	Tadewald	31
5	Rodrigo	Vanzelotti	21

Desse modo filtramos pela condição de interesse. Podemos também utilizar `.loc` para realizar a mesma operação:

```
df.loc[df['Nome'] == 'Rodrigo']
```

	Nome	Sobrenome	Idade Atual
1	Rodrigo	Tadewald	31
5	Rodrigo	Vanzelotti	21

Não há diferença entre as abordagens e ambas podem ser utilizadas.

```
df.loc[df['Nome'] == 'Rodrigo', 'Sobrenome']
```

```
1    Tadewald
5    Vanzelotti
Name: Sobrenome, dtype: object
```

Com `.loc` podemos fazer simultaneamente a seleção de colunas também!

### Filtrando por múltiplas condições (.loc)

```
df.loc[(df['Nome'] == 'Rodrigo') & (df['Idade Atual'] < 30)]
```

	Nome	Sobrenome	Idade Atual
5	Rodrigo	Vanzelotti	21

## Filtrando por lista de valores (.isin)

```
nomes_filtro = ['Adriano', 'Luiza']  
  
df.loc[df['Nome'].isin(nomes_filtro)]
```

	Nome	Sobrenome	Idade Atual
0	Adriano	Soares	31
4	Luiza	Cherobini	22

## Filtrando por Series

Algo que não falamos ainda no curso são das `pd.Series`. Elas não passam de DataFrames com apenas uma dimensão e não duas. > Um DataFrame é composto por um conjunto de Series e uma Series é composta por um conjunto de valores

Podemos filtrar sempre um DataFrame através de uma Series com valores booleanos (True e False)

```
df['Nome'].str.contains("Adriano")
```

```
0    True  
1   False  
2   False  
3   False  
4   False  
5   False  
6   False  
7   False  
Name: Nome, dtype: bool
```

```
df.loc[df['Nome'].str.contains("Adriano")]
```

	Nome	Sobrenome	Idade Atual
0	Adriano	Soares	31

## 7. Adicionando novas colunas

Como tudo em DataFrames, existe várias formas de adicionarmos uma coluna nova a nossa tabela. Passaremos aqui da forma mais simples às mais avançadas. ## Criando colunas a partir de um valor

A primeira forma é simplesmente a partir de um único valor que será replicado por todas as linhas da nova coluna. Começamos com o mesmo DataFrame que estávamos utilizando anteriormente.

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus', 'Luiza', 'Rodrigo', 'Ayres',
               ↪ 'Aninha'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle', 'Cherobini', 'Vanzelotti',
               ↪ 'Júnior', 'Tadewald'],
        'Idade Atual': [31, 31, 25, 35, 22, 21, 28, 20],
        }

df = pd.DataFrame(data)
```

E agora adicionamos a coluna “É gente boa” à tabela:

```
df['É gente boa'] = 'Sim'
df
```

	Nome	Sobrenome	Idade Atual	É gente boa
0	Adriano	Soares	31	Sim
1	Rodrigo	Tadewald	31	Sim
2	Juliano	Faccioni	25	Sim
3	Mateus	Kienzle	35	Sim
4	Luiza	Cherobini	22	Sim
5	Rodrigo	Vanzelotti	21	Sim
6	Ayres	Júnior	28	Sim
7	Aninha	Tadewald	20	Sim

### Criando colunas a partir de listas

Podemos também criar uma coluna a partir de uma lista, com o cuidado de que essa lista deve ter o mesmo tamanho que o número de linhas do DataFrame.

```
df['Cidade que Nasceu'] = ['Lajeado', 'Porto Alegre', 'Lajeado', 'Porto Alegre', 'Faxinaldo',
                           ↪ 'Porto Alegre', 'Porto Alegre', 'Porto Alegre']
df
```

	Nome	Sobrenome	Idade Atual	É gente boa	Cidade que Nasceu
0	Adriano	Soares	31	Sim	Lajeado
1	Rodrigo	Tadewald	31	Sim	Porto Alegre
2	Juliano	Faccioni	25	Sim	Lajeado
3	Mateus	Kienzle	35	Sim	Porto Alegre
4	Luiza	Cherobini	22	Sim	Faxinaldo
5	Rodrigo	Vanzelotti	21	Sim	Porto Alegre
6	Ayres	Júnior	28	Sim	Porto Alegre
7	Aninha	Tadewald	20	Sim	Porto Alegre

### Criando colunas com o método .insert

O uso de insert não é muito utilizado, mas fica aqui para conhecimento!

```
df.insert(3, 'Signo', ['Leão', 'Áries', 'Peixes', 'Virgem', 'Libra', 'Áries', 'Peixes',  
↪ 'Virgem'], True)  
df
```

	Nome	Sobrenome	Idade Atual	Signo	É gente boa	Cidade que Nasceu
0	Adriano	Soares	31	Leão	Sim	Lajeado
1	Rodrigo	Tadewald	31	Áries	Sim	Porto Alegre
2	Juliano	Faccioni	25	Peixes	Sim	Lajeado
3	Mateus	Kienzle	35	Virgem	Sim	Porto Alegre
4	Luiza	Cherobini	22	Libra	Sim	Faxinaldo
5	Rodrigo	Vanzelotti	21	Áries	Sim	Porto Alegre
6	Ayres	Júnior	28	Peixes	Sim	Porto Alegre
7	Aninha	Tadewald	20	Virgem	Sim	Porto Alegre

### Criando colunas a partir de Series

Uma das formas mais utilizada é a criação de colunas a partir de Series que são derivadas do próprio DataFrame. Por exemplo, podemos criar uma coluna dobro da idade a partir da operação da coluna 'Idade Atual':

```
df['Dobro Idade'] = df['Idade Atual'] * df['Idade Atual']  
df
```

	Nome	Sobrenome	Idade Atual	Signo	É gente boa	Cidade que Nasceu	Dobro Idade
0	Adriano	Soares	31	Leão	Sim	Lajeado	961
1	Rodrigo	Tadewald	31	Áries	Sim	Porto Alegre	961
2	Juliano	Faccioni	25	Peixes	Sim	Lajeado	625
3	Mateus	Kienzle	35	Virgem	Sim	Porto Alegre	1225
4	Luiza	Cherobini	22	Libra	Sim	Faxinaldo	484
5	Rodrigo	Vanzelotti	21	Áries	Sim	Porto Alegre	441
6	Ayres	Júnior	28	Peixes	Sim	Porto Alegre	784
7	Aninha	Tadewald	20	Virgem	Sim	Porto Alegre	400

Em colunas numéricas, podemos fazer todas as operações aritméticas normais e elas serão aplicadas em todas as linhas do DataFrame. Podemos portanto fazer soma, subtração, divisão, multiplicação, exponenciação de forma muito fácil:

```
df['Metade Idade'] = df['Idade Atual'] / 2
df
```

	Nome	Sobrenome	Idade Atual	Signo	É gente boa	Cidade que Nasceu	Dobro Idade	Metade Idade
0	Adriano	Soares	31	Leão	Sim	Lajeado	961	15.5
1	Rodrigo	Tadewald	31	Áries	Sim	Porto Alegre	961	15.5
2	Juliano	Faccioni	25	Peixes	Sim	Lajeado	625	12.5
3	Mateus	Kienzle	35	Virgem	Sim	Porto Alegre	1225	17.5
4	Luiza	Cherobini	22	Libra	Sim	Faxinaldo	484	11.0
5	Rodrigo	Vanzelotti	21	Áries	Sim	Porto Alegre	441	10.5
6	Ayres	Júnior	28	Peixes	Sim	Porto Alegre	784	14.0
7	Aninha	Tadewald	20	Virgem	Sim	Porto Alegre	400	10.0

Em colunas que contém strings, podemos fazer as mesmas operações de string. Como exemplo, fazemos uma concatenação de strings gerando a coluna “Nome Completo”:

```
df['Nome Completo'] = df['Nome'] + ' ' + df['Sobrenome']
df
```

	Nome	Sobrenome	Idade Atual	Signo	É gente boa	Cidade que Nasceu	Dobro Idade	Metade Idade	Nome Completo
0	Adriano	Soares	31	Leão	Sim	Lajeado	961	15.5	Adriano Soares
1	Rodrigo	Tadewald	31	Áries	Sim	Porto Alegre	961	15.5	Rodrigo Tadewald
2	Juliano	Faccioni	25	Peixes	Sim	Lajeado	625	12.5	Juliano Faccioni
3	Mateus	Kienzle	35	Virgem	Sim	Porto Alegre	1225	17.5	Mateus Kienzle
4	Luiza	Cherobini	22	Libra	Sim	Faxinaldo	484	11.0	Luiza Cherobini
5	Rodrigo	Vanzelotti	21	Áries	Sim	Porto Alegre	441	10.5	Rodrigo Vanzelotti
6	Ayres	Júnior	28	Peixes	Sim	Porto Alegre	784	14.0	Ayres Júnior
7	Aninha	Tadewald	20	Virgem	Sim	Porto Alegre	400	10.0	Aninha Tadewald

## 8. Adicionando novas linhas

Vamos mostrar duas formas de adicionar novas linhas em DataFrames. Como sempre, sempre, quando falamos de DataFrames, sempre existem várias formas de fazer uma operação. A que mais utilizo é a última (pd.concat), mas deixo opções para caso você prefira outras

### Com uma lista utilizando .loc

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus', 'Luiza', 'Rodrigo', 'Ayres',
↪ 'Aninha'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle', 'Cherobini', 'Vanzelotti',
↪ 'Júnior', 'Tadewald'],
        'Idade Atual': [31, 31, 25, 35, 22, 21, 28, 20],
        }

df = pd.DataFrame(data)

df.loc[len(df)] = ['Roberta', 'De Souza', 27]
df
```

	Nome	Sobrenome	Idade Atual
0	Adriano	Soares	31
1	Rodrigo	Tadewald	31
2	Juliano	Faccioni	25
3	Mateus	Kienzle	35
4	Luiza	Cherobini	22
5	Rodrigo	Vanzelotti	21
6	Ayres	Júnior	28
7	Aninha	Tadewald	20
8	Roberta	De Souza	27

```
df.loc['Roberta'] = ['Roberta', 'De Souza', 27]
df
```



	Nome	Sobrenome	Idade Atual
0	Adriano	Soares	31
1	Rodrigo	Tadewald	31
2	Juliano	Faccioni	25
3	Mateus	Kienzle	35
4	Luiza	Cherobini	22
5	Rodrigo	Vanzelotti	21
6	Ayres	Júnior	28
7	Aninha	Tadewald	20
8	Roberta	De Souza	27
	Roberta	De Souza	27

Como podemos ver, o parâmetro que passamos no loc é o nome do índice da nova linha inserido. No primeiro exemplo, passamos como índice o len do DataFrame inicial. Como ele possuía 8 valores, o retorno de len(df) foi igual a 8 foi adicionado a linha com o índice 8. Na segunda vez, adicionamos o valor 'Roberta' no loc, portanto foi adicionada uma linha com o índice igual a 'Roberta'.

### Com um DataFrame e utilizando o método pd.concat

```
import pandas as pd

data = {'Nome': ['Adriano', 'Rodrigo', 'Juliano', 'Mateus', 'Luiza', 'Rodrigo', 'Ayres',
↪ 'Aninha'],
        'Sobrenome': ['Soares', 'Tadewald', 'Faccioni', 'Kienzle', 'Cherobini', 'Vanzelotti',
↪ 'Júnior', 'Tadewald'],
        'Idade Atual': [31, 31, 25, 35, 22, 21, 28, 20],
        }

df = pd.DataFrame(data)

data2 = {'Nome': ['Roberta'],
        'Sobrenome': ['De Souza'],
        'Idade Atual': [27],
        }
df2 = pd.DataFrame(data2)

df = pd.concat([df, df2], ignore_index=True)
df
```

	Nome	Sobrenome	Idade Atual
0	Adriano	Soares	31
1	Rodrigo	Tadewald	31
2	Juliano	Faccioni	25
3	Mateus	Kienzle	35
4	Luiza	Cherobini	22
5	Rodrigo	Vanzelotti	21
6	Ayres	Júnior	28
7	Aninha	Tadewald	20
8	Roberta	De Souza	27

## 9. PROJETO 2 - Seleção de colunas e filtro de linhas

Nessa aula desenvolveremos uma nova página no nosso WebApp que permite a visualização dos dados da tabela vendas, com a opção de selecionar as colunas de interesse e filtrar linhas da tabela:

```
from pathlib import Path

import streamlit as st
import pandas as pd

pasta_datasets = Path(__file__).parent.parent / 'datasets'
df_vendas = pd.read_csv(pasta_datasets / 'vendas.csv', decimal=',', sep=';', index_col=0)

colunas = list(df_vendas.columns)
colunas_selecionadas = st.sidebar.multiselect(
    'Selecione as colunas da tabela:',
    colunas,
    colunas,
)

st.sidebar.divider()
col1, col2 = st.sidebar.columns([0.4, 0.6])
coluna_filtro = col1.selectbox('Selecione a coluna', [c for c in colunas if not c in
    ↪ 'id_vendas'])
opcoes_filtro = list(df_vendas[coluna_filtro].unique())
opcoes_filtro_selecionadas = col2.multiselect(
    'Selecione os valores',
    opcoes_filtro,
)

col11, col22 = st.sidebar.columns([0.4, 0.6])
col22.button('Limpar')
if col11.button('Filtrar'):
    ↪ st.dataframe(df_vendas[df_vendas[coluna_filtro].isin(opcoes_filtro_selecionadas)][colunas_selecionadas],
    ↪ height=800)
else:
    st.dataframe(df_vendas[colunas_selecionadas], height=800)
```

