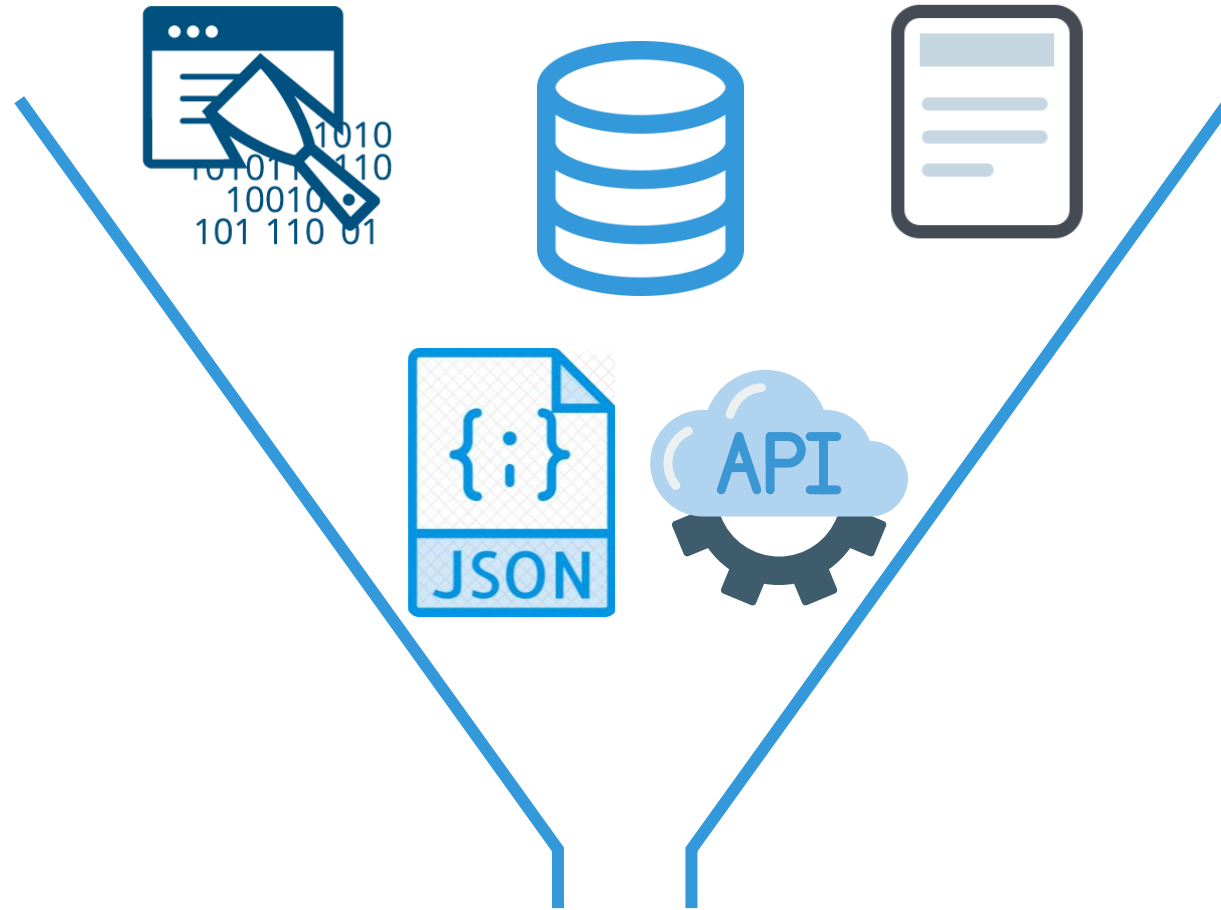


Pandas, Fontes de dados e Agregação



Algumas importantes fontes de Dados

- Arquivo CSV (ou XLS/XLSX)
- Arquivo JSON
 - API REST
- Bancos de Dados
 - Via linguagem SQL
- Web Scrapy
 - Coleta/"Raspagem" de dados de página web
 - ✓ automatizado via programação (robô)

