**Introdução de unidade**



Objetivos da Unidade

Ao longo desta Unidade, você irá:

* definir o pacote pandas;
* demonstrar as series.str;
* analisar os conceitos da biblioteca seaborn.

Olá estudantes, sejam bem-vindos!

Nesta unidade vamos abordar conceitos relacionados à estrutura de dados, biblioteca panda, manipulação de dados e a visualização de dados em Python.

Não deixe de praticar e buscar as informações que são recomendadas para que seu aproveitamento desta disciplina seja completo.

Bons estudos.

**Introdução da aula**



Qual é o foco da aula?

Nesta aula, vamos aprender conceitos importantes sobre o pandas, além de conhecer melhor as estruturas de um DataFrame e suas colunas.

Objetivos gerais de aprendizagem

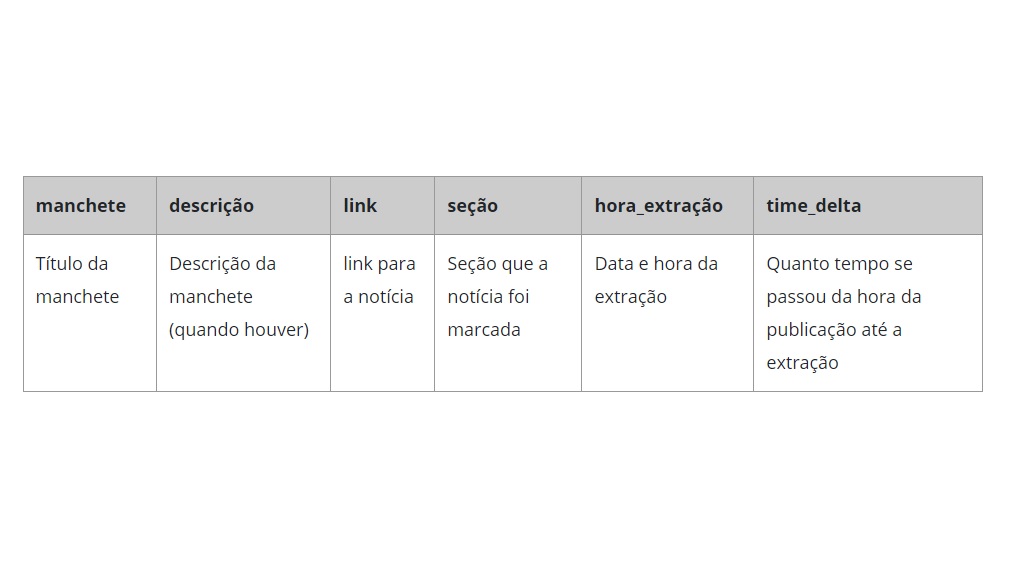
Ao longo desta aula, você irá:

* descrever o pacote Pandas;
* demonstrar informações de uma Series;
* ilustrar a seleção de coluna de um DataFrame.

**Situação-problema**

Em um artigo publicado no dia 06 de março de 2019, no portal Computer World, o autor fala sobre o profissional que deseja seguir a carreira de analista de dados, o qual deve ter habilidades em: filtrar dados, construir APIs, web scraping e ter conhecimento nas linguagens Git, MySQL e Python. (MOTIM, Raphael Bueno da. A [carreira](https://computerworld.com.br/2019/03/06/carreira-de-analista-de-dados-oferece-salarios-de-ate-r-125-mil/" \t "_blank)de analista de dados oferece salários de até R$12,5 mil. 2019.

Como desenvolvedor em uma empresa de consultoria de software, você foi alocado em uma equipe de marketing analítico em uma marca esportiva, que necessita fazer a coleta das principais notícias do mundo do esporte em um determinado portal. O cliente pediu para que o [portal](https://globoesporte.globo.com/" \t "_blank) fosse o escolhido. O cliente deseja um componente capaz de fazer a extração dos dados em forma tabular, com os seguintes campos: manchete, descrição, link, seção, hora da extração, tempo decorrido da publicação até a hora da extração. O quadro abaixo apresenta, visualmente, como os dados devem ser organizados e exibidos.

Resultado esperado. Fonte: elaborada pela autora.

O grande desafio no trabalho de web scraping é descobrir qual o padrão nas tags HTML e atributos CSS usados. Pois somente através deles é que conseguiremos alcançar a informação solicitada. Como o cliente já trabalha com o portal de notícias, foram-lhe passadas as seguintes informações técnicas que o ajudarão a fazer a extração.

Para extração de todas as informações localize todas as div com atributo 'class':'feed-post-body'. De cada item localizado extraia:

* A manchete que ocupa a primeira posição do conteúdo.
* O link que pode ser localizado pela tag "a" e pelo atributo "href".
* A descrição pode estar na terceira posição conteúdo ou dentro de uma div com atributo 'class':'bstn-related'
* A seção está dentro de uma div com atributo 'class':'feed-post-metadata'. Localize o span com atributo 'class': 'feed-post-metadata-section'.
* O tempo decorrido está uma div com atributo 'class':'feed-post-metadata'. Localize o span com atributo 'class': 'feed-post-datetime'.

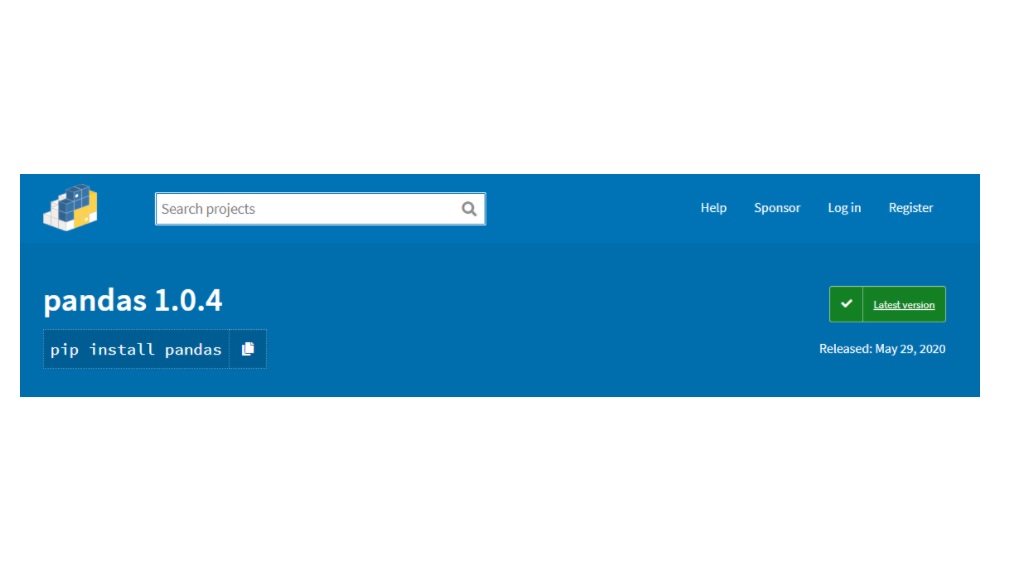
Caso tente acessar o texto de uma tag não localizada, um erro é dado, para evitar esses casos, os campos descrição, seção e time\_delta devem ser tratados para esses casos, retornando None (nulo). Agora é com você, faça a implementação e gere um DataFrame com as informações solicitadas.