E o segundo para adição de linhas ao DataFrame:

```
from pathlib import Path
from datetime import datetime

import streamlit as st
import pandas as pd

pasta_datasets = Path(__file__).parent.parent / 'datasets'
df_vendas = pd.read_csv(pasta_datasets / 'vendas.csv', decimal=',', sep=';', index_col=0,
    ↪ parse_dates=True)
df_filiais = pd.read_csv(pasta_datasets / 'filiais.csv', decimal=',', sep=';')
df_produtos = pd.read_csv(pasta_datasets / 'produtos.csv', decimal=',', sep=';')
```

```
df_filiais['cidade/estado'] = df_filiais['cidade'] + '/' + df_filiais['estado']
filiais = df_filiais['cidade/estado'].to_list()
filial_selecionada = st.sidebar.selectbox(
    'Selecione a filial:',
    filiais)

vendedores = df_filiais.loc[df_filiais['cidade/estado'] == filial_selecionada,
    ↪ 'vendedores'].iloc[0]
vendedores = vendedores.strip('[').replace('"', '').split(', ')
vendedor_selecionada = st.sidebar.selectbox(
    'Selecione o vendedor:',
    vendedores)

produtos = df_produtos['nome'].to_list()
produto_selecionada = st.sidebar.selectbox(
    'Selecione o produto:',
    produtos)

nome_cliente = st.sidebar.text_input(
    'Nome do cliente:')

genero_cliente = st.sidebar.selectbox(
    'Gênero do cliente:',
    ['feminino', 'masculino'])

forma_pagamento = st.sidebar.selectbox(
    'Forma de pagamento',
    ['pix', 'credito', 'boleto'])

if st.sidebar.button('Adicionar nova venda'):
    df_vendas_adicionar = pd.DataFrame([[df_vendas['id_venda'].max()+1,
                                         filial_selecionada.split('/')[0],
                                         vendedor_selecionada,
                                         produto_selecionada,
                                         nome_cliente,
                                         genero_cliente,
                                         forma_pagamento]])

    df_vendas_adicionar.index = [datetime.now()]
    df_vendas_adicionar.index.name = 'data'
    df_vendas_adicionar.columns = df_vendas.columns
    df_vendas = pd.concat([df_vendas, df_vendas_adicionar])
    df_vendas.to_csv(pasta_datasets / 'vendas.csv', decimal=',', sep=';')
    st.success('Venda adicionada!')

st.dataframe(df_vendas, height=800)
```

## 10. PROJETO 3 - Adição de linhas à tabela

Nessa aula desenvolveremos mais uma página do nosso WebApp que permite adicionar novas linhas de dados à tabela:

```
from pathlib import Path
from datetime import datetime

import streamlit as st
import pandas as pd

pasta_datasets = Path(__file__).parent.parent / 'datasets'
df_vendas = pd.read_csv(pasta_datasets / 'vendas.csv', decimal=',', sep=';', index_col=0,
    ↪ parse_dates=True)
df_filiais = pd.read_csv(pasta_datasets / 'filiais.csv', decimal=',', sep=';')
df_produtos = pd.read_csv(pasta_datasets / 'produtos.csv', decimal=',', sep=';')

df_filiais['cidade/estado'] = df_filiais['cidade'] + '/' + df_filiais['estado']
filiais = df_filiais['cidade/estado'].to_list()
filial_selecionada = st.sidebar.selectbox(
    'Selecione a filial:',
    filiais)

vendedores = df_filiais.loc[df_filiais['cidade/estado'] == filial_selecionada,
    ↪ 'vendedores'].iloc[0]
vendedores = vendedores.strip('[]').replace('"', '').split(', ')
vendedor_selecionada = st.sidebar.selectbox(
    'Selecione o vendedor:',
    vendedores)

produtos = df_produtos['nome'].to_list()
produto_selecionada = st.sidebar.selectbox(
    'Selecione o produto:',
    produtos)

nome_cliente = st.sidebar.text_input(
    'Nome do cliente:')

genero_cliente = st.sidebar.selectbox(
    'Gênero do cliente:',
    ['feminino', 'masculino'])

forma_pagamento = st.sidebar.selectbox(
    'Forma de pagamento',
    ['pix', 'credito', 'boleto'])

if st.sidebar.button('Adicionar nova venda'):
    df_vendas_adicionar = pd.DataFrame([[df_vendas['id_venda'].max()+1,
        filial_selecionada.split('/')[0],
```

```
        vendedor_selecionada,
        produto_selecionada,
        nome_cliente,
        genero_cliente,
        forma_pagamento]))

df_vendas_adicionar.index = [datetime.now()]
df_vendas_adicionar.index.name = 'data'
df_vendas_adicionar.columns = df_vendas.columns
df_vendas = pd.concat([df_vendas, df_vendas_adicionar])
df_vendas.to_csv(pasta_datasets / 'vendas.csv', decimal=',', sep=';')
st.success('Venda adicionada!')

st.dataframe(df_vendas, height=800)
```

## 11. Mesclando tabelas (o PROCV no Python)

Uma das principais funções do excel é o PROCV e bastante gente se complica ao utilizá-lo. Ele serve para mesclar duas tabelas, ou seja, para substituir os valores de uma coluna conforma dados de outra tabela. Mostraremos isso melhor com exemplos e ficará claro que não é uma operação complicada em Python!

Mas antes, quero explicar um pouco melhor o nosso desafio. Essas são as duas tabelas de dados que serão necessárias mesclar para continuarmos o desenvolvimento do nosso WebApp:

Tabela de vendas:

	A	B	C	D	E
1	data	id_venda	filial	vendedor	produto
2	2023-01-12 07:42:38.311304	1	Canoas	Luis Fernando	Tenis Nike
3	2023-01-12 08:21:37.427917	2	Florianópolis	Claudia dos Santos	Tenis Adidas
4	2023-01-12 08:27:36.903347	3	São Paulo	Cassia Moraes	Tenis Fila
5	2023-01-12 09:32:38.748455	4	Florianópolis	Mario Sérgio	Tenis Nike
6	2023-01-12 11:01:38.047889	5	Canoas	Luis Fernando	Tenis Fila
7	2023-01-12 14:41:39.235334	6	Canoas	Carlos Henrique	Tenis Adidas
8	2023-01-13 06:40:39.161308	7	Florianópolis	Claudia dos Santos	Tenis Adidas
9	2023-01-13 07:21:39.554138	8	São Paulo	Cassia Moraes	Tenis NB
10	2023-01-13 09:43:37.001444	9	Canoas	Luis Fernando	Tenis Adidas
11	2023-01-13 09:58:39.090014	10	Caxias do Sul	Rodrigo Vanzeloti	Tenis Nike
12	2023-01-13 10:44:38.252917	11	São Paulo	Claudio Bueno	Tenis NB

Tabela de produtos:

	A	B	C	D	E
1		nome	id	preco	
2	0	Tenis Nike	0	300	
3	1	Tenis Adidas	1	450	
4	2	Tenis NB	2	500	
5	3	Tenis Fila	3	250	
6					

Podemos notar que na tabela de vendas, não temos o preço dos produtos e, portanto, não conseguimos calcular o faturamento que tivemos com as nossas vendas.

A unica forma de fazermos isso é mesclando as duas tabelas, de forma que criemos uma coluna nova na tabela de vendas com o preço da venda, a partir do produto que foi vendido. Portanto, é esperado que quando o produto vendido for Tênis Nike, o preço da venda na tabela de vendas seja 300, para um Tênis Adidas seria de 450, e assim por diante!