CSV e JSON

CSV (Comma-Separated Value)

```
one,two,three  
1,2,3  
4,3,2
```



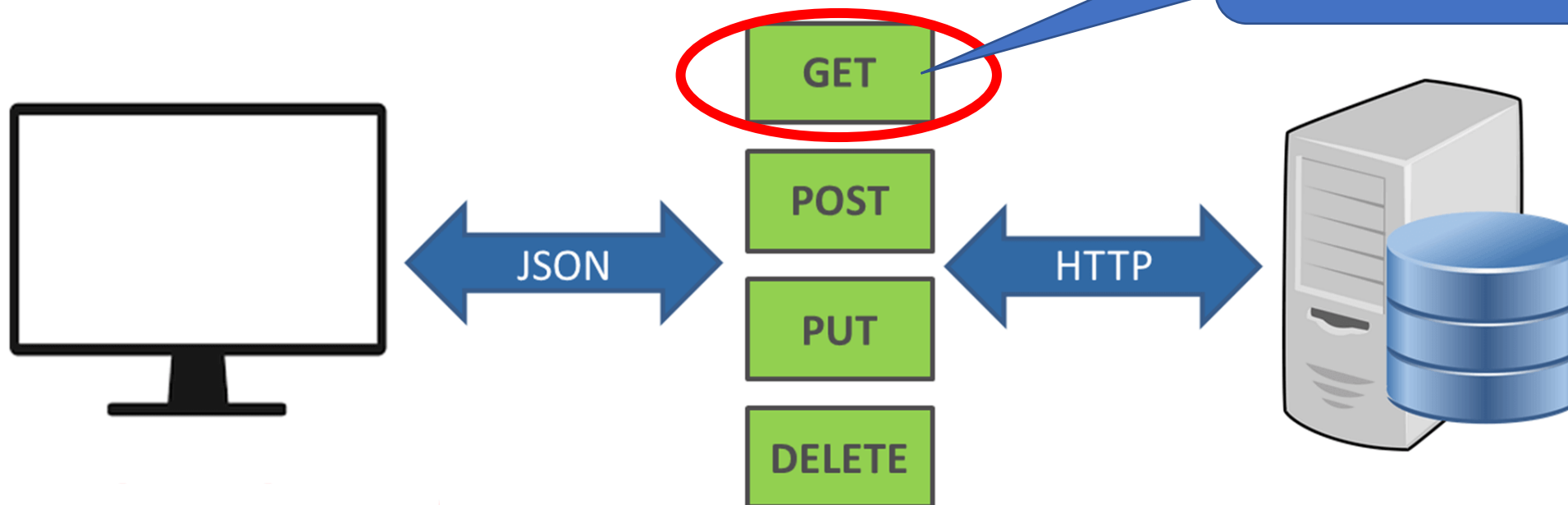
JSON (JavaScript Object Notation)

```
[{"one":"1","two":"2","three":"3"},  
{"one":"4","two":"3","three":"2"}]
```

- TSV (Separado por Tabulação)
- Ou outro separador (Ex.: ponto e vírgula)

API REST

Método mais importante
pra ciência de dados



Cliente envia uma requisição

Métodos HTTP

Servidor envia uma resposta

- POST e PUT: Enviam dados
- GET: Solicita dado

Exemplo de requisição GET: <https://www.servicos.gov.br/api/v1/servicos/9029>

SQL (Structured Query Language)

- Um curso completo de banco de dados e SQL
 - Está fora do escopo deste curso.
 - ✓ Teremos uma breve Introdução.
 - **CONCENTRE-SE NESSES POUCOS MINUTOS!!**
 - ✓ Não tente repetir os comandos enquanto o professor explica!!
- Linguagem padronizada de acesso a Bancos de Dados
 - Algumas de instruções: **SELECT**, **UPDATE**, **DELETE**, **INSERT**
- Para executar uma Query SQL
 - Você precisa conectar ao servidor de banco de dados
- Exemplo simplificado:
 - https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all

