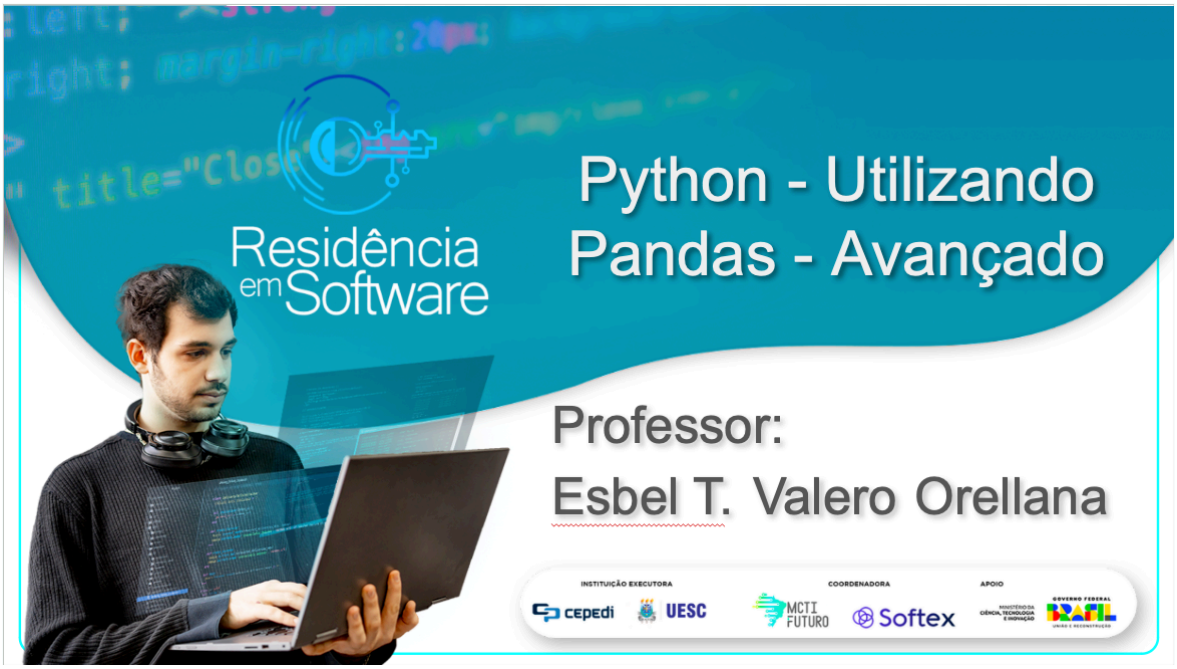
Módulo de Programação Python

Trilha Python - Aula 18: Utilizando Pandas - Avançado



Objetivo: Trabalhar com pacotes e módulos disponíveis em **Python**: **Pandas** avançado. Apresentar recursos do Pandas para trabalhar conjuntos de dados.

Conteúdo: Combinando conjuntos de dados: concatenação de matrizes NumPy. Concatenação Simples com pd.concat. Álgebra Relacional. Categorias de junções. Especificação da chave de mesclagem. Especificando conjunto aritmético para junções. Nomes de colunas sobrepostos: a palavra-chave sufixos.

Combinando datasets

Frequentemente, so realizar análise de dados, se faz necessário combinar de alguma forma diferentes conjuntos de dados. Este tipo operações pode envolver desde uma concatenação muito simples de dois conjuntos de dados diferentes até junções mais complicadas no estilo ao que pode ser feito com banco de dados, que lidam corretamente com quaisquer sobreposições entre os conjuntos de dados.

As classes `Series e DataFrame` foram construídos com este tipo de operação em mente, e **Pandas** inclui funções e métodos que tornam esse tipo de manipulação de dados rápida e direta.

In [1]:
```
import numpy as np
import pandas as pd
print("NumPy: ", np.__version__)
print("Pandas: ", pd.__version__)
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions
4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated.
Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Adva
nced Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions
4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated.
Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Adva
nced Vector Extensions (Intel(R) AVX) instructions.
NumPy:  1.26.2
Pandas:  2.1.4
```

Para entender como estas operações podem ser utilizadas em **Pandas**, vamos começar por implementar uma função que cria um `DataFrame` de um formato particular e qui utilizaremos nos nossos exemplos posteriores.

In [2]:
```
def cria_df(colunas, indices):
    data = {c: [str(c) + str(i) for i in indices] for c in colunas}
    return pd.DataFrame(data, indices)

# Cria um DataFrame com 3 colunas e 3 linhas
cria_df(['alpha', 'beta', 'gamma'], ['A', 'B', 'C'])
```

Out[2]:

	alpha	beta	gamma
A	alphaA	betaA	gammaA
B	alphaB	betaB	gammaB
C	alphaC	betaC	gammaC

Além disso, criaremos uma classe que nos permitirá exibir vários `DataFrames` lado a lado.

