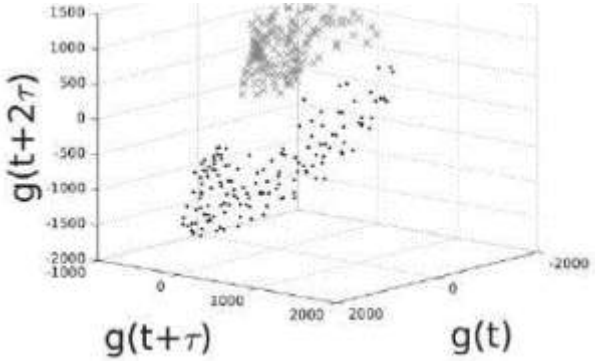
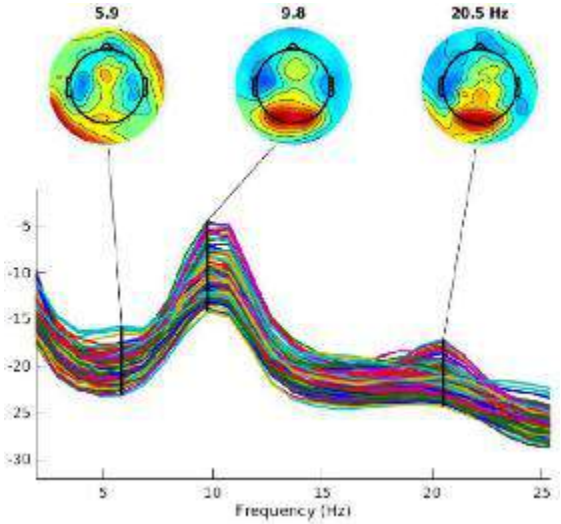




UNICAMP

Instituto de Computação

MC102: Algoritmos e Programação de Computadores (turmas 4,5,6,7)
Extra – Trabalhando com tabelas: pacote Pandas

ARTHUR VALENCIO

Pós-doutorando IC/Unicamp

Fapesp CEPID NeuroMat

Campinas, 17 de Maio de 2020

>>Instalando e utilizando

1. Pandas costuma estar incluso no pacote **Anaconda**. Porém se em sua versão não estiver, basta abrir o **prompt de comando** (Windows) ou **terminal** (Linux) e digitar:

   ```
   conda install pandas
   ```

2. Alternativamente, pode-se instalar através do recurso **pip**. Para isso, basta abrir o **prompt de comando** (Windows) ou **terminal** (Linux) e digitar:

   ```
   pip3 install pandas
   ```

3. Para utilizar o pacote pandas, em geral importamos da seguinte forma:

   ```
   import pandas as pd
   ```

>>Tipos de objetos

Os principais tipos de objetos que o Pandas trabalha são:

- *Series*: sequências com 1 dimensão, por exemplo uma série temporal ou sequência de RNA
- *DataFrame*: estruturas de tabela, isto é com 2 dimensões

Em ambos os casos, os dados são indexados, isto é, existe um valor/texto que indica a primeira/segunda/terceira/n-ésima linha e o mesmo para as colunas.

>>Tipos de objetos

```
import pandas as pd
import numpy as np
```

```
s=pd.Series(range(5,10))
```

$$s = \begin{bmatrix} 0 & 5 \\ 1 & 6 \\ 2 & 7 \\ 3 & 8 \\ 4 & 9 \end{bmatrix}$$

índices

valores

>>Tipos de objetos

```
import pandas as pd
import numpy as np

s=pd.Series(range(5,10))

df1=pd.DataFrame(np.random.randomn(5,4), columns=['a','b','c','d'])
```

$$df1 = \begin{bmatrix} & a & b & c & d \\ 0 & & & & \\ 1 & & & & \\ 2 & & & & \\ 3 & & & & \\ 4 & & & & \end{bmatrix}$$

Nomes atribuídos para as colunas

índices das linhas

>>Tipos de objetos

```
import pandas as pd
import numpy as np

s=pd.Series(range(5,10))

df1=pd.DataFrame(np.random.randomn(5,4), columns=['a','b','c','d'])

df2=pd.DataFrame(np.random.rand(3,5), index=[15.2,17.3,20.5])
```

$$df1 = \begin{bmatrix} & 0 & 1 & 2 & 3 & 4 \\ 15.2 & & & & & \\ 17.3 & & & & & \\ 20.5 & & & & & \end{bmatrix}$$

>>Tipos de objetos

```
import pandas as pd
import numpy as np

s=pd.Series(range(5,10))

df1=pd.DataFrame(np.random.randomn(5,4), columns=['a','b','c','d'])

df2=pd.DataFrame(np.random.rand(3,5), index=[15.2,17.3,20.5])

df3= pd.DataFrame({'a':[1,1.5,3], 'b':[3,5,2], 'c':['bla', 'blabla', 'bla'*3]})
```

$$\begin{bmatrix} & a & b & c \\ 0 & 1 & 3 & bla \\ 1 & 1.5 & 5 & blabla \\ 2 & 3 & 2 & blablabla \end{bmatrix}$$

