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

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)
```

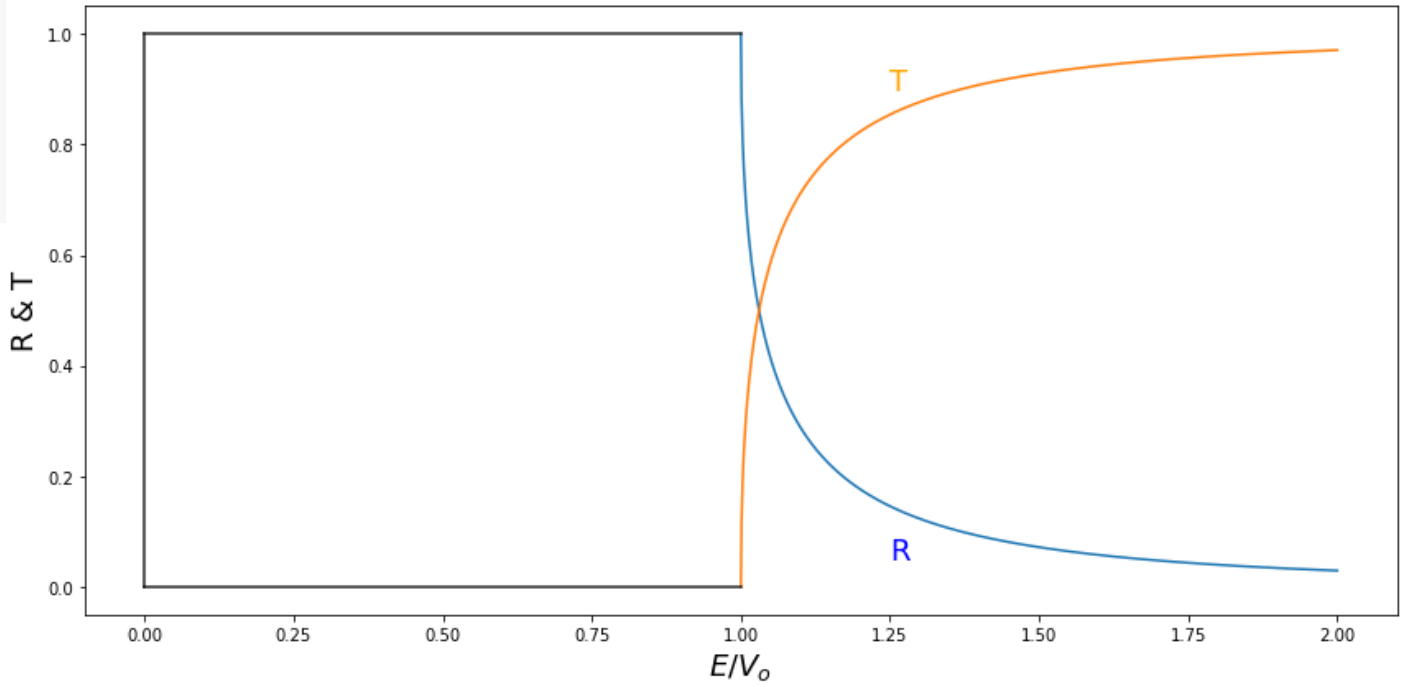
```
ax.plot(np.linspace(0,1,2), np.linspace(0,0,2), color='black', alpha=0.85)
ax.plot(np.linspace(0,0,2), np.linspace(0,1,2),color='black', alpha=0.85)
ax.plot(np.linspace(0,1,2), np.linspace(1,1,2), color='black', alpha=0.85)

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.