O código faz uso do método especial *repr_html*, que o IPython usa para implementar a exibição de objetos.

In [3]:
```
class Display(object):
    """Permite exibir representação HTML de vários objetos"""

    template = """<div style="float: left; padding: 10px;">
    <p style='font-family:"Courier New", Courier, monospace'>{0}</p
    </div>"""
    def __init__(self, *args):
        self.args = args

    def _repr_html_(self):
        return '\n'.join(self.template.format(a, eval(a)._repr_html_
                         for a in self.args)

    def __repr__(self):
        return '\n\n'.join(a + '\n' + repr(eval(a))
                           for a in self.args)
```

In [4]:
```
exe01_DF = cria_df(['alpha', 'beta', 'gamma'], ['A', 'B', 'C'])
exe02_DF = cria_df('ABC', range(3))
Display('exe01_DF', 'exe02_DF')
```

Out[4]:

exe01_DF

	alpha	beta	gamma
A	alphaA	betaA	gammaA
B	alphaB	betaB	gammaB
C	alphaC	betaC	gammaC

exe02_DF

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2

Relembrando concatenação de matrizes NumPy

A concatenação de objetos `Series` e `DataFrame` é muito semelhante à concatenação de arrays **Numpy**, o que pode ser feito através da função `np.concatenate` conforme anteriormente.

Lembre-se de que com esta função você pode combinar o conteúdo de dois ou mais arrays em um único array.

In [5]:
```
# Concatenando três listas num ndarray
l1 = [1, 2, 3]
l2 = [4, 5, 6]
l3 = [7, 8, 9]
np.concatenate([l1, l2, l3])
```

Out[5]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

No exemplo anterior vemos que primeiro argumento da função é uma lista ou uma tupla de arrays a serem concatenados.

Além disso, pode ser utilizada a palavra-chave axis que permite especificar o eixo ao longo do qual o resultado será concatenado.

In [6]:
```
mat1 = np.concatenate([l1, l2, l3])
mat1 = mat1.reshape(3, 3)
mat1
```

Out[6]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [7]:
```
mat2 = mat1.T
mat2
```

Out[7]:
```
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

In [8]:
```
# por padrão axis=0
np.concatenate([mat1, mat2])
```

Out[8]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9],
       [1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

In [9]:
```
np.concatenate([mat1, mat2], axis=1)
```

Out[9]:
```
array([[1, 2, 3, 1, 4, 7],
       [4, 5, 6, 2, 5, 8],
       [7, 8, 9, 3, 6, 9]])
```

Concatenando em Pandas

Pandas tem uma função, `pd.concat()`, que tem uma sintaxe semelhante a `np.concatenate` mas contém uma série de opções adicionais.

```
# Pandas v2.2.4
pandas.concat(objs, *, axis=0, join='outer', ignore_index=False,
              keys=None, levels=None, names=None, verify_integrity=False,
              sort=False, copy=None)

# Numpy v1.26
numpy.concatenate((a1, a2, ...), axis=0, out=None, dtype=None, casting="same_kind")
```

In [10]:
```
ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])
ser2 = pd.Series(['D', 'E', 'F'], index=[4, 5, 6])
pd.concat([ser1, ser2])
```

Out[10]:
```
1    A
2    B
3    C
4    D
5    E
6    F
dtype: object
```

In [11]:
```
df1 = cria_df('AB', [1, 2])
df2 = cria_df('AB', [3, 4])
df12 = pd.concat([df1, df2])
Display('df1', 'df2', 'df12')
```

Out[11]:

df1

	A	B
1	A1	B1
2	A2	B2

df2

	A	B
3	A3	B3
4	A4	B4

df12

	A	B
1	A1	B1
2	A2	B2
3	A3	B3
4	A4	B4

Por padrão, a concatenação ocorre por linha dentro do `DataFrame` (`axix = 0`). Assim como `np.concatenate`, `pd.concat` permite a especificação de um eixo ao longo do qual será feita a concatenação.

In [12]:
```
df3 = cria_df('AB', [0, 1])
df4 = cria_df('CD', [0, 1])
df34 = pd.concat([df3, df4], axis='columns')
Display('df3', 'df4', 'df34')
```

Out[12]:

df3

	A	B
0	A0	B0
1	A1	B1

df4

	C	D
0	C0	D0
1	C1	D1

df34

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1

Índices duplicados

Uma diferença importante entre `np.concatenate` e `pd.concat` é que a concatenação do **Pandas** preserva os índices, mesmo que o resultado tenha índices duplicados!

In [13]:
```
df1 = cria_df('AB', [0, 1])
df2 = cria_df('AB', [2, 3])
Display('df1', 'df2')
```

Out[13]:

df1

	A	B
0	A0	B0
1	A1	B1

df2

	A	B
2	A2	B2
3	A3	B3

In [14]:
```
df2.index = df1.index  # duplicando os índices!
Display('df1', 'df2')
```

Out[14]:

df1

	A	B
0	A0	B0
1	A1	B1

df2

	A	B
0	A2	B2
1	A3	B3

In [15]:
```
df12 = pd.concat([df1, df2])
Display('df1', 'df2', 'df12')
```

Out[15]:

df1

	A	B
0	A0	B0
1	A1	B1

df2

	A	B
0	A2	B2
1	A3	B3

df12

	A	B
0	A0	B0
1	A1	B1
0	A2	B2
1	A3	B3

Repare nos índices repetidos no resultado.

Embora isso seja válido em um `DataFrame`, o resultado geralmente é indesejável. A função `pd.concat()` nos dá algumas maneiras de lidar com este tipo de situação.

Se você quiser simplesmente verificar se os índices no resultado não se sobrepõem, você pode especificar a flag `verify_integrity` definido como `True` (por pdrão ele é `False`). Neste caso a concatenação gerará uma exceção se houver índices duplicados.

In [16]:
```
try:
    pd.concat([df1, df2], verify_integrity=True)
except ValueError as e:
    print("ValueError:", e)
```