>>Tipos de objetos

Também é comum trabalhar com variáveis representando tempo e variáveis representando categorias

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=5)
```

Cria um objeto *DatetimeIndex:*
[2020-05-15, 2020-05-16, 2020-05-17, 2020-05-18, 2020-05-19]

Posso usar isso, por exemplo, como índice das linhas em uma *Series* ou *DataFrame*

>>Tipos de objetos

Também é comum trabalhar com variáveis representando tempo e variáveis representando categorias

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=5)

comorbidades=pd.Categorical(["pressão alta", "diabetes", "outra",
                             "não há"])
```

Cria objeto comorbidades com as 4 categorias que definimos

>>Visualização rápida

Pandas foi construído pensando na análise de conjuntos muito grandes de dados. Assim, existem algumas ferramentas que permitem só visualizar um trecho dos dados, para saber do que eles constituem, sem precisar imprimir toda a tabela

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=50)
df=pd.DataFrame(np.random.rand(50,10), index=datas)
```

`df.head()` → Mostra as 5 primeiras linhas, junto com os índices de linha e coluna

`df.tail(3)` → Mostra as 3 últimas linhas, junto com os índices de linha e coluna

>>Visualização rápida

Pandas foi construído pensando na análise de conjuntos muito grandes de dados. Assim, existem algumas ferramentas que permitem só visualizar um trecho dos dados, para saber do que eles constituem, sem precisar imprimir toda a tabela

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=50)
df=pd.DataFrame(np.random.rand(50,10), index=datas)
```

`df.head()` → Mostra as 5 primeiras linhas, junto com os índices de linha e coluna

`df.tail(3)` → Mostra as 3 últimas linhas, junto com os índices de linha e coluna

>>Conversão para numpy

A conversão de um objeto de Pandas para NumPy pode ser feita através da função to_numpy(). Somente os valores são copiados, e os índices não.

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=50)
df=pd.DataFrame(np.random.rand(50,10), index=datas)

dados_numpy=df.to_numpy()
```

>>Estatísticas rápidas

Um único comando, describe(), lhe fornece as estatísticas mais utilizadas, calculadas para cada coluna dos seus dados de DataFrame

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=50)
df=pd.DataFrame(np.random.rand(50,10), index=datas)
```

df.describe() →

	0	1	2	...	7	8	9
count	50.0000	50.00000	50.00000	...	50.00000	50.00000	50.00000
mean	0.483801	0.522020	0.542103	...	0.522749	0.497735	0.490762
std	0.289410	0.281000	0.308155	...	0.308249	0.262450	0.285114
min	0.006347	0.013854	0.001195	...	0.022700	0.021504	0.005443
25%	0.242506	0.328788	0.326779	...	0.228950	0.290372	0.243922
50%	0.481804	0.547512	0.552628	...	0.513654	0.551229	0.448929
75%	0.812335	0.738834	0.812781	...	0.799341	0.697840	0.724995
max	0.978875	0.969799	0.997039	...	0.968961	0.999500	0.996496

>>Seleção de colunas

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=50)
df=pd.DataFrame(np.random.rand(50,4), index=datas,columns=['a','b','c','d'])
```

```
A=df.a
B=df['a']
```

→ Formas alternativas de se retornar a mesma coluna 'a' do *DataFrame*

→ Tanto A quanto B são *Series* com o mesmo índice das linhas que havia no *DataFrame* original

>>Seleção de linhas

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=50)
df=pd.DataFrame(np.random.rand(50,4), index=datas,columns=['a','b','c','d'])

A=df[0:5]
B=df['2020-05-15':'2020-05-19']
```

Formas alternativas de se retornar o mesmo intervalo de linhas do *DataFrame*

Tanto A quanto B são *DataFrames* com os mesmos índices das linhas e colunas que havia no original

>>Seleção condicional

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=50)
df=pd.DataFrame(np.random.rand(50,4), index=datas,columns=['a','b','c','d'])
```

```
A=df[df['a']>0.5]
```

→ Retorna um *DataFrame* apenas com as linhas cujos valores na coluna 'a' são maiores do que 0.5
A é um *DataFrames* com os mesmos índices das linhas e colunas que havia no original df

>>Seleção condicional

```
import pandas as pd
import numpy as np

datas=pd.date_range("2020-05-15", periods=50)
df=pd.DataFrame(np.random.rand(50,4), index=datas,columns=['a','b','c','d'])
```

```
A=df[df['a']>0.5]
```

→ Retorna um *DataFrame* apenas com as linhas cujos valores na coluna 'a' são maiores do que 0.5
A é um *DataFrames* com os mesmos índices das linhas e colunas que havia no original df

>>Dados faltantes

O Pandas costuma completar dados faltantes com Not-a-Number (NaN) do NumPy (np.nan), e, por padrão, eles não entram nos cálculos. Porém, isso pode ser alterado:

#excluir linhas com NaN:

```
A=df.dropna(how="any")
```

#substituir NaN por algum valor numérico:

```
A=df.fillna(value=5)
```

#fazer o teste booleano em todos os elementos para verificar se é NaN:

```
pd.isna(df)
```

>>Concatenação de dados

#adicionando linhas novas:

```
A=pd.concat( (df1, df2) )
```

#adicionando colunas novas:

```
A=pd.concat((df1, df2) ,axis=1)
```

>>Tabelas dinâmicas (Pivot tables)

Vamos primeiro construir a nossa tabela de dados original:

```
df = pd.DataFrame({'A': ['one', 'one', 'two', 'three'] * 3,
                   'B': ['A', 'B', 'C'] * 4,
                   'C': ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 2,
                   'D': np.random.randn(12),
                   'E': np.random.randn(12)})
```

df=

	A	B	C	D	E
0	one	A	foo	-1.202872	0.047609
1	one	B	foo	-1.814470	-0.136473
2	two	C	foo	1.018601	-0.561757
3	three	A	bar	-0.595447	-1.623033
4	one	B	bar	1.395433	0.029399
5	one	C	bar	-0.392670	-0.542108
6	two	A	foo	0.007207	0.282696
7	three	B	foo	1.928123	-0.087302
8	one	C	foo	-0.055224	-1.575170
9	one	A	bar	2.395985	1.771208
10	two	B	bar	1.552825	0.816482
11	three	C	bar	0.166599	1.100230

>>Tabelas dinâmicas (Pivot tables)

Agora podemos fazer a tabela dinâmica:

```
df = pd.DataFrame({'A': ['one', 'one', 'two', 'three'] * 3,
                   'B': ['a', 'b', 'c'] * 4,
                   'C': ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 2,
                   'D': np.random.randn(12),
                   'E': np.random.randn(12)})
df_pivot=pd.pivot_table(df,values="D",index=["A", "B"],columns="C")
```

df_pivot=

C:		bar	foo
A	B		
one	a	2.395985	-1.202872
	b	1.395433	-1.814470
	c	-0.392670	-0.055224
three	a	-0.595447	NaN
	b	NaN	1.928123
	c	0.166599	NaN
two	a	NaN	0.007207
	b	1.552825	NaN
	c	NaN	1.018601

>>Leitura/Escrita de arquivos

CSV:

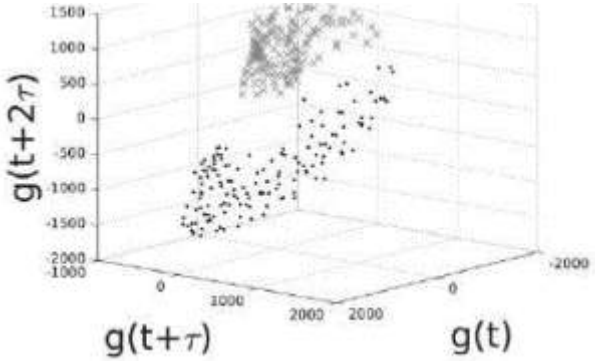
- **Escrita:**
  df.to_csv('minha_tabela.csv')
- **Leitura:**
  pd.read_csv('minha_tabela.csv')

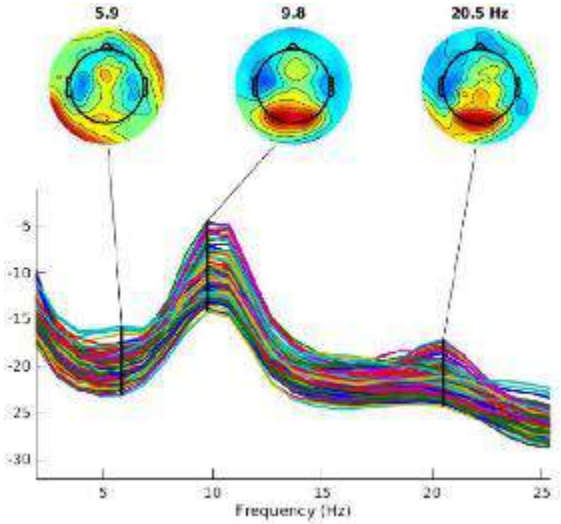
Excel (xlsx):

- **Escrita:**
  df.to_excel('minha_tabela.xlsx', sheet_name='Planilha 1')
- **Leitura:**
  pd.read_excel('minha_tabela.xlsx', 'Planilha 1', index_col=None, na_values=['NA'])

HDF5 (Big Data):

- **Escrita:**
  pd.hdf('minha_tabela.h5', 'df')
- **Leitura:**
  pd.read_hdf('minha_tabela.h5', 'df')





UNICAMP

Instituto de Computação

That´s all folks!

arthur_valencio@physics.org

http://www.arthurvalencio.com/mc102

http://www.ic.unicamp.br/~mc102