Python Numpy e Matplotlib
Fundamentos para Análise de Dados

Eduardo Destefani Stefanato[1]*
Vitor Souza Premoli Pinto de Oliveira[1]*

[1]Universidade Federal do Espírito Santo

Inteligência Artificial Aplicada em Imagens

Sumário

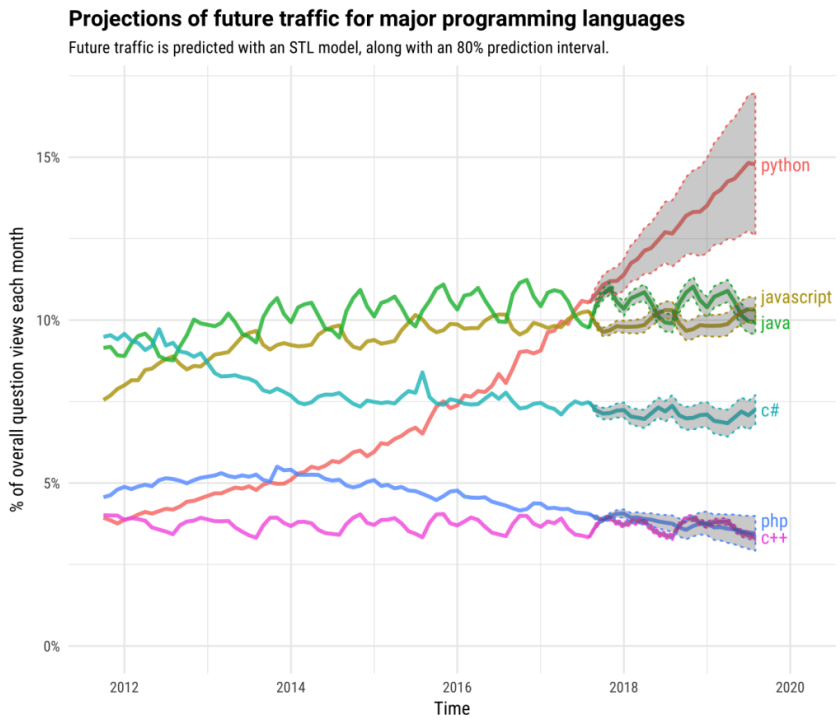
- Introdução – Linguagem Python.....3
  - Oque é Python?.....4
  - Módulos Python.....5
- Biblioteca e Módulos Python – Numpy.....6
  - O que é o Numpy e para que serve?.....7
  - Atributos e aplicações de um *array*.....8
  - Métodos estatísticos.....9
  - Métodos numéricos..... 10
- Biblioteca e Módulos Python – Matplotlib.....14
  - O que é o Matplotlib e para que serve?.....15
- Referências.....20

INTRODUÇÃO

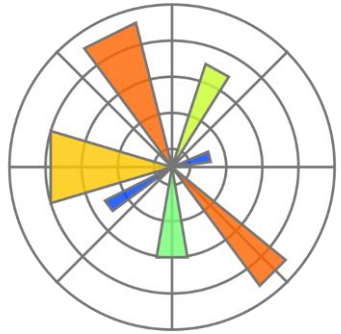
Linguagem Python

O que é Python?

Python é uma linguagem de programação de alto nível, interpretada, de script, interativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991, mas alcançou maior popularidade somente 2 décadas depois.



Por ser uma linguagem Open Source, o uso dessa linguagem é difundido entre pesquisadores de várias áreas. Tal linguagem de programação é conhecida pela sua simplicidade e facilidade de aprendizado, essa conclusão indica que o uso de Python e suas ferramentas, especialmente Numpy e Matplotlib, são extremamente vantajosas para pesquisa em Física.

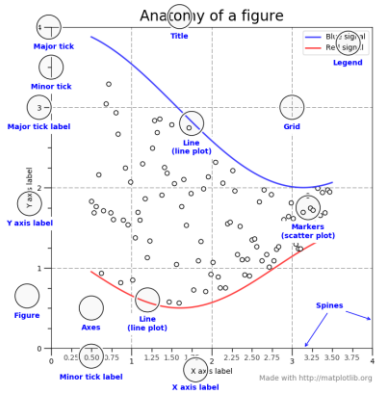


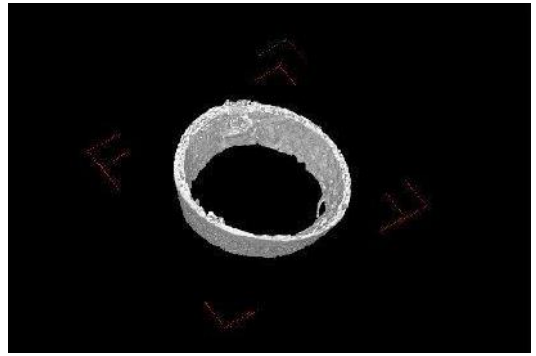
matplotlib

Módulos Python

O forte da linguagem Python é sua grande riqueza de bibliotecas, chamadas de módulos, que podem ser oficiais ou da comunidade. Um módulo Python é um arquivo com código-fonte Python, com definições de dados e funções, normalmente com certa finalidade. Atualmente Python tem centenas de milhares de módulos, a maioria da comunidade, vide site PiPy, que em 21/07/2021 listava 316.970 módulos/pacotes Python.

Logo, um programa Python tipicamente importa módulos com as devidas funcionalidades para resolver determinados problemas.





BIBLIOTECAS E MÓDULOS PYTHON

Numpy

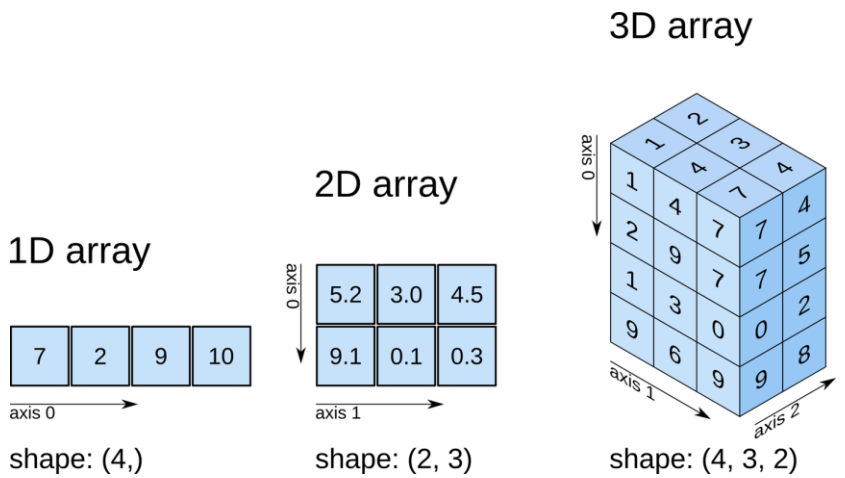
O que é o Numpy e para que serve?