```
ValueError: Indexes have overlapping values: Index([0, 1], dtype='
int64')
```

Às vezes, o índice em si não importa e você prefere que ele seja simplesmente ignorado. Esta opção pode ser especificada usando a flag `ignore_index` como `True`. Desta forma a concatenação criará um novo índice inteiro para a `Série` resultante.

In [17]:
```
Display('df1', 'df2', 'pd.concat([df1, df2], ignore_index=True)')
```

Out[17]:

df1

	A	B
0	A0	B0
1	A1	B1

df2

	A	B
0	A2	B2
1	A3	B3

pd.concat([df1, df2], ignore_index=True)

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3

Outra alternativa é usar a opção `keys` para especificar um rótulo para cada uma das fontes de dados. O resultado será uma série indexada hierarquicamente contendo os dados originais.

In [18]:
```
Display('df1', 'df2', "pd.concat([df1, df2], keys=['df1', 'df2'])")
```

Out[18]:

df1

	A	B
0	A0	B0
1	A1	B1

df2

	A	B
0	A2	B2
1	A3	B3

pd.concat([df1, df2], keys=['df1', 'df2'])

		A	B
df1	**0**	A0	B0
	1	A1	B1
df2	**0**	A2	B2
	1	A3	B3

Concatenação com junções

Nos exemplos simples que acabamos de ver, estávamos principalmente concatenando `DataFrame` s com nomes de colunas compartilhadas.

Na prática, dados de diferentes fontes podem ter diferentes conjuntos de nomes de colunas, e a função `pd.concat` oferece diversas opções para lidar com estes casos.

In [19]:
```
df5 = cria_df('ABC', [1, 2])
df6 = cria_df('BCD', [3, 4])
Display('df5', 'df6', 'pd.concat([df5, df6])')
```

Out[19]:

df5

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

df6

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

pd.concat([df5, df6])

	A	B	C	D
1	A1	B1	C1	NaN
2	A2	B2	C2	NaN
3	NaN	B3	C3	D3
4	NaN	B4	C4	D4

Por padrão, as entradas para as quais não há dados disponíveis são preenchidas como valores ausentes.

Para mudar isso, podemos especificar o parâmetro `join` da função `pd.concat` . A junção é uma, por padrão, a união das colunas de entrada (`join='outer'`), mas podemos mudar isso para uma interseção das colunas usando `join='inner'` .

In [21]:
```
Display('df5', 'df6',
        "pd.concat([df5, df6], join='inner')")
```

Out[21]:

df5

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

df6

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

pd.concat([df5, df6], join='inner')

	B	C
1	B1	C1
2	B2	C2
3	B3	C3
4	B4	C4

Álgebra Relacional

Outro recurso essencial oferecido pelo Pandas são suas operações de *join* e *merge* de alto desempenho na memória.

A interface principal para isso é a função `pd.merge` , e veremos alguns exemplos de como isso pode funcionar na prática.

O comportamento implementado em `pd.merge()` é um subconjunto do que é conhecido como *álgebra relacional*, que é um conjunto formal de regras para manipulação de dados relacionais e forma a base conceitual das operações disponíveis na maioria dos bancos de dados.

A força da abordagem da álgebra relacional é que ela propõe diversas operações primitivas, que se tornam os blocos de construção de operações mais complicadas em qualquer conjunto de dados.

Com este léxico de operações fundamentais implementado de forma eficiente em um banco de dados ou outro programa, uma ampla gama de operações compostas bastante complicadas pode ser executada.

O **Pandas** implementa vários desses blocos de construção fundamentais na função `pd.merge()` e no método `join()` de `Series` e `Dataframe` .

Categorias de `join`

A função `pd.merge()` implementa vários tipos de junções:

- *um-para-um*,
- *muitos-para-um*
- *muitos-para-muitos*.

Todos os três tipos de joins são acessados através de uma chamada idêntica à interface `pd.merge()` .

O tipo de junção realizada depende da forma dos dados de entrada.

Junções um-para-um

Talvez o tipo mais simples de expressão de mesclagem seja a junção um para um, que é em muitos aspectos muito semelhante à concatenação em colunas.

Como exemplo concreto, considere os dois `DataFrames` a seguir que contêm informações sobre vários funcionários de uma empresa:

In [34]:
```
df1 = pd.DataFrame({'residente': ['Thiago', 'Rafaela', 'Allana', 'A
                    'form. Esp': ['Eng. Elétrica', 'Eng. Mecânica '
df2 = pd.DataFrame({'residente': ['Rafaela', 'Allana', 'Thiago', 'A
                    'ano_nascimento': [2000, 2004, 1998, 2002]})
Display('df1', 'df2')
```

Out[34]:

df1

	residente	form. Esp
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df2

	residente	ano_nascimento
0	Rafaela	2000
1	Allana	2004
2	Thiago	1998
3	Arthur	2002

Para combinar essas informações em um único `DataFrame` , podemos usar a função `pd.merge()`

In [35]:
```
df3 = pd.merge(df1, df2)
df3
```

Out[35]:

	residente	form. Esp	ano_nascimento
0	Thiago	Eng. Elétrica	1998
1	Rafaela	Eng. Mecânica	2000
2	Allana	Cien. Computação	2004
3	Arthur	Eng. Civil	2002

A função `pd.merge()` identifica que cada `DataFrame` possui uma coluna "residente" e une automaticamente usando esta coluna como chave.

O resultado da fusão é um novo `DataFrame` que combina as informações das duas entradas.

Observe que a ordem das entradas em cada coluna não é necessariamente mantida: neste caso, a ordem da coluna "residente" difere entre `df1` e `df2`, e a função `pd.merge()` leva em conta isso corretamente.

