

NÃO PODE FALTAR

INTRODUÇÃO A BIBLIOTECA PANDAS

Vanessa Cadan Scheffer

ESTRUTURA DE DADOS

Pandas é um pacote Python que fornece estruturas de dados projetadas para facilitar o trabalho com dados estruturados (tabelas) e de séries temporais.



Fonte: Shutterstock.

Deseja ouvir este material?

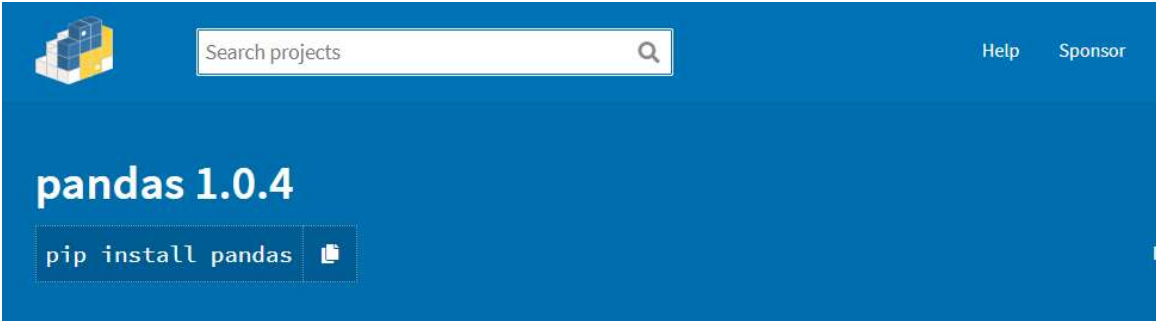
Áudio disponível no material digital.

INTRODUÇÃO A BIBLIOTECA PANDAS

Dentre as diversas bibliotecas disponíveis no repositório PyPI, pandas é uma biblioteca de Python que fornece estruturas de dados projetadas para facilitar o trabalho com dados estruturados (tabelas) e de séries temporais (<https://pandas.py>

versão 1.0.4 e teve sua última atualização disponibilizada no dia 29 de 2020 (Figura 4.1).

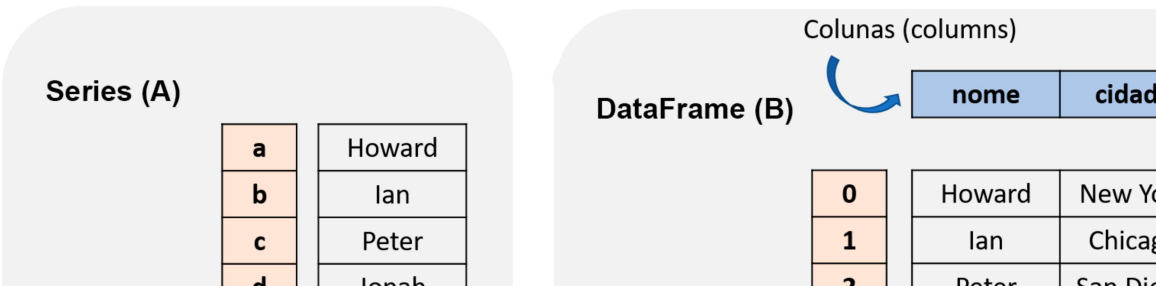
Figura 4.1 - Instalação do pandas



Fonte: Pypi. Disponível em: <https://pypi.org/project/pandas/>.

Como uma ferramenta de alto nível, pandas possui duas estruturas de dados. São as principais para a análise/manipulação de dados: a Series e o DataFrame. Uma **Series** é um vetor de dados (unidimensional), capaz de armazenar diferentes tipos de dados. Um **DataFrame** é um conjunto de Series, ou como a documentação apresenta, um contêiner para Series. Ambas estruturas possuem, como grande característica, a indexação das linhas, ou seja, cada linha possui um rótulo (nome) que o identifica, o qual pode ser uma string, uma inteiro ou uma data. A Figura 4.2 ilustra uma Series (A) e um DataFrame (B). A Series possui somente "uma coluna" de informação e seus rótulos (índices). O DataFrame pode ter uma ou mais colunas e além dos índices, também possui um rótulo de identificação com o nome da coluna. Podemos comparar um DataFrame como uma planilha eletrônica, como o Excel (da Microsoft) ou o Calc (da LibreOffice).