Numpy é uma biblioteca Python que reúne vários pacotes e funções direcionadas para *arrays* multidimensionais. O Numpy foi desenvolvido para computação numérica em alto desempenho, logo, possui otimizações no seu código-fonte para aproveitar todo o potencial do hardware. O Numpy também é conhecido como *computação orientada a arrays.*

Mas afinal, o que são *arrays*?

- *Arrays* são, basicamente, uma estrutura e organização de dados alinhados e divididos em linhas e colunas de maneira uni, bi, tri e n-dimensional;

- Os dados podem ser:
  - Pixels de uma imagem (escala de cinza ou cor);
  - Sinais discretos e/ou contínuos de um sensor, simulação ou experimento;
  - Dados 3D referentes a cada eixo como os dados oriundos de uma ressonância magnética;
  - E entre outros.



Atributos e aplicações de um *array*

```
import numpy as np
import matplotlib.pyplot as plt
```

```
img = plt.imread('dc_metro.png'); print('Temos um %s: \n\n %s \n\ncom shape (dimensão) igual à %s, portanto uma matriz 2D.'
%(type(img), img, np.shape(img)))
```

```
Temos um <class 'numpy.ndarray'>:

 [[0.6039216  0.6431373  0.6784314  ... 0.59607846 0.5921569  0.63529414]
 [0.6666667  0.654902   0.654902   ... 0.654902   0.64705884 0.6392157 ]
 [0.68235296 0.6784314  0.6666667  ... 0.61960787 0.57254905 0.5764706 ]
 ...
 [0.14509805 0.15294118 0.16078432 ... 0.1254902  0.1254902  0.1254902 ]
 [0.14117648 0.15686275 0.16470589 ... 0.12156863 0.1254902  0.12156863]
 [0.14117648 0.15294118 0.16470589 ... 0.12156863 0.1254902  0.12156863]]

com shape (dimensão) igual à (461, 615), portanto uma matriz 2D.
```

Métodos estatísticos

Funções Built-in do Numpy. Funções e métodos que permitem trabalhar e analisar os *arrays* (sua estrutura de dados) dentro do Numpy.

- Estatística básica:
  - Média;
  - Desvio Padrão;
  - Mediana;
  - Máximo;
  - Mínimo;
  - Etc.

```
import numpy as np

# array de 100 valores aleatórios de [0 a 1) :
arr = np.array(np.random.rand(100), dtype='float32')

print('Média = {}\nMediana = {}\nDesvio Padrão = {}\nMáximo = {}\nMínimo = {}'
.format(np.mean(arr), np.median(arr), np.std(arr), np.max(arr), np.min(arr)))
```

```
Média = 0.4543342888355255
Mediana = 0.4197880029678345
Desvio Padrão = 0.2962743043899536
Máximo = 0.9850571751594543
Mínimo = 0.002677689539268613
```

Algoritmos numéricos

Também existem métodos algébricos e algoritmos Built-in no Numpy. Ou seja, métodos bem usuais para análise de dados que permitem trabalhar e analisar os *arrays* (sua estrutura de dados) dentro do Numpy, entregando uma nova estrutura de dados para análise gráfica posteriormente.

São bons exemplos:

- FFT (Fast Furrier Transform);
- IFFT (Inverse Fast Furrier Transform);

$$x[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}}$$

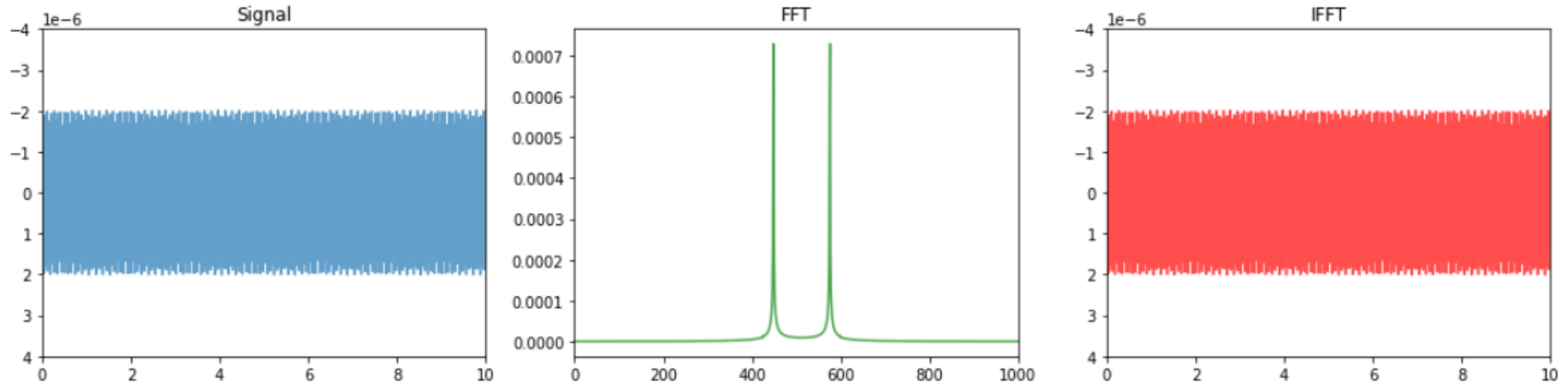
```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 10, 1024); f = 1/t; z = 1
E1 = 2e-6; w1 = 1e4*(np.pi/2); k1 = 2*np.pi/380e-9
E2 = 1e-8; w2 = 1e6*np.pi; k2 = 2*np.pi/380e-9

# Signal
y = E1*np.sin(k1*z + w1*t) + E2*np.sin(k2*z + w2*t)

# FFT of signal
y_fft = np.fft.fft(y)
yr_fft = abs(y_fft)

# IFFT of signal
y_ifft = np.fft.ifft(y_fft)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4:
  after removing the cwd from sys.path.
```