Tabela com Linhas e Colunas


CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
17	Drachenblut Delikatessend	Sven Ottlieb	Walserweg 21	Aachen	52066	Germany
25	Frankenversand	Peter Franken	Berliner Platz 43	München	80805	Germany
39	Königlich Essen	Philip Cramer	Maubelstr. 90	Brandenburg	14776	Germany
44	Lehmanns Marktstand	Renate Messner	Magazinweg 7	Frankfurt a.M.	60528	Germany
52	Morgenstern Gesundkost	Alexander Feuer	Heerstr. 22	Leipzig	04179	Germany
56	Ottilies Käseladen	Henriette Pfalzheim	Mehrheimerstr. 369	Köln	50739	Germany

Exemplo de Queries

- `SELECT * from Customers`
 - Selecciona todas as colunas (*) da tabela Customers
- `SELECT * FROM Customers WHERE Country="Germany";`
 - Selecciona todas as colunas (*) filtrando por registros
 - ✓ onde (where) Country é igual a Germany
- `SELECT CustomerID FROM Customers WHERE Country="Germany";`
 - Selecciona a coluna CustomerID filtrando por registros
 - ✓ onde (where) Country é igual a Germany

Seleção colunas e Filtro de linhas

SELECT {....}



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
17	Drachenblut Delikatessend	Sven Ottlieb	Walserweg 21	Aachen	52066	Germany
25	Frankenversand	Peter Franken	Berliner Platz 43	München	80805	Germany
39	Königlich Essen	Philip Cramer	Maubelstr. 90	Brandenburg	14776	Germany
44	Lehmanns Marktstand	Renate Messner	Magazinweg 7	Frankfurt a.M.	60528	Germany
52	Morgenstern Gesundkost	Alexander Feuer	Heerstr. 22	Leipzig	04179	Germany
56	Ottilies Käseladen	Henriette Pfalzheim	Mehrheimerstr. 369	Köln	50739	Germany



WHERE {....}

Group by (Agregação)

OrderDetailID	OrderID	ProductID	Quantity
1	10248	1	2
2	10248	2	10
3	10248	7	5
4	10249	4	5
5	10249	1	4
6	10250	2	5
7	10250	1	6
8	10250	4	15

SELECT ProductID,
sum(Quantity) as Qtd
FROM OrderDetails
GROUP BY ProductID;

ProductID	Qtd
1	12
2	15
4	20
7	5

SELECT ProductID, sum(Quantity) as Qtd FROM OrderDetails group by ProductID;

Prática no Jupyter Notebook (15 min)

- Revise o notebook de teoria, testando pequenas variações; ou
- Faça os exercícios da aula (referente ao conteúdo já visto – até o exercício 3.3).

Descartando valores faltantes (NA ou NaN)

- Aceita o argumento `inplace=True`

```
In [15]: from numpy import nan as NA
```

```
In [16]: data = pd.Series([1, NA, 3.5, NA, 7])
```

```
In [17]: data.dropna()
```

```
Out[17]:
```

```
0    1.0
```

```
2    3.5
```

```
4    7.0
```

```
In [18]: data[data.notnull()]
```

```
Out[18]:
```

```
0    1.0
```

```
2    3.5
```

```
4    7.0
```



Equivalentes

Preenchendo valores faltantes

- `fillna` também aceita o argumento `inplace=True`

```
In [27]: df = pd.DataFrame(np.random.randn(7, 3))
```

```
In [33]: df.fillna(0)
```

```
Out[33]:
```

	0	1	2
0	-0.204708	0.000000	0.000000
1	-0.555730	0.000000	0.000000
2	0.092908	0.000000	0.769023
3	1.246435	0.000000	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

```
In [34]: df.fillna({1: 0.5, 2: 0})
```

```
Out[34]:
```

	0	1	2
0	-0.204708	0.500000	0.000000
1	-0.555730	0.500000	0.000000
2	0.092908	0.500000	0.769023
3	1.246435	0.500000	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

Preenchendo valores faltantes

- `fillna` possui um método de preenchimento `ffill`

```
In [37]: df = pd.DataFrame(np.random.randn(6, 3))
```

```
In [38]: df.iloc[2:, 1] = NA
```

```
In [39]: df.iloc[4:, 2] = NA
```

```
In [40]: df
```

```
Out[40]:
```