Esperamos um resultado da seguinte forma:

A	B	C	D	E	F
data	id_venda	filial	vendedor	produto	valor venda
2023-01-12 07:42:38.311304	1	Canoas	Luis Fernando	Tenis Nike	300
2023-01-12 08:21:37.427917	2	Florianópolis	Claudia dos Santos	Tenis Adidas	450
2023-01-12 08:27:36.903347	3	São Paulo	Cassia Moraes	Tenis Fila	250
2023-01-12 09:32:38.748455	4	Florianópolis	Mario Sérgio	Tenis Nike	300
2023-01-12 11:01:38.047889	5	Canoas	Luis Fernando	Tenis Fila	250

E é isso que vamos aprender a fazer agora!

Antes de tudo, vamos importar nossas tabelas:

```
import pandas as pd
```

```
df_vendas = pd.read_csv('datasets/vendas.csv', sep=';', decimal=',')  
df_produtos = pd.read_csv('datasets/produtos.csv', sep=';', decimal=',', index_col=0)
```

## Utilizando o método .merge

Para utilizarmos o .merge, precisamos de uma coluna de referência que sirva como âncora para a mesclagem. No nosso caso, temos que nos guiar pela coluna do nome do produto, pois é a informação comum entre as duas tabelas. É necessário que essa coluna tenha o mesmo nome em ambas as tabelas. Por isso, vamos utilizar o método .rename para modificar o nome da tabela de produtos de 'nome' para 'produto':

```
df_produtos = df_produtos.rename(columns={'nome': 'produto'})  
df_produtos
```

	produto	id	preco
0	Tenis Nike	0	300
1	Tenis Adidas	1	450
2	Tenis NB	2	500
3	Tenis Fila	3	250

Depois vamos selecionar apenas as colunas nossas de interesse, que são 'produto' e 'preço':

```
df_produtos = df_produtos[['produto', 'preco']]  
df_produtos
```

E por último, vamos utilizar o método merge:

```
df_vendas_mesclado = pd.merge(left=df_vendas,  
                               right=df_produtos,  
                               on='produto',  
                               how='left')  
  
df_vendas_mesclado.head(10)
```



	data	id_venda	filial	vendedor	produto	cliente_nome	cliente_genero	forma_pagamento	preco
0	2023-01-12 07:42:38.311304	1	Canoas	Luis Fernando	Tenis Nike	Anthony Moon	masculino	credito	300
1	2023-01-12 08:21:37.427917	2	Florianópolis	Claudia dos Santos	Tenis Adidas	Bud Loughlin	masculino	pix	450
2	2023-01-12 08:27:36.903347	3	São Paulo	Cassia Moraes	Tenis Fila	Gonzalo Kroes	masculino	credito	250
3	2023-01-12 09:32:38.748455	4	Florianópolis	Mario Sérgio	Tenis Nike	Lisa Thomas	feminino	credito	300
4	2023-01-12 11:01:38.047889	5	Canoas	Luis Fernando	Tenis Fila	Charles Bowlin	masculino	credito	250
5	2023-01-12 14:41:39.235334	6	Canoas	Carlos Henrique	Tenis Adidas	Shari Craig	feminino	credito	450
6	2023-01-13 06:40:39.161308	7	Florianópolis	Claudia dos Santos	Tenis Adidas	Frank Felker	masculino	credito	450
7	2023-01-13 07:21:39.554138	8	São Paulo	Cassia Moraes	Tenis NB	Eric Ryan	masculino	credito	500
8	2023-01-13 09:43:37.001444	9	Canoas	Luis Fernando	Tenis Adidas	Myron Wallen	masculino	credito	450
9	2023-01-13 09:58:39.090014	10	Caxias do Sul	Rodrigo Vanzeloti	Tenis Nike	Dawn Cobb	feminino	credito	300

Explicando brevemente os parâmetros:

- `left`: é a tabela que vamos querer como resultado, é nela que estamos adicionando dados novos (no nosso caso é a tabela de vendas)
- `right`: é a tabela usada para capturar os dados que serão adicionados (no nosso caso, tabela de produtos para pegar o preço)
- `on`: a coluna que deve ter em comum entre as duas tabelas e será utilizada como referência na transferência de dados de uma tabela para outra
- `how`: é o modo de transferência. Quando utilizado o parâmetro 'left' ele sempre mantém no final todos os dados da tabela principal (mesmo que valor compatíveis não tenham sido encontrados na tabela right)

## Utilizando o método .map

Para utilizarmos o map, precisamos antes de um dicionário cujas keys são o nome dos produtos e os valores os preços. Podemos obter isso da seguinte forma:

```
dict_produtos = df_produtos.set_index('produto').to_dict()['preco']
dict_produtos
```

```
{'Tenis Nike': 300, 'Tenis Adidas': 450, 'Tenis NB': 500, 'Tenis Fila': 250}
```

Dado que temos o dicionário, podemos utilizar o método map:

```
df_vendas['preco_venda'] = df_vendas['produto'].map(dict_produtos)
df_vendas.head(10)
```

	data	id_venda	filial	vendedor	produto	cliente_nome	cliente_genero	forma_pagamento	preco_venda
0	2023-01-12 07:42:38.311304	1	Canoas	Luis Fernando	Tenis Nike	Anthony Moon	masculino	credito	300
1	2023-01-12 08:21:37.427917	2	Florianópolis	Claudia dos Santos	Tenis Adidas	Bud Loughlin	masculino	pix	450
2	2023-01-12 08:27:36.903347	3	São Paulo	Cassia Moraes	Tenis Fila	Gonzalo Kroes	masculino	credito	250
3	2023-01-12 09:32:38.748455	4	Florianópolis	Mario Sérgio	Tenis Nike	Lisa Thomas	feminino	credito	300
4	2023-01-12 11:01:38.047889	5	Canoas	Luis Fernando	Tenis Fila	Charles Bowlin	masculino	credito	250
5	2023-01-12 14:41:39.235334	6	Canoas	Carlos Henrique	Tenis Adidas	Shari Craig	feminino	credito	450
6	2023-01-13 06:40:39.161308	7	Florianópolis	Claudia dos Santos	Tenis Adidas	Frank Felker	masculino	credito	450
7	2023-01-13 07:21:39.554138	8	São Paulo	Cassia Moraes	Tenis NB	Eric Ryan	masculino	credito	500
8	2023-01-13 09:43:37.001444	9	Canoas	Luis Fernando	Tenis Adidas	Myron Wallen	masculino	credito	450

## **Utilizando o método .replace**

## 12. Realizando cálculos na tabela

A ideia dessa aula é mostrar como é simples realizar análises e cálculos simples em tabelas DataFrame.

Vamos começar com análises simples da estrutura dos dados e depois vamos para as operações básicas.