Figura 4.2 - Series (A) e DataFrame (B)



trabalhar com fontes externas de dados. Após criarmos as estruturas vamos ver como extrair informações estatísticas básicas, bem como informações gerais da estrutura e selecionar colunas específicas.

DICA

Na Internet você pode encontrar diversas "cheat sheet" (folha de dicas) sobre a biblioteca pandas. Recomendamos a cheat sheet oficial da biblioteca, disponível no endereço https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

Vamos importar a biblioteca antes de começar nossa primeira linha de código. Por convenção, a biblioteca é importada com o apelido (as) `pd`. Logo, para utilizar as funcionalidades, vamos utilizar a sintaxe `pd.funcionalidade`.

In [1]:

```
import pandas as pd
```

SERIES

Para construir um objeto do tipo `Series`, precisamos utilizar o método `Series` do pacote `pandas`. O método possui o seguinte construtor: `pandas.Series(index=None, dtype=None, name=None, copy=False, fastpath=False)`. Veja que os parâmetros possuem valores padrões (default) o que permite instanciar o objeto de diferentes formas. Para entender cada parâmetro, a melhor fonte de informações é a documentação oficial: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html>.

Dentre todos os parâmetros esperados, somente um é obrigatório para criar uma `Series` com dados (se for uma `Series` sem dados, nenhum parâmetro é obrigatório), o parâmetro `data=xxxx`. Esse parâmetro pode receber, um valor (inteiro, string, float), uma lista de valores, ou um dicionário, veja alguns exemplos.

```
In [2]: pd.Series(data=5) # Cria uma Series com o valor a

Out[2]: 0    5
dtype: int64

In [3]: lista_nomes = 'Howard Ian Peter Jonah Kellie'.split()

pd.Series(lista_nomes) # Cria uma Series com uma lista d

Out[3]: 0    Howard
1      Ian
2    Peter
3    Jonah
4    Kellie
dtype: object

In [4]: dados = {
    'nome1': 'Howard',
    'nome2': 'Ian',
    'nome3': 'Peter',
    'nome4': 'Jonah',
    'nome5': 'Kellie',
}

pd.Series(dados) # Cria uma Series com um dicionário

Out[4]: nome1    Howard
nome2      Ian
nome3    Peter
nome4    Jonah
nome5    Kellie
dtype: object
```

Na entrada 2, criamos uma Series com um único valor, veja que apareceu o índice 0 e 5 como valor. Quando não deixamos explícito os rótulos (índices), o pandas queremos usar é construído um range de 0 até N-1, onde N é a quantidade de valores. Outro detalhe interessante é o dtype (data type), que foi identificado como int64, nesse caso, já que temos somente um valor inteiro no objeto.

Na entrada 3, criamos uma Series a partir de uma lista de nomes, veja que os índices variam de 0 até 4 e o dtype é "object". Esse tipo de dado é usado para representar texto ou valores numéricos e não numéricos combinados.

Na entrada 4, criamos uma Series a partir de um dicionário, a grande diferença aqui é que o índice não é um range, mas sim os nomes das chaves do dicionário.

```
555.555.555-55    Kellie  
dtype: object
```

Rotular as Series (e como veremos os DataFrames), é interessante para a localização e manipulação dos dados. Por exemplo, se quiséssemos saber o nome da pessoa com cpf 111.111.111-11, poderíamos localizar facilmente essa informação com o atributo **loc**, usando a seguinte sintaxe: `series_dados.loc[rotulo]`, onde rótulo é índice a ser localizado. Veja o código a seguir na entrada 6, criamos uma Series com a lista de nomes e guardamos em uma variável chamada `series_dados`. Na linha 3, com o atributo `loc`, localizamos a informação com índice '111.111.111-11'. Veremos mais sobre essa questão de filtrar informações, ao longo das aulas.

```
In [6]: series_dados = pd.Series(lista_nomes, index=cpfs)  
        series_dados.loc['111.111.111-11']  
  
Out[6]: 'Howard'
```

