

# UNIDADE 3

## Infraestrutura para Manipulação e Visualização de Dados

Disciplina: Tópicos Especiais III (DEC7553)

Prof. Alexandre L. Gonçalves

E-mail: [a.l.goncalves@ufsc.br](mailto:a.l.goncalves@ufsc.br)



## ■ Tratamento de Dados: Junção, Combinação e Reformatação

- Para muitas aplicações, os dados podem estar espalhados em diferentes arquivos ou bancos de dados e em formatos variados;
- Neste caso, torna-se necessário a combinação, junção e reorganização dos dados;



## ■ Indexação Hierárquica

- Recurso importante do Pandas, pois permite ter vários níveis de índices em um eixo;
- Provê uma maneira de trabalhar com dados de dimensões mais elevadas em um formato de dimensões menores ou reduzidas.

# ■ Indexação Hierárquica

- Criando uma Série com uma lista de lista (ou de arrays) como índice.

```
data = pd.Series(np.random.randn(9),  
                 index=[['a', 'a', 'a', 'b', 'b', 'c', 'c', 'd', 'd'],  
                       [1, 2, 3, 1, 3, 1, 2, 2, 3]])
```

data

```
a 1 -0.204708  
   2  0.478943  
   3 -0.519439  
b 1 -0.555730  
   3  1.965781  
c 1  1.393406  
   2  0.092908  
d 2  0.281746  
   3  0.769023  
dtype: float64
```

## ■ Indexação Hierárquica

- De modo geral, tem-se uma Série com um MultiIndex como índice. Isto possibilita trabalhar com os vários rótulos da estrutura para acessar os dados.

`data.index`

```
MultiIndex(levels=[['a', 'b', 'c', 'd'], [1, 2, 3]],  
            codes=[[0, 0, 0, 1, 1, 2, 2, 3, 3], [0, 1, 2, 0, 2, 0, 1, 1, 2]])
```

# ■ Indexação Hierárquica

- Com um objeto hierarquicamente indexado, torna-se possível a indexação parcial possibilitando selecionar subconjuntos de dados.

- `data['b']`  
1 -0.555730  
3 1.965781  
dtype: float64

- `data['a':'c']`  
a 1 -0.204708  
2 0.478943  
3 -0.519439  
b 1 -0.555730  
3 1.965781  
c 1 1.393406  
2 0.092908  
dtype: float64

- `data.loc[['b', 'd']]`  
b 1 -0.555730  
3 1.965781  
d 2 0.281746  
3 0.769023  
dtype: float64

## ■ Indexação Hierárquica

- A seleção é também possível a partir de um nível mais interno.

```
data.loc[:, 2]
```

```
a    0.478943  
c    0.092908  
d    0.281746  
dtype: float64
```

## ■ Indexação Hierárquica

- A indexação hierárquica desempenha um papel importante na reformatação dos dados e nas operações baseadas em grupos.
- A reorganização dos dados em um DataFrame pode ser realizado através do método *unstack*.

`data.unstack()`

	1	2	3
a	-0.204708	0.478943	-0.519439
b	-0.555730	NaN	1.965781
c	1.393406	0.092908	NaN
d	NaN	0.281746	0.769023



# ■ Indexação Hierárquica

- A operação inversa de *unstack* é *stack*.

```
data.unstack().stack()
```

```
a 1 -0.204708  
   2  0.478943  
   3 -0.519439  
b 1 -0.555730  
   3  1.965781  
c 1  1.393406  
   2  0.092908  
d 2  0.281746  
   3  0.769023  
dtype: float64
```

# ■ Indexação Hierárquica

- Em um DataFrame, qualquer eixo pode ser um índice hierárquico.

```
frame = pd.DataFrame(np.arange(20).reshape((4, 5)),  
                      index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],  
                      columns=[['SC', 'SC', 'RS', 'RS', 'PR'],  
                               ['Verde', 'Vermelho', 'Verde', 'Vermelho', 'Verde']])
```

frame

		SC		RS		PR
		Verde	Vermelho	Verde	Vermelho	Verde
a	1	0	1	2	3	4
	2	5	6	7	8	9
b	1	10	11	12	13	14
	2	15	16	17	18	19

# ■ Indexação Hierárquica

- Os níveis hierárquicos podem ter nomes.

```
frame.index.names = ['Chave1', 'Chave2']  
frame.columns.names = ['Estado', 'Cor']  
frame
```