Para isso, vamos utilizar nossa tabela de vendas já com os dados de preço do produto vendido adicionado:

```
import pandas as pd

df_vendas = pd.read_csv('datasets/vendas.csv', sep=';', decimal=',')
df_produtos = pd.read_csv('datasets/produtos.csv', sep=';', decimal=',')

df_produtos = df_produtos.rename(columns={'nome': 'produto'})
df_produtos = df_produtos[['produto', 'preco']]
df_vendas = pd.merge(left=df_vendas,
                     right=df_produtos,
                     on='produto',
                     how='left')
```

	data	id_venda	filial	vendedor	produto	cliente_nome	cliente_genero	forma_pagamento	preco
0	2023-01-12 07:42:38.311304	1	Canoas	Luis Fernando	Tenis Nike	Anthony Moon	masculino	credito	300
1	2023-01-12 08:21:37.427917	2	Florianópolis	Claudia dos Santos	Tenis Adidas	Bud Loughlin	masculino	pix	450
2	2023-01-12 08:27:36.903347	3	São Paulo	Cassia Moraes	Tenis Fila	Gonzalo Kroes	masculino	credito	250
3	2023-01-12 09:32:38.748455	4	Florianópolis	Mario Sérgio	Tenis Nike	Lisa Thomas	feminino	credito	300
4	2023-01-12 11:01:38.047889	5	Canoas	Luis Fernando	Tenis Fila	Charles Bowlin	masculino	credito	250
5	2023-01-12 14:41:39.235334	6	Canoas	Carlos Henrique	Tenis Adidas	Shari Craig	feminino	credito	450
6	2023-01-13 06:40:39.161308	7	Florianópolis	Claudia dos Santos	Tenis Adidas	Frank Felker	masculino	credito	450
7	2023-01-13 07:21:39.554138	8	São Paulo	Cassia Moraes	Tenis NB	Eric Ryan	masculino	credito	500
8	2023-01-13 09:43:37.001444	9	Canoas	Luis Fernando	Tenis Adidas	Myron Wallen	masculino	credito	450
9	2023-01-13 09:58:39.090014	10	Caxias do Sul	Rodrigo Vanzeloti	Tenis Nike	Dawn Cobb	feminino	credito	300

### Vendo as características gerais dos dados

Podemos utilizar o método `.describe` para ver as principais características estatísticas das colunas que possuem valor numérico:

```
df_vendas.describe()
```

	id_venda	preco
count	2000.000000	2000.000000
mean	1000.500000	374.500000
std	577.494589	103.247664
min	1.000000	250.000000
25%	500.750000	250.000000
50%	1000.500000	300.000000
75%	1500.250000	450.000000
max	2000.000000	500.000000

Nesse caso, como só temos duas colunas numéricas, um DataFrame com as características terá duas colunas.

Nele podemos ver estatísticas como: - count: número de aparições - mean: média dos valores na coluna - std: desvio padrão dos valores na coluna - min: valor mínimo na coluna - max: valor máximo na coluna - percentis: 25%, 50% e 75% para entendermos como esses valores estão distribuídos

Podemos também utilizar o método .info para entendermos os tipos dos dados que temos na nossa tabela:

```
df_vendas.info()
```

```
.. <class 'pandas.core.frame.DataFrame'>
   RangeIndex: 2000 entries, 0 to 1999
   Data columns (total 9 columns):
    #   Column                Non-Null Count  Dtype
   ---  -
    0   data                   2000 non-null  object
    1   id_venda               2000 non-null  int64
    2   filial                 2000 non-null  object
    3   vendedor               2000 non-null  object
    4   produto                2000 non-null  object
    5   cliente_nome           2000 non-null  object
    6   cliente_genero         2000 non-null  object
    7   forma_pagamento       2000 non-null  object
    8   preco                  2000 non-null  int64
   dtypes: int64(2), object(7)
   memory usage: 140.8+ KB
```

Ele confirma que só temos duas colunas com dados numéricos e as outras estão sendo vistas como objetos pelo pandas.

## Contando valores únicos

Muitas vezes gostaríamos de saber quantos valores diferentes temos em uma coluna da nossa tabela e a proporção que esse valor aparece. Podemos utilizar dois métodos para isso:

```
df_vendas['filial'].value_counts()
```

```
filial
Florianópolis    356
Canoas           354
Caxias do Sul    333
São Paulo        325
Lajeado          318
Porto Alegre     314
Name: count, dtype: int64
```

Aqui já podemos ver facilmente quantas vendas cada filial fez.

Se queremos apenas os valores únicos, podemos utilizar também o unique:

```
df_vendas['filial'].unique()
```

```
array(['Canoas', 'Florianópolis', 'São Paulo', 'Caxias do Sul', 'Lajeado',
       'Porto Alegre'], dtype=object)
```

## Operações por toda a tabela e por coluna

Temos diversos métodos para realizar outras análises de dados bem fáceis de serem aplicadas, como:

```
df_vendas.sum(axis=0)
```

```
data          2023-01-12 07:42:38.3113042023-01-12 08:21:37....
id_venda                                2001000
filial      CanoasFlorianópolisSão PauloFlorianópolisCanoa...
vendedor    Luis FernandoClaudia dos SantosCassia MoraesMa...
produto     Tennis NikeTennis AdidasTennis FilaTennis NikeTeni...
cliente_nome Anthony MoonBud LoughlinGonzalo KroesLisa Thom...
cliente_genero masculinomasculinomasculinofemininomasculinofe...
forma_pagamento creditopixcreditocreditocreditocreditocreditoc...
preco                                749000
dtype: object
```

Realiza a soma de todos os valores das colunas. No caso de colunas que contêm string, é feita a concatenação dessas strings.

Essa mesma operação pode ser feita para uma coluna desejada:

```
df_vendas['preco'].sum()
```



749000

Além disso, temos diversas outras como:

média

```
df_vendas['preco'].mean()
```

mediana

```
df_vendas['preco'].median()
```

max

```
df_vendas['preco'].max()
```

min

```
df_vendas['preco'].min()
```

count

```
df_vendas['preco'].count()
```

## 13. PROJETO 4 - Visualizando volume de vendas e comissões

Nessa aula desenvolveremos uma página do nosso WebApp que permite analisar o volume de vendas da nossa empresa e analisar as principais filiais e vendedores.

```
from pathlib import Path
from datetime import date, timedelta

import streamlit as st
import pandas as pd

PERCENTUAL_COMISSAO = 0.08

pasta_datasets = Path(__file__).parent.parent / 'datasets'
df_vendas = pd.read_csv(pasta_datasets / 'vendas.csv', decimal=',', sep=';', index_col=0,
    ↪ parse_dates=True)
df_filiais = pd.read_csv(pasta_datasets / 'filiais.csv', decimal=',', sep=';', index_col=0)
df_produtos = pd.read_csv(pasta_datasets / 'produtos.csv', decimal=',', sep=';', index_col=0)

df_produtos = df_produtos.rename(columns={'nome': 'produto'})
df_vendas = pd.merge(df_vendas.reset_index(), df_produtos[['produto', 'preco']],
    ↪ on='produto').set_index('data')
df_vendas['comissao'] = df_vendas['preco'] * PERCENTUAL_COMISSAO

st.markdown('# Números Gerais')
col11, col12 = st.columns(2)
data_inicio = st.sidebar.date_input("Data Inicial", date.today() - timedelta(days=7))
data_final = st.sidebar.date_input("Data Final", date.today() - timedelta(days=1))

df_vendas_corte = df_vendas[(df_vendas.index.date >= data_inicio) & (df_vendas.index.date <=
    ↪ data_final + timedelta(days=1))]
valor_vendas = f"R$ {df_vendas_corte['preco'].sum():.2f}"
col11.metric(label="Valor de vendas no período", value=valor_vendas)
col12.metric(label="Quantidade itens vendidos", value=df_vendas_corte['preco'].count())

if data_inicio <= data_final:
    st.divider()
    principal_filial = df_vendas_corte['filial'].value_counts().sort_values().index[-1]
    st.markdown(f'# Principal Filial: {principal_filial}')
    col21, col22 = st.columns(2)
    valor_vendas = f"R$ {df_vendas_corte[df_vendas_corte['filial'] ==
    ↪ principal_filial]['preco'].sum():.2f}"
    col21.metric(label=f"Valor de vendas de {principal_filial}", value=valor_vendas)
    col22.metric(label="Quantidade itens vendidos",
    ↪ value=df_vendas_corte[df_vendas_corte['filial'] == principal_filial]['preco'].count())

    st.divider()
    principal_vendedor = df_vendas_corte['vendedor'].value_counts().sort_values().index[-1]

    st.markdown(f'# Principal Vendedor: {principal_vendedor}')
    col31, col32 = st.columns(2)
    valor_vendas = f"R$ {df_vendas_corte[df_vendas_corte['vendedor'] ==
    ↪ principal_vendedor]['preco'].sum():.2f}"
```

```
col31.metric(label=f"Valor de vendas de {principal_vendedor}", value=valor_vendas)

valor_comissao = f"R$ {df_vendas_corte[df_vendas_corte['vendedor'] ==
↪ principal_vendedor]['comissao'].sum():.2f}"
col32.metric(label=f"Comissão", value=valor_comissao)
```



## 14. Pivotando e agrupando tabelas (as TABELAS DINÂMICAS)

Para conseguir explicar esse tema, preciso que tenhamos um objetivo claro em mente. Digamos que estamos com as seguintes questões em mente: Para conseguir explicar esse tema, preciso que tenhamos um objetivo claro em mente. Digamos que estamos com as seguintes questões em mente:

- Quantas vendas fizemos por filial de cada produto?
- Quanto cada vendedor representou das vendas de cada filial?
- O gênero do cliente influencia na forma de pagamento dele?

Essas perguntas podem ser facilmente respondidas ao reagruparmos nossos dados de uma forma que as respostas fiquem mais evidentes. Por exemplo, para primeira questão seria melhor se tivéssemos uma tabela cujo índice fosse as filiais, as colunas os produtos e os valores fosse as quantidades de vendas.

Todas informações já estão presentes no nosso dado original, e para reorganizá-lo dessa forma que queremos, basta utilizarmos o método `.pivot_table()`!

### Utilizando o método `pivot_table`

```
df_vendas.pivot_table(values='preco',  
                        index='filial',  
                        columns='produto',  
                        aggfunc='count')
```

produto	Tenis Adidas	Tenis Fila	Tenis NB	Tenis Nike
filial				
Canoas	88	81	93	92
Caxias do Sul	95	85	73	80
Florianópolis	86	89	101	80
Lajeado	77	78	85	78
Porto Alegre	74	84	73	83
São Paulo	77	90	74	84

E já conseguimos responder nossa pergunta.

O método toma os seguintes argumentos: - `values`: o valor a ser analisado. Precisa ser uma string com o nome de uma das colunas - `index`: é a coluna que será usada como índice do DataFrame final. Pode ser uma string com o nome da coluna ou uma lista de strings com nomes de colunas - `columns`:

é a coluna que será usada como a coluna do DataFrame final. Pode ser uma string com o nome da coluna ou uma lista de strings com nomes de colunas - *aggfunc*: é o cálculo que será realizado na coluna *values*. Pode tomar diversos valores como: - *mean*: para média - *sum*: para somar - *median*: para a mediana - *max*: para obter o valor máximo - *min*: para obter o valor mínimo - *std*: para obter o desvio padrão - *count*: para contar a quantidade de valores

### Utilizando o método `groupby`

Outra forma de realizar análises similares é utilizando o método `groupby`:

```
df_vendas.groupby(['filial', 'produto'])[['preco']].count()
```

		preço
filial	produto	
Canoas	Tenis Adidas	88
	Tenis Fila	81
	Tenis NB	93
	Tenis Nike	92
Caxias do Sul	Tenis Adidas	95
	Tenis Fila	85
	Tenis NB	73
	Tenis Nike	80
Florianópolis	Tenis Adidas	86
	Tenis Fila	89
	Tenis NB	101
	Tenis Nike	80
Lajeado	Tenis Adidas	77
	Tenis Fila	78
	Tenis NB	85
	Tenis Nike	78
Porto Alegre	Tenis Adidas	74
	Tenis Fila	84
	Tenis NB	73
	Tenis Nike	83
São Paulo	Tenis Adidas	77
	Tenis Fila	90
	Tenis NB	74
	Tenis Nike	84

## 15. PROJETO 5 - Tabela dinâmica

Nessa aula criaremos uma tabela dinâmica no nosso webApp.

```
from pathlib import Path

import streamlit as st
import pandas as pd

PERCENTUAL_COMISSAO = 0.08

pasta_datasets = Path(__file__).parent.parent / 'datasets'
df_vendas = pd.read_csv(pasta_datasets / 'vendas.csv', decimal=',', sep=';', index_col=0,
    ↪ parse_dates=True)
df_filiais = pd.read_csv(pasta_datasets / 'filiais.csv', decimal=',', sep=';', index_col=0)
df_produtos = pd.read_csv(pasta_datasets / 'produtos.csv', decimal=',', sep=';', index_col=0)

df_produtos = df_produtos.rename(columns={'nome': 'produto'})
df_vendas = pd.merge(df_vendas.reset_index(), df_produtos[['produto', 'preco']],
    ↪ on='produto').set_index('data')
df_vendas['comissao'] = df_vendas['preco'] * PERCENTUAL_COMISSAO
df_vendas['dia_da_semana'] = df_vendas.index.dayofweek
df_vendas['dia_da_semana'] = df_vendas['dia_da_semana'].map({0: 'segunda-feira',
    1: 'terça-feira',
    2: 'quarta-feira',
    3: 'quinta-feira',
    4: 'sexta-feira',
    5: 'sábado',
    6: 'domingo'})

colunas_analise = ['filial', 'vendedor', 'produto', 'dia_da_semana']
colunas_valor = ['preco', 'comissao']
funcoes_agg = {'soma': 'sum', 'contagem': 'count'}

index_selecionado = st.sidebar.multiselect('Selecione os índices:',
    colunas_analise)
coluna_selecionado = st.sidebar.multiselect('Selecione as colunas:',
    [c for c in colunas_analise if not c in index_selecionado])
valor_selecionado = st.sidebar.selectbox('Selecione o valor de análise:',
    colunas_valor)
metrica_selecionada = st.sidebar.selectbox('Selecione a métrica:',
    funcoes_agg.keys())

if len(index_selecionado) > 0 and len(coluna_selecionado) > 0:
    metrica_selecionada = funcoes_agg[metrica_selecionada]
    vendas_pivotada = pd.pivot_table(df_vendas, index=index_selecionado,
    ↪ columns=coluna_selecionado, values=valor_selecionado, aggfunc=metrica_selecionada)
    vendas_pivotada['TOTAL GERAL'] = vendas_pivotada.sum(axis=1)
    vendas_pivotada.loc['TOTAL GERAL', :] = vendas_pivotada.sum(axis=0).to_list()

    st.dataframe(vendas_pivotada)
```

## 16. Criando gráficos com plotly

Chegando a parte final do nosso curso, falta apenas uma funcionalidade para tornarmos nosso WebApp uma ferramenta útil: adicionar gráfico. Com isso, estaremos prontos para substituir nossas antigas planilhas de Excel por belos WebApps em Python!