**Introdução a biblioteca pandas**

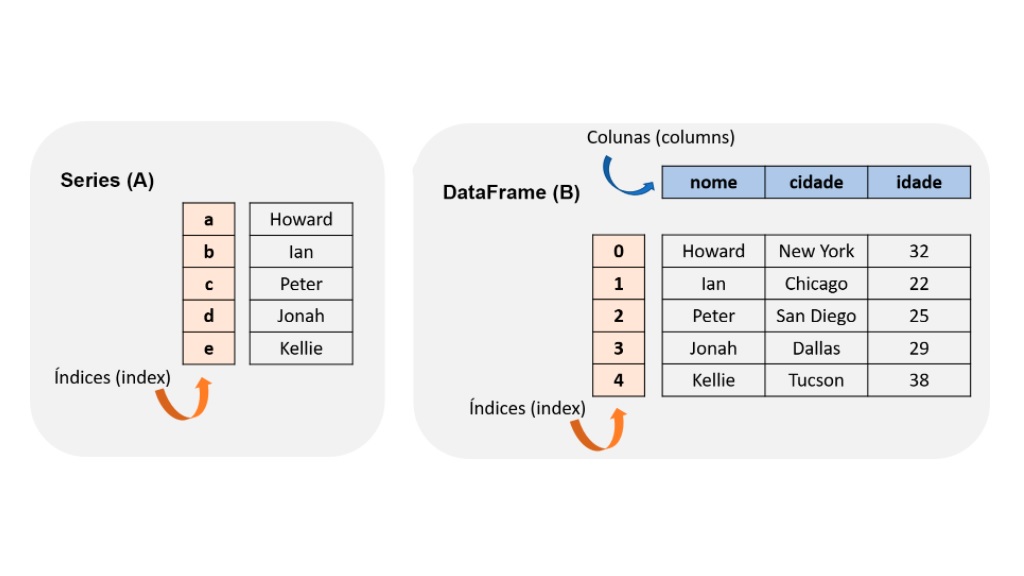


Dentre as diversas bibliotecas disponíveis no repositório PyPI, pandas é um pacote Python que fornece estruturas de dados projetadas para facilitar o trabalho com dados estruturados (tabelas) e de séries [temporais](https://pandas.pydata.org/docs/getting_started/overview.html" \t "_blank). Esse pacote começou a ser desenvolvido em 2008, tornando-se uma solução open source no final de 2009. Desde 2015, o projeto pandas é patrocinado pela [NumFOCUS](https://pandas.pydata.org/about/" \t "_blank). Segundo sua documentação oficial, pandas pretende ser o alicerce de alto nível fundamental para uma análise prática, a dos dados do mundo real em Python. Além disso, ele 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 de programação.

Para utilizar a biblioteca pandas é preciso fazer a instalação, como mostra a figura: pip install pandas (caso esteja usando conda, verificar documentação). No momento em que esse material está sendo produzido, a biblioteca encontra-se na versão 1.0.4 e teve sua última atualização disponibilizada no dia 29 de maio de 2020 (figura abaixo).

Instalação do pandas. Fonte: Pypi.

Como uma ferramenta de alto nível, pandas possui duas estruturas de dados que são as principais para a análise/manipulação de dados: a Series e o DataFrame. Uma **Series**é um como um vetor de dados (unidimencional), capaz de armazenar diferentes tipos de dados. Um **DataFrame** é 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, um decimal ou uma data. A figura abaixo ilustra uma Series (A) e um DataFrame (B). Veja que uma Series possui somente "uma coluna" de informação e seus rótulos (índices). Um DataFrame pode ter uma ou mais colunas e além dos índices, também há um rótulo de identificação com o nome da coluna. Podemos comparar um DataFrame como uma planilha eletrônico, como o Excel (da Microsoft) ou o Calc (do Open Office).

Series (A) e DataFrame (B). Fonte: elaborada pela autora.

\_\_\_\_\_\_\_

**⭐ Dica**

Na Internet você pode encontrar diversas "cheat sheet" (folha de dicas) sobre a biblioteca pandas. Recomendamos a cheat sheet oficial da [biblioteca](https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf" \t "_blank).

\_\_\_\_\_\_\_

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(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False). Veja que todos os parâmetros possuem valores padrões (default) o que permite instanciar um objeto de diferentes formas. Para endender cada parâmetro, a melhor fonte de informações é a documentação [oficial](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html" \t "_blank).

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

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 aparece 0 como índice e 5 como valor. Quando não deixamos explícito os rótulos (índices) que 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 agora 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 desse tipo de dado na construção é que a chave do dicionário é usada como índice.

Outra forma de construir a Series é passando os dados e os rótulos que desejamos usar. Veja na entrada 5 essa construção, na qual utilizaoms uma lista de supostos cpfs para rotular os valores da Series.

In [5]: cpfs **=** '111.111.111-11 222.222.222-22 333.333.333-33 444.444.444-44 555.555.555-55'**.**split()

pd**.**Series(lista\_nomes, index**=**cpfs)

Out [5]: 111.111.111-11    Howard

222.222.222-22       Ian

333.333.333-33     Peter

444.444.444-44     Jonah

555.555.555-55    Kellie

dtype: object

Rotular as Series (e como veremos os DataFrames), é interessante para facilitar 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 guardados dentro 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 e que, diferentes estruturas possuem diferentes atributos e métodos. Com as estruturas de dados do pandas não é diferente, tais objetos possuem atributos e métodos específicos, vamos conhecer alguns. Na entrada 7, criamos uma série contando números e um valor nulo (None). As informações extraídas das linhas 3 a 7, são mais com relação a "forma" dos dados, portanto poderiam ser usadas independente do tipo de dado armazenado na Series, inclusive em um cenário de dados com diferentes tipos. Já as informações das linhas 9 a 15, como se tratam de funções matemáticas e estatísticas, podem fazer mais sentido quando utilizadas 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 count(). O primeiro verifica quantas linhas a Series possui (quantos índices), já o segundo, conta quantos dados não nulos existem.