		Estado	SC		RS		PR
		Cor	Verde	Vermelho	Verde	Vermelho	Verde
Chave1	Chave2						
a	1		0	1	2	3	4
	2		5	6	7	8	9
b	1		10	11	12	13	14
	2		15	16	17	18	19

## ■ Indexação Hierárquica

- É possível ainda selecionar grupos de colunas.

```
frame['SC']
```

	cor	Verde	Vermelho
Chave1	Chave2		
a	1	0	1
	2	5	6
b	1	10	11
	2	15	16

## ■ Reorganizando e Ordenando Níveis

- Eventualmente torna-se necessário reorganizar os níveis em um eixo ou ordenar os dados de acordo com os valores em um nível específico.
- Isso pode ser realizado utilizando o método *swaplevel* que aceita dois números ou nomes de níveis e devolve um novo objeto com os níveis alterados.

```
frame.swaplevel('Chave1', 'Chave2')
```

	estado	SC		RS		PR
	cor	Verde	Vermelho	Verde	Vermelho	Verde
Chave2	Chave1					
1	a	0	1	2	3	4
2	a	5	6	7	8	9
1	b	10	11	12	13	14
2	b	15	16	17	18	19

## ■ Reorganizando e Ordenando Níveis

- Já o método `sort_index()` ordena os dados utilizando somente um nível.

```
frame.sort_index(level=1)
```

```
frame.swaplevel(0, 1).sort_index(level=0)
```

	estado	SC		RS		PR
		Verde	Vermelho	Verde	Vermelho	Verde
Chave2	Chave1					
1	a	0	1	2	3	4
	b	10	11	12	13	14
2	a	5	6	7	8	9
	b	15	16	17	18	19

## ■ Estatísticas de Resumo por Nível

- Muitas estatísticas descritivas ou de resumo em DataFrame e em Series possuem uma opção *level*;
- Com ela pode-se especificar o nível de acordo com o qual se deseja realizar uma agregação, em um eixo em particular.

```
frame.sum(level='Chave2')
```

estado	SC		RS		PR
cor	Verde	Vermelho	Verde	Vermelho	Verde
Chave2					
1	10	12	14	16	18
2	20	22	24	26	28

## ■ Estatísticas de Resumo por Nível

```
frame.sum(level='cor', axis=1)
```

	cor	Verde	Vermelho
Chave1	Chave2		
a	1	6	4
	2	21	14
b	1	36	24
	2	51	34



## ■ Indexando com as Colunas de um DataFrame

- Não é incomum utilizar uma ou mais colunas de um DataFrame como índice de linha. De modo alternativo, talvez seja necessário mover o índice das linhas para as colunas do DataFrame. Considere o DataFrame abaixo:

```
frame = pd.DataFrame({'a': range(7), 'b': range(7, 0, -1),  
                      'c': ['Um', 'Um', 'Um', 'Dois', 'Dois',  
                           'Dois', 'Dois'],  
                      'd': [0, 1, 2, 0, 1, 2, 3]})
```

frame

## ■ Indexando com as Colunas de um DataFrame

	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>
<b>0</b>	0	7	Um	0
<b>1</b>	1	6	Um	1
<b>2</b>	2	5	Um	2
<b>3</b>	3	4	Dois	0
<b>4</b>	4	3	Dois	1
<b>5</b>	5	2	Dois	2
<b>6</b>	6	1	Dois	3

## ■ Indexando com as Colunas de um DataFrame

- A função `set_index` de `DataFrame` cria um novo `DataFrame` usando uma ou mais de suas colunas como índice.

```
frame2 = frame.set_index(['c', 'd'])
```

```
frame2
```

		a	b
c	d		
Um	0	0	7
	1	1	6
	2	2	5
Dois	0	3	4
	1	4	3
	2	5	2
	3	6	1

## ■ Indexando com as Colunas de um DataFrame

- Por padrão, as colunas são removidas do DataFrame, embora possam ser mantidas.

```
frame.set_index(['c', 'd'], drop=False)
```

		a	b	c	d
c	d				
Um	0	0	7	Um	0
	1	1	6	Um	1
	2	2	5	Um	2
Dois	0	3	4	Dois	0
	1	4	3	Dois	1
	2	5	2	Dois	2
	3	6	1	Dois	3