Para isso, utilizaremos uma biblioteca muito poderosa chamada **Plotly**.

### Introdução a Plotly

Plotly é uma biblioteca para visualização e compreensão de dados de forma simples e fácil. Suporta vários tipos de gráficos, como gráficos de linhas, gráficos de dispersão, histogramas, gráficos de cox, etc.

Existem diversas ferramentas em Python para geração de gráficos, então acho importante ressaltar alguns fatores importantes que nos levaram a utilizar Plotly neste curso:

- É visualmente atraente e pode ser aceito por uma ampla gama de públicos.
- Permite-nos uma personalização infinita dos nossos gráficos, o que torna o nosso gráfico mais significativo e compreensível para os outros.
- Streamlit possui integração nativa com Plotly, sendo simples a adição de gráficos aos nossos WebApps.

### Criando gráficos com Plotly

Nesse curso, iremos explorar os três tipos principais de gráficos: gráficos de linha, de barras e de pizza.

Antes de qualquer coisa, vamos importar nossos dados e adicionar a coluna de preço, como já fizemos nas outras aulas.

```
import pandas as pd

df_vendas = pd.read_csv('datasets/vendas.csv', sep=';', decimal=',', index_col=0,
    ↪ parse_dates=True)
df_produtos = pd.read_csv('datasets/produtos.csv', sep=';', decimal=',', index_col=0)

df_produtos = df_produtos.rename(columns={'nome': 'produto'})
df_produtos = df_produtos[['produto', 'preco']]
df_vendas = pd.merge(left=df_vendas.reset_index(),
    right=df_produtos,
    on='produto',
    how='left')
df_vendas = df_vendas.set_index('data')
df_vendas.head(10)
```

## Criando gráficos de linha

Criaremos um gráfico de linha para saber quanto vendemos por dia e, para isso, precisaremos colocar o dia da venda no eixo x do gráfico e o valor total de vendas no dia no eixo y. Primeira coisa que precisamos é criar primeiro precisamos criar uma tabela com o valor total de vendas por dia

Começamos criando uma coluna com os dias sem a hora da venda:

```
df_vendas['dia_venda'] = df_vendas.index.date
df_vendas.head()
```

	id_venda	filial	vendedor	produto	cliente_nome	cliente_genero	forma_pagamento	preco	dia_venda
data									
2023-01-12 07:42:38.311304	1	Canoas	Luis Fernando	Tenis Nike	Anthony Moon	masculino	credito	300	2023-01-12
2023-01-12 08:21:37.427917	2	Florianópolis	Claudia dos Santos	Tenis Adidas	Bud Loughlin	masculino	pix	450	2023-01-12
2023-01-12 08:27:36.903347	3	São Paulo	Cassia Moraes	Tenis Fila	Gonzalo Kroes	masculino	credito	250	2023-01-12
2023-01-12 09:32:38.748455	4	Florianópolis	Mario Sérgio	Tenis Nike	Lisa Thomas	feminino	credito	300	2023-01-12
2023-01-12 11:01:38.047889	5	Canoas	Luis Fernando	Tenis Fila	Charles Bowlin	masculino	credito	250	2023-01-12

A coluna dia\_venda foi criada com sucesso!

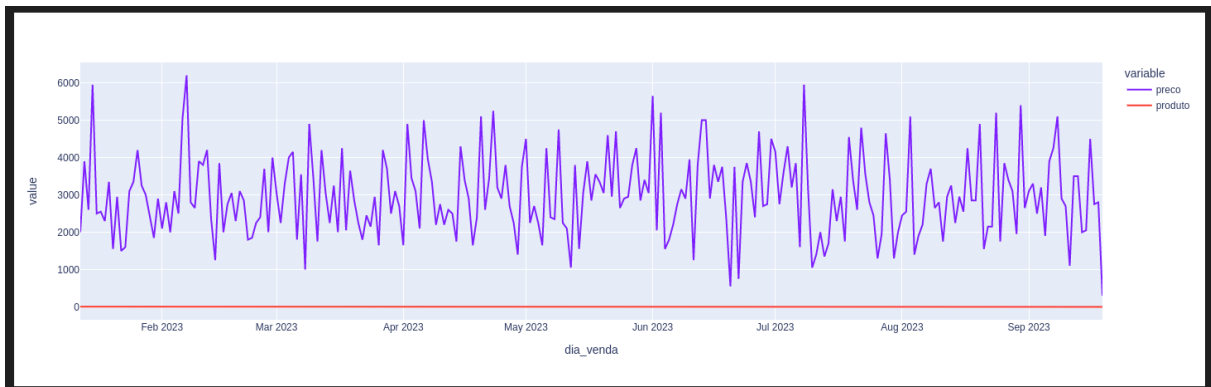
Agora vamos criar um agrupamento de dados, como visto anteriormente. Nele teremos tanto a contagem de vendas quanto o valor financeiro total por dia.

```
df_vendas_dia = df_vendas.groupby('dia_venda').agg({'preco': 'sum', 'produto': 'count'})
df_vendas_dia.head()
```

	preco	produto
dia_venda		
2023-01-12	2000	6
2023-01-13	3900	9
2023-01-14	2600	8
2023-01-15	5950	15
2023-01-16	2500	7

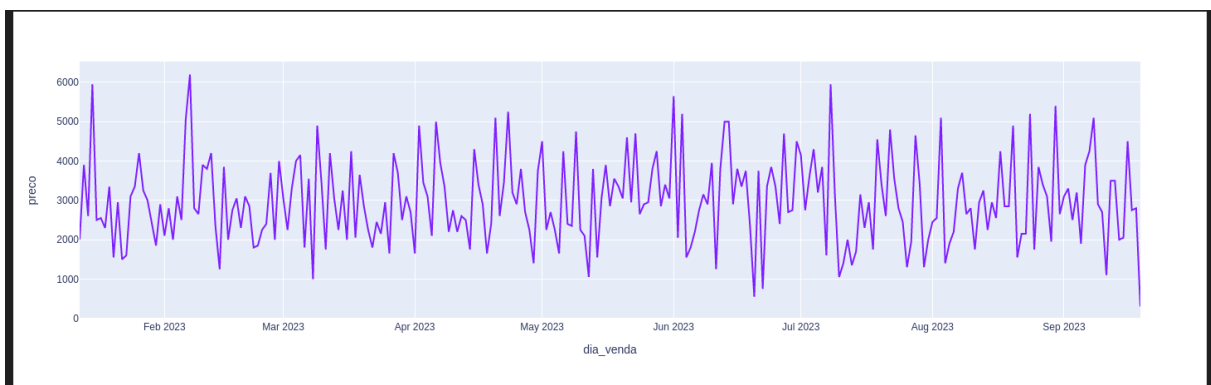
Podemos notar que na coluna 'preco' temos o valor financeiro total de vendas no dia e na coluna 'produto' temos a quantidade de produtos vendidos no dia. Para plotar, basta utilizar as seguintes linhas:

```
px.line(df_vendas_dia)
```