Algoritmos numéricos

```
fig, ax = plt.subplots(1, 3, figsize=(18, 4))

ax[0].plot(t, y, alpha=0.7)
ax[0].set_title('Signal')
ax[0].set_ylim(4e-6, -4e-6)
ax[0].set_xlim(0, 10)

ax[1].plot(yr_fft, alpha=0.7, color='g')
ax[1].set_title('FFT')
ax[1].set_xlim(0, 1000)

ax[2].plot(t, y_ifft, alpha=0.7, color='r')
ax[2].set_title('IFFT')
ax[2].set_ylim(4e-6, -4e-6)
ax[2].set_xlim(0, 10)

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



Criação e operação de Matrizes;

- Transposta;
- Adição;
- Subtração;
- Multiplicação;
- Produtos vetoriais, escalares e tensoriais;
- Álgebra Linear;
- etc.

```
import numpy as np

# Dois arrays(A, B) de valores aleatórios entre [0 a 1) :
A = np.array(np.random.rand(3,1), dtype='float32')
B = np.array(np.random.rand(3,3), dtype='float32')

# Dois vetores :
vec_A = np.array([1,2,3])
vec_B = np.array([5,7,2])

# Dois tensores de dimensão (3,4,5) e (4,3,2) :
a = np.arange(60.).reshape(3,4,5)
b = np.arange(24.).reshape(4,3,2)

print('Adição: \n {}\n\nSubtração: \n {}\n\nMultiplicação: \n {}\n\nTransposta: \n {}'
.format(A+B, A-B, A*B, B.T))

print('\nProduto vetorial: \n{}\n\nProduto vetorial: \n{}\n\nProduto escalar entre tensores: \n{}'
        .format(np.cross(vec_A, vec_B), np.dot(vec_A, vec_B), np.tensordot(a,b,axes=([1,0],[0,1]))))
```

Algoritmos numéricos

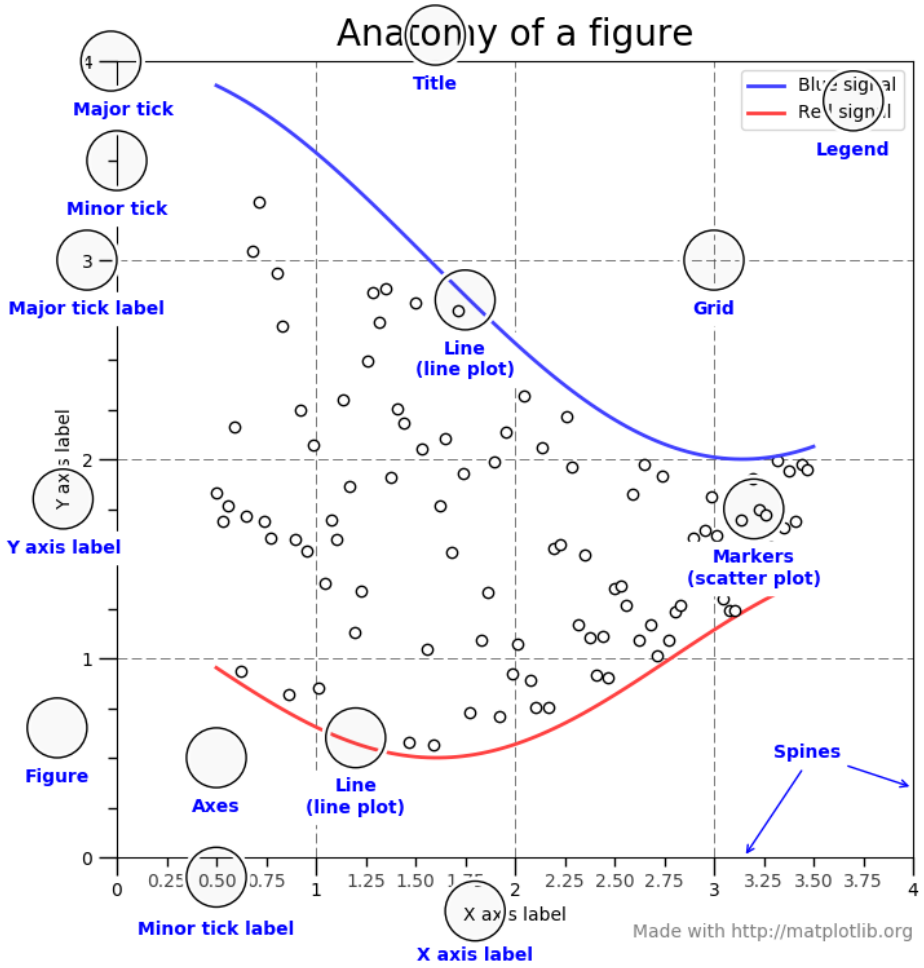
```python
import numpy as np

# Dois arrays(A, B) de valores aleatórios entre [0 a 1) :
A = np.array(np.random.rand(3,1), dtype='float32')
B = np.array(np.random.rand(3,3), dtype='float32')

# Dois vetores :
vec_A = np.array([1,2,3])
vec_B = np.array([5,7,2])

# Dois tensores de dimensão (3,4,5) e (4,3,2) :
a = np.arange(60.).reshape(3,4,5)
b = np.arange(24.).reshape(4,3,2)

print('Adição: \n {}\n\nSubtração: \n {}\n\nMultiplicação: \n {}\n\nTransposta: \n {}'
.format(A+B, A-B, A*B, B.T))

print('\nProduto vetorial: \n{}\n\nProduto vetorial: \n{}\n\nProduto escalar entre tensores: \n{}'
        .format(np.cross(vec_A, vec_B), np.dot(vec_A, vec_B), np.tensordot(a,b,axes=([1,0],[0,1]))))
```