Junções muitos-para-um

Junções muitos-para-um são junções nas quais uma das duas colunas-chave contém entradas duplicadas. Para o caso muitos-para-um, o `DataFrame` resultante preservará essas entradas duplicadas conforme seja apropriado.

In [37]:
```
df4 = pd.DataFrame({'form. Esp': ['Eng. Elétrica', 'Eng. Mecânica '
                    'form. Geral': ['Engenharia', 'Engenharia', 'Co
df5 = pd.merge(df3, df4)
Display('df3', 'df4', 'df5')
```

Out[37]:

df3

	residente	form. Esp	ano_nascimento
0	Thiago	Eng. Elétrica	1998
1	Rafaela	Eng. Mecânica	2000
2	Allana	Cien. Computação	2004
3	Arthur	Eng. Civil	2002

df4

	form. Esp	form. Geral
0	Eng. Elétrica	Engenharia
1	Eng. Mecânica	Engenharia
2	Cien. Computação	Computação
3	Eng. Civil	Engenharia

df5

	residente	form. Esp	ano_nascimento	form. Geral
0	Thiago	Eng. Elétrica	1998	Engenharia
1	Rafaela	Eng. Mecânica	2000	Engenharia
2	Allana	Cien. Computação	2004	Computação
3	Arthur	Eng. Civil	2002	Engenharia

Junções muitos-para-muitos

As junções muitos-para-muitos são um pouco confusas conceitualmente, mas ainda assim são bem definidas.

Se a coluna-chave na matriz esquerda e direita contiver duplicatas, o resultado será uma mesclagem muitos para muitos.

In [38]:
```
df6 = pd.DataFrame({'form. Geral': ['Engenharia', 'Engenharia', 'Co
                    'equipes': ['Inter_eqEng', 'Pyth_eqMista', 'Int
Display('df5', 'df6', "pd.merge(df5, df6)")
```

Out[38]:

df5

	residente	form. Esp	ano_nascimento	form. Geral
0	Thiago	Eng. Elétrica	1998	Engenharia
1	Rafaela	Eng. Mecânica	2000	Engenharia
2	Allana	Cien. Computação	2004	Computação
3	Arthur	Eng. Civil	2002	Engenharia

df6

	form. Geral	equipes
0	Engenharia	Inter_eqEng
1	Engenharia	Pyth_eqMista
2	Computação	Inter_eqComp
3	Computação	Pyth_eqMista

pd.merge(df5, df6)

	residente	form. Esp	ano_nascimento	form. Geral	equipes
0	Thiago	Eng. Elétrica	1998	Engenharia	Inter_eqEng
1	Thiago	Eng. Elétrica	1998	Engenharia	Pyth_eqMista
2	Rafaela	Eng. Mecânica	2000	Engenharia	Inter_eqEng
3	Rafaela	Eng. Mecânica	2000	Engenharia	Pyth_eqMista
4	Arthur	Eng. Civil	2002	Engenharia	Inter_eqEng
5	Arthur	Eng. Civil	2002	Engenharia	Pyth_eqMista
6	Allana	Cien. Computação	2004	Computação	Inter_eqComp
7	Allana	Cien. Computação	2004	Computação	Pyth_eqMista

Estes três tipos de junções podem ser usados com outras ferramentas **Pandas** para implementar uma ampla gama de funcionalidades na hora de preparar os conjuntos de dados que serão utilizados.

Entretanto, na prática, os conjuntos de dados raramente são tão limpos quanto aquele com o qual estamos trabalhando aqui.

A seguir consideraremos algumas das opções fornecidas por `pd.merge()` que permitem ajustar como as operações de junção funcionam.

Especificando a chave

Já vimos o comportamento padrão de `pd.merge()` : ele procura por um ou mais nomes de colunas correspondentes entre as duas entradas e usa isso como chave.

No entanto, muitas vezes os nomes das colunas não combinam tão bem, e `pd.merge()` fornece uma variedade de opções para lidar com isso.

Usando a palavra-chave `on`

Você pode especificar explicitamente o nome da coluna chave usando a palavra-chave `on` , que recebe um nome de coluna ou uma lista de nomes de colunas

In [40]:
```
Display('df1', 'df2', "pd.merge(df1, df2, on='residente')")
```

Out[40]:

df1

	residente	form. Esp
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df2

	residente	ano_nascimento
0	Rafaela	2000
1	Allana	2004
2	Thiago	1998
3	Arthur	2002

`pd.merge(df1, df2, on='residente')`

	residente	form. Esp	ano_nascimento
0	Thiago	Eng. Elétrica	1998
1	Rafaela	Eng. Mecânica	2000
2	Allana	Cien. Computação	2004
3	Arthur	Eng. Civil	2002

Entretanto, esta opção funciona apenas se os dois `DataFrame` s tiverem o nome de coluna `residente` .

As palavras-chave `left_on` e `right_on`

Às vezes você pode desejar mesclar dois conjuntos de dados com nomes de colunas diferentes. Por exemplo, podemos ter um conjunto de dados em que o nome dos residentes é rotulado como "nome" em vez de "residente".

Neste caso, podemos usar as palavras-chave `left_on` e `right_on` para especificar os dois nomes de colunas a serem utilizados.

In [41]:
```
df2 = pd.DataFrame({'nome': ['Rafaela', 'Allana', 'Thiago', 'Arthur
                    'ano_nascimento': [2000, 2004, 1998, 2002]})
Display('df1', 'df2', 'pd.merge(df1, df2, left_on="residente", righ
```

Out[41]:

df1

	residente	form. Esp
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df2

	nome	ano_nascimento
0	Rafaela	2000
1	Allana	2004
2	Thiago	1998
3	Arthur	2002

`pd.merge(df1, df2, left_on="residente", right_on="nome")`

	residente	form. Esp	nome	ano_nascimento
0	Thiago	Eng. Elétrica	Thiago	1998
1	Rafaela	Eng. Mecânica	Rafaela	2000
2	Allana	Cien. Computação	Allana	2004
3	Arthur	Eng. Civil	Arthur	2002

Veja que como resultado temos uma coluna redundante que podemos eliminar, se desejarmos, usando o método `drop()` do `DataFrame` .

In [42]:
```
pd.merge(df1, df2, left_on="residente", right_on="nome").drop('nome
```

Out[42]:

	residente	form. Esp	ano_nascimento
0	Thiago	Eng. Elétrica	1998
1	Rafaela	Eng. Mecânica	2000
2	Allana	Cien. Computação	2004
3	Arthur	Eng. Civil	2002

O método