Se quiséssemos plotar apenas a coluna 'preco', podemos fazer o seguinte:

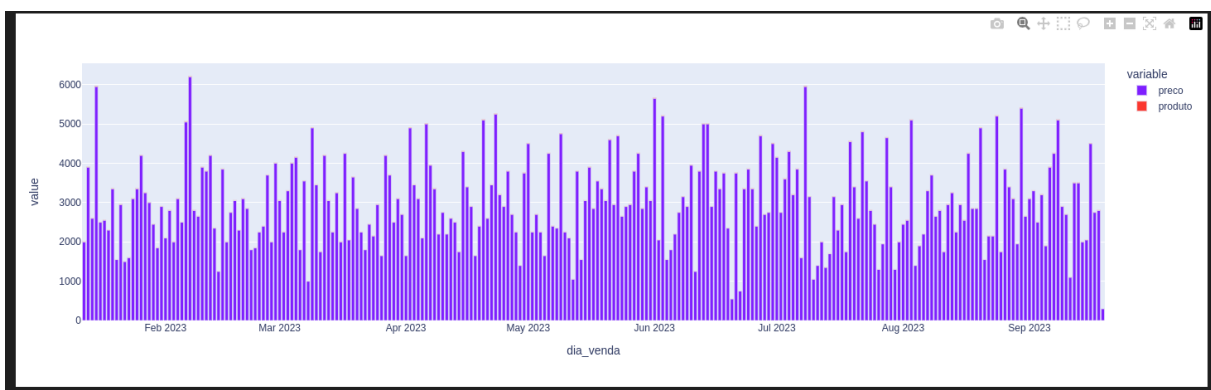
```
px.line(df_vendas_dia, y='preco')
```



### Criando gráficos de barra

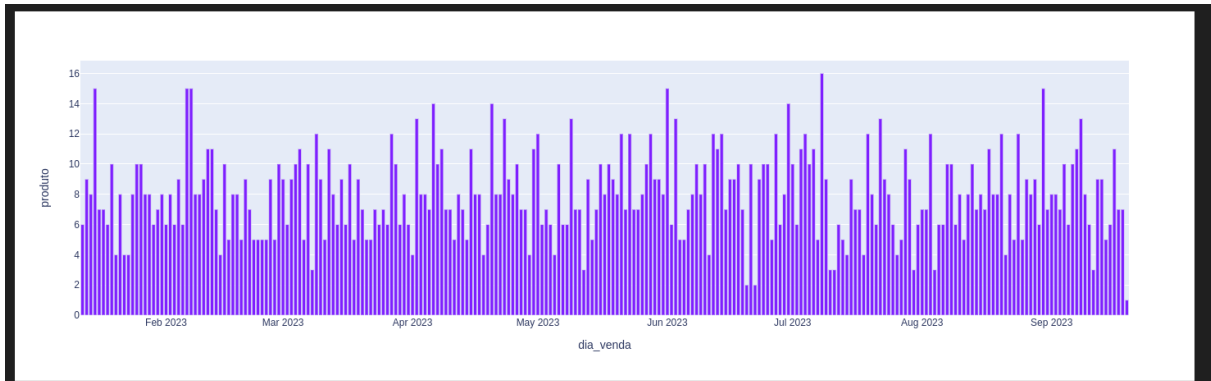
O funcionamento do gráfico de barras é bem similar ao de linhas.

```
px.bar(df_vendas_dia)
```



Para plotar apenas uma coluna, o funcionamento é o mesmo.

```
px.bar(df_vendas_dia, y='produto')
```



### Criando gráficos de pizza

Para criar um gráfico de pizza é muito simples. Basta passar a variável que queremos analisar (no caso o tipo do produto) e a variável que será utilizada para análise (no caso o volume financeiro das vendas 'preco'):

```
px.pie(df_vendas, names='produto', values='preco')
```





## 17. PROJETO 6 - Adicionando gráficos

Nessa aula adicionaremos uma página de gráficos no nosso webApp.

```
from pathlib import Path
from datetime import date, timedelta

import streamlit as st
import pandas as pd
import plotly.express as px

PERCENTUAL_COMISSAO = 0.08

st.set_page_config(layout="wide")

pasta_datasets = Path(__file__).parent.parent / 'datasets'
df_vendas = pd.read_csv(pasta_datasets / 'vendas.csv', decimal=',', sep=';', index_col=0,
    ↪ parse_dates=True)
df_filiais = pd.read_csv(pasta_datasets / 'filiais.csv', decimal=',', sep=';', index_col=0)
df_produtos = pd.read_csv(pasta_datasets / 'produtos.csv', decimal=',', sep=';', index_col=0)

df_produtos = df_produtos.rename(columns={'nome': 'produto'})
df_vendas = pd.merge(df_vendas.reset_index(), df_produtos[['produto', 'preco']],
    ↪ on='produto').set_index('data')
df_vendas['comissao'] = df_vendas['preco'] * PERCENTUAL_COMISSAO

col11, col12, col13 = st.columns([0.5, 0.25, 0.25])
col11.markdown('# Números Gerais')
data_inicio = st.sidebar.date_input("Data Inicial", date.today() - timedelta(days=7))
data_final = st.sidebar.date_input("Data Final", date.today() - timedelta(days=1))

df_vendas_corte = df_vendas[(df_vendas.index.date >= data_inicio) & (df_vendas.index.date <=
    ↪ data_final + timedelta(days=1))]
valor_vendas = f"R$ {df_vendas_corte['preco'].sum():.2f}"

col12.metric(label="Valor de vendas no período", value=valor_vendas)
col13.metric(label="Quantidade itens vendidos", value=df_vendas_corte['preco'].count())

if data_inicio <= data_final:
    st.divider()
    col21, col22 = st.columns(2)

    vendas_por_dia = df_vendas_corte.groupby(df_vendas_corte.index.date)['preco'].sum()
    vendas_por_dia.name = 'Valor Vendas'
    fig = px.line(vendas_por_dia)
    col21.plotly_chart(fig)

    analise_selecionada = st.sidebar.selectbox('Analisar:',
```

```
        ['Filial', 'Vendedor', 'Produto'])

analise_selecionada = analise_selecionada.lower()
vendas_por_filial =
↪ df_vendas_corte.groupby(analise_selecionada)['preco'].sum().to_frame().reset_index()
    fig = px.pie(vendas_por_filial, values='preco', names=analise_selecionada)
    col22.plotly_chart(fig)

st.divider()
```