```
Adição:
 [[0.6628479  0.6986138  1.2615874 ]
 [0.9305967  0.15957609 0.28207493]
 [0.4887628  0.28935975 1.0981607 ]]

Subtração:
 [[ 0.6038631   0.5680972   0.00512362]
 [-0.64012825  0.1308923   0.00839347]
 [ 0.05459739  0.25400043 -0.5548005 ]]

Multiplicação:
 [[0.01867916 0.04133173 0.3978941 ]
 [0.11406149 0.00208293 0.01987395]
 [0.05897705 0.00480321 0.22453833]]

Transposta:
 [[0.02949237 0.7853625  0.2170827 ]
 [0.06525834 0.01434189 0.01767967]
 [0.6282319  0.13684073 0.8264806 ]]

Produto vetorial:
[-17  13  -3]

Produto  escalar:
25

Produto escalar entre tensores:
[[4400. 4730.]
 [4532. 4874.]
 [4664. 5018.]
 [4796. 5162.]
 [4928. 5306.]]
```

BIBLIOTECAS E MÓDULOS PYTHON

Matplotlib

O que é o Matplotlib e para que serve?



Matplotlib é uma biblioteca python para criação de gráficos e visualizações de dados em geral, direcionada para sua extensão de matemática Numpy.

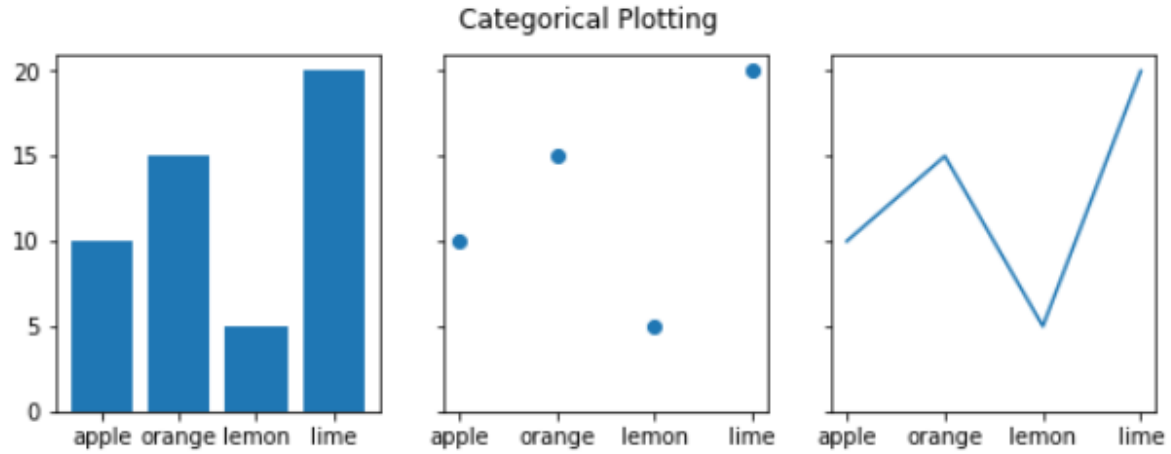
Através do Matplotlib, podemos:

- Analisar imagens;
- Renderizar imagens;
- Criar simulações;
- Criar gráficos interativos.
- E entre outros.

O que é o Matplotlib e para que serve?

```
import matplotlib.pyplot as plt

data = {'apple': 10, 'orange': 15, 'lemon': 5, 'lime': 20}
names = list(data.keys())
values = list(data.values())

fig, axs = plt.subplots(1, 3, figsize=(9, 3), sharey=True)
axs[0].bar(names, values)
axs[1].scatter(names, values)
axs[2].plot(names, values)
fig.suptitle('Categorical Plotting')
```