EXTRAINDO INFORMAÇÕES DE UMA SERIES

Já sabemos que estruturas de dados são utilizadas para armazenar dados. Como diferentes estruturas possuem diferentes atributos e métodos. Com a estrutura de dados do pandas não é diferente, tais objetos possuem atributos e métodos específicos, vamos conhecer alguns. Na entrada 7, criamos uma série com valores numéricos e um valor nulo (None). As informações extraídas das linhas 1 a 5 são mais com relação a "forma" dos dados, portanto poderiam ser usadas independentemente do tipo de dado armazenado na Series, inclusive em uma Series com dados com diferentes tipos. Já as informações das linhas 9 a 15, como as funções matemáticas e estatísticas, podem fazer mais sentido quando aplicadas para tipos numéricos. Verifique no comentário a frente de cada comando o que ele faz. Vale a pena ressaltar a diferença entre o atributo `shape` e o método `len`. O primeiro verifica quantas linhas a Series possui (quantos índices), já o

In [7]:

```

series_dados = pd.Series([10.2, -1, None, 15, 23.4])

print('Quantidade de linhas = ', series_dados.shape) # Retorna a
tupla com o número de linhas
print('Tipo de dados', series_dados.dtypes) # Retorna o
dados, se for misto será object
print('Os valores são únicos?', series_dados.is_unique) #
se os valores são únicos (sem duplicações)
print('Existem valores nulos?', series_dados.hasnans) #
existem valores nulos
print('Quantos valores existem?', series_dados.count()) #
quantos valores existem (exclui os nulos)

print('Qual o menor valor?', series_dados.min()) # Extrai o
valor da Series (nesse caso os dados precisam ser do mesmo
print('Qual o maior valor?', series_dados.max()) # Extrai o
máximo, com a mesma condição do mínimo
print('Qual a média aritmética?', series_dados.mean()) #
média aritmética de uma Series numérica
print('Qual o desvio padrão?', series_dados.std()) # Extrai o
padrão de uma Series numérica
print('Qual a mediana?', series_dados.median()) # Extrai a
de uma Series numérica

print('\nResumo:\n', series_dados.describe()) # Exibe um
sobre os dados na Series

```

```

Quantidade de linhas = (5,)
Tipo de dados float64
Os valores são únicos? True
Existem valores nulos? True
Quantos valores existem? 4
Qual o menor valor? -1.0
Qual o maior valor? 23.4
Qual a média aritmética? 11.899999999999999
Qual o desvio padrão? 10.184301645179211
Qual a mediana? 12.6

```

Resumo:

```

count      4.000000
mean       11.900000
std        10.184302
min        -1.000000
25%         7.400000
50%        12.600000
75%        17.100000
max        23.400000
dtype: float64

```

Dentre todos os parâmetros esperados, somente um é obrigatório para criar um DataFrame com dados, o parâmetro `data=xxxx`. Esse parâmetro pode ser um objeto iterável, como uma lista, tupla, um dicionário ou um DataFrame. Vejamos os exemplos.

CONSTRUTOR DATAFRAME COM LISTA

Na entrada 8, criamos 4 listas, com mesmo tamanho (5 valores) que vamos usar como fonte de dados para criar os primeiros DataFrames. Na entrada 9, invocando o método `DataFrame` e passando como parâmetro a lista de nomes, criando um nome (rótulo) para a coluna. Veja o resultado, temos os dados na ordem dos índices, que como não especificamos é atribuído o range de 0 a N-1. Na entrada 10, criamos o mesmo DataFrame, mas agora passando a lista de cpfs como parâmetro. Na entrada 11, usamos a função `zip()` para criar tuplas, cada uma composta por um valor de cada lista, e a transformamos em uma lista de tuplas. Fizemos a mesma construção para criar um DataFrame, no qual cada lista passe a ser um parâmetro, conforme pode ser observado no resultado.