```
DataFrame.drop(labels=None, *, axis=0, index=None, columns=
None,
               level=None, inplace=False, errors='raise')
```

Elimina rótulos especificados de linhas ou colunas.

Remove linhas ou colunas especificando nomes de rótulos e eixos correspondentes ou especificando diretamente nomes de índices ou colunas. Ao usar um índice múltiplo, os rótulos em níveis diferentes podem ser removidos especificando o nível.

As palavras-chave `left_index e right_index`

Às vezes, em vez de mesclar em uma coluna, você gostaria de mesclar em um índice.

In [43]:
```
Display('df1', 'df2')
```

Out[43]:

df1

	residente	form. Esp
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df2

	nome	ano_nascimento
0	Rafaela	2000
1	Allana	2004
2	Thiago	1998
3	Arthur	2002

In [44]:
```
df1a = df1.set_index('residente')
df2a = df2.set_index('nome')
Display('df1a', 'df2a')
```

Out[44]:

df1a

	form. Esp
residente	
Thiago	Eng. Elétrica
Rafaela	Eng. Mecânica
Allana	Cien. Computação
Arthur	Eng. Civil

df2a

	ano_nascimento
nome	
Rafaela	2000
Allana	2004
Thiago	1998
Arthur	2002

In [45]:
```
df2a.index.name = 'residente'
Display('df1a', 'df2a')
```

Out[45]:

df1a

residente	form. Esp
Thiago	Eng. Elétrica
Rafaela	Eng. Mecânica
Allana	Cien. Computação
Arthur	Eng. Civil

df2a

residente	ano_nascimento
Rafaela	2000
Allana	2004
Thiago	1998
Arthur	2002

Agora você pode usar o índice como chave para mesclagem, especificando os sinalizadores left_index e/ou right_index em `pd.merge()` .

In [47]:
```
Display('df1a', 'df2a',
        "pd.merge(df1a, df2a, left_index=True, right_index=True)")
```

Out[47]:

df1a

residente	form. Esp
Thiago	Eng. Elétrica
Rafaela	Eng. Mecânica
Allana	Cien. Computação
Arthur	Eng. Civil

df2a

residente	ano_nascimento
Rafaela	2000
Allana	2004
Thiago	1998
Arthur	2002

`pd.merge(df1a, df2a, left_index=True, right_index=True)`

residente	form. Esp	ano_nascimento
Thiago	Eng. Elétrica	1998
Rafaela	Eng. Mecânica	2000
Allana	Cien. Computação	2004
Arthur	Eng. Civil	2002

Por conveniência, na classe `DataFrame` se implementam o método `join()` , que executa uma mesclagem cujo padrão é juntar em índices.

In [49]:
```
Display('df1a', 'df2a', 'df1a.join(df2a)')
```

Out[49]:

df1a

residente	form. Esp
Thiago	Eng. Elétrica
Rafaela	Eng. Mecânica
Allana	Cien. Computação
Arthur	Eng. Civil

df2a

residente	ano_nascimento
Rafaela	2000
Allana	2004
Thiago	1998
Arthur	2002

df1a.join(df2a)

residente	form. Esp	ano_nascimento
Thiago	Eng. Elétrica	1998
Rafaela	Eng. Mecânica	2000
Allana	Cien. Computação	2004
Arthur	Eng. Civil	2002

Se você quiser misturar índices e colunas, você pode combinar `left_index` com `right_on` ou `left_on` com `right_index` para obter o comportamento desejado.

In [52]:
```
Display('df1a', 'df2', "pd.merge(df1a, df2, left_index=True, right_
```

Out[52]:

df1a

residente	form. Esp
Thiago	Eng. Elétrica
Rafaela	Eng. Mecânica
Allana	Cien. Computação
Arthur	Eng. Civil

df2

	nome	ano_nascimento
0	Rafaela	2000
1	Allana	2004
2	Thiago	1998
3	Arthur	2002