```
Text(0.5, 0.98, 'Categorical Plotting')
```



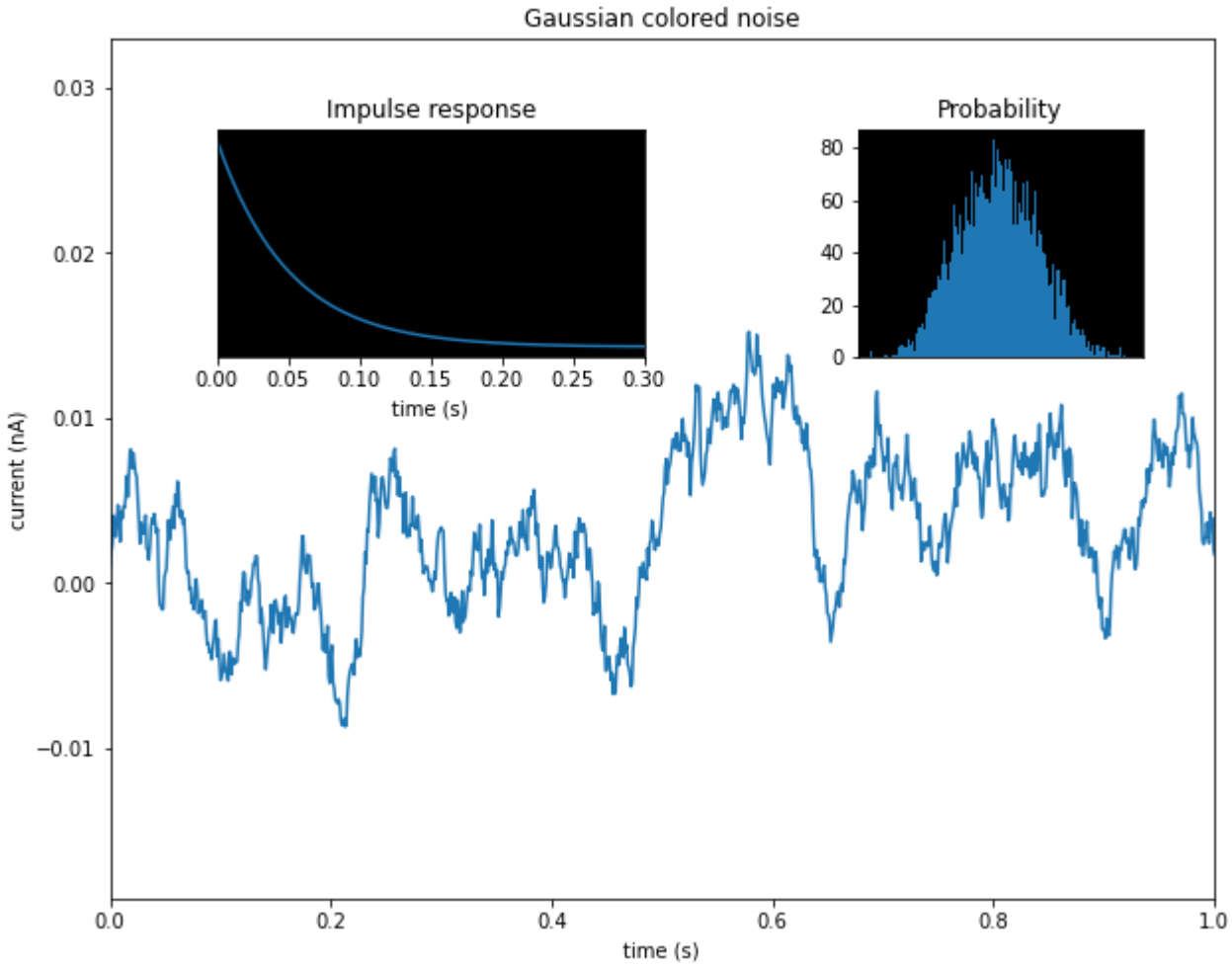
Temos um exemplo de dados em dicionário exibidos graficamente em três categorias; barras, pontos e linha.

Tal exemplo é simples e pode ser desenvolvido, implementando legenda, cores, tipo de linha/ponto/barra, transparência e etc.

Também podemos apresenta-los de forma única, como nos exemplos adiante.

O que é o Matplotlib e para que serve?

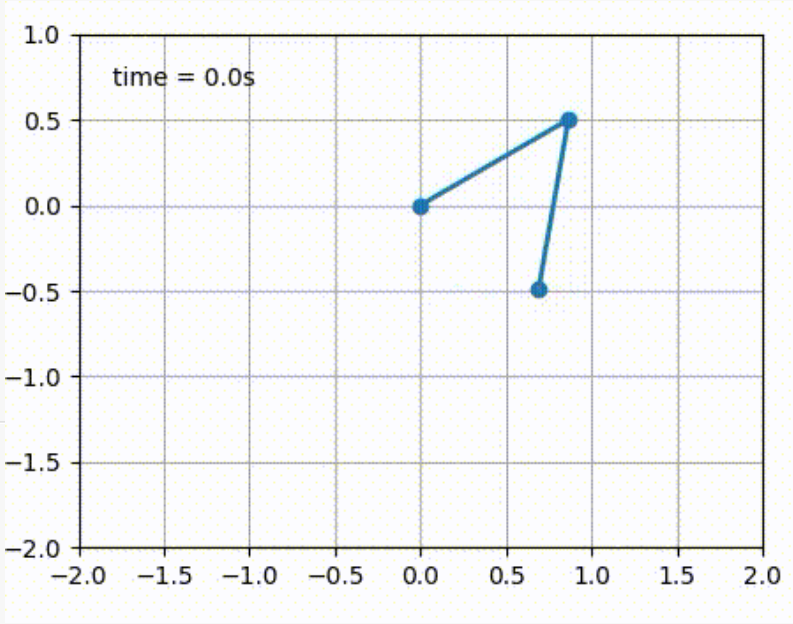
```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(19680801)  # Fixing random state for reproducibility.

# create some data to use for the plot
dt = 0.001
t = np.arange(0.0, 10.0, dt)
r = np.exp(-t[:1000] / 0.05)  # impulse response
x = np.random.randn(len(t))
s = np.convolve(x, r)[:len(x)] * dt  # colored noise

fig, main_ax = plt.subplots(figsize=(10,8))
main_ax.plot(t, s)
main_ax.set_xlim(0, 1)
main_ax.set_ylim(1.1 * np.min(s), 2 * np.max(s))
main_ax.set_xlabel('time (s)')
main_ax.set_ylabel('current (nA)')
main_ax.set_title('Gaussian colored noise')

# this is an inset axes over the main axes
right_inset_ax = fig.add_axes([.65, .6, .2, .2], facecolor='k')
right_inset_ax.hist(s, 400, density=True)
right_inset_ax.set(title='Probability', xticks=[])

# this is another inset axes over the main axes
left_inset_ax = fig.add_axes([.2, .6, .3, .2], facecolor='k')
left_inset_ax.plot(t[:len(r)], r)
left_inset_ax.set(title='Impulse response', xlim=(0, .3), yticks=[])
left_inset_ax.set_xlabel('time (s)')

plt.show()
```



O que é o Matplotlib e para que serve?

```
from numpy import sin, cos
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import matplotlib.animation as animation
from collections import deque

G = 9.8  # acceleration due to gravity, in m/s^2
L1 = 1.0  # length of pendulum 1 in m
L2 = 1.0  # length of pendulum 2 in m
L = L1 + L2  # maximal length of the combined pendulum
M1 = 1.0  # mass of pendulum 1 in kg
M2 = 1.0  # mass of pendulum 2 in kg
t_stop = 5  # how many seconds to simulate
history_len = 500  # how many trajectory points to display


def derivs(state, t):

    dydx = np.zeros_like(state)
    dydx[0] = state[1]

    delta = state[2] - state[0]
    den1 = (M1+M2) * L1 - M2 * L1 * cos(delta) * cos(delta)
    dydx[1] = ((M2 * L1 * state[1] * state[1] * sin(delta) * cos(delta)
                + M2 * G * sin(state[2]) * cos(delta)
                + M2 * L2 * state[3] * state[3] * sin(delta)
                - (M1+M2) * G * sin(state[0]))
               / den1)

    dydx[2] = state[3]

    den2 = (L2/L1) * den1
    dydx[3] = ((- M2 * L2 * state[3] * state[3] * sin(delta) * cos(delta)
                + (M1+M2) * G * sin(state[0]) * cos(delta)
                - (M1+M2) * L1 * state[1] * state[1] * sin(delta)
                - (M1+M2) * G * sin(state[2]))
               / den2)

    return dydx

# create a time array from 0..t_stop sampled at 0.02 second steps
dt = 0.02
t = np.arange(0, t_stop, dt)

# th1 and th2 are the initial angles (degrees)
# w10 and w20 are the initial angular velocities (degrees per second)
th1 = 120.0
w1 = 0.0
th2 = -10.0
w2 = 0.0