In [2]:

```
lista_nomes = 'Howard Ian Peter Jonah Kellie'.split()
lista_cpfs = '111.111.111-11 222.222.222-22 333.333.333-33 444.444.444-44 555.555.555-55'.split()
lista_emails = 'risus.varius@dictumPhasellusin.ca Nunc@volutaFames.ac.turpis@cursusa.org non@felisullamcorper.org eget.dictum.placerat@necluctus.co.uk'.split()
lista_idades = [32, 22, 25, 29, 38]
```

In [9]:

```
pd.DataFrame(lista_nomes, columns=['nome'])
```

Out[9]:

	nome
0	Howard
1	Ian
2	Peter
3	Jonah
4	Kellie

In [10]:

```
pd.DataFrame(lista_nomes, columns=['nome'], index=lista_cpfs)
```

Out[10]:

CONSTRUTOR DATAFRAME COM DICIONÁRIO

DataFrames também podem ser construídos a partir de estruturas de tipo dicionário. Cada chave será uma coluna e pode ter atribuída uma lista de valores. **Obs: cada chave deve estar associada a uma lista de mesmo tamanho.** Na entrada 12, criamos nosso dicionário de dados, veja que cada chave tem uma lista de mesmo tamanho e criamos nosso DataFrame, passando o dicionário como fonte de dados. Dessa forma o construtor já consegue identificar as colunas.

In [6]:

```
dados = {
    'nomes': 'Howard Ian Peter Jonah Kellie'.split(),
    'cpfs' : '111.111.111-11 222.222.222-22 333.333.333-33 444.444.444-44 555.555.555-55'.split(),
    'emails' : 'risus.varius@dictumPhasellusin.ca Nunc@vulputate.ca fames.ac.turpis@cursusa.org non@felisullamcorper.org eget.dictum.placerat@necluctus.co.uk'.split(),
    'idades' : [32, 22, 25, 29, 38]
}

pd.DataFrame(dados)
```

Out[6]:

	nomes	cpfs	emails	idades
0	Howard	111.111.111-11	risus.varius@dictumPhasellusin.ca	32
1	Ian	222.222.222-22	Nunc@vulputate.ca	22
2	Peter	333.333.333-33	fames.ac.turpis@cursusa.org	25
3	Jonah	444.444.444-44	non@felisullamcorper.org	29
4	Kellie	555.555.555-55	eget.dictum.placerat@necluctus.co.uk	38

EXTRAINDO INFORMAÇÕES DE UM DATAFRAME

Como já mencionamos, cada objeto possui seus próprios atributos e métodos. Logo, embora Series e DataFrame tenham recursos em comum, eles também possuem suas particularidades. No DataFrame temos o método `info()` que nos diz quantas linhas e colunas existem. Também exibe o tipo de cada coluna e se os valores não nulos existem ali. Esse método também retorna uma informação sobre a quantidade de memória RAM essa estrutura está ocupando. Faça a

In [13]:

```
df_dados = pd.DataFrame(dados)

print('\nInformações do DataFrame:\n')
print(df_dados.info()) # Apresenta informações sobre a e
DF

print('\nQuantidade de linhas e colunas = ', df_dados.sh
Retorna uma tupla com o número de linhas e colunas
print('\nTipo de dados:\n', df_dados.dtypes) # Retorna o
dados, para cada coluna, se for misto será object

print('\nQual o menor valor de cada coluna?\n', df_dados
Extrai o menor de cada coluna
print('\nQual o maior valor?\n', df_dados.max()) # Extra
máximo e cada coluna
print('\nQual a média aritmética?\n', df_dados.mean()) #
média aritmética de cada coluna numérica
print('\nQual o desvio padrão?\n', df_dados.std()) # Ext
padrão de cada coluna numérica
print('\nQual a mediana?\n', df_dados.median()) # Extrai
de cada coluna numérica

print('\nResumo:\n', df_dados.describe()) # Exibe um res

df_dados.head() # Exibe os 5 primeiros registros do Data
```