	0	1	2
0	0.476985	3.248944	-1.021228
1	-0.577087	0.124121	0.302614
2	0.523772	NaN	1.343810
3	-0.713544	NaN	-2.370232
4	-1.860761	NaN	NaN
5	-1.265934	NaN	NaN

```
In [41]: df.fillna(method='ffill')  
Out[41]:
```

	0	1	2
0	0.476985	3.248944	-1.021228
1	-0.577087	0.124121	0.302614
2	0.523772	0.124121	1.343810
3	-0.713544	0.124121	-2.370232
4	-1.860761	0.124121	-2.370232
5	-1.265934	0.124121	-2.370232

Remover duplicatas

```
In [45]: data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],  
.....:                      'k2': [1, 1, 2, 3, 3, 4, 4]})
```

```
In [46]: data
```

```
Out[46]:
```

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

```
In [47]: data.duplicated()
```

```
Out[47]:
```

0	False
1	False
2	False
3	False
4	False
5	False
6	True

dtype: bool

```
In [48]: data.drop_duplicates()
```

```
Out[48]:
```

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

Indexação Hierárquica

- Possibilita mais de um nível de indexação num eixo

```
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]])
```

```
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
```

Filtro com lista

```
In [14]: data.loc[['b', 'd']]  
Out[14]:  
b  1  -0.555730  
   3   1.965781  
d  2   0.281746  
   3   0.769023
```

Filtro no 2º Nível

```
In [15]: data.loc[:, 2]  
Out[15]:  
a    0.478943  
c    0.092908  
d    0.281746
```

Resumo estatístico por nível

state		Ohio		Colorado
color		Green	Red	Green
key2	key1			
1	a	0	1	2
	b	6	7	8
2	a	3	4	5
	b	9	10	11

```
In [27]: frame.sum(level='key2')
```

```
Out[27]:
```

state	Ohio		Colorado
color	Green	Red	Green
key2			
1	6	8	10
2	12	14	16

merge (fundir/juntar)

- A chave de junção (identificador único) foi inferida
 - a partir do contexto da interseção entre as colunas
 - ✓ Também pode ser especificada com o argumento **on** (Ex.: on='key')

```
In [37]: df1
```

```
Out[37]:
```

	data1	key
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	a
6	6	b

```
In [38]: df2
```

```
Out[38]:
```

	data2	key
0	0	a
1	1	b
2	2	d

```
In [39]: pd.merge(df1, df2)
```

```
Out[39]:
```

	data1	key	data2
0	0	b	1
1	1	b	1
2	6	b	1
3	2	a	0
4	4	a	0
5	5	a	0

join (fundir/juntar)

- Semelhante ao merge, mas a chave de junção é
 - o índice do DataFrame

```
In [70]: left2
```

```
Out[70]:
```

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

```
In [71]: right2
```

```
Out[71]:
```

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

```
In [73]: left2.join(right2, how='outer')
```

```
Out[73]:
```

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
b	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	13.0	14.0

join (fundir/juntar)

- Com how='left' somente os registros do dataframe da esquerda
 - aparecem no resultado

```
In [70]: left2
```

```
Out[70]:
```

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

```
In [71]: right2
```

```
Out[71]:
```

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

```
left2.join(right2, how='left')
```

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
c	3.0	4.0	9.0	10.0
e	5.0	6.0	13.0	14.0

Maneiras de Armazenar vs Analisar os dados

Melhor para Armazenar

	Aluno	Disciplina	Objetiva	Discursiva
0	AlunoA	Portugues	8.5	6
1	AlunoA	Matematica	7.5	6.5
2	AlunoB	Geografia	9	7.5
3	AlunoB	História	10	7

Melhor para Analisar

Disciplina	Geografia	História	Matematica	Portugues
Aluno				
AlunoA	NaN	NaN	7.5	8.5
AlunoB	9	10	NaN	NaN

Reshaping / Pivoting (Pivotar)

- Método pivot

- 3 argumentos: **index**, **columns**, **values**

- ✓ `df.pivot(index='Prova', columns='Disciplina', values='Objetiva')`

- a função `melt()` faz a operação de despivotar

	Aluno	Disciplina	Objetiva	Discursiva
0	AlunoA	Portugues	8.5	6
1	AlunoA	Matematica	7.5	6.5
2	AlunoB	Geografia	9	7.5
3	AlunoB	História	10	7

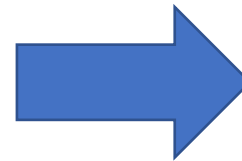
	Disciplina	Geografia	História	Matematica	Portugues
Aluno					
AlunoA		NaN	NaN	7.5	8.5
AlunoB		9	10	NaN	NaN

Pivotar

E quando houver valores repetidos ?