# initial state
state = np.radians([th1, w1, th2, w2])

# integrate your ODE using scipy.integrate.
y = integrate.odeint(derivs, state, t)

x1 = L1*sin(y[:, 0])
y1 = -L1*cos(y[:, 0])

x2 = L2*sin(y[:, 2]) + x1
y2 = -L2*cos(y[:, 2]) + y1

fig = plt.figure(figsize=(5, 4))
ax = fig.add_subplot(autoscale_on=False, xlim=(-L, L), ylim=(-L, 1.))
ax.set_aspect('equal')
ax.grid()

line, = ax.plot([], [], 'o-', lw=2)
trace, = ax.plot([], [], ',-', lw=1)
time_template = 'time = %.1fs'
time_text = ax.text(0.05, 0.9, '', transform=ax.transAxes)
history_x, history_y = deque(maxlen=history_len), deque(maxlen=history_len)
```

```
def animate(i):
    thisx = [0, x1[i], x2[i]]
    thisy = [0, y1[i], y2[i]]

    if i == 0:
        history_x.clear()
        history_y.clear()

    history_x.appendleft(thisx[2])
    history_y.appendleft(thisy[2])

    line.set_data(thisx, thisy)
    trace.set_data(history_x, history_y)
    time_text.set_text(time_template % (i*dt))
    return line, trace, time_text


ani = animation.FuncAnimation(
    fig, animate, len(y), interval=dt*1000, blit=True)
plt.show()
```



Exemplo de animação, ilustrando o problema de um pêndulo duplo

O que é o Matplotlib e para que serve?

```
import matplotlib.pyplot as plt

img = plt.imread('primeiro-raio-x-2.jpg'); print(
    'Temos um %s: \n\n %s \n\ncom shape (dimensão) igual à %s, portanto uma tensor 3D de uma imagem RGB.'
%(type(img), img[:2], np.shape(img)))
```

```
Temos um <class 'numpy.ndarray'>:

 [[[244 244 244]
  [239 239 239]
  [236 236 236]
  ...
  [252 252 252]
  [252 252 252]
  [252 252 252]]

 [[242 242 242]
  [235 235 235]
  [229 229 229]
  ...
  [252 252 252]
  [252 252 252]
  [252 252 252]]]

com shape (dimensão) igual à (449, 311, 3), portanto uma tensor 3D de uma imagem RGB.
```

Exemplo de tradução de uma imagem RGB em um array de dados, nesse caso, cada pixel é descrito por um array (3,) contendo valores de 0 a 255, a escala de cor.

Python Funções e Pandas
Fundamentos para Análise de Dados

Eduardo Destefani Stefanato[1]*
Vitor Souza Premoli Pinto de Oliveira[1]*

[1]Universidade Federal do Espírito Santo

Inteligência Artificial Aplicada em Imagens

Sumário

- Linguagem Python – Função Lambda.....21
  - Funções.....23
  - Lambda e palavras reservadas.....24
  - Aplicação.....25
- Biblioteca e Módulos Python – Pandas.....26
  - O que é o Pandas e para que serve?.....27
  - Destaques da biblioteca.....28
- Referências.....33

LINGUAGEM PYTHON

Função Lambda

Funções

```
def func(data):
    err_func = []
    for i in data:
        err_func.append(i)
    return err_func

def lista(*data):
    err_func = []
    for i in data:
        err_func.append(i)
    return err_func

def mean(*data):
    x = np.array(data)
    y = np.mean(x)
    return y

print('média: %s\nlista: %s'%(mean(1,2,3,4,5,6), lista(1,2,3)))
```

```
média: 3.5
lista: [1, 2, 3]
```

```
func(1,2,3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-2036b2cfd233> in <module>
----> 1 func(1,2,3)

TypeError: func() takes 1 positional argument but 3 were given
```

Na programação, funções são blocos de código que realizam determinadas tarefas (cálculos numéricos, simulações e etc) que normalmente precisam ser executadas diversas vezes dentro de uma aplicação.

Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um nome e que poderá ser chamada/executada em diferentes partes do programa.

Mas o que, estruturalmente, define uma função na linguagem Python e demais?

Lambda e palavras reservadas

Oque é a Função lambda?

- Palavra reservada ~~de programação~~ para criação de **funções**;
- Função anônima.

Para que serve?

- Suas principais funcionalidades e atributos;
  - Basicamente a criação de funções compactadas;
- Vantagens e Limitações;
- Exemplos de aplicação.

int	open	with
float	sum	while
round	input	else
format	return	elife
type	try	for
Len	If	except
yield	def	lambda

```
arr = np.linspace(0,4,10)

lan = lambda *x: np.mean(np.array(x))
f_par = lambda x: True if x%2 == 0 else False