Informações do DataFrame:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
nomes      5 non-null object
cpfs       5 non-null object
emails     5 non-null object
idades     5 non-null int64
dtypes: int64(1), object(3)
memory usage: 240.0+ bytes
None
```

Quantidade de linhas e colunas = (5, 4)

Tipo de dados:

```
nomes      object
cpfs       object
emails     object
idades     int64
dtype: object
```

Qual o menor valor de cada coluna?

```
nomes      Howard
```

dtype: float64

Qual a mediana?
idades 29.0
dtype: float64

Resumo:

	idades
count	5.000000
mean	29.200000
std	6.220932
min	22.000000
25%	25.000000
50%	29.000000
75%	32.000000
max	38.000000

Out[13]:

	nomes	cpfs	emails	idades
0	Howard	111.111.111-11	risus.varius@dictumPhasellusin.ca	32
1	Ian	222.222.222-22	Nunc@vulputate.ca	22
2	Peter	333.333.333-33	fames.ac.turpis@cursusa.org	25
3	Jonah	444.444.444-44	non@felisullamcorper.org	29
4	Kellie	555.555.555-55	eget.dictum.placerat@neductus.co.uk	38

Agora que você aprendeu como criar dataframes e extrair informações, vamos agora utilizar o Jupyter Notebook no emulador a seguir para testar o exemplo apresentado e crie o seu próprio DataFrame e extraia informações.



SELEÇÃO DE COLUNAS EM UM DATAFRAME

Podemos realizar operações em colunas específicas de um DataFrame. Para isso, podemos criar um novo objeto contendo somente as colunas que serão usadas para a determinada análise. Para selecionar uma coluna, as duas possíveis sintaxes são:

1. `nome_df.nome_coluna`
2. `nome_df[nome_coluna]`

A primeira forma é familiar aos desenvolvedores que utilizam a linguagem Python, porém ela não aceita colunas com espaços entre as palavras. Já a segunda forma aceita. Se precisarmos selecionar mais do que uma coluna, então precisamos passar uma lista, da seguinte forma: `nome_df[['col1', 'col2', 'col3']]`. A lista pode ser criada fora da seção e passada como parâmetro.

Ao selecionar uma coluna, obtemos uma Series, consequentemente, podemos realizar operações

```
In [14]: df_uma_coluna = df_dados['idades']
print(type(df_uma_coluna))

print('Média de idades = ', df_uma_coluna.mean())

df_uma_coluna
```

```
<class 'pandas.core.series.Series'>
Média de idades = 29.2
```

```
Out[14]: 0    32
1    22
2    25
3    29
4    38
Name: idades, dtype: int64
```

```
In [15]: colunas = ['nomes', 'cpfs']
df_duas_colunas = df_dados[colunas]
print(type(df_duas_colunas))
df_duas_colunas
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Out[15]:
```

	nomes	cpfs
0	Howard	111.111.111-11
1	Ian	222.222.222-22
2	Peter	333.333.333-33
3	Jonah	444.444.444-44
4	Kellie	555.555.555-55

EXEMPLIFICANDO

Vamos utilizar tudo que já aprendemos e fazer uma atividade de web (web scraping). Vamos acessar a seguinte página de notícia do New York Times: <https://nyti.ms/3aHRu2D>. A partir dessa fonte de informações vamos trabalhar para criar um DataFrame contendo a notícia, o comentário que foi feito, a explicação que foi dada e o nome da notícia.

Vamos começar nossa raspagem utilizando um recurso que já não é tão familiar, a biblioteca requests! Fazer a extração da notícia com o `requests.get()` convertendo tudo para uma única string, por isso usamos a propriedade `text`. Na linha 4, da entrada 16 imprimimos o

Como temos um conteúdo em HTML é conveniente utilizar a biblioteca BeautifulSoup, para converter a string em uma estrutura HTML e filtrar determinadas tags. Veja na entrada 17, estamos importando a biblioteca e através da classe BeautifulSoup, instanciamos um objeto passando o texto, em string, e o parâmetro '**html.parser**'. Agora, como o objeto do tipo BeautifulSoup, podemos usar o método **find_all()** para encontrar todas as ocorrências de uma determinada tag, no nosso caso estamos buscando pelas tags span, que contenham um atributo 'class': 'short-desc'. O resultado dessa busca é um conjunto iterável (class 'bs4.element.ResultSet'), como se fosse uma lista, então na linha 18 estamos exibindo a notícia no índice 5 desse iterável e na linha 19 estamos exibindo o "conteúdo" desse mesmo elemento, veja que contentamos com uma lista do conteúdo. Obs: para saber qual tag buscar, antes é necessário examinar o código fonte da página que se deseja "raspar".