```
pd.merge(df1a, df2, left_index=True, right_on='nome')
```

	form. Esp	nome	ano_nascimento
2	Eng. Elétrica	Thiago	1998
0	Eng. Mecânica	Rafaela	2000
1	Cien. Computação	Allana	2004
3	Eng. Civil	Arthur	2002

Especificando conjunto aritmético para junções

Em todos os exemplos anteriores, deixamos de lado uma consideração importante ao realizar uma junção: o tipo de conjunto aritmético usado na junção.

Isso surge quando um valor aparece em uma coluna-chave, mas não na outra.

In [53]:
```
#df1 = pd.DataFrame({'residente': ['Thiago', 'Rafaela', 'Allana', '
#                    'form. Esp': ['Eng. Elétrica', 'Eng. Mecânica
df7 = pd.DataFrame({'residente': ['Paulo', 'Allana', 'Myllena', 'Ar
                    'tempo_formado': [1, 2, 2, 5]})
Display('df1', 'df7', 'pd.merge(df1, df7)')
```

Out[53]:

df1

	residente	form. Esp
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df7

	residente	tempo_formado
0	Paulo	1
1	Allana	2
2	Myllena	2
3	Arthur	5

pd.merge(df1, df7)

	residente	form. Esp	tempo_formado
0	Allana	Cien. Computação	2
1	Arthur	Eng. Civil	5

Aqui estamos mesclando dois conjuntos de dados que possuem apenas duas entradas de “residente” em comum.

Por padrão, o resultado contém a *interseção* dos dois conjuntos de entradas. Este comportamento é conhecido como *inner join*.

Podemos definir este comportamento explicitamente usando a palavra-chave `how`, cujo padrão é `"inner"`.

Outras opções para a palavra-chave `how` são `'outer'`, `'left'` e `'right'`. Uma *outer join* retorna uma junção com a união das colunas de entrada e preenche todos os valores ausentes com ausentes.

In [54]:
```
Display('df1', 'df7', "pd.merge(df1, df7, how='outer')")
```

Out[54]:

df1

	residente	form. Esp
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df7

	residente	tempo_formado
0	Paulo	1
1	Allana	2
2	Myllena	2
3	Arthur	5

pd.merge(df1, df7, how='outer')

	residente	form. Esp	tempo_formado
0	Thiago	Eng. Elétrica	NaN
1	Rafaela	Eng. Mecânica	NaN
2	Allana	Cien. Computação	2.0
3	Arthur	Eng. Civil	5.0
4	Paulo	NaN	1.0
5	Myllena	NaN	2.0

O *left join* e o *right join* retornam junções sobre as entradas esquerda e direita, respectivamente.

In [55]:
```
Display('df1', 'df7', "pd.merge(df1, df7, how='left')")
```

Out[55]:

df1

	residente	form. Esp
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df7

	residente	tempo_formado
0	Paulo	1
1	Allana	2
2	Myllena	2
3	Arthur	5

pd.merge(df1, df7, how='left')

	residente	form. Esp	tempo_formado
0	Thiago	Eng. Elétrica	NaN
1	Rafaela	Eng. Mecânica	NaN
2	Allana	Cien. Computação	2.0
3	Arthur	Eng. Civil	5.0

In [56]:
```
Display('df1', 'df7', "pd.merge(df1, df7, how='right')")
```

Out[56]:

df1

	residente	form. Esp
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df7

	residente	tempo_formado
0	Paulo	1
1	Allana	2
2	Myllena	2
3	Arthur	5

pd.merge(df1, df7, how='right')

	residente	form. Esp	tempo_formado
0	Paulo	NaN	1
1	Allana	Cien. Computação	2
2	Myllena	NaN	2
3	Arthur	Eng. Civil	5

Sobreposição de nomes de colunas

Finalmente, você pode acabar em um caso onde seus dois `DataFrame` s de entrada têm nomes de colunas conflitantes.

In [58]:
```
df8 = pd.DataFrame({'residente': ['Thiago', 'Rafaela', 'Allana', 'A
                    'formação': ['Eng. Elétrica', 'Eng. Mecânica ',
df9 = pd.DataFrame({'residente': ['Thiago', 'Rafaela', 'Allana', 'A
                    'formação': ['Engenharia', 'Engenharia ', 'Compu
Display('df8', 'df9', 'pd.merge(df8, df9, on="residente")')
```

Out[58]:

df8

	residente	formação
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df9

	residente	formação
0	Thiago	Engenharia
1	Rafaela	Engenharia
2	Allana	Computação
3	Arthur	Engenharia

`pd.merge(df8, df9, on="residente")`

	residente	formação_x	formação_y
0	Thiago	Eng. Elétrica	Engenharia
1	Rafaela	Eng. Mecânica	Engenharia
2	Allana	Cien. Computação	Computação
3	Arthur	Eng. Civil	Engenharia

Como a saída teria dois nomes de coluna conflitantes, a função anexa automaticamente um sufixo `_x` ou `_y` para tornar as colunas de saída diferentes. Se esses padrões forem inadequados, é possível especificar um sufixo personalizado usando a palavra-chave suffixes.

In [61]:
```
Display('df8', 'df9', 'pd.merge(df8, df9, on="residente", suffixes=
```

Out[61]:

df8

	residente	formação
0	Thiago	Eng. Elétrica
1	Rafaela	Eng. Mecânica
2	Allana	Cien. Computação
3	Arthur	Eng. Civil

df9

	residente	formação
0	Thiago	Engenharia
1	Rafaela	Engenharia
2	Allana	Computação
3	Arthur	Engenharia