In [7]: series\_dados **=** pd**.**Series([10.2, **-**1, None, 15, 23.4])

print('Quantidade de linhas = ', series\_dados**.**shape) # Retorna uma tupla com o número de linhas

print('Tipo de dados', series\_dados**.**dtypes) # Retorna o tipo de dados, se for misto será object

print('Os valores são únicos?', series\_dados**.**is\_unique) # Verifica se os valores são únicos (sem duplicações)

print('Existem valores nulos?', series\_dados**.**hasnans) # Verifica se existem valores nulos

print('Quantos valores existem?', series\_dados**.**count()) # Conta quantas valores existem (excluí os nulos)

print('Qual o menor valor?', series\_dados**.**min()) # Extrai o menor valor da Series (nesse caso os dados precisam ser do mesmo tipo)

print('Qual o maior valor?', series\_dados**.**max()) # Extrai o valor máximo, com a mesma condição do mínimo

print('Qual a média aritmética?', series\_dados**.**mean()) # Extrai a média aritmética de uma Series numérica

print('Qual o desvio padrão?', series\_dados**.**std()) # Extrai o desvio padrão de uma Series numérica

print('Qual a mediana?', series\_dados**.**median()) # Extrai a mediana de uma Series numérica

print('\nResumo:\n', series\_dados**.**describe()) # Exibe um resumo 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

**Dataframe**- Para construir um objeto do tipo DataFrame, precisamos utilizar o método DataFrame() do pacote pandas. O método possui o seguinte construtor: pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False). Veja que todos os parâmetros possuem valores padrões (default) o que permite instanciar um objeto de diferentes formas. Para endender cada parâmetro, a melhor fonte de informações é a documentação [oficial](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \t "_blank).

Dentre todos os parâmetros esperados, somente um é obrigatório para se criar um DataFrame com dados, o parâmetro data=XXXX. Esse parâmetro pode receber, 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 o mesmo tamanho (5 valores) que vamos usar como fonte de dados para criar os primeiros DataFrames. Na entrada 9, estamos invocando o método DataFrame e passando como parâmetro a lista de nomes e um nome (rótulo) para a coluna. Veja o resultado, temos os dados na coluna e os 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 índice. 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 essa construção para criar um DataFrame, no qual cada lista passe a ser uma coluna, conforme pode ser observado no resultado.

In [8]: lista\_nomes **=** 'Howard Ian Peter Jonah Kellie'**.**split()

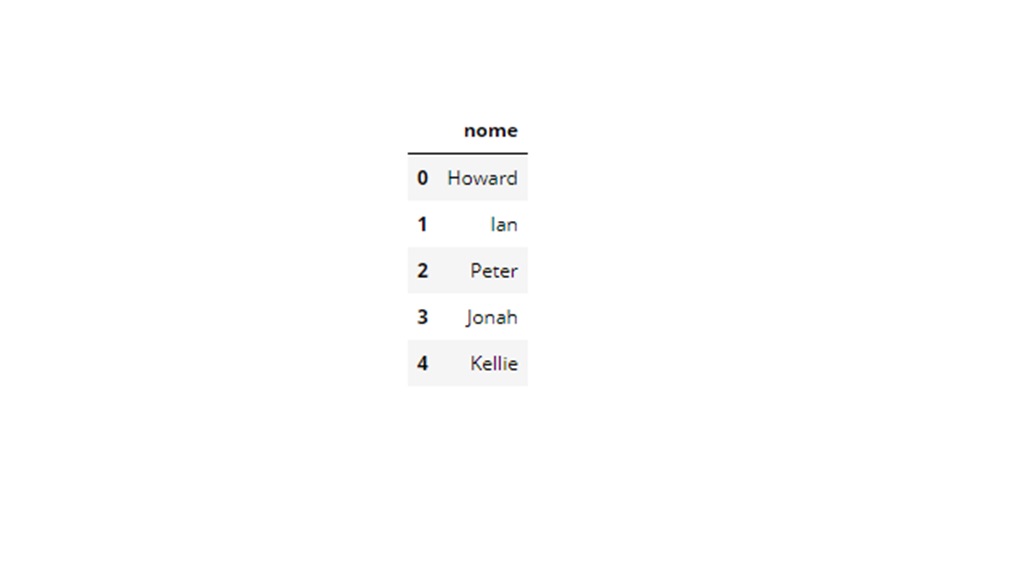
lista\_cpfs **=** '111.111.111-11 222.222.222-22 333.333.333-33 444.444.444-44 555.555.555-55'**.**split()

lista\_emails **=** 'risus.varius@dictumPhasellusin.ca Nunc@vulputate.ca fames.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]:



In [10]: pd**.**DataFrame(lista\_nomes, columns**=**['nome'], index**=**lista\_cpfs)

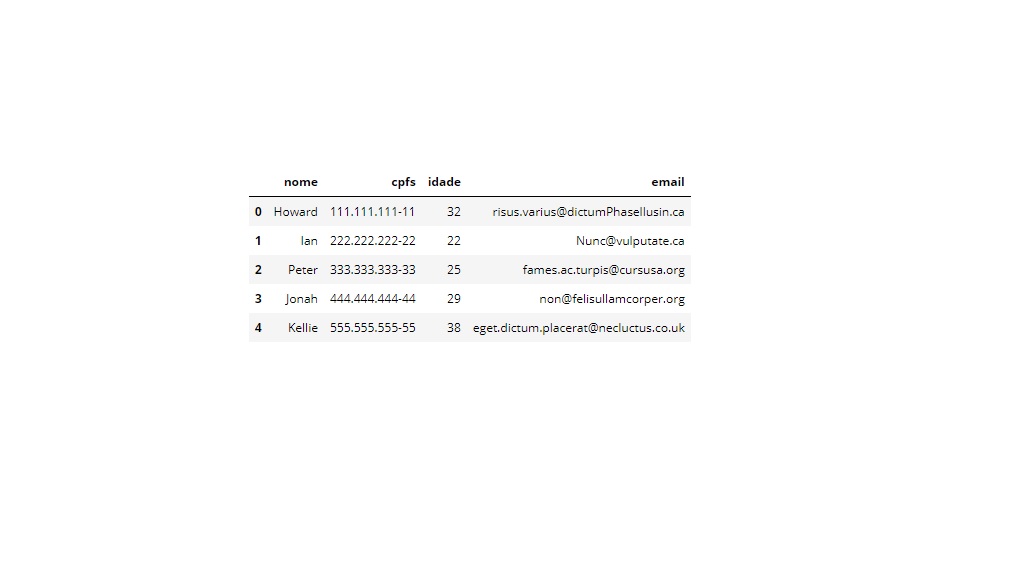
Out [10]:



In [7]: dados **=** list(zip(lista\_nomes, lista\_cpfs, lista\_idades, lista\_emails)) # cria uma lista de tuplas

pd**.**DataFrame(dados, columns**=**['nome', 'cpfs', 'idade', 'email'])

Out [7]:



**Construtor DataFrame com dicionário**- DataFrames também podem ser construídos a partir de estruturas de dados do 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 possui uma lista de mesmo tamanho e criamos nosso DataFrame, passando o dicionário como fonte de dados. Dessa forma o construtor já consegue identificar o nome das 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(),

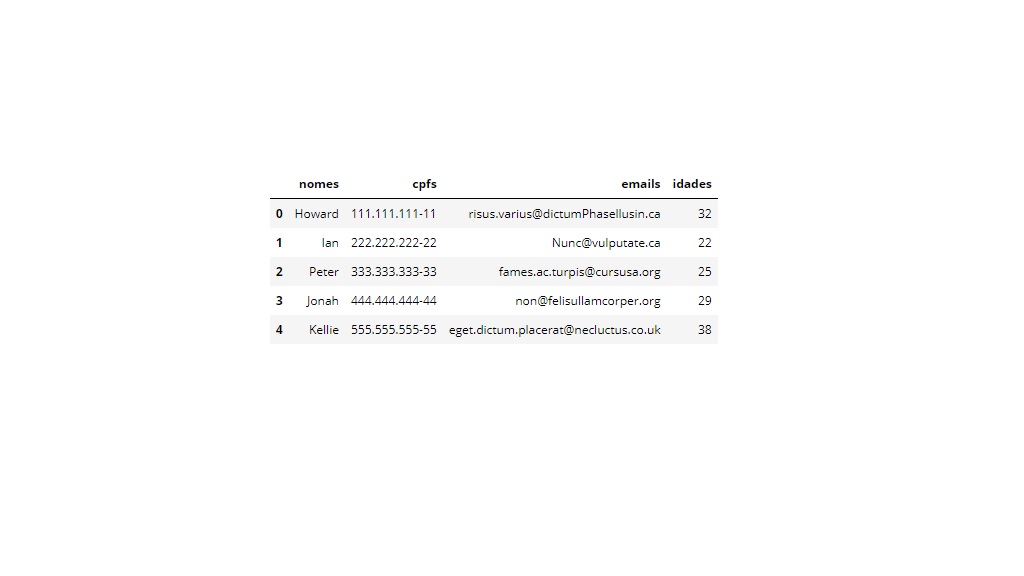
    'emails' : 'risus.varius@dictumPhasellusin.ca Nunc@vulputate.ca fames.ac.turpis@cursusa.org non@felisullamcorper.org eget.dictum.placerat@necluctus.co.uk'**.**split(),

    'idades' : [32, 22, 25, 29, 38]

}

pd**.**DataFrame(dados)

Out [6]:



**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 mostra quantas linhas e colunas existem. Também exibe o tipo de cada coluna e quanto valores não nulos existem ali. Esse método também retorna uma informação sobre a quantidade de memória RAM essa estrutura está ocupando. Faça a leitura dos comentários e veja o que cada atributo e método retorna.

In [13]: df\_dados **=** pd**.**DataFrame(dados)

print('\nInformações do DataFrame:\n')

print(df\_dados**.**info()) # Apresenta informações sobre a estrutura do DF

print('\nQuantidade de linhas e colunas = ', df\_dados**.**shape) # Retorna uma tupla com o número de linhas e colunas

print('\nTipo de dados:\n', df\_dados**.**dtypes) # Retorna o tipo de dados, para cada coluna, se for misto será object

print('\nQual o menor valor de cada coluna?\n', df\_dados**.**min()) # Extrai o menor de cada coluna

print('\nQual o maior valor?\n', df\_dados**.**max()) # Extrai o valor máximo e cada coluna

print('\nQual a média aritmética?\n', df\_dados**.**mean()) # Extrai a média aritmética de cada coluna numérica

print('\nQual o desvio padrão?\n', df\_dados**.**std()) # Extrai o desvio padrão de cada coluna numérica

print('\nQual a mediana?\n', df\_dados**.**median()) # Extrai a mediana de cada coluna numérica

print('\nResumo:\n', df\_dados**.**describe()) # Exibe um resumo

df\_dados**.**head() # Exibe os 5 primeiros registros do DataFrame

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

cpfs         111.111.111-11

emails    Nunc@vulputate.ca

idades                   22

dtype: object

Qual o maior valor?

 nomes                                 Peter

cpfs                         555.555.555-55

emails    risus.varius@dictumPhasellusin.ca

idades                                   38

dtype: object

Qual a média aritmética?

 idades    29.2

dtype: float64

Qual o desvio padrão?

 idades    6.220932

dtype: float64

Qual a mediana?

 idades    29.0

dtype: float64

Resumo:

           idades

count   5.000000

mean   29.200000

std     6.220932

min    22.000000

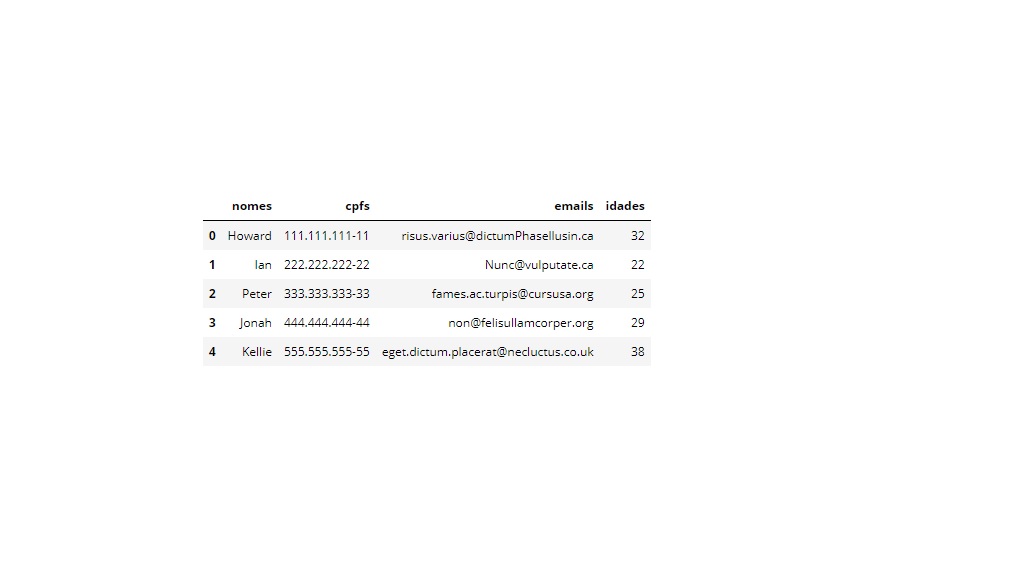
25%    25.000000

50%    29.000000

75%    32.000000

max    38.000000

Out [13]:



Agora que você aprendeu como criar dataframes e extrair informações. Utilize o [emulador](https://trinket.io/python3/40426b53a0" \t "_blank)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 ou ainda criar um novo objeto contendo somente as colunas que serão usadas em uma 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 utilizar a linguagem SQL, porém ela não aceita colunas com espaços entre as palavras. Já a segunda aceita.Se precisarmos selecionar mais do que uma coluna, então precisamos passar uma lista, da seguinte forma: nome\_df[['col1', 'col2', 'col3']], se preferir a lista pode ser criada fora da seção e passada como parâmetro.