- Pivotar com o mesmo método pivot() gera exceção
 - Neste caso, use o método pivot_table
 - ✓ mean é a métrica padrão de cálculo sobre a de agregação

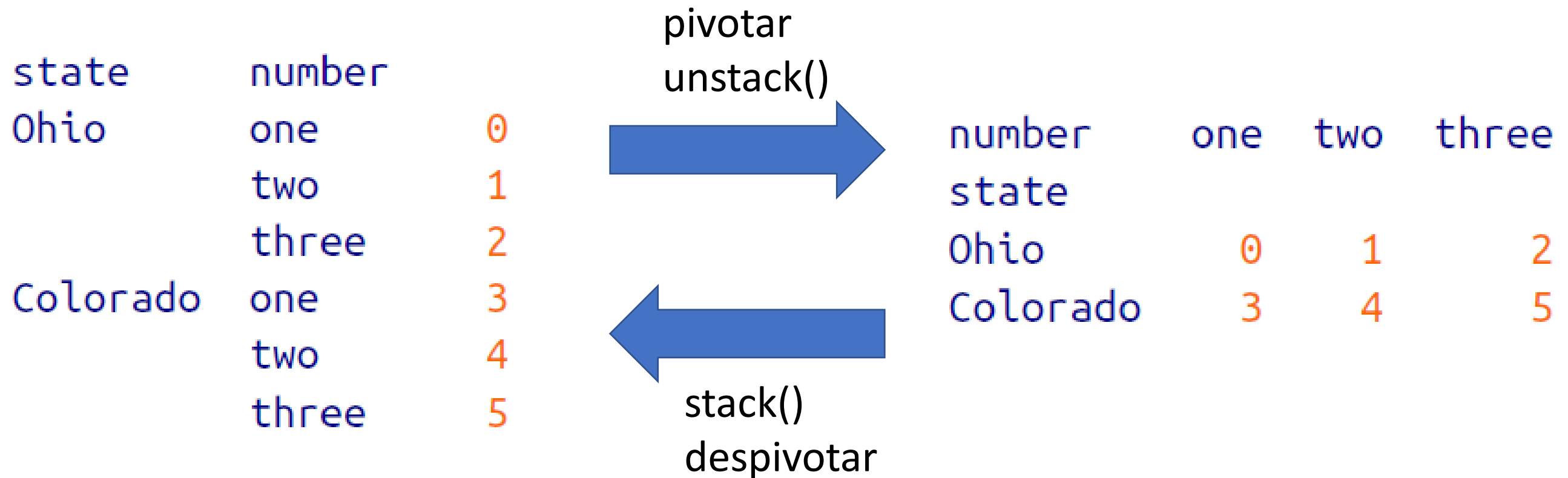
	Aluno	Disciplina	Objetiva	Discursiva
0	AlunoA	Portugues	8.5	6.0
1	AlunoA	Matematica	7.5	6.5
2	AlunoA	Geografia	9.0	7.5
3	AlunoA	Geografia	10.0	7.0
4	AlunoA	História	9.0	8.0
5	AlunoB	Portugues	8.5	8.5
6	AlunoB	Matematica	7.5	7.5
7	AlunoB	Geografia	9.0	9.0
8	AlunoB	História	10.0	10.0



Disciplina	Geografia	História	Matematica	Portugues
Aluno				
AlunoA	9.5	9.0	7.5	8.5
AlunoB	9.0	10.0	7.5	8.5

Reshaping / Pivoting com Índice Hierárquico

- Método stack/unstack (Pivotar com índice hierárquico)
 - stack = empilhar



Prática no Jupyter Notebook

- Faça o restante dos exercícios da aula;
- Há exercícios extra.