```
pd.merge(df8, df9, on="residente", suffixes=["_espec",
"_geral"])
```

	residente	formação_espec	formação_geral
0	Thiago	Eng. Elétrica	Engenharia
1	Rafaela	Eng. Mecânica	Engenharia
2	Allana	Cien. Computação	Computação
3	Arthur	Eng. Civil	Engenharia

Um caso de estudo

As operações de mesclagem e junção surgem com mais frequência ao combinar dados de diferentes fontes. Aqui consideraremos um exemplo de alguns dados de desenvolvimento regional com países da America do Sul.

Primeiramente vamos dar uma olhada em como funciona a função `read_csv()` do **Pandas**.

```
pandas.read_csv(filepath_or_buffer, *, sep=_NoDefault.no_default, delimiter=None,
                header='infer', names=_NoDefault.no_default, index_col=None,
                usecols=None, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0,
                nrows=None, na_values=None, keep_default_na=True, na_filter=True,
                verbose=False, skip_blank_lines=True, parse_dates=None, infer_datetime_format=_NoDefault.no_default, keep_date_col=False, date_parser=_NoDefault.no_default, date_format=None, dayfirst=False,
                cache_dates=True, iterator=False, chunksize=None, compression='infer',
                thousands=None, decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None, on_bad_lines='error',
                delim_whitespace=False, low_memory=True, memory_map=False,
                float_precision=None, storage_options=None, dtype_backend=_NoDefault.no_default)
```

In [91]:
```
hci = pd.read_csv("datasets/Data_Index.csv")
#hci_Meta = pd.read_csv("datasets/Series_Metadata_Index.csv")
educação = pd.read_csv("datasets/Data_Edu.csv")
#educação_Meta = pd.read_csv("datasets/Series_Metadata_Edu.csv")
```

In [92]:
```
Display('hci.head()', 'educação.head()')
```

Out[92]:

`hci.head()`

	Series Name	Series Code	Country Name	Country Code	2010 [YR2010]	2014 [YR2014]	2018 [YR2
0	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Argentina	ARG	0.588557064533234	..	0.617366731
	Human Capital Index						

1	Index (HCI) (scale 0-1)	HD.HCI.OVRL	Bolivia	BOL		..	..
2	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Brazil	BRA	0.532952487468719	..	0.545723736286
3	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Chile	CHL	0.62635999917984	..	0.665265142917
4	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Colombia	COL	0.580249488353729	..	0.598653852939