print('Média: %d \n '%lan(arr))
print('São pares os  valores:')
for i in range(0,10):
    print(i,'={}' .format(f_par(i)))
```

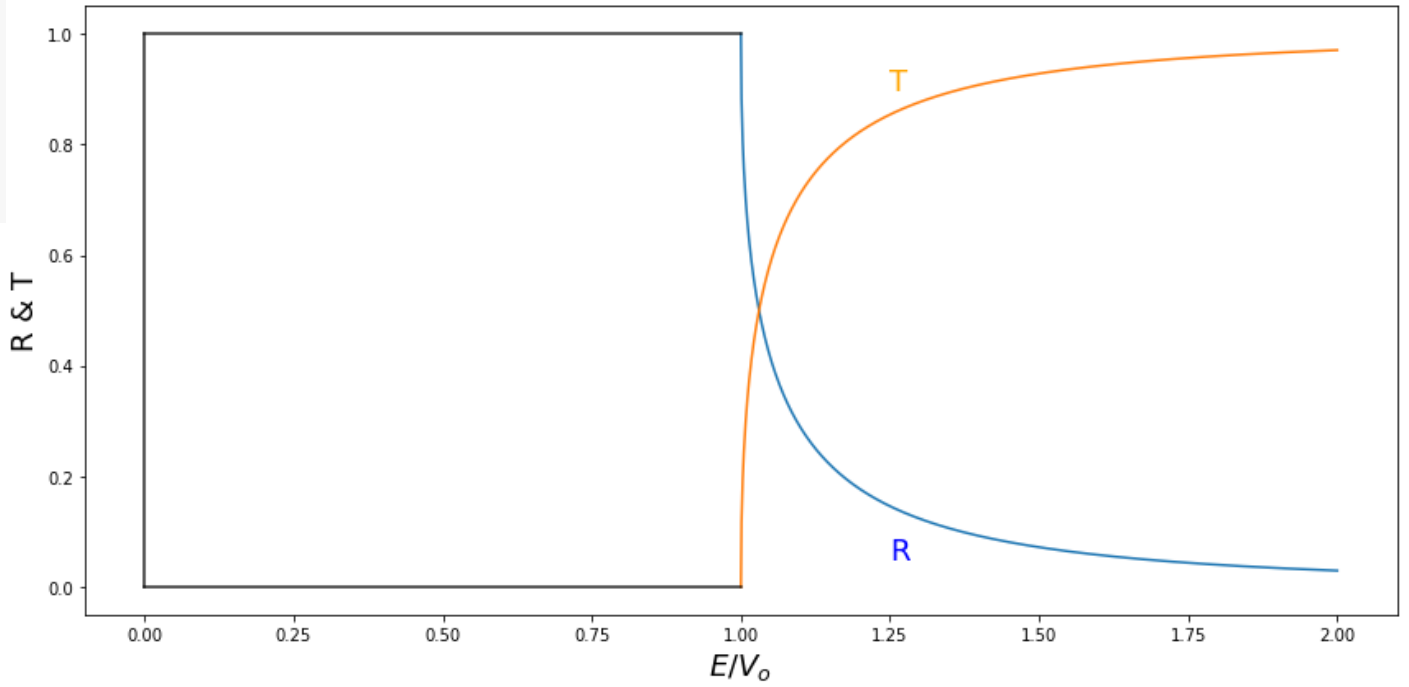
```
Média: 2

São pares os  valores:
0 =True
1 =False
2 =True
3 =False
4 =True
5 =False
6 =True
7 =False
```

Aplicação

```
import matplotlib.pyplot as plt

# Movimento de várias particulas (neutrons) sobre a influência de um potêncial
# degrau

Vo = 50e6 # Vo = potencial da barreira;
E = np.linspace(50e6, 100e6, 1024) # E = energia total (K + V)

#Coeficiente de Reflexão
R = (lambda E: ((1-(1-(Vo/E))**(1/2))/(1+(1-(Vo/E))**(1/2)))**2)(E)
#Coeficiente de Transmissão
T = (lambda E: 1-((1-(1-(Vo/E))**(1/2))/(1+(1-(Vo/E))**(1/2)))**2)(E)

x = E/Vo

fig, ax = plt.subplots(figsize=(12,6))
ax.plot(x, R)
ax.plot(x, T)
```

```
ax.plot(np.linspace(0,1,2), np.linspace(0,0,2), color='black', alpha=0.85)
ax.plot(np.linspace(0,0,2), np.linspace(0,1,2),color='black', alpha=0.85)
ax.plot(np.linspace(0,1,2), np.linspace(1,1,2), color='black', alpha=0.85)

ax.text(1.25,0.9,'T', fontsize=18, color='orange')
ax.text(1.25,0.05,'R', fontsize=18, color='blue')

ax.set_xlabel('$E/V_{o}$', fontsize=18)
ax.set_ylabel('R & T', fontsize=18)

plt.tight_layout()
```

Exemplo de aplicação da função Lambda para calculo do fluxo de probabilidade de um conjunto de nêutrons incidentes a um potencial degrau.



BIBLIOTECA E MÓDULOS PYTHON

Pandas

O que é o Pandas e para que serve?

Pandas é uma biblioteca que fornece ferramentas para ler, gravar e manipular dados entre estruturas na memória, em diferentes formatos: arquivos CSV e de texto, Microsoft Excel, bancos de dados SQL e o formato rápido HDF5**.**

É muito comum que dados que analisamos estejam dispostos em formato de tabelas, como uma *rede neural* ou quaisquer algoritmos de *machine learning,* sobretudo quando falamos em analise estatística de dados.

Esses dados, no formato de tabelas, podem apresentar, por exemplo, colunas que trazem diferentes atributos do dado, e linhas que trazem um conjunto de observações.

DataFrame

Para nos ajudar a lidar com dados em formato de tabelas, o Pandas nos fornece um objeto, chamado *DataFrame*, que é capaz de armazenar e manipular esse tipo de dado de uma forma equivalente a uma planilha de Excel, onde suas linhas e colunas são chamadas de *Series*.

pandas

Destaques da biblioteca

```
data = np.genfromtxt("SN_m_tot_V2.0.txt",skip_footer=0)
data = pd.DataFrame(data, columns=['Ano','NaN1','Data','Nums Spots', 'NaN2','NaN3']).copy(); data
```

	Ano	NaN1	Data	Nums Spots	NaN2	NaN3
0	1749.0	1.0	1749.042	96.7	-1.0	-1.0
1	1749.0	2.0	1749.123	104.3	-1.0	-1.0
2	1749.0	3.0	1749.204	116.7	-1.0	-1.0
3	1749.0	4.0	1749.288	92.8	-1.0	-1.0
4	1749.0	5.0	1749.371	141.7	-1.0	-1.0
...	...	...	...	...	...	...
3208	2016.0	5.0	2016.373	52.1	4.7	810.0
3209	2016.0	6.0	2016.456	20.9	2.2	886.0
3210	2016.0	7.0	2016.540	32.5	3.7	910.0
3211	2016.0	8.0	2016.624	50.7	4.4	879.0
3212	2016.0	9.0	2016.708	44.7	3.8	742.0

3213 rows × 6 columns

Exemplo de uma tabela de dados retirado do domínio da NASA, o *Solar Cycle Prediction*, a partir do arquivo no formato *.txt*. A mesma nos informa as médias mensais de manchas solares ao longo dos anos.

A tabela possui 3213 linhas e 6 colunas.

Destaques da biblioteca