## ■ Indexando com as Colunas de um DataFrame

- `reset_index`, por outro lado, faz o inverso de `set_index`; os níveis dos índices hierárquicos são passados para as colunas.

```
frame2.reset_index()
```

	<b>c</b>	<b>d</b>	<b>a</b>	<b>b</b>
<b>0</b>	Um	0	0	7
<b>1</b>	Um	1	1	6
<b>2</b>	Um	2	2	5
<b>3</b>	Dois	0	3	4
<b>4</b>	Dois	1	4	3
<b>5</b>	Dois	2	5	2
<b>6</b>	Dois	3	6	1

## ■ Combinando e Mesclando Conjuntos de Dados

□ Os dados contidos em objetos do Pandas podem ser combinados de várias maneiras:

- `pandas.merge`: conecta linhas em DataFrames com base em uma ou mais chaves. É similar a operação de junção (join) de bancos de dados relacionais;
- `pandas.concat`: concatena ou “empilha” objetos ao longo de um eixo;
- `pandas.combine_first`: permite combinar dados que se sobrepõem a fim de preencher valores ausentes em um objeto com valores de outro objeto.

## ■ Junções no DataFrame no Estilo de Banco de Dados

- Operações de mesclagem (*merge*) ou de junção (*join*) combinam conjuntos de dados associando linhas por meio de uma ou mais chaves:

```
df1 = pd.DataFrame({'Chave': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],  
                    'Coluna 1': range(7)})
```

```
df2 = pd.DataFrame({'Chave': ['a', 'b', 'd'],  
                    'Coluna 2': range(3)})
```

```
df1
```

```
df2
```

## ■ Junções no DataFrame no Estilo de Banco de Dados

df1

	Chave	Coluna 1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

df2

	Chave	Coluna 2
0	a	0
1	b	1
2	d	2



## ■ Junções no DataFrame no Estilo de Banco de Dados

- O exemplo a seguir é de junção de muitos-para-um (*many-to-one*);
- Os dados em df1 possuem várias linhas de rótulos **a** e **b**, enquanto df2 tem apenas uma linha para cada valor na coluna chave.

`pd.merge(df1, df2)`

	Chave	Dados1	Dados2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

## ■ Junções no DataFrame no Estilo de Banco de Dados


- No exemplo anterior nenhuma coluna foi especificada para realizar a junção;
- Se essa informação não for especificada, *merge* utilizará como chaves os nomes das colunas que se sobrepõem;
- Contudo, a indicação explícita da coluna é uma boa prática.

## ■ Junções no DataFrame no Estilo de Banco de Dados

- Se os nomes das colunas forem diferentes em cada objeto, as mesmas devem ser especificadas separadamente.

```
df3 = pd.DataFrame({'Chave E': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],  
                    'Coluna 1': range(7)})  
df4 = pd.DataFrame({'Chave D': ['a', 'b', 'd'],  
                    'Coluna 2': range(3)})  
pd.merge(df3, df4, left_on='Chave E', right_on='Chave D')
```

	Chave E	Coluna 1	Chave D	Coluna 2
0	b	0	b	1
1	b	1	b	1
2	b	6	b	1
3	a	2	a	0
4	a	4	a	0
5	a	5	a	0

- 
- Junções no DataFrame no Estilo de Banco de Dados
    - Nos exemplos anteriores, os valores 'c' e 'd' e os dados associados estão ausentes no resultado;
    - Por padrão, *merge* executa uma junção do tipo *inner* (interna);
    - As chaves no resultado representam a intersecção ou o conjunto comum que se encontra nas duas tabelas;
    - Outras opções possíveis são 'left', 'right' e 'outer'.

## ■ Junções no DataFrame no Estilo de Banco de Dados

- A junção externa (outer join) efetua a união das chaves combinando as junções tanto a esquerda quanto a direita.

```
pd.merge(df1, df2, how='outer')
```

	Chave	Coluna 1	Coluna 2
0	b	0.0	1.0
1	b	1.0	1.0
2	b	6.0	1.0
3	a	2.0	0.0
4	a	4.0	0.0
5	a	5.0	0.0
6	c	3.0	NaN
7	d	NaN	2.0

## ■ Junções no DataFrame no Estilo de Banco de Dados

- Outra possibilidade são as mesclagens em conjuntos do tipo muitos-para-muitos.

```
df1 = pd.DataFrame({'Chave': ['b', 'b', 'a', 'c', 'a', 'b'],  
                    'Coluna 1': range(6)})  
df2 = pd.DataFrame({'Chave': ['a', 'b', 'a', 'b', 'd'],  
                    'Coluna 2': range(5)})  
  
df1  
df2  
pd.merge(df1, df2, on='Chave', how='left')
```