```
educação.head()
```

	Country Name	Country Code	Series	Series Code	2010 [YR2010]	2014 [YR2014]	2018 [YR2018]
0	Argentina	ARG	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	161473	160786	166717
1	Bolivia	BOL	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	..	..	..
2	Brazil	BRA	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	9201677	7758773	6306355
3	Chile	CHL	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	181946	..	..
4	Colombia	COL	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	1284400	1191477	1054679

Escolhemos dois datasets obtidos no [The World Banck (https://databank.worldbank.org/home.aspx)](https://databank.worldbank.org/home.aspx). O primeiro mostra como evoluiu i *Human Capital Index* nos respectivos países entre 2010 e 2018. O ano 2014 não apresenta dados para nenhum dos países elencados.

O segundo dataset mostra a população analfabeta entre 25 e 64 anos. Aqui tem dados de 2010, 2014 e 2018.

Uma vez analisados os dois conjuntos de dados podemos, por exemplo, excluir os dados de 2014 já que não podemos fazer relação entre as informações dos datasets neste ano.

In [93]:
```
hci.drop("2014 [YR2014]", axis=1, inplace=True)
hci.head()
```

Out[93]:

	Series Name	Series Code	Country Name	Country Code	2010 [YR2010]	2018 [YR2018]
0	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Argentina	ARG	0.588557064533234	0.61736673116684
1	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Bolivia	BOL	..	..
2	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Brazil	BRA	0.532952487468719	0.545723736286163
3	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Chile	CHL	0.62635999917984	0.665265142917633
4	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Colombia	COL	0.580249488353729	0.598653852939606

In [94]:
```
educação.drop("2014 [YR2014]", axis=1, inplace=True)
educação.head()
```

Out[94]:

	Country Name	Country Code	Series	Series Code	2010 [YR2010]	2018 [YR2018]
0	Argentina	ARG	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	161473	166717
1	Bolivia	BOL	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	..	..
2	Brazil	BRA	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	9201677	6306355
3	Chile	CHL	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	181946	..
4	Colombia	COL	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	1284400	1054679

Vamos unificar os nomes das colunas nas duas tabelas. Neste caso podemos mudar o nome da coluna `Series` da tabela `educação` .

In [96]:
```
educação.rename(columns={'Series': 'Series Name'}, inplace=True)
educação.head()
```

Out[96]:

	Country Name	Country Code	Series Name	Series Code	2010 [YR2010]	2018 [YR2018]
0	Argentina	ARG	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	161473	166717
1	Bolivia	BOL	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	..	..
2	Brazil	BRA	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	9201677	6306355
3	Chile	CHL	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	181946	..
4	Colombia	COL	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	1284400	1054679

Podemos remover a coluna Country Code que está repetida nas duas tabela.

In [98]:
```
educação.drop("Country Code", axis=1, inplace=True)
educação.head()
```

Out[98]:

	Country Name	Series Name	Series Code	2010 [YR2010]	2018 [YR2018]
0	Argentina	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	161473	166717
1	Bolivia	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	..	..
2	Brazil	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	9201677	6306355
3	Chile	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	181946	..
4	Colombia	Illiterate population, 25-64 years, both sexes...	UIS.ILLPOP.AG25T64	1284400	1054679

Agora vamos juntar os dados das duas tabelas em uma nova.

In [99]:
```
tudo = pd.merge(hci, educação, on="Country Name")
tudo.head()
```

Out[99]:

	Series Name_x	Series Code_x	Country Name	Country Code	2010 [YR2010]_x	2018 [YR2018]_x	S Na
0	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Argentina	ARG	0.588557064533234	0.61736673116684	Illit popul 2 years, se
1	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Bolivia	BOL	..	..	Illit popul 2 years, se
2	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Brazil	BRA	0.532952487468719	0.545723736286163	Illit popul 2 years, se
3	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Chile	CHL	0.62635999917984	0.665265142917633	Illit popul 2 years, se
4	Human Capital Index (HCI) (scale 0-1)	HD.HCI.OVRL	Colombia	COL	0.580249488353729	0.598653852939606	Illit popul 2 years, se