Ao selecionar uma coluna, obtemos uma Series, consequentemente, podemos aplicar os atributos e métodos que aprendemos, por exemplo, para obter a média aritmética de uma determinada coluna. Observe os códigos a seguir. Na entrada 14, fizemos a seleção de uma única coluna "idades", veja na impressão que o tipo do objeto agora é uma Series. Na linha 4, a partir desse novo objeto, imprimimos a média de idade. Já na entrada 15, criamos uma lista com as colunas que queremos selecionar e na linha 2, passamos essa lista para seleção (df\_dados[colunas]), consequentemente, obtivemos um novo DataFrame, mas agora com duas colunas.

Através da seleção de certas colunas podemos extrair informações específicas e até compará-las com outras colunas ou com outros dados. Esse recurso é muito utilizado por quem trabalha na área de dados.

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

Iin [15]: colunas **=** ['nomes', 'cpfs']

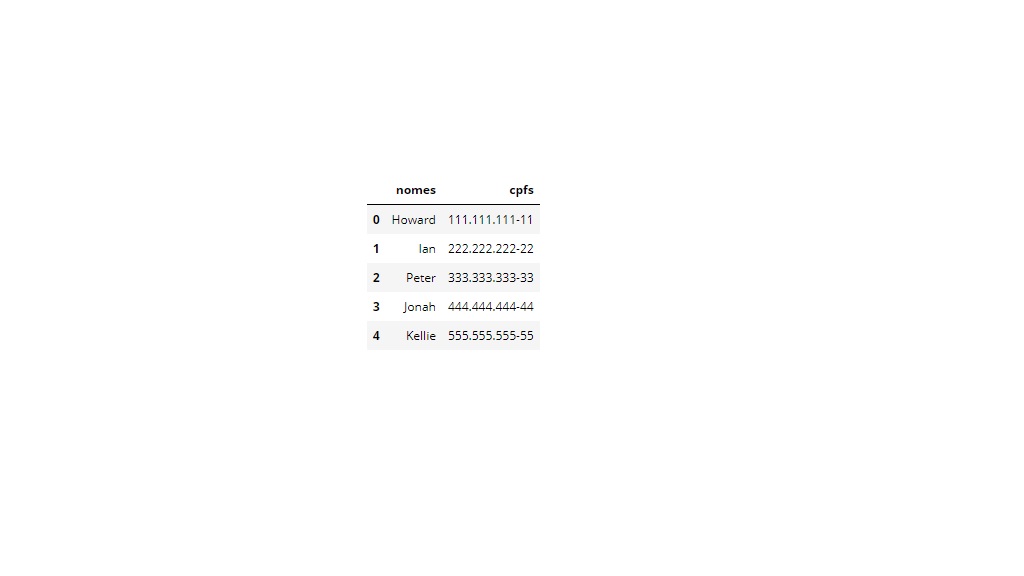
df\_duas\_colunas **=** df\_dados[colunas]

print(type(df\_duas\_colunas))

df\_duas\_colunas

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

Out [15]:



### **\_\_\_\_\_\_\_**

**📝 Exemplificando**

Vamos utilizar tudo que já aprendemos e fazer uma atividade de raspagem web (web scraping). Vamos acessar a seguinte página de notícias do jornal [New York Times](https://www.nytimes.com/interactive/2017/06/23/opinion/trumps-lies.html). A partir dessa fonte de informações vamos trabalhar para criar um DataFrame contendo o dia da notícia, o comentário que foi feito, a explicação que foi dada e o link da notícia.

Vamos começar nossa raspagem utilizando um recurso que já nos é familiar, a biblioteca requests! Fazer a extração da notícia com o requestes.get() convertendo tudo para uma única string, por isso vamos usar a propriedade text. Na linha 4, da entrada 16 imprimimos os 100 primeiros caracteres do texto que capturamos. Veja que foi lido o conteúdo HTML da página.

In [16]: import requests

texto\_string **=** requests**.**get('https://www.nytimes.com/interactive/2017/06/23/opinion/trumps-lies.html')**.**text

print(texto\_string[:100])

<!DOCTYPE html>

<!--[if (gt IE 9)|!(IE)]> <!--><html lang="en" class="no-js page-interactive section

Como temos um conteúdo em HTML é conveniente utilizar a biblioteca Beautiful Soup, para converter a string em uma estrutura HTML e então 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, com o objeto do tipo BeautifulSoup, podemos usar o método **find\_all**() para buscar 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 uma conjunto iterável (class 'bs4.element.ResultSet'), como se fosse uma lista, então na linha 8, estamos exibindo a notícia no índice 5 desse iterável e na linha 10, estamos exibindo o "conteúdo" desse mesmo elemento, veja que contents, retorna uma lista do conteúdo. Obs: para saber qual tag buscar, antes é preciso 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>“You had millions of people that now aren't insured anymore.” <span class="short-truth"><a href="https://www.nytimes.com/2017/03/13/us/politics/fact-check-trump-obamacare-health-care.html" target="\_blank">(The real number is less than 1 million, according to the Urban 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 href="https://www.nytimes.com/2017/03/13/us/politics/fact-check-trump-obamacare-health-care.html" target="\_blank">(The real number is less than 1 million, according to the Urban Institute.)</a></span>]

Na entrada 18, criamos uma estrutura de repetição que vai percorrer cada notícia do objeto iterável do tipo bs4.element.ResultSet, extraindo as informações que queremos. Para explicar cada linha de comando, vamos considerar a saída dos dados obtidas anteriormente, ou seja, a notícia na posição 5.