## ■ Junções no DataFrame no Estilo de Banco de Dados

	Chave	Dados1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

	Chave	Dados2
0	a	0
1	b	1
2	a	2
3	b	3
4	d	4

	Chave	Dados1	Dados2
0	b	0	1.0
1	b	0	3.0
2	b	1	1.0
3	b	1	3.0
4	a	2	0.0
5	a	2	2.0
6	c	3	NaN
7	a	4	0.0
8	a	4	2.0
9	b	5	1.0
10	b	5	3.0

## ■ Junções no DataFrame no Estilo de Banco de Dados

- Junções de muitos-para-muitos formam o produto cartesiano das linhas. Como existem três linhas 'b' no DataFrame da esquerda e duas no da direita, há seis linhas 'b' no resultado.

```
pd.merge(df1, df2, how='inner')
```

	Chave	Dados1	Dados2
0	b	0	1
1	b	0	3
2	b	1	1
3	b	1	3
4	b	5	1
5	b	5	3
6	a	2	0
7	a	2	2
8	a	4	0
9	a	4	2



## ■ Junções no DataFrame no Estilo de Banco de Dados

- Para um merge com várias chaves, é necessário passar uma lista de nomes de coluna.

```
left = pd.DataFrame({'Chave1': ['foo', 'foo', 'bar'],  
                    'Chave2': ['one', 'two', 'one'],  
                    'ValorE': [1, 2, 3]})  
right = pd.DataFrame({'Chave1': ['foo', 'foo', 'bar', 'bar'],  
                     'Chave2': ['one', 'one', 'one', 'two'],  
                     'ValorR': [4, 5, 6, 7]})  
pd.merge(left, right, on=['Chave1', 'Chave2'], how='outer')
```

## ■ Junções no DataFrame no Estilo de Banco de Dados

Chave1 Chave2 ValorE

```
0  foo  one    1
1  foo  two    2
2  bar  one    3
```

Chave1 Chave2 ValorR

```
0  foo  one    4
1  foo  one    5
2  bar  one    6
3  bar  two    7
```

	Chave1	Chave2	ValorE	ValorR
0	Foo	one	1.0	4.0
1	foo	one	1.0	5.0
2	foo	two	2.0	NaN
3	bar	one	3.0	6.0
4	bar	two	NaN	7.0

- Junções no DataFrame no Estilo de Banco de Dados
  - Uma última questão a ser considerada em operações de merge é o tratamento dos nomes de coluna que se sobrepõem;
  - Para isso existe a opção *suffixes* que permite especificar *strings* a serem concatenadas nos nomes que se sobrepõem, nos objetos DataFrame à esquerda ou à direita.

```
pd.merge(left, right, on='Chave1')
```

```
pd.merge(left, right, on='Chave1', suffixes=('_left', '_right'))
```

## ■ Junções no DataFrame no Estilo de Banco de Dados

	Chave1	Chave2_x	ValorE	Chave2_y	ValorR
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

	Chave1	Chave2_left	ValorE	Chave2_right	ValorR
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

## ■ Realizando o Merge com Base no Índice

- Em alguns casos, a(s) chave(s) do *merge* (fusão) em um DataFrame serão encontradas em seu índice. Nesta situação, pode-se passar `left_index=True` ou `right_index=True` (ou ambos).

```
left1 = pd.DataFrame({'Chave': ['a', 'b', 'a', 'a', 'b', 'c'],  
                      'Valor': range(6)})  
right1 = pd.DataFrame({'Grupo_val': [3.5, 7]}, index=['a', 'b'])  
print(left1)  
print(right1)  
pd.merge(left1, right1, left_on='Chave', right_index=True)
```

## ■ Realizando o Merge com Base no Índice

Chave Valor

0 a 0

1 b 1

2 a 2

3 a 3

4 b 4

5 c 5

Grupo\_val

a 3.5

b 7.0

	Chave	Valor	Grupo_val
<b>0</b>	a	0	3.5
<b>2</b>	a	2	3.5
<b>3</b>	a	3	3.5
<b>1</b>	b	1	7.0
<b>4</b>	b	4	7.0

## ■ Realizando o Merge com Base no Índice

□ Como o método padrão de *merge* consiste em fazer uma intersecção das chaves de junção, pode-se, de maneira alternativa, compor a união delas fazendo uma junção externa.