```
# retorna um novo DataFrame dos valores da coluna especificada do DataFrame principal
data_main = data[['Ano', 'Data', 'Nums Spots']]; data_main
```

	Ano	Data	Nums Spots
0	1749.0	1749.042	96.7
1	1749.0	1749.123	104.3
2	1749.0	1749.204	116.7
3	1749.0	1749.288	92.8
4	1749.0	1749.371	141.7
...	...	...	...
3208	2016.0	2016.373	52.1
3209	2016.0	2016.456	20.9
3210	2016.0	2016.540	32.5
3211	2016.0	2016.624	50.7
3212	2016.0	2016.708	44.7

3213 rows × 3 columns

slicing, **indexing**, e **subsetting**

```
# Informações sobre os dados:

data_main.info()
data_main.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3213 entries, 0 to 3212
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Ano         3213 non-null   float64
 1   Data        3213 non-null   float64
 2   Nums Spots  3213 non-null   float64
dtypes: float64(3)
memory usage: 75.4 KB
(3213, 3)
```

Destaques da biblioteca

Remodelagem do conjunto de dados

```
# função exemplo:
def test(x, e):
    return e*x**2
```

```
# aplicamos à uma das colunas do DataFrame a função test()
data_main['Ano'].apply(test, e=4)
```

```
0       12236004.0
1       12236004.0
2       12236004.0
3       12236004.0
4       12236004.0
           ...
3208    16257024.0
3209    16257024.0
3210    16257024.0
3211    16257024.0
3212    16257024.0
Name: Ano, Length: 3213, dtype: float64
```

```
# função teste
def avg_3_apply(col):
    return (np.mean(col))
```

```
# aplicamos a função avg_3_apply que retorna a média das colunas
data_main.apply(avg_3_apply)
```

```
Ano           1882.375350
Data          1882.873012
Nums Spots      82.923561
dtype: float64
```

```
# Aplicados a função lambda em uma das colunas do DataFrame
data_main.apply(lambda col: np.mean(col))
```

```
Ano           1882.375350
Data          1882.873012
Nums Spots      82.923561
dtype: float64
```

Destaques da biblioteca

Agregação e **transformação** por meio do **groupby**, permitindo dividir e aplicar operações em um conjunto de dados

```
# Agrupamento de banco de dados e agregação por funções
data_main.groupby(['Ano', 'Data'])[['Value', 'Nums Spots']].agg([np.mean, np.std])
```

		Value		Nums Spots	
		mean	std	mean	std
Ano	Data				
1749.0	1749.042	-1.0	NaN	96.7	NaN
	1749.123	-1.0	NaN	104.3	NaN
	1749.204	-1.0	NaN	116.7	NaN
	1749.288	-1.0	NaN	92.8	NaN
	1749.371	-1.0	NaN	141.7	NaN
...	...	...	...	...	...
2016.0	2016.373	810.0	NaN	52.1	NaN
	2016.456	886.0	NaN	20.9	NaN
	2016.540	910.0	NaN	32.5	NaN
	2016.624	879.0	NaN	50.7	NaN
	2016.708	742.0	NaN	44.7	NaN

```
# Agrupamento de banco de dados e agregação por função lambda
data_main.groupby(['Ano', 'Data'])[['Value', 'Nums Spots']].agg(lambda x:
                                                    abs(x.astype(float)))
```

		Value	Nums Spots
Ano	Data		
1749.0	1749.042	1.0	96.7
	1749.123	1.0	104.3
	1749.204	1.0	116.7
	1749.288	1.0	92.8
	1749.371	1.0	141.7
...	...	...	...
2016.0	2016.373	810.0	52.1
	2016.456	886.0	20.9
	2016.540	910.0	32.5
	2016.624	879.0	50.7
	2016.708	742.0	44.7

Destaques da biblioteca

A indexação de eixo hierárquica

1)
```
# função loc de um DataFrame
data_main.loc[700]
```

```
Ano           1807.000
Data          1807.371
Nums Spots      16.700
Name: 700, dtype: float64
```

2)
```
data_main.loc[[0,700]]
```

	Ano	Data	Nums Spots
0	1749.0	1749.042	96.7
700	1807.0	1807.371	16.7

3)
```
data_main.loc[[0,700], ['Data', 'Ano']]
```

	Data	Ano
0	1749.042	1749.0
700	1807.371	1807.0

4)
```
data_main['Nums Spots'] == 20.9
```

```
0       False
1       False
2       False
3       False
4       False
        ...
3208    False
3209     True
3210    False
3211    False
3212    False
Name: Nums Spots, Length: 3213, dtype: bool
```

5)
```
# bitwise operators; & = and , | = or
data_main.loc[(data_main['Nums Spots'] == 20.9) & (data_main['Data'] > 2006.)]
```

	Ano	Data	Nums Spots
3084	2006.0	2006.042	20.9
3209	2016.0	2016.456	20.9

6)
```
data_main.loc[data_main['Nums Spots'] == 20.9]
```

	Ano	Data	Nums Spots
1518	1875.0	1875.538	20.9
3084	2006.0	2006.042	20.9
3209	2016.0	2016.456	20.9

Python Fundamentos para Análise de Dados

Eduardo Destefani Stefanato[1]*
Vitor Souza Premoli Pinto de Oliveira[1]*

[1]Universidade Federal do Espírito Santo

Inteligência Artificial Aplicada em Imagens

REFERÊNCIAS

PiPy. **Find, install and publish Python packages with the Python Package Index**. Disponível em: https://pypi.org/. Acesso em: 22 Jul. 2021.

Matplotlib. **Matplotlib: Python plotting — Matplotlib 3.4.2 documentation.** Disponível em: https://matplotlib.org/. Acesso em: 22 Jul. 2021.

Numpy. **NumPy Reference — NumPy v1.21 Manual.** Disponível em: https://numpy.org/doc/stable/reference/. Acesso em: 22 Jul. 2021.

NumFocus. **Pandas.** Disponível em: https://pandas.pydata.org/. Acesso em: 29 Jul. 2021.