* **Linha 1**: Criamos uma lista vazia.
* **Linha 4**: O código noticia.contents[0] retorna: <strong>Jan. 25</strong>, ao acessar a propriedade text, eliminamos as tags, então temos Jan. 25. Usamos a função strip() para eliminar espaço em branco na string e concatenamos com o ano.
* **Linha 5**: O código contents[1] retorna: "“You had millions of people that now aren't insured anymore.” " usamos o strip() para eliminar espaços em branco e a função replace para substituir os caracteres especiais por nada.
* **Linha 6**: O código noticia.contents[2] retorna: <a href="https://www.nytimes.com/2017/03/13/us/politics/fact-check-trump-obamacare-health-care.html" target="\_blank">(The real number is less than 1 million, according to the Urban Institute.)</a></span>, ao acessar a propriedade text, eliminamos as tags então temos (The real number is less than 1 million, according to the Urban Institute.), o qual ajustamos para elimar espaços e os parênteses.
* **Linha 7**: o código noticia.find('a')['href'] retorna: https://www.nytimes.com/2017/03/13/us/politics/fact-check-trump-obamacare-health-care.html
* Aprendamos 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() **+** ', 2017' # Dessa informação <strong>Jan. 25 </strong> vai extrair Jan. 25, 2017

    comentario **=** noticia**.**contents[1]**.**strip()**.**replace("“", '')**.**replace("”", '')

    explicacao **=** noticia**.**contents[2]**.**text**.**strip()**.**replace("(", '')**.**replace(")", '')

    url **=** noticia**.**find('a')['href']

    dados**.**append((data, comentario, explicacao, url))

dados[1]

Out [18]: ('Jan. 21, 2017',

 'A reporter for Time magazine — and I have been on their cover 14 or 15 times. I think we have the all-time record in the history of Time magazine.',

 'Trump was on the cover 11 times and Nixon appeared 55 times.',

 'http://nation.time.com/2013/11/06/10-things-you-didnt-know-about-time/')

Agora que temos nossa lista de tuplas com os dados, podemos criar o DataFrame e disponibilizar para um cientista de dados fazer a análise de sentimentos. Veja que na entrada 19, usamos o construtor DataFrame passando os dados e o nome das colunas. Pelo atributo shape conseguimos de 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]:



**Leitura de dados estruturados com a biblioteca pandas**



Um dos grandes recursos da biblioteca pandas é sua capacidade de fazer 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 uma tabela em uma página web, utilizando o método pandas.read\_html(). A documentação desse método está [disponível](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_html.html" \t "_blank)e seu construtor possui os seguintes parâmetros: pandas.read\_html(io, match='.+', flavor=None, header=None, index\_col=None, skiprows=None, attrs=None, parse\_dates=False, thousands=',', encoding=None, decimal='.', converters=None, na\_values=None, keep\_default\_na=True, displayed\_only=True). Dentre todos, somente o "io" é o que recebe a URL a ser usada. Esse método procura por tags <table>  na estrutura do código HTML e devolve uma lista de DataFrames contendo as tabelas que localizou.

Na [URL](https://www.fdic.gov/bank/individual/failed/banklist.html), encontra-se uma tabela com bancos norte-americanos que faliram desde 1º de outubro de 2000, cada linha representa um banco. Vamos utilizar o método read\_html() para capturar os dados e carregar em um DataFrame. Observe o código na entrada 20, 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 tipo do 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/banklist.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 obter a tabela que queremos, basta acessar a posição 0 da lista. Observe na entrada 21, guardamos o único DataFrame da lista em uma nova variável, verificamos quantas linhas existem e quais os tipos de cada coluna, com excessão da coluna CERT, 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