□ `pd.merge(left1, right1, left_on='Chave', right_index=True, how='outer')`

	Chave	Valor	Grupo_val
0	A	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

## ■ Concatenando ao Longo de um Eixo

- Outro tipo de operação de combinação de dados é chamado de concatenação, vinculação (*binding*) ou empilhamento (*stacking*).
- A função `concatenate` do NumPy é capaz de fazer isso com arrays NumPy.

```
arr = np.arange(12).reshape((3, 4))  
arr  
np.concatenate([arr, arr], axis=1)
```

```
array([[ 0,  1,  2,  3,  0,  1,  2,  3],  
       [ 4,  5,  6,  7,  4,  5,  6,  7],  
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```



## ■ Concatenando ao Longo de um Eixo

- Outra possibilidade é o método `concat` do Pandas que permite concatenar vários conjuntos de dados.

```
s1 = pd.Series([0, 1], index=['a', 'b'])  
s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])  
s3 = pd.Series([5, 6], index=['f', 'g'])
```

```
pd.concat([s1, s2, s3])
```

```
a    0
```

```
b    1
```

```
c    2
```

```
d    3
```

```
e    4
```

```
f    5
```

```
g    6
```

```
dtype: int64
```

## ■ Concatenando ao Longo de um Eixo

- Por padrão, concat atua em axis=0 gerando outra Série. Se axis=1 for utilizado, o resultado será um DataFrame (axis=1 são as colunas).

```
pd.concat([s1, s2, s3], axis=1, sort=False)
```

	0	1	2
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

## ■ Combinando Dados com Sobreposição

- Existe outra situação de combinação de dados que não pode ser expressa nem como uma operação de *merge*, nem como uma operação de concatenação;
- Pode-se ter dois conjuntos de dados cujos índices se sobreponham de forma total ou parcial;
- Para tal, utiliza-se a função *where* do NumPy, que executa o equivalente a uma expressão *if-else*.

## ■ Combinando Dados com Sobreposição

```
a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],  
              index=['f', 'e', 'd', 'c', 'b', 'a'])
```

```
b = pd.Series(np.arange(len(a), dtype=np.float64),  
              index=['f', 'e', 'd', 'c', 'b', 'a'])
```

```
print(a)
```

```
print(b)
```

```
np.where(pd.isnull(a), b, a)
```

```
f  NaN  
e   2.5  
d  NaN  
c   3.5  
b   4.5  
a  NaN  
dtype: float64
```

```
f   0.0  
e   1.0  
d   2.0  
c   3.0  
b   4.0  
a   5.0  
dtype: float64
```

```
array([0. , 2.5, 2. , 3.5, 4.5, 5. ])
```

## ■ Combinando Dados com Sobreposição

- Um *Series* tem um método *combine\_first* que executa o equivalente ao método *where*.

```
a.combine_first(b)
```

```
f    0.0  
e    2.5  
d    2.0  
c    3.5  
b    4.5  
a    5.0  
dtype: float64
```

## ■ Combinando Dados com Sobreposição

- Em um DataFrame a mesma operação é realizada coluna a coluna.

```
df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],  
                    'b': [np.nan, 2., np.nan, 6.],  
                    'c': range(2, 18, 4)})  
df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],  
                    'b': [np.nan, 3., 4., 6., 8.]})  
print(df1)  
print(df2)  
df1.combine_first(df2)
```

## ■ Combinando Dados com Sobreposição

	a	b	c
0	1.0	NaN	2
1	NaN	2.0	6
2	5.0	NaN	10
3	NaN	6.0	14

	a	b
0	5.0	NaN
1	4.0	3.0
2	NaN	4.0
3	3.0	6.0
4	7.0	8.0

	a	b	c
<b>0</b>	1.0	NaN	2.0
<b>1</b>	4.0	2.0	6.0
<b>2</b>	5.0	4.0	10.0
<b>3</b>	3.0	6.0	14.0
<b>4</b>	7.0	8.0	NaN

## ■ Conclusão

- A partir dos conceitos básicos do Pandas para importação, limpeza e reorganização de dados, pode-se seguir na direção da visualização de dados.





# **Bons Estudos!**