```
In [17]: from bs4 import BeautifulSoup

        bsp_texto = BeautifulSoup(texto_string, 'html.parser')
        lista_noticias = bsp_texto.find_all('span', attrs={'class': 'short-desc'})

        print(type(bsp_texto))
        print(type(lista_noticias))
        print(lista_noticias[5])

        lista_noticias[5].contents

        <class 'bs4.BeautifulSoup'>
        <class 'bs4.element.ResultSet'>
        <span class="short-desc"><strong>Jan. 25 </strong><br>
        millions of people that now aren't insured anymore<br>
        class="short-truth"><a
        href="https://www.nytimes.com/2017/03/13/us/politics/
        check-trump-obamacare-health-care.html" target="_blank">
        real number is less than 1 million, according to the
        Institute.</a></span></span>

Out[17]: [<strong>Jan. 25 </strong>,
          "“You had millions of people that now aren't insured
          anymore.” ",
          <span class="short-truth"><a
```

- **Linha 4:** O código `noticia.contents[0]` retorna: `Jan.`
``, ao acessar a propriedade `text`, eliminamos as tags
temos `Jan. 25`. Usamos a função `strip()` para eliminar espaço
na string e concatenamos com o ano.
- **Linha 5:** O código `contents[1]` retorna: `"'You had millions of p`
`now aren't insured anymore.'" "` usamos o `strip()` para eliminar
branco e a função `replace` para substituir os caracteres espe
nada.
- **Linha 6:** O código `noticia.contents[2]` retorna: `<a href="http`
`mes.com/2017/03/13/us/politics/fact-check-trump-obamacare-h`
`tml" target="_blank"`
`>(The real number is less than 1 million, according to the Urban`
`a>`, ao acessar a propriedade `text`, eliminamos as tags
os `(The real number is less than 1 million, according to the Urba`
o qual ajustamos para eliminar espaços e os parênteses.
- Linha 7: o código `noticia.find('a')['href']` retorna: `https://v`
`com/2017/03/13/us/politics/fact-check-trump-obamacare-health`
- Apendamos a nossa lista de dados uma tupla com as quatro
informações que extraímos.

In [18]:

```
dados = []

for noticia in lista_noticias:
    data = noticia.contents[0].text.strip() + ', 20
    Dessa informação <strong>Jan. 25 </strong> vai ext
    25, 2017
    comentario = noticia.contents[1].strip().repla
    '').replace('"', '')
    explicacao = noticia.contents[2].text.strip().
    ('', '').replace('"', '')
    url = noticia.find('a')['href']
    dados.append((data, comentario, explicacao, url)

dados[1]
```

Out[18]:

(17, 2017, 21, 2017)

passando os dados e o nome das colunas. Pelo atributo shape conseguimos saber que foram extraídas 180 notícias e, que cada coluna possui o tipo object (que já era esperado por ser texto).

In [19]:

```
df_noticias = pd.DataFrame(dados, columns=['data',
      'comentário', 'explicação', 'url'])

print(df_noticias.shape)
print(df_noticias.dtypes)
df_noticias.head()
```

```
(180, 4)
data          object
comentário     object
explicação     object
url            object
dtype: object
```

Out[19]:

	data	comentário	explicação	
0	Jan. 21, 2017	I wasn't a fan of Iraq. I didn't want to go in...	He was for an invasion before he was against it.	https://www.buzzfeed.com
1	Jan. 21, 2017	A reporter for Time magazine — and I have been...	Trump was on the cover 11 times and Nixon appe...	http://nation.time.com/2013
2	Jan. 23, 2017	Between 3 million and 5 million illegal votes ...	There's no evidence of illegal voting.	https://www.nytimes.com/20
3	Jan. 25, 2017	Now, the audience was the biggest ever. But th...	Official aerial photos show Obama's 2009 inaug...	https://www.nytimes.com/20
4	Jan. 25, 2017	Take a look at the Pew reports (which show vot...	The report never mentioned voter fraud.	https://www.nytimes.com/20

LEITURA DE DADOS ESTRUTURADOS COM A BIBLIOTECA PANDAS

Um dos grandes recursos da biblioteca pandas é sua capacidade de fazer a leitura de dados estruturados, através de seus métodos, guardando em um Dataframe. A biblioteca possui uma série de métodos "read", cuja sintaxe é: `pandas.read_xxxxx()` onde a sequência de X representa as diversas opções disponíveis. Para finalizar nossa aula, vamos ver como fazer a leitura de dados de uma página web, utilizando o método `pandas.read_html()`. A documentação desse método está disponível em <https://pandas.pydata.org/pandas-docs/stable/10min.html>.

capturar os dados e carregar em um DataFrame. Observe o código na linha 3, onde o método `read_html` capturou todas as tabelas no endereço passado como parâmetro, sendo que cada tabela é armazenada em um DataFrame e o método retorna uma lista com todos eles. Veja na linha 4, que ao imprimirmos o resultado guardado na variável `dfs`, obtemos uma lista e ao verificarmos quantos DataFrames foram criados (`len(dfs)`), somente uma tabela foi encontrada, pois o tamanho da lista é 1.

In [20]:

```
url = 'https://www.fdic.gov/bank/individual/failed/bank1.html'
dfs = pd.read_html(url)

print(type(dfs))
print(len(dfs))

<class 'list'>
1
```

Sabendo que o tamanho da lista resultado do método é 1, então para acessar a única tabela que queremos, basta acessar a posição 0 da lista. Observe na linha 5, onde guardamos o único DataFrame da lista em uma nova variável, verificamos quantas linhas existem e quais os tipos de cada coluna, com exceção da coluna `Bank Name`, todas as demais são texto. Usamos o método `head` para ver os cinco primeiros registros do DataFrame.

In [21]:

```
df_bancos = dfs[0]

print(df_bancos.shape)
print(df_bancos.dtypes)

df_bancos.head()

(561, 6)
Bank Name      object
City           object
ST            object
CERT          int64
Acquiring Institution  object
Closing Date   object
```


REFERÊNCIAS E LINKS ÚTEIS

PyPI. Python Package Index. Disponível em: <https://pypi.org/>. Acesso em: 17 jun. 2020.

Leonard Richardson. BeautifulSoup Documentation. Disponível em: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>. Acesso em: 17 jun. 2020.

Pandas Team. About pandas. Disponível em: <https://pandas.pydata.org/pandas-docs/stable/10min.html>. Acesso em: 17 jun. 2020.

Pandas Team. DataFrame. Disponível em: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>. Acesso em: 17 jun. 2020.

Pandas Team. pandas documentation. Disponível em: <https://pandas.pydata.org/pandas-docs/stable/index.html>. Acesso em: 17 jun. 2020.

Pandas Team. Package overview. Disponível em: <https://pandas.pydata.org/pandas-docs/stable/10min.html>. Acesso em: 17 jun. 2020.

Pandas Team. Series. Disponível em: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html>. Acesso em: 17 jun. 2020.