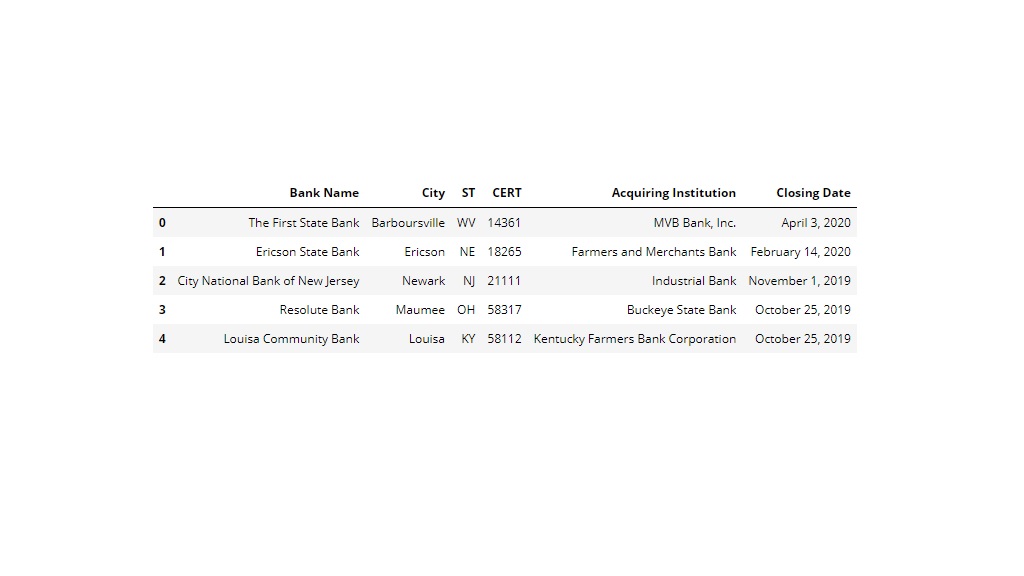
CERT                      int64

Acquiring Institution    object

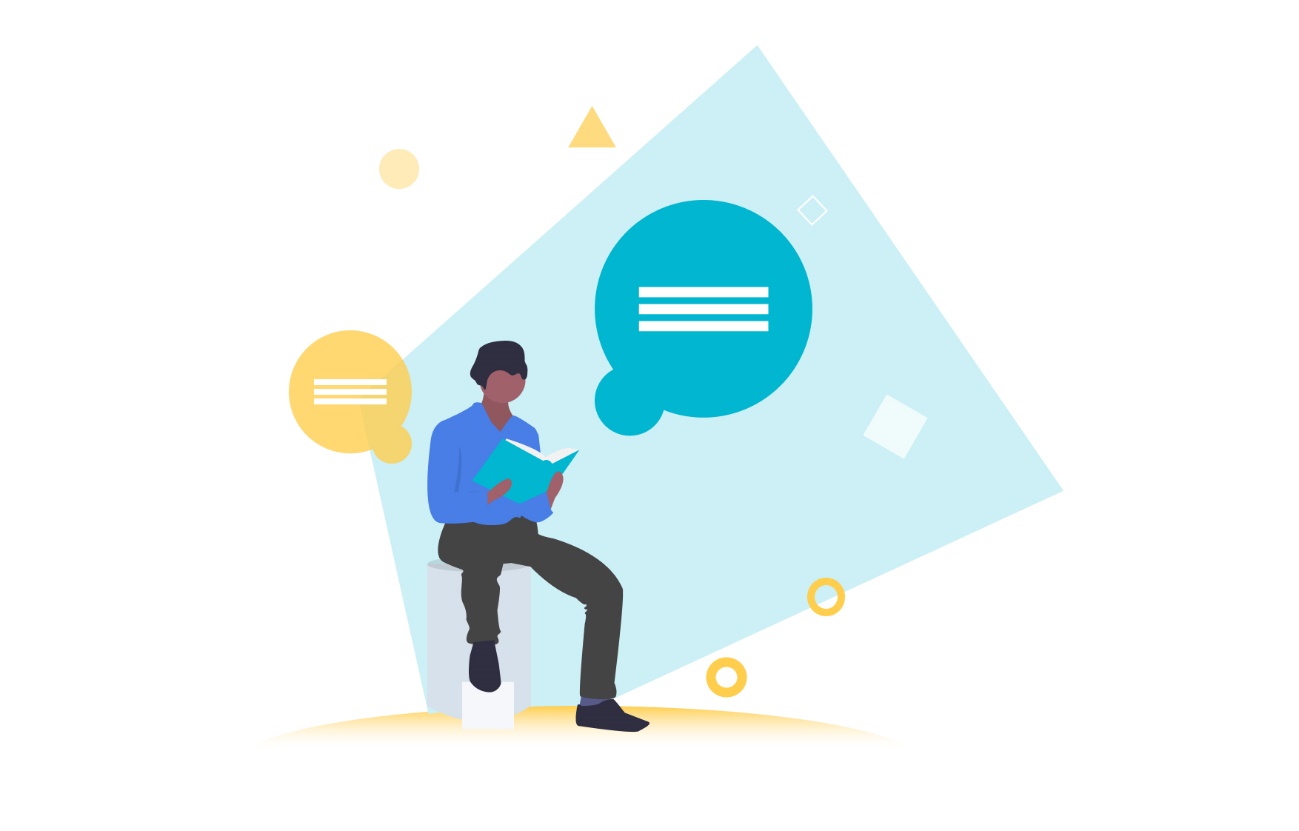
Closing Date             object

dtype: object

Out [21]:



**Conclusão**



Para fazer o web scraping solicitado, vamos utilizar as bibliotecas requests, BeautifulSoup, pandas e datetime. As duas primeiras serão usadas para fazer a captura do conteúdo da página, pandas para entregar os resultados em forma estruturada e datetime para marcar o dia e hora da extração.

In [22]: from datetime import datetime

import requests

from bs4 import BeautifulSoup

import pandas as pd

Com as bibliotecas importadas, vamos acessar o portal e utilizar a propriedade text da biblioteca requests para capturar em formato de string. Em seguida, vamos transformar essa string em formato html, para que possamos localizar as tags de nosso interesse. Na linha 2, registramos o horário da extração. Na linha 5, procuramos todas as tags div com o atributo que nos foi indicado. Essa linha retornará uma lista com cada notícia. Veja que na linha 6 imprimimos quantas notícias foram encontradas e na linha 7 imprimimos o conteúdo da primeira notícia. Lembre-se que contents transforma cada início e final da div em um elemento da lista.

In [23]: texto\_string **=** requests**.**get('https://globoesporte.globo.com/')**.**text

hora\_extracao **=** datetime**.**now()**.**strftime("%d-%m-%Y %H:%M:%S")

bsp\_texto **=** BeautifulSoup(texto\_string, 'html.parser')

lista\_noticias **=** bsp\_texto**.**find\_all('div', attrs**=**{'class':'feed-post-body'})

print("Quantidade de manchetes = ", len(lista\_noticias))

Lista\_noticias[0]**.**contents

Quantidade de manchetes =  10

Out [23]: [<div class="feed-post-header"></div>,

 <div class="\_label\_event"><div class="feed-post-body-title gui-color-primary gui-color-hover"><div class="\_ee"><a class="feed-post-link gui-color-primary gui-color-hover" href="https://globoesporte.globo.com/futebol/futebol-internacional/futebol-italiano/jogo/17-06-2020/napoli-juventusita.ghtml">VICE SENHORA</a></div></div></div>,

 <div class="\_label\_event"><div class="feed-post-body-resumo">Napoli vence Juventus nos pênaltis e leva Copa da Itália</div></div>,

 <div class="feed-media-wrapper"><div class="\_label\_event"><a class="feed-post-figure-link gui-image-hover" href="https://globoesporte.globo.com/futebol/futebol-internacional/futebol-italiano/jogo/17-06-2020/napoli-juventusita.ghtml"><div class="bstn-fd-item-cover"><picture class="bstn-fd-cover-picture"><img alt="Foto: (Alberto Lingria/Reuters)" class="bstn-fd-picture-image" src="https://s2.glbimg.com/BeTGAixT5O\_Cvs4hQA88PdHiCsY=/0x0:5406x3041/540x304/smart/https://i.s3.glbimg.com/v1/AUTH\_bc8228b6673f488aa253bbcb03c80ec5/internal\_photos/bs/2020/G/4/GZIncSSPmVRWLaYiNGIg/2020-06-17t212444z-1091152315-rc29bh9icqss-rtrmadp-3-soccer-italy-nap-juv-report.jpg" srcset="https://s2.glbimg.com/BeTGAixT5O\_Cvs4hQA88PdHiCsY=/0x0:5406x3041/540x304/smart/https://i.s3.glbimg.com/v1/AUTH\_bc8228b6673f488aa253bbcb03c80ec5/internal\_photos/bs/2020/G/4/GZIncSSPmVRWLaYiNGIg/2020-06-17t212444z-1091152315-rc29bh9icqss-rtrmadp-3-soccer-italy-nap-juv-report.jpg 1x,https://s2.glbimg.com/A6dTbIFD8sDl\_t7eMHvA-2ONF0Y=/0x0:5406x3041/810x456/smart/https://i.s3.glbimg.com/v1/AUTH\_bc8228b6673f488aa253bbcb03c80ec5/internal\_photos/bs/2020/G/4/GZIncSSPmVRWLaYiNGIg/2020-06-17t212444z-1091152315-rc29bh9icqss-rtrmadp-3-soccer-italy-nap-juv-report.jpg 1.5x,https://s2.glbimg.com/n\_XVqiX\_3nn\_wSar4FYy5I-cPUw=/0x0:5406x3041/1080x608/smart/https://i.s3.glbimg.com/v1/AUTH\_bc8228b6673f488aa253bbcb03c80ec5/internal\_photos/bs/2020/G/4/GZIncSSPmVRWLaYiNGIg/2020-06-17t212444z-1091152315-rc29bh9icqss-rtrmadp-3-soccer-italy-nap-juv-report.jpg 2x" title="Foto: (Alberto Lingria/Reuters)"/></picture></div></a></div></div>,<div class="feed-post-metadata"><span class="feed-post-datetime">Há 3 horas</span><span class="feed-post-metadata-section"> futebol italiano </span></div>]

Dentro dessa estrutura, procurando pelas tags corretas, vamos encontrar todas as informações que foram solicitadas. Pela saída anterior podemos ver que a manchete ocupa a posição 2 da lista de conteúdos, logo para guardar a manchete devemos fazer:

In [24]: lista\_noticias[0]**.**contents[1]**.**text**.**replace('"',"")

Out [24]: 'VICE SENHORA'

Para extração do link para notícia, como ele se encontra também na posição 1 da lista, vamos utilizar o método find('a') para localizá-lo e extrair da seguinte forma:

In [25]: lista\_noticias[0]**.**find('a')**.**get('href')

Out [25]: 'https://globoesporte.globo.com/futebol/futebol-internacional/futebol-italiano/jogo/17-06-2020/napoli-juventusita.ghtml'

Para a descrição, como ela pode estar na terceira posição ou em outra tag, vamos ter que testar em ambas e caso não esteja, então retornar None (nulo). Veja a seguir.

In [26]: descricao **=** lista\_noticias[0]**.**contents[2]**.**text

if not descricao:

    descricao **=** noticia**.**find('div', attrs**=**{'class': 'bstn-related'})

    descricao **=** descricao**.**text if descricao else None # Somente acessará a propriedade text caso tenha encontrado ("find")

descricao

Out [26]: 'Napoli vence Juventus nos pênaltis e leva Copa da Itália'

Para extração da seção e do tempo decorrido, vamos acessar primeiro o atributo 'feed-post-metadata' e guardar em uma variável, para em seguida, dentro desse novo subconjunto, localizar os atributos 'feed-post-datetime' e 'feed-post-metadata-section'. Como existe a possibilidade dessa informação não existir, precisamos garantir que somente acessaremos a propriedade text (linhas 6 e 7) caso tenha encontrando ("find"). Veja a seguir:

In [27]: metadados **=** lista\_noticias[0]**.**find('div', attrs**=**{'class':'feed-post-metadata'})

time\_delta **=** metadados**.**find('span', attrs**=**{'class': 'feed-post-datetime'})

secao **=** metadados**.**find('span', attrs**=**{'class': 'feed-post-metadata-section'})

time\_delta **=** time\_delta**.**text if time\_delta else None

secao **=** secao**.**text if secao else None

    print('time\_delta = ', time\_delta)

print('seção = ', secao)

time\_delta =  Há 3 horas

seção =   futebol italiano

Veja que para a notícia 0 extraímos todas as informações solicitadas, mas precisamos extrair de todas, portanto cada extração deve ser feita dentro de uma estrutura de repetição. Para criar um DataFrame com os dados, vamos criar uma lista vazia e a cada iteração apendar uma tupla com as informações extraídas. Com essa lista, podemos criar nosso DataFrame, passando os dados e os nomes das colunas. Veja a seguir:

In [28]: dados **=** []

for noticia in lista\_noticias:

    manchete **=** noticia**.**contents[1]**.**text**.**replace('"',"")

    link **=** noticia**.**find('a')**.**get('href')

    descricao **=** noticia**.**contents[2]**.**text

    if not descricao:

        descricao **=** noticia**.**find('div', attrs**=**{'class': 'bstn-related'})

        descricao **=** descricao**.**text if descricao else None

    metadados **=** noticia**.**find('div', attrs**=**{'class':'feed-post-metadata'})

    time\_delta **=** metadados**.**find('span', attrs**=**{'class': 'feed-post-datetime'})

    secao **=** metadados**.**find('span', attrs**=**{'class': 'feed-post-metadata-section'})

    time\_delta **=** time\_delta**.**text if time\_delta else None

    secao **=** secao**.**text if secao else None

    dados**.**append((manchete, descricao, link, secao, hora\_extracao, time\_delta))

df **=** pd**.**DataFrame(dados, columns**=**['manchete', 'descrição', 'link', 'seção', 'hora\_extração', 'time\_delta'])

df**.**head()

Out [28]:



Vamos tornar nossa entrega mais profissional e transformar a solução em uma classe, assim toda vez que for preciso fazer a extração, basta instanciar um objeto e executar o método de extração.

In [29]: from datetime import datetime

import requests

from bs4 import BeautifulSoup

import pandas as pd

class ExtracaoPortal:

    def \_\_init\_\_(self):

        self**.**portal **=** None

        def extrair(self, portal):

        self**.**portal **=** portal

        texto\_string **=** requests**.**get('https://globoesporte.globo.com/')**.**text

        hora\_extracao **=** datetime**.**now()**.**strftime("%d-%m-%Y %H:%M:%S")

        bsp\_texto **=** BeautifulSoup(texto\_string, 'html.parser')

        lista\_noticias **=** bsp\_texto**.**find\_all('div', attrs**=**{'class':'feed-post-body'})

        dados **=** []

        for noticia in lista\_noticias:

            manchete **=** noticia**.**contents[1]**.**text**.**replace('"',"")

            link **=** noticia**.**find('a')**.**get('href')

            descricao **=** noticia**.**contents[2]**.**text

            if not descricao:

                descricao **=** noticia**.**find('div', attrs**=**{'class': 'bstn-related'})

                descricao **=** descricao**.**text if descricao else None

            metadados **=** noticia**.**find('div', attrs**=**{'class':'feed-post-metadata'})

            time\_delta **=** metadados**.**find('span', attrs**=**{'class': 'feed-post-datetime'})

            secao **=** metadados**.**find('span', attrs**=**{'class': 'feed-post-metadata-section'})

            time\_delta **=** time\_delta**.**text if time\_delta else None

            secao **=** secao**.**text if secao else None

            dados**.**append((manchete, descricao, link, secao, hora\_extracao, time\_delta))

        df **=** pd**.**DataFrame(dados, columns**=**['manchete', 'descrição', 'link', 'seção', 'hora\_extração', 'time\_delta'])

        return df

In [30]:

 df **=** ExtracaoPortal()**.**extrair("https://globoesporte.globo.com/")

df**.**head()

Out [30]:

