

Introdução ao Python para Tratamento de Dados

Hugo Everaldo Salvador Bezerra

10 de julho de 2024

Sumário

1	Introdução	2
2	Bibliografia sugerida	2
3	Introdução ao Python	3
3.1	Contexto	3
3.1.1	Características da linguagem	3
3.1.2	Usando Python	3
3.2	Variáveis	4
3.2.1	Números	4
3.2.2	String	9
3.2.3	Listas	13
3.2.4	Tupla	20
3.2.5	Dicionário	20
3.2.6	Conjunto (<i>set</i>)	21
3.3	Condicional	22
3.4	Laços de repetição (<i>Loops</i>)	24
3.4.1	<i>List Comprehension</i> (Listcomps)	26
3.4.2	Estrutura de paradigma funcional com função <code>map</code>	28
3.5	Tratamento de erro (<code>try/except/else</code>)	28
3.6	Funções	29
4	NumPy	30
4.1	Métodos de <code>ndarray</code>	36
4.2	Indexing / Slicing	37
4.3	Broadcasting	40
4.4	Mais Rotinas	41
4.5	Resolução de sistemas lineares com <code>numpy.linalg</code>	42
4.6	Matplotlib	43
4.7	Distribuições probabilísticas em <code>numpy.random</code>	46
5	Programação Orientada a Objeto	49
6	pandas	54
6.1	Aquisição de dados	61
6.2	Entendendo a base de dados	67
6.3	Seaborn	80

7 Anexos	85
7.1 Google Colab	85
7.2 Easter egg no Pytnon - Zen of Python	86

1 Introdução

Este material é composto de notas de aulas ministradas para pessoas que conhecem teoria de algoritmos ou já tem experiência com outra linguagem de programação, mas tem pouca ou nenhuma experiência com Python. O objetivo é fazer uma introdução ao Python e citar referências importantes na área de computação científica e tratamento de dados.

Um dos pontos fortes do material são links das funções e pacotes citados para acesso a tutoriais, páginas oficiais dos pacotes, textos complementares e manuais. A maioria das funções citadas tem link para acesso a texto de suporte para utilização da função.

Para geração do PDF foi utilizado o Colab, realizado tratamento em LaTeX e então gerado o arquivo PDF.

Este trabalho tem licença Creative Commons sendo do tipo BY-NC-ND, que pode ser visto no site da [CC Brasil](#). Você pode realizar download e compartilhar desde que atribuam crédito ao autor, mas sem que possam alterá-los de nenhuma forma ou utilizá-los para fins comerciais.



```
[1]: # Versão do Python utilizada neste material
import sys
sys.version
```

```
[1]: '3.11.9 | packaged by conda-forge | (main, Apr 19 2024, 18:27:10) [MSC v.1938 64
bit (AMD64)]'
```

2 Bibliografia sugerida

Listamos algumas referências de material de apoio para aprofundamento nos temas abordados.

- [Documentação Oficial do Python](#): documentação completa sobre a linguagem com tutoriais e referências em várias versões do Python.
- [Pense em Python](#): Livro em português disponibilizado online gratuitamente.
- [Python Fluente](#): Segunda edição de livro disponibilizada gratuitamente no site com detalhes sobre Python. Escrito pelo renomado autor da área, Luciano Ramalho, teve a primeira edição do livro publicado em nove idiomas.

- [Guide to Numpy](#): Livro escrito por Travis E. Oliphant, criador do NumPy, escrito em 2006 que é disponibilizado gratuitamente.
- NumPy User Guide ([web](#)/[pdf](#)): Guia on-line e em arquivo PDF, atualizado pela equipe que mantém o pacote.
- [NumPy Reference](#): Referência disponibilizada on-line pela equipe que mantém o pacote.
- Pandas Documentation([web](#)/[pdf](#)): Guia disponibilizado on-line e em arquivo PDF pela equipe que mantém o pacote.
- [Python for Data Analys](#): terceira edição lançada em 2022 do livro escrito pelo criador do pandas, Wes Mckinney. Versão aberta disponibilizada [on-line](#).

3 Introdução ao Python

3.1 Contexto

3.1.1 Características da linguagem

- Licença Open Source (pode ser reproduzida, analisada, testada, executar e/ou exibida publicamente, preparado trabalhos derivados, distribuída e usada)
- Fácil de iniciar programando
- Inglês das linguagens de programação, transpassa por várias áreas como computação científica, Inteligência Artificial, desenvolvimento web, manutenção de máquina e **Automação Robótica de Processos** - RPA (do inglês *Robotic Process Automation*) entre outras.
- Linguagem de alto nível, onde não é necessário se preocupar ao escrever o código com detalhes como gerenciamento de utilização de memória da máquina.
- Linguagem interpretada, não havendo necessidade de compilar o código antes de executar. Esta característica ajuda bastante em testar previamente função que será utilizada. Podemos rodar apenas a linha que estamos estudando para entender sua utilização e testá-la antes de inserir no código.
- Multiparadigma: fortemente Orientado a Objetos, Procedural e Funcional.
- Tipagem dinâmica de variável, diferente de linguagens como C/C++ ou Java onde as variáveis devem ser declaradas antes de sua utilização.
- Possui um vasto repertório de bibliotecas (mais de 500 mil pacotes listados do repositório *Python Package Index* - [Pypi](#) em dezembro de 2023).
- Expansível com C/C++ ou Fortran, melhorando o desempenho e performance.
- Extremamente portátil, podendo rodar o código em sistemas operacionais como Unix/Linux, Windows, Mac ou embarcado em [microcontroladores](#).

3.1.2 Usando Python

Por ser uma linguagem interpretada, o Python pode ser usada com comando interativos em um prompt de comando padrão da linguagem. Uma boa alternativa é o [IPython](#) que conta com auto-completar, histórico de comandos e gráficos integrados, ideal para análise de dados.

É possível escrever código Python em qualquer editor de texto (ex. Bloco de notas do Windows, [Notepad++](#), [Visual Studio Code](#) da Microsoft, [Vim](#)), mas existem Ambientes Integrados de Desenvolvimento ou IDEs (*Integrated Development Environment*) específicos como o [Spyder](#) e [Jupyter](#). O Spyder é uma IDE familiar para pessoas que estão acostumadas com IDEs do Matlab e RStudio.

Jupyter é um IDE que roda diretamente em um navegador de internet e é muito utilizado para tratamento de dados. Atualmente o Visual Studio Code está sendo muito utilizado para edição de códigos Python com [ajuda de extensões](#) que facilitam e agilizam a codificação.

O [Colab](#) é um IDE disponibilizado on-line para qualquer um que tenha uma conta no Google e é muito semelhante ao Jupyter. É uma boa alternativa para quem quer iniciar aprendendo Python sem instalar qualquer programa na máquina local. Além disso, o ambiente já conta com várias bibliotecas como NumPy e Matplotlib. Tanto o Colab como o Jupyter usam o IPython como núcleo.

No caso de se optar por instalar o Python em uma máquina local, as melhores opções são a utilização do [Anaconda](#) que instala não só o Python como também vários [pacotes](#) e ferramentas adicionais como o Jupyter e o Spyder. No [repositório](#) do Anaconda existem várias versões disponíveis. Se a intenção não é instalar uma solução completa como o Anaconda, uma boa alternativa é o [Miniconda](#) que é uma instalação mais enxuta, mas já vem com funcionalidades importantes como gerenciador de pacotes melhor que o original do Python e controle de ambientes isolados.

3.2 Variáveis

O Python trabalha com uma grande variedade de [Modelos de Dados](#), mas nesta seção serão considerados os tipos básicos para texto, numérico e sequência.

Tipos básicas	Descrição	Exemplo
strings	Texto	'spam', "Bob's", "" "Python é massa" "" "1234"
int	Número Inteiro	1234
float	Número Real	3.14159
complex	Número Complexo	3+4j
bool	Lógico	True, False
lists	Lista	[1, [2, 'three'], 4.5], [1,2,3,4], ['casa', 'carro', 'bola']
dict	Dicionário	{'food': 'spam', 'taste': 'yum'}, {'nome': 'João', 'idade': 32}
tuples	Tupla	(1, 'spam', 4, 'U'), (1,2,3)
set	Conjunto	{'r', 'g', 'b'}, {10, 20, 40}

3.2.1 Números

O Python trabalha com 4 tipos básicos de valores numéricos: inteiros (`int`), números reais (`float`), números complexos (`complex`) e booleanos (`bool`). A precisão e intervalo de armazenamento em cada tipo varia de acordo com a arquitetura da máquina onde o script Python está sendo executado.

Vamos passar de forma rápida os principais operadores no Python utilizados com números. Note que o texto após o símbolo `#` é considerado um comentário, ou seja, o Python não tenta interpretar e executar este texto que serve apenas para ajudar as pessoas que escrevem e leem os códigos documenta-lo.

```
[2]: 1.222 + 5.32 # Soma
```

[2]: 6.542

```
[3]: 3 * 5.5 # Multiplicação
```

[3]: 16.5

```
[4]: 7/3 # Divisão
```

[4]: 2.3333333333333335

```
[5]: 7 // 3 # Divisão com resultado inteiro
```

[5]: 2

```
[6]: 7 % 3 # Resto da divisão
```

[6]: 1

```
[7]: 2 ** 4 # Potência
```

[7]: 16

O Python é uma linguagem de propósito geral, diferente de linguagens como MATLAB, Octave e Julia voltadas para computação científica ou R criada para análise de dados e estatística. Muitas funções matemáticas básicas já são carregadas na memória e disponibilizadas para utilização ao iniciar o ambiente destas linguagens, por vezes disponibilizando até base de dados para testes.

Por ser uma linguagem de propósito geral, o Python necessita que sejam utilizados pacotes que tem funções específicas para cada aplicação. Existem pacotes voltados para desenvolvimento Web, análise e modelagem estatística, inteligência artificial, finanças, automatização de tarefas, desenvolvimento de jogos, bioinformática, entre outras. Para trabalhar com computação científica no Python existem pacotes como math, NumPy e SciPy. O pacote math faz parte do conjunto de pacotes básico na maioria das instalações do Python, outros como o NumPy e SciPy usualmente necessitam de instalação após a instalação do Python. Instaladores como o Anaconda já trazem pacotes necessários à computação científica.

```
[8]: import math
from math import pi, cos

print('pi = ', math.pi)

print('sen( $\pi/2$ ) = ', math.sin(math.pi/2))

print('cos( $\pi/2$ ) = ', cos(pi))

print('Tipo da variável math.pi: ', type(pi))
```

```
pi = 3.141592653589793
sen( $\pi/2$ ) = 1.0
```

```
cos(π/2) = -1.0
Tipo da variável math.pi: <class 'float'>
```

Acima, importamos o pacote `math` onde estão as funções básicas de matemática em Python. Os pacotes são uma forma de agrupar funções e variáveis para um fim específico. Os pacotes não são carregados por padrão pelo Python para evitar que existem muitas funções carregadas na memória que não serão utilizadas.

Existem três formas de importar pacotes no Python:

```
import <pacote>                # utilização: <pacote>.<função>
from <pacote> import <função>  # utilização: <função>
from <pacote> import *         # utilização: <função>
```

Quando importamos um pacote utilizando a palavra `import` seguido o nome do pacote sempre precisamos definir o pacote e a função que queremos usar. No exemplo acima, importamos o pacote `math` utilizando `import` e para utilizar a função `sin` foi necessário a utilização na forma `math.sin`.

Na linha seguinte usamos `from math import pi, cos`. Ao utilizar a função não foi necessário definir o nome do pacote de onde a função foi importada, já que desta forma a função e a variável foram carregadas direto na memória. Da mesma forma poderíamos utilizar apenas a função sem definir o pacote se a importação fosse feita utilizando `from <pacote> import *`. A desvantagem da última forma é que todas as funções existentes no pacote seriam carregadas na memória e se importarmos desta forma mais de um pacote correremos o risco de ter funções com mesmo nome em pacotes distintos, podendo causar confusão para saber de que pacote a função utilizada pertence.

Como opção se pode definir uma *alias*, um apelido para não precisar usar o nome inteiro do pacote:

```
import <pacote> as <alias>    # utilização: <alias>.<função>
```

Neste caso podemos resumir o nome do pacote e ter certeza de que pacote a função que estamos usando pertence.

Daqui para frente teremos vários exemplos de importação e de utilização de funções de pacotes específicos.

```
[9]: # Exemplo de utilização de número complexo
c1 = 3 + 4j

print(type(c1)) # Mostrar o tipo de variável

c2 = 5 + 8j

print(c1*c2)    # Resultado da multiplicação de dois complexos
```

```
<class 'complex'>
(-17+44j)
```

Existem formas de trabalhar com números em outros formatos como em forma de fração, em sistema binário, hexadecimais e octadecimais.

```
[10]: # Trabalhando com números em forma de fração
from fractions import Fraction
```

```
print(Fraction(1,5) + Fraction(4,10) + 1)
```

8/5

```
[11]: # Trabalhando com números binários, hexadecimais e octadecimais

# Print de números no sistema decimal
print(0b101010) # binário
print(0xb0ca)   # hexadecimal
print(0o177)    # octadecimal
print()         # Linha em branco

# Print de números do sistema decimal em binários, hexadecimais ou octadecimais
print(bin(42))
print(hex(15815114))
print(oct(127))
```

42
45258
127

0b101010
0xf151ca
0o177

Abaixo vemos a versatilidade de trabalhar de forma direta em sistemas decimais, binários, hexadecimais e octadecimais. Estamos somando um número no sistema binário, com um número no sistema decimal, com número no sistema octadecimal e dando o resultado no sistema hexadecimal.

```
[12]: hex(0b101010 + 100 + 0o177)
```

```
[12]: '0x10d'
```

Podemos fazer operações matemáticas entre números do tipo `bool`, `int`, `float` e `complex` de forma transparente, sem nos preocuparmos em converter o tipo do número. Se um número `complex` estiver envolvido na operação, o resultado será um número complexo, se um número `float` estiver em uma operação sem um número `complex` presente o resultado será `float` e assim por diante, seguindo a priorização `bool -> int -> float -> complex`.

```
[13]: # bool -> int -> float -> complex

etcha = True + 2 * 1.1 / 4j
print(etcha)
print(type(etcha))
```

(1-0.55j)
<class 'complex'>

Existem atalhos para se trabalhar com variáveis que facilitam o desenvolvimento de scripts. Uma

que sempre é citado é troca de valores de variáveis utilizando uma linha, que nem toda linguagem de programação consegue realizar.

```
[14]: a = 3
      b = 5

      a, b = b, a

      print('a: ', a)
      print('b: ', b)
```

```
a: 5
b: 3
```

No exemplo a seguir importamos as funções `mean` (média) e `pstdev` (desvio padrão populacional) do pacote `statistics`. Utilizamos como nome da variável μ que não faz parte dos caracteres no padrão ASCII, mas faz parte do padrão Unicode. Isso possibilita que utilizemos em nossos scripts variáveis com letras gregas, letras com acentos e caracteres como ç.

```
[15]: from statistics import mean, pstdev
      # Python aceita Unicode no código para, por exemplo, nomear variáveis
      ações = [1,2,3,4,5,6]
       $\mu$  = mean(ações)
       $\sigma$  = pstdev(ações)
      print('Média = ',  $\mu$ )
      print('Desvio padrão = ',  $\sigma$ )
```

```
Média = 3.5
Desvio padrão = 1.707825127659933
```

A partir do Python 3.6 foi adicionada a [string formatada](#), uma forma muito fácil e poderosa de formatar strings utilizando uma [minilinguagem de especificação de formato](#). Existem outros meios de formatar string no Python, mas a string formatada é a mais utilizada por desenvolvedores Python. Abaixo temos a definição de como formatar uma `string`, mais no intuito de ser usado como mnemônico. Até se habituar com a utilização da string formatada é útil recorrer a um tutorial ou documentação de referência.

Para indicar que o Python de interpretar como uma string formatada é colocado um `f` imediatamente antes da string.

```
f'{' : [[preencher]alinhamento] [sinal] ["z"] ["#"] ["0"] [tamanho] [agrupamento] [".precisão] [tipo]]'
```

preencher : <qualquer caracter>

alinhamento : "<" | ">" | "=" | "^"

sinal : "+" | "-" | ""

tamanho : quantidade de dígitos

agrupamento : "_" | ","

precisão : quantidade de dígitos

tipo : "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"


```
[16]: a = 0.1298731
      b = 35466.3108012
      c = 1231

      print(f'Números {a+1}, {b} e {c**2} serão mostrados.')
```

Números 1.1298731, 35466.3108012 e 1515361 serão mostrados.

```
[17]: print(f'Número a {a:->20.2%} em percentual')
      print(f'Número b {b:+<30.2e} expoente')
      print(f'Número b {b:*~35,.4f} float')
      print(f'Número c {c:"~20_b} em binário')
      print(f'Número c {c:^20_b} novamente')
```

Número a -----12.99% em percentual
 Número b 3.55e+04+++++++ expoente
 Número b *****35,466.3108***** float
 Número c ""100_1100_1111"" em binário
 Número c 100_1100_1111 novamente

```
[18]: print(f'Número a {a*2:->20,.2f} foi multiplicado por 2')
      print(f'Número b {b*3:->20,.2f} foi multiplicado por 3')
      print(f'Número c {c*4:->20,.2f} foi multiplicado por 4')
```

Número a -----0.26 foi multiplicado por 2
 Número b -----106,398.93 foi multiplicado por 3
 Número c -----4,924.00 foi multiplicado por 4

Para definir a configuração brasileira de numeração (“.” para separação de milhar e “,” para separação decimal) a biblioteca padrão locale pode ser utilizada e definida formação com tipo n:

```
[19]: import locale
      locale.setlocale(locale.LC_NUMERIC, 'pt_BR')

      print(f'Número real {a:n}')
      print(f'Número real {b:n}')
      print(f'Número inteiro {c:n}')
```

Número real 0,129873
 Número real 35.466,3
 Número inteiro 1.231

Abaixo temos um modo fácil e direto de imprimir nome e valor da variável.

```
[20]: print(f'{a=} e {b=}')

a=0.1298731 e b=35466.3108012
```

3.2.2 String

Variáveis de texto são declaradas utilizando aspas simples ' ou duplas ". **strings** em Python são por padrão da Classe Unicode, ou seja, aceitam caracteres como letras com acento, ç e caracteres especiais.

Abaixo temos um exemplo associação de uma **string** com a variável **s**.

```
[21]: # Nova variável s como string
s = '!!! Python é "massa"! Fácil, versátil e 100% grátis.   !!!'
print(s)
```

```
!!! Python é "massa"! Fácil, versátil e 100% grátis.   !!!
```

Em Python, tudo é um objeto, incluindo variáveis. Isso significa que quando você cria uma variável, na verdade está criando um objeto que contém um valor específico. Toda variável conta com métodos que podem ser utilizados para modificar ou realizar testes na variável. Isso significa que você pode realizar operações e acessar informações sobre esses objetos através de métodos associados a eles.

Abaixo temos exemplo de métodos que transformam o texto salvo na variável em minúscula, em maiúscula e com as primeiras letras em maiúsculas.

```
[22]: print(s.lower())      # Texto em minúsculo
print(s.upper())          # Texto em maiúsculo
print(s.title())          # Texto em formato de título
```

```
!!! python é "massa"! fácil, versátil e 100% grátis.   !!!
!!! PYTHON É "MASSA"! FÁCIL, VERSÁTIL E 100% GRÁTIS.   !!!
!!! Python É "Massa"! Fácil, Versátil E 100% Grátis.   !!!
```

Abaixo estão listados os métodos que podem ser utilizados em uma variável do tipo **string**.

capitalize	index	isspace	removesuffix	startswith
casefold	isalnum	istitle	replace	strip
center	isalpha	isupper	rfind	swapcase
count	isascii	join	rindex	title
encode	isdecimal	ljust	rjust	translate
endswith	isdigit	lower	rpartition	upper
expandtabs	isidentifier	lstrip	rsplit	zfill
find	islower	maketrans	rstrip	
format	isnumeric	partition	split	
format_map	isprintable	removeprefix	splitlines	

Mais alguns exemplos de modificação do valor da variável utilizando métodos.

```
[23]: print(s)
print(s.strip('!'))      # Limpar espaços vazios dos extremos
print(s.rstrip('!'))     # Limpar espaços vazios da direita
print(s.strip('!').strip()) # Limpar espaços e exclamações
s = s.strip('!').strip().upper() # Limpar espaços, ! e colocar maiúsculo
print(s)
```

```

!!! Python é "massa"! Fácil, versátil e 100% grátis.   !!!
Python é "massa"! Fácil, versátil e 100% grátis.
!!! Python é "massa"! Fácil, versátil e 100% grátis.
Python é "massa"! Fácil, versátil e 100% grátis.
PYTHON É "MASSA"! FÁCIL, VERSÁTIL E 100% GRÁTIS.

```

```

[24]: print(s.replace('I', 'i')) # Substituir I por i
      print(s.count('R'))        # Contar quantidade de R
      print(s.split())           # Separar usando espaço
      print(s.split('R'))        # Separar usando R

```

```
PYTHON É "MASSA"! FÁCIL, VERSÁTIL E 100% GRÁTIS.
```

```
2
```

```

['PYTHON', 'É', '"MASSA"', 'FÁCIL,', 'VERSÁTIL', 'E', '100%', 'GRÁTIS.']
['PYTHON É "MASSA"! FÁCIL, VE', 'SÁTIL E 100% G', 'ÁTIS.']

```

Uma forma muito poderosa de trabalhar com **strings** oir meio de padrões são as Expressões Regulares. Expressões Regulares é uma metalinguagem de definição de padrões de texto utilizados para lidar com combinações de caracteres em uma **string**. Trata-se de um assunto extenso com livros dedicados ao assunto.

Aqui nos vamos nos ater a como a implantação de Expressões Regulares no Python por meio da biblioteca **re**. Seguem exemplos de como realizar seleção, separação e substituição de parte de texto.

```

[25]: import re

# ? - 0 ou 1 ocorrências
# * - 0 ou mais ocorrências
# + - 1 ou mais ocorrências
# \w - caracteres do alfabeto inglês
# \W - caracteres que não estão no conjunto definido por \w
# \d - dígitos numéricos
# \D - caracteres que não estão no conjunto definido por \d
# \s - caractere espaço simples
# \S - caracteres que não estão no conjunto definido por \s
# \t - caractere tab
# \n - caractere de nova linha

```

```

[26]: # Encontrar entre duas e 4 ocorrências consecutivas do dígito 5
      re.findall('5{2,4}', 'adft12355554855759')

```

```
[26]: ['5555', '55']
```

```

[27]: # Encotrar número 12 ou 123
      re.findall('123?', '123 12 124 132')

```

```
[27]: ['123', '12', '12']
```

```
[28]: # Listar caracteres diferentes de a, b e c que seguem o número 12
# Note que os parênteses neste exemplo servem para definir qual dígito será
↳ listado
re.findall('12([~a-c])', '12a 12c 12d 125 13d')
```

```
[28]: ['d', '5']
```

```
[29]: # Encontra números no texto
re.findall('[0-9]+', 'Em 2022 exitiam na Chesf em torno de 3200 funcionários.')
```

```
[29]: ['2022', '3200']
```

```
[30]: # Encontrar e-mails em um texto
re.findall(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b',
'Esta semana meu e-mail mudou de funcionario@empresa.gov.br para
↳ funcionario@cempresa.com.br.')
```

```
[30]: ['funcionario@empresa.gov.br', 'funcionario@cempresa.com.br']
```

```
[31]: # Encontrar mais 3 ou mais dígitos consecutivos que seja formado por 0, 1, 2, 3,
↳ 4 ou 5
re.search('[0-5]{3,}', 'adft12354879').group()
```

```
[31]: '12354'
```

```
[32]: # Encontrar mais 3 ou mais dígitos consecutivos que seja formados por números
re.search('\d{3,}', 'adft12354879').group()
```

```
[32]: '12354879'
```

```
[33]: # Encontrar mais 3 ou mais dígitos consecutivos que não seja formados por números
re.search('\D{3,}', 'adft12354879').group()
```

```
[33]: 'adft'
```

```
[34]: # Separar texto usando caracteres entre a e f
re.split('[a-f]', '0a3b9z5f99p')
```

```
[34]: ['0', '3', '9z5', '99p']
```

```
[35]: # Separar texto usando string formado de caracteres entre a e f
re.split('[a-f]+', '0ad3kbd9t')
```

```
[35]: ['0', '3k', '9t']
```

```
[36]: re.sub(':', '|', '.', '0:3;9,10')
```

```
[36]: '0.3.9.10'
```

```
[37]: re.sub('\.+', '.', '0.....3...9..10.....5')
```

```
[37]: '0.3.9.10.5'
```

Pode-se tratar textos lidos de um arquivo em padrão texto ou mesmo de uma lista de um arquivo MS Excel com muitas células e criação de uma nova coluna listando os textos tratados.

3.2.3 Listas

A forma mais comum de tratar listas no Python é utilizando a classe `list`. Muitos pacotes usam como base o tipo de lista `ndarray` do pacote NumPy em computação científica. Os objetos `ndarray` e `list` tem comportamentos e utilizações bem distintas. Nesta seção veremos funcionalidades básicas da classe `list` do Python e abordaremos listas do tipo `ndarray` quando falarmos do NumPy.

Métodos de uma Listas

append	index	sort
clear	insert	
copy	pop	
count	remove	
extend	reverse	

A forma básica de utilizar uma lista em Python é usar chaves para delimitar o início e fim da lista tendo seus elementos separados por vírgula. A lista pode contar elementos de diversos tipos, inclusive outras listas.

```
[38]: l1 = [1, [2, 'three'], 4.5] # Lista com número inteiro, float e outra lista
      l2 = range(2, 20, 2) # Lista de números inteiros no intervalo [2,20) com
      ↪ intervalo de 2 elementos
      print(l1, '\n')
      print(l2, '\n')
      print(list(l2), '\n')
```

```
[1, [2, 'three'], 4.5]
```

```
range(2, 20, 2)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
[39]: lista = ['a', 'b', 'c', 'd', 'e', 'f']
```

```
[40]: lista.append('c') # Adicionar "c" na lista
      lista
```

```
[40]: ['a', 'b', 'c', 'd', 'e', 'f', 'c']
```

```
[41]: lista.count('c') # Contar "c" na lista
```

```
[41]: 2
```

```
[42]: lista.extend([3, 'Python', 3.14]) # Expandir lista com outra lista
      lista
```

```
[42]: ['a', 'b', 'c', 'd', 'e', 'f', 'c', 3, 'Python', 3.14]
```

```
[43]: print(lista)
      lista.pop(-1) # Retirar item de lista por posição (último elemento)
      lista.pop(2)  # Retirar item de lista por posição (terceito elemento)
      lista.remove('a')
      lista.remove(3)
      print(lista)
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'c', 3, 'Python', 3.14]
['b', 'd', 'e', 'f', 'c', 'Python']
```

```
[44]: a = [3.14, 5.1, 1.73, 9, 4, 2.7182]
      a.sort()
      a
```

```
[44]: [1.73, 2.7182, 3.14, 4, 5.1, 9]
```

A biblioteca padrão do Python conta com o módulo `random` com algumas funções básicas de geração de números randômicos (ou pseudo randômicos como os mais rigorosos gostam de definir), bem como escolha aleatória em uma lista de números, mas existe a biblioteca `random` do NumPy com muito mais recursos disponíveis.

Funções disponíveis na biblioteca `random` padrão do Python.

betavariate	lognormvariate	seed
choice	normalvariate	setstate
choices	paretovariate	shuffle
expovariate	randbytes	triangular
gammavariate	randint	uniform
gauss	random	vonmisesvariate
getrandbits	randrange	weibullvariate
getstate	sample	

```
[45]: import random

      random.random() # Número float randômico
```

```
[45]: 0.7739024503719453
```

```
[46]: b = ['abacate', 'banana', 'côco', 'damasco', 'embaúba', 'figo']

      random.shuffle(b) # Misturas a lista b
      print(b)
```

```

b.sort() # Ordenar a lista "b"
print(b)

print(random.choice(b)) # Escolhe um dos elementos da lista de forma aleatória

print(random.sample(b, 3)) # Escolhe uma amostra de 3 elementos

```

```

['figo', 'côco', 'abacate', 'damasco', 'embaúba', 'banana']
['abacate', 'banana', 'côco', 'damasco', 'embaúba', 'figo']
embaúba
['embaúba', 'figo', 'banana']

```

```
[47]: sorted(b, key=len) # Ordenar a lista "b" com base no tamanho da palavra
```

```
[47]: ['côco', 'figo', 'banana', 'abacate', 'damasco', 'embaúba']
```

```
[48]: [1,2,3] + [4,5,6] # Juntar listas
```

```
[48]: [1, 2, 3, 4, 5, 6]
```

```
[49]: 3 * [1,2,3] # Repetir listas
```

```
[49]: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

A seleção de elementos de uma lista é uma funcionalidade importante e deve-se entender bem como funciona. O *slice* de uma lista tem a seguinte estrutura:

Lista[de: até: passo]

Na figura abaixo é detalhado como se pode selecionar elementos de uma lista. A contagem dos elementos inicia do zero e pode-se selecionar um elemento ou um intervalo de elementos. Para a seleção de intervalo ficar mais natural, pense que o intervalo é definido pelo índice entre os elementos e não o elemento em si, como sugere a figura.

Index from rear:	-6	-5	-4	-3	-2	-1
Index from front:	0	1	2	3	4	5
	+---	+---	+---	+---	+---	+---
	a	b	c	d	e	f
	+---	+---	+---	+---	+---	+---
Slice from front:	:	1	2	3	4	5 :
Slice from rear:	:	-5	-4	-3	-2	-1 :

Pode-se definir o índice do elemento da esquerda para direita utilizando números inteiros positivos ou da direita para esquerda utilizando números inteiros negativos.

```
[50]: lista = ['a', 'b', 'c', 'd', 'e', 'f']
```

```
[51]: lista[3]  # Quarto elemento
```

```
[51]: 'd'
```

```
[52]: lista[-2]  # Penúltimo elemento
```

```
[52]: 'e'
```

```
[53]: # Da posição entre b e c até a posição entre penúltimo e último elemento  
lista[2:-1]
```

```
[53]: ['c', 'd', 'e']
```

```
[54]: # Da posição entre a e b, até a posição entre e e f, de dois em dois elementos  
lista[1:-1:2]
```

```
[54]: ['b', 'd']
```

Não colocar o número de índice da posição “de” do *slice* é equivalente a colocar zero. Da mesma forma não colocar o número de índice da posição “até” é equivalente a colocar a última posição da lista.

```
[55]: print(lista[0:3])  
print(lista[:3])  # Igual ao anterior
```

```
['a', 'b', 'c']  
['a', 'b', 'c']
```

```
[56]: print(lista[3:6])  
print(lista[3:])  # Igual ao anterior
```

```
['d', 'e', 'f']  
['d', 'e', 'f']
```

Uma forma fácil de inverter uma lista é pedir a lista do início ao final com passo -1.

```
[57]: lista[::-1]  # Inverte lista, passo = -1
```

```
[57]: ['f', 'e', 'd', 'c', 'b', 'a']
```

No Python se poder fazer muita manipulação de lista em apenas uma linha, utilizando métodos sobre outros métodos ou, no caso deste exemplo, faz uma segunda seleção em cima de uma seleção já realizada. Cuidado para não cair na armadilha de ao enxugar o script não comprometer o entendimento, a facilidade de entender o código.

```
[58]: lista[::-1][:-3]  # Da lista invertida, pegar até o antepenúltimo elemento
```

```
[58]: ['f', 'e', 'd']
```


Além da funcionalidade de seleção, a técnica de *slice* pode servir para alterar a lista.

```
[59]: # Redefinindo calor de elementos
lista[0] = 'Primeiro'
lista[2:4] = ['Bola', 'Casa']
lista
```

```
[59]: ['Primeiro', 'b', 'Bola', 'Casa', 'e', 'f']
```

Temos agora um exemplo de modificação de um elemento de uma lista dentro de outra lista. Em `l1del[1][2]` definimos que deve ser selecionado o segundo elemento da primeira lista (índice 1) e deste elemento selecionado deve-se selecionar o terceiro elemento (índice 2).

```
[60]: l1del = [['_', '_', '_', '_'],
              ['_', '_', '_'],
              ['_', '_']]
l1del[1][2] = 'X'
l1del
```

```
[60]: [['_', '_', '_', '_'], ['_', '_', 'X'], ['_', '_']]
```

A técnica de *slice* também funciona da mesma forma para variáveis do tipo `string` e `tuple`.

```
[61]: s = '!!! Python é de "torar"! Fácil, versátil e 100% grátis. !!!'
print(s[6:12])
print(s[::-1])
```

Python

```
!!! .sitárg %001 e litásrev ,licáF !"rarot" ed é nohtyP !!!
```

```
[62]: # Salvar os elementos da lista em variáveis independentes
a, b, c = ['Python', 'C++', 'javascript']

print(a)
print(b)
print(c)
```

Python

C++

javascript

Uma funcionalidade muito útil nas listas é chamada **desempacotamento de variáveis** que é utilizada através do símbolo `*`. Abaixo temos um exemplo de junção de valores em uma variável. No exemplo usamos de forma separada as variáveis `a` e `b` e colocamos os demais valores na variável `c`.

```
[63]: a, b, *c = range(5) # * => as demais em "c"
print(a)
print(b)
print(c)
```

```
0
1
[2, 3, 4]
```

No próximo exemplo pegamos o primeiro valor e associamos a variável **a**, o último valor associado a variável **c** e os demais valores a variável **b**.

```
[64]: a, *b, c = range(5)  # * => as demais em "b"
      print(a)
      print(b)
      print(c)
```

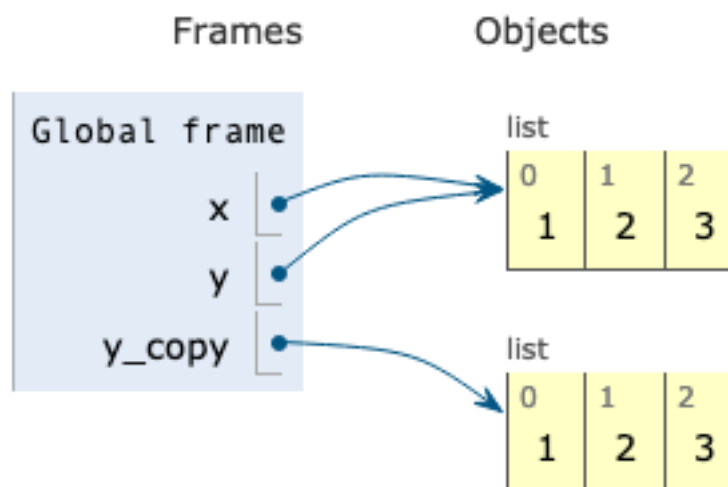
```
0
[1, 2, 3]
4
```

Podemos utilizar o ***** para desempacotar uma lista a ser inserida em uma função. No exemplo abaixo o comando `print(b)` tem como saída a lista **b**, mas se colocamos um asterisco antes da variável a saída é equivalente a `print(b[0], b[1], b[2])`.

```
[65]: print(b)
      print(*b)
      print(b[0], b[1], b[2])
```

```
[1, 2, 3]
1 2 3
1 2 3
```

As variáveis devem ser pensadas como etiquetas de valores e não como caixas. As variáveis são referências que apontam para endereços de memória e não para um valor em si. Se o valor gravado em um endereço de memória é alterado, a variável tem seu valor alterado. O comportamento que vamos exemplificar ocorre com todas as variáveis mutáveis (listas, set e dicionários)



```
[66]: x = [1, 2, 3]
print(f'x = {x}', '\n')

y = x
y_cópia = x.copy()

print(f'Identidade de x      = {id(x)}')
print(f'Identidade de y      = {id(y)}')
print(f'Identidade de y_cópia = {id(y_cópia)}', '\n')

print('Mudando x[0]', '\n')

x[0] = 5

print(f'x = {x}')
print(f'y = {y}')
print(f'y_cópia = {y_cópia}')
```

x = [1, 2, 3]

Identidade de x = 3102194138816

Identidade de y = 3102194138816

Identidade de y_cópia = 3102194138944

Mudando x[0]

x = [5, 2, 3]

y = [5, 2, 3]

y_cópia = [1, 2, 3]

```
[67]: x = [1, 2, 3]
y = [1, 2, 3]

print(f'x = {x}')
print(f'y = {y}', '\n')

print(f'Identidade de x = {id(x)}')
print(f'Identidade de y = {id(y)}', '\n')

print('Mudando x[0]', '\n')

x[0] = 5

print(f'x = {x}')
print(f'y = {y}')
```

x = [1, 2, 3]

```
y = [1, 2, 3]
```

```
Identidade de x = 3102194137728
```

```
Identidade de y = 3102194137664
```

```
Mudando x[0]
```

```
x = [5, 2, 3]
```

```
y = [1, 2, 3]
```

```
[68]: a = 1 # Variável imutável
      b = 1 # Variável imutável

      # Os valores de a e b são iguais e estão no mesmo local na memória
      print('a is b: ', a is b)
      print('a == b: ', a == b)
      print()

      a1 = [1] # Variável mutável
      b1 = [1] # Variável mutável

      # Os valores de a1 e b1 são iguais, mas não estão no mesmo local na memória
      print('a1 is b1: ', a1 is b1)
      print('a1 == b1: ', a1 == b1)
```

```
a is b: True
```

```
a == b: True
```

```
a1 is b1: False
```

```
a1 == b1: True
```

3.2.4 Tupla

Tuplas são listas imutáveis.

```
[69]: tupla = (1, 'spam', 4, 'U')
      print(tupla)
      (type(tupla))
```

```
(1, 'spam', 4, 'U')
```

```
[69]: tuple
```

A tupla pode ser mais eficiente que a lista do ponto de vista de utilização de memória.

3.2.5 Dicionário

```
[70]: d = {'Python': 4, 'C++':5, 'R':0}
      d
```

```
[70]: {'Python': 4, 'C++': 5, 'R': 0}
```

```
[71]: print(d.keys())
      print(d.values())
```

```
dict_keys(['Python', 'C++', 'R'])
dict_values([4, 5, 0])
```

```
[72]: d['Python']
```

```
[72]: 4
```

```
[73]: d['Python'] += 1 # d['Python'] = d['Python'] + 1
      d
```

```
[73]: {'Python': 5, 'C++': 5, 'R': 0}
```

```
[74]: d['Julia'] = 'nova'
      d
```

```
[74]: {'Python': 5, 'C++': 5, 'R': 0, 'Julia': 'nova'}
```

Métodos de dicionário

clear	pop
copy	popitem
fromkeys	setdefault
get	update
items	values
keys	

```
[75]: print(d.get('Python', 'Não achei')) # Se chave Python existir, senão 'Não achei'
      print(d.get('Java', 'Não achei'))  # Se chave Java existir, senão 'Não achei'
```

```
5
Não achei
```

3.2.6 Conjunto (set)

Conjunto	Python
$A \setminus B$	A - B
$A \cup B$	A B
$A \cap B$	A & B

Conjunto	Python
$A \subset B$	<code>A < B</code>
$A \triangle B$	<code>A ^ B</code>
$e \in B$	<code>e in B</code>

Em notebooks do Colab ou Jupyter podem ser utilizados símbolos matemáticos utilizando o padrão do \LaTeX

```
[76]: A = {'a', 'b', 'c', 'd', 1, 2, 3, 4}
      B = {'c', 'd', 'e', 'f', 3, 4, 5, 6}
      C = {'c', 5}

      print(type(A))

      print(f'A não em B: {A-B}')
      print(f'A união B: {A | B}')
      print(f'A intersecção B: {A & B}')
      print(f'C está contido em A: {C < A}')
      print(f'C está contido em B: {C < B}')
      print(f'ou em A ou em B: {A ^ B}')
      print(f'3 pertence a B: {3 in B}')
```

```
<class 'set'>
A não em B: {1, 2, 'b', 'a'}
A união B: {'f', 1, 2, 'b', 3, 4, 5, 6, 'c', 'd', 'a', 'e'}
A intersecção B: {3, 4, 'c', 'd'}
C está contido em A: False
C está contido em B: True
ou em A ou em B: {'f', 1, 2, 5, 6, 'b', 'a', 'e'}
3 pertence a B: True
```

Métodos de set

clear	discard	issubset	symmetric_difference
copy	intersection	issuperset	symmetric_difference_update
difference	intersection_update	pop	union
difference_update	isdisjoint	remove	update

```
[77]: lista = [1,2,3,4,3,2,3,4,5,3,1,5,6,4,2,7,3,2]
      D = set(lista)
      D
```

```
[77]: {1, 2, 3, 4, 5, 6, 7}
```

3.3 Condicional

A indentação é uma característica importante no Python, pois de acordo com a indentação se define o que está dentro da declaração condicional. Não se utiliza marcadores como `begin` e `end` ou outro

delimitador como chaves.

A indentação também é utilizada em Loops e funções.

```
[78]: x = 3

if 1 <= x < 4: # x em [1,4)
    print('Dentro do intervalo')
elif x%2==0:
    print('Fora do intervalo, mas é par')
elif type(x)==int:
    print('Fora do intervalo, não é par, mas é inteiro')
else:
    print('Fora do intervalo e não é inteiro')
```

Dentro do intervalo

```
[79]: x = 4

if 1 <= x < 4: # x em [1,4)
    print('Dentro do intervalo')
elif x%2==0:
    print('Fora do intervalo, mas é par')
elif type(x)==int:
    print('Fora do intervalo, não é par, mas é inteiro')
else:
    print('Fora do intervalo e não é inteiro')
```

Fora do intervalo, mas é par

```
[80]: ls = []

if ls:
    print('Lista vazia')
else:
    ls.append(4)
    if ls:
        print(f'Agooooora: {ls}')
```

Agooooora: [4]

```
[81]: a = 2
      b = 4

if a%2==0 and b%2==0:
    print('Ambos pares')
```

Ambos pares

```
[82]: print(f'[]: {bool([])!s:>10}')
      print(f'(): {bool(())!s:>10}')
      print(f'set(): {bool(set())!s:>7}')
      print(f'0: {bool(0)!s:>11}')
      print(f'": {bool("")!s:>10}')
```

```
[]:      False
():      False
set():   False
0:       False
"":      False
```

Atribuição condicional de variável

```
[83]: k = 1
      x = 5 if k==1 else 4
      print(x)

      k = 0
      x = 5 if k==1 else 4
      print(x)
```

```
5
4
```

3.4 Laços de repetição (*Loops*)

Declarações	Utilização
<code>pass</code>	Reserva de espaço vazio
<code>break</code>	Saída de laço
<code>continue</code>	Continuar o laço

O `for` do Python funciona como o `foreach` existente em algumas linguagens de programação.

```
[84]: for letra in ['a', 'b', 'c']:
      print(letra)
```

```
a
b
c
```

```
[85]: for i in range(5):
      print(5*'-')
      print(f'Valor {i}')

      print('Acabou o for')
```

```
-----
Valor 0
```



```
-----  
Valor 1  
-----  
Valor 2  
-----  
Valor 3  
-----  
Valor 4  
Acabou o for
```

```
[86]: letra_proibida = 'c'  
  
for l in ['a', 'b', 'c', 'd', 'e']:  
    if l != letra_proibida:  
        print(f'{l}, não é o {letra_proibida}')  
    else:  
        print(f'Chegou o {letra_proibida}')  
        break  
else: # só se não for dado o comando break  
    print('Não precisei para')  
  
print('E acabou-se')
```

```
a, não é o c  
b, não é o c  
Chegou o c  
E acabou-se
```

```
[87]: letra_proibida = 'f'  
  
for l in ['a', 'b', 'c', 'd', 'e']:  
    if l != letra_proibida:  
        print(f'{l}, não é o {letra_proibida}')  
    else:  
        print(f'Chegou o {letra_proibida}')  
        break  
else: # só se não for dado o comando break  
    print('Não precisei para')  
  
print('E acabou-se')
```

```
a, não é o f  
b, não é o f  
c, não é o f  
d, não é o f  
e, não é o f  
Não precisei para  
E acabou-se
```

```
[88]: for n,c in zip([1,2,3], ['a', 'b','c']): # Juntar duas listas no for
      print(f'Número {n} e letra {c}')
```

Número 1 e letra a
Número 2 e letra b
Número 3 e letra c

```
[89]: for n,c in zip([1,2,3,4], ['a', 'b','c','d']):
      print(n*c)
```

a
bb
ccc
dddd

```
[90]: # for dentro de for
      for n in [1,2,3]:
          for c in ['a', 'b','c']:
              print(n*c)
```

a
b
c
aa
bb
cc
aaa
bbb
ccc

```
[91]: for n,c in enumerate(['a', 'b','c', 'd', 'e']): # Enumeração de elementos
      print(f'{n+1}) letra {c}')
```

1) letra a
2) letra b
3) letra c
4) letra d
5) letra e

```
[92]: k = 0
      while k <= 5:
          print(f'0 número é {k}')
          k += 1
```

0 número é 0
0 número é 1
0 número é 2
0 número é 3
0 número é 4
0 número é 5

3.4.1 List Comprehension (Listcomps)

```
[93]: # Lista com números de 0 a 9
      lista = range(10)

      # Para cada valor da lista, transformar em string e repetir 3 vezes
      [str(i)*3 for i in lista]
```

```
[93]: ['000', '111', '222', '333', '444', '555', '666', '777', '888', '999']
```

```
[94]: multi = []
      for i in range(1,5):
          for j in range(11,15):
              multi.append(i*j)
      multi
```

```
[94]: [11, 12, 13, 14, 22, 24, 26, 28, 33, 36, 39, 42, 44, 48, 52, 56]
```

```
[95]: # Equivalente a estrutura anterior
      [i*j for i in range(1,5) for j in range(11,15)]
```

```
[95]: [11, 12, 13, 14, 22, 24, 26, 28, 33, 36, 39, 42, 44, 48, 52, 56]
```

```
[96]: par_pow = []
      for i in lista:
          if i%2==0:
              par_pow.append(i**2)
      par_pow
```

```
[96]: [0, 4, 16, 36, 64]
```

```
[97]: # Equivalente a estrutura anterior
      [i**2 for i in lista if i%2==0]
```

```
[97]: [0, 4, 16, 36, 64]
```

```
[98]: # Produto Cartesiano de listas
      [(i,j) for i in 'abcd'
          for j in range(4)]
```

```
[98]: [('a', 0),
      ('a', 1),
      ('a', 2),
      ('a', 3),
      ('b', 0),
      ('b', 1),
      ('b', 2),
      ('b', 3),
```

```
('c', 0),
('c', 1),
('c', 2),
('c', 3),
('d', 0),
('d', 1),
('d', 2),
('d', 3)]
```

```
[99]: # Dicionário
      {k:v**2 for k,v in zip('abcdef', range(1,7))}
```

```
[99]: {'a': 1, 'b': 4, 'c': 9, 'd': 16, 'e': 25, 'f': 36}
```

```
[100]: # Tupla
      tupla = (abs(i) for i in range(-3,4))
      print(type(tupla))
      print(list(tupla))
```

```
<class 'generator'>
[3, 2, 1, 0, 1, 2, 3]
```

3.4.2 Estrutura de paradigma funcional com função map

```
[101]: fx = lambda x: x**2 -3*x + 5

      print(fx(10))

      print(list(map(fx, [1,2,3,4,5])))
```

```
75
[3, 3, 5, 9, 15]
```

```
[102]: def tratar(s):
      s = str(s).upper()
      s = s.replace('4', 'X')
      return 'EQ: ' + s

      print(tratar('04de'))

      entrada = ['05c1', '03T2', '04p2', '14d1']
      print(list(map(tratar, entrada)))
```

```
EQ: OXDE
['EQ: 05C1', 'EQ: 03T2', 'EQ: OXP2', 'EQ: 1XD1']
```

3.5 Tratamento de erro (try/except/else)

```
[103]: try:
        print('Vamos ver se Python é bom mesmo')
        print(2 * 'Python')
    except:
        print('Aí tais querendo muito')
    else:
        print('Deu certo')

    print('Teste finalizado')
```

Vamos ver se Python é bom mesmo

PythonPython

Deu certo

Teste finalizado

```
>>> 2 + 'Python'
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-1-e645bad84159> in <module>
----> 1 2 + 'Python'
```

TypeError: unsupported operand type(s) for +: 'int' and 'str'

```
[104]: try:
        print('Vamos ver se Python é bom mesmo')
        print(2 + 'Python')
    except:
        print('Aí tais querendo muito')
    else:
        print('Deu certo')

    print('Teste finalizado')
```

Vamos ver se Python é bom mesmo

Aí tais querendo muito

Teste finalizado

3.6 Funções

```
[105]: def func1(a=2, b=3):
        return a**b

print(func1(3,4))      # 3**4
print(func1())          # 2**3
print(func1(b=2))      # 2**2
print(func1(3))         # 3**3
```

```
print(func1(b=3,a=4))    # 4**3
```

```
81
8
4
27
64
```

```
[106]: def func2(a, b, *c):
        print(a,b)
        print(c)
        return a*b + sum(c)

func2(2, 3, 4, 5, 6)    # 2**3 + (4+5+6)
```

```
2 3
(4, 5, 6)
```

```
[106]: 23
```

```
[107]: def factorial(n):
        if n<2:
            return 1
        else:
            return n * factorial(n-1)

factorial(5)
```

```
[107]: 120
```

4 NumPy

O Python é uma linguagem de programação de aplicação geral, não é uma linguagem desenvolvida originalmente para utilização em computação científica. Se valendo da característica de relativa facilidade de criação de pacotes para o Python aplicando linguagens como C/C++ e Fortran, o NumPy foi desenvolvido com intuito de facilitar a computação científica no Python de forma performática. Desde a segunda metade da década de 2000 se tornou um pacote fundamental para computação científica em Python.

O NumPy é utilizado como base de outros importantes pacotes utilizados em computação científica como [Matplotlib](#), [SciPy](#), [pandas](#), [TensorFlow](#), [Scikit-Learn](#), [Statsmodels](#), [CVXPY](#), [PyWavelets](#), entre outros. São disponibilizadas funções pré-compiladas em C, C++ e Fortran, muitas provenientes de pacotes matemáticos já consolidados como [BLAS](#) e [LAPACK](#).

Não vamos abordar neste material o [SciPy](#), mas este pacote é importante na área de computação científica com várias bibliotecas como por exemplo [scipy.integrate](#) (Integração e *Ordinary Differential Equations* - ODEs), [scipy.interpolate](#) (Interpolação), [scipy.optimize](#) (Otimização e zeros da função), [scipy.signal](#) (Processamento de Sinais) e [scipy.stats](#) (funções Estatísticas).

Os dois grandes diferenciais do NumPy são o objeto `ndarray` e as funções do tipo `ufunc`. O `ndarray` (*N-dimensional array*) é um objeto que representa um array multidimensional, com tipagem homogênea e com itens de tamanho fixo na memória. As `ufunc` (*Universal Function*) processam `ndarray`, evitando utilização de estrutura de laços, otimizando a execução do código.

Aos que conhecem Matlab e R, podem usar as referências [NumPy for MATLAB users](#) e [NumPy for R \(and S-Plus\) users](#) respectivamente.

Alguns conjuntos de funções disponíveis

Pacote	Descrição
numpy.polynomial	Polinômios
numpy.linalg	Álgebra Linear
numpy.random	Amostras Randômicas
numpy.fft	Transformada Discreta de Fourier

Executar no ipython/Jupyter/Colab `%pylab` ou `from pylab import *` em script Python é equivalente a importações de 27 módulos (entre eles `numpy` como `np`, `matplotlib.pyplot` como `plt`, `numpy.random` como `random`, `numpy.fft` como `fft`, `numpy.linalg` como `linalg`, entre outros). Seria algo como:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import random
from numpy import linalg
.
.
.
```

Além disso, são importados quase 900 funções e constantes para serem utilizadas diretamente. São importadas constantes (`pi`, `e`, `Inf`, `NaN`), funções trigonométricas (`sin`, `sinh`, `arcsin`, `deg2rad`, etc), estatísticas e probabilidade (`mean`, `median`, `std`, `cov`, `rand`, `randn`, `choice`, `poisson`, etc), álgebra linear e manipulação de matrizes (`det`, `inv`, `solve`, `tensorinv`, etc), polinômios (`poly`, `root`, `polyfit`, etc). Seria algo equivalente a:

```
from matplotlib.pyplot import *
from numpy import *
from numpy.fft import *
from numpy.linalg import *
from numpy.polynomial import *
from numpy.random import *
.
.
.
```

```
[108]: from pylab import *
```

```
[109]: # Versão do NumPy utilizada neste material
np.__version__
```

```
[109]: '1.26.4'
```

```
[110]: import sys
x1 = linspace(0, 2*pi, 128) # Array de 0 a 2pi com 128 amostras
x2 = [float(i) for i in x1] # Lista baseado em x1 com elementos float
print('x1:', type(x1), 'com elementos do tipo ', x1.dtype)
print('x2:', type(x2), 'com elementos do tipo ', type(x2[0]))
```

```
x1: <class 'numpy.ndarray'> com elementos do tipo float64
x2: <class 'list'> com elementos do tipo <class 'float'>
```

ndarray é mais eficiente utilizando operações de forma vetorial que list utilizando loops. No exemplo abaixo a operação de elevar 10.000 elementos de um ndarray é medido em μ s (microsegundos) e a mesma operação utilizando list é medida em ms (milissegundos), lembrando que 1.000 μ s é equivalente a 1 ms.

```
[111]: x1 = linspace(0, 10*pi, 10000) # Array de 0 a 10pi com 10.000 amostras
x2 = [float(i) for i in x1] # Lista baseado em x1 com elementos float

print(f'Tipo de x1: {type(x1)}')
print(f'Tipo de x2: {type(x2)}')
```

```
Tipo de x1: <class 'numpy.ndarray'>
Tipo de x2: <class 'list'>
```

```
[112]: %timeit x1**2
```

```
4.09  $\mu$ s  $\pm$  139 ns per loop (mean  $\pm$  std. dev. of 7 runs, 100,000 loops each)
```

```
[113]: def fsqrt(x):
    list_temp = []
    for i in x:
        list_temp.append(i**2)
    return list_temp

%timeit fsqrt(x2)
```

```
2.12 ms  $\pm$  231  $\mu$ s per loop (mean  $\pm$  std. dev. of 7 runs, 100 loops each)
```

Existem várias formas de [criar ndarray no NumPy](#).

```
[114]: # Criando array
a = array([[10, 2, 1],
           [1, 5, 1],
           [2, 3, 10]]) # ndarray 3x3
b = arange(0, 20, 0.5).reshape(8, 5) # (8, -1) seria calculado as 5 colunas
c = linspace(0, 2*np.pi, 32)
d = ones([3,3], dtype=complex) # dtype poderia ser usado nas outras técnicas
```



```
[115]: print(a)
print()
print(b)
print()
print(c)
print()
print(d)
```

```
[[10  2  1]
 [ 1  5  1]
 [ 2  3 10]]
```

```
[[ 0.  0.5  1.  1.5  2. ]
 [ 2.5  3.  3.5  4.  4.5]
 [ 5.  5.5  6.  6.5  7. ]
 [ 7.5  8.  8.5  9.  9.5]
 [10. 10.5 11. 11.5 12. ]
 [12.5 13. 13.5 14. 14.5]
 [15. 15.5 16. 16.5 17. ]
 [17.5 18. 18.5 19. 19.5]]
```

```
[0.          0.2026834  0.40536679 0.60805019 0.81073359 1.01341699
 1.21610038 1.41878378 1.62146718 1.82415057 2.02683397 2.22951737
 2.43220076 2.63488416 2.83756756 3.04025096 3.24293435 3.44561775
 3.64830115 3.85098454 4.05366794 4.25635134 4.45903473 4.66171813
 4.86440153 5.06708493 5.26976832 5.47245172 5.67513512 5.87781851
 6.08050191 6.28318531]
```

```
[[1.+0.j 1.+0.j 1.+0.j]
 [1.+0.j 1.+0.j 1.+0.j]
 [1.+0.j 1.+0.j 1.+0.j]]
```

Além dos métodos mostrados para criar `ndarray` podemos carregar ou salvar dados em arquivos em formato específico do NumPy/Python ([load/save](#)) ou de arquivos texto ([loadtxt/savetxt](#)). Também é possível criar `dnarray` proveniente de arquivo no padrão do Matlab, mas utilizando função do pacote SciPy ([scipy.io.loadmat](#)).

Como exemplo vamos gravar o `ndarray` de `b` em um arquivo “`matr.dat`” e recarregar.

```
[116]: savetxt('matr.dat', b)

b1 = loadtxt('matr.dat')

print(b1)  # Mostrar os valores do ndarray b1
print()
print(f'Tipo de dados em b1: {b1.dtype}')  # Tipo dos dados salvo no ndarray b1
```

```
[[ 0.  0.5  1.  1.5  2. ]
 [ 2.5  3.  3.5  4.  4.5]]
```

```
[ 5.   5.5  6.   6.5  7. ]
[ 7.5  8.   8.5  9.   9.5]
[10.   10.5 11.   11.5 12. ]
[12.5 13.   13.5 14.   14.5]
[15.   15.5 16.   16.5 17. ]
[17.5 18.   18.5 19.   19.5]]
```

Tipo de dados em b1: float64

As listas do tipo `ndarray` têm elementos do mesmo tipo e este tipo é normalmente definido durante a criação da lista, mas podemos forçar um tipo para os elementos para, por exemplo, ajustarmos o uso de memória mais adequado em nosso script.

```
[117]: # ndarray com inteiros de 8 bits (inteiros de -128 a 127)
a1 = array([[10, 2, 1, 5, 20],
            [1, 5, 1, 20, 18],
            [2, 3, 10, 8, 40],
            [7, 2, 50, 2, 50],
            [0, 8, 15, 9, 3]], dtype=int8)

a2 = array([[10, 2, 1, 5, 20],
            [1, 5, 1, 20, 18],
            [2, 3, 10, 8, 40],
            [7, 2, 50, 2, 50],
            [0, 8, 15, 9, 3]])

a1s = sys.getsizeof(a1) # Memória utilizada em x1
a2s = sys.getsizeof(a2) # Memória utilizada em x2
print(f'a1 com elementos {a1.dtype} está usando {a1s:_d} bytes')
print(f'a2 com elementos {a2.dtype} está usando {a2s:_d} bytes')
print(f'a2 está usando {a2s/a1s:.2f} mais memória que a1')
```

```
a1 com elementos int8 está usando 153 bytes
a2 com elementos int32 está usando 228 bytes
a2 está usando 1.49 mais memória que a1
```

Mas cuidado que uma vez definido `ndarray` com inteiro de 32 bits não se consegue guardar inteiros maiores que a memória definida para a variável (neste caso número maior que 255).

Para inteiros podem ser definidos `uint8`, `uint16`, `uint32`, `uint64`, `int8`, `int16`, `int32` e `int64` e para números reais `float16`, `float32`, `float64` e `float128` (quando suportado). Os `dtypes` que iniciam com `u` são *unsigned*, ou seja, apenas números positivos. O `int8` aceita números entre -128 e 127, os `uint8` números entre 0 e 255, por exemplo.

```
[118]: print(iinfo(uint8)) # Inteiro positivos de 8 bits
print(iinfo(int8))
print(iinfo(int64))

print('\n\n')
```

```
print(finfo(float16))
print(finfo(float64))
```

Machine parameters for uint8

```
-----
min = 0
max = 255
-----
```

Machine parameters for int8

```
-----
min = -128
max = 127
-----
```

Machine parameters for int64

```
-----
min = -9223372036854775808
max = 9223372036854775807
-----
```

Machine parameters for float16

```
-----
precision =   3    resolution = 1.00040e-03
machep =    -10    eps =        9.76562e-04
negexp =    -11    epsneg =     4.88281e-04
minexp =    -14    tiny =       6.10352e-05
maxexp =     16    max =        6.55040e+04
nexp =       5     min =       -max
smallest_normal = 6.10352e-05    smallest_subnormal = 5.96046e-08
-----
```

Machine parameters for float64

```
-----
precision =  15    resolution = 1.0000000000000001e-15
machep =    -52    eps =        2.2204460492503131e-16
negexp =    -53    epsneg =     1.1102230246251565e-16
minexp =   -1022    tiny =       2.2250738585072014e-308
maxexp =    1024    max =        1.7976931348623157e+308
nexp =      11     min =       -max
smallest_normal = 2.2250738585072014e-308    smallest_subnormal =
4.9406564584124654e-324
-----
```

O `ndarray` é um objeto multidimensional que pode representar lista no espaço \mathbb{R}^n . Podemos representar vetores $A_{(i)}$ com uma dimensão com i elementos, matrizes $A_{(i,j)}$ com $i \times j$ elementos e tensores com n dimensões.

Descrição	Exemplo	Espaço Vetorial
Vetor	$A_{(3)} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$	\mathbb{R}
Matriz	$A_{(2,2)} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	\mathbb{R}^2
Tensor	$A_{(2,2,2)} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$	\mathbb{R}^n

4.1 Métodos de ndarray

Abaixo listamos os métodos de variáveis do tipo `ndarray` salientando em negrito alguns métodos muito utilizados.

T	choose	diagonal	imag	nonzero	round	sum
all	clip	dot	item	partition	searchsorted	swapaxes
any	compress	dtype	itemset	prod	setfield	take
argmax	conj	dump	itemsizes	ptp	setflags	tobytes
argmin	conjugate	dumps	max	put	shape	tofile
argpartition	copy	fill	mean	ravel	size	tolist
argsort	ctypes	flags	min	real	sort	tostring
astype	cumprod	flat	nbytes	repeat	squeeze	trace
base	cumsum	flatten	ndim	reshape	std	transpose
byteswap	data	getfield	newbyteorder	resize	strides	var
						view

```
[119]: b
```

```
[119]: array([[ 0. ,  0.5,  1. ,  1.5,  2. ],
              [ 2.5,  3. ,  3.5,  4. ,  4.5],
              [ 5. ,  5.5,  6. ,  6.5,  7. ],
              [ 7.5,  8. ,  8.5,  9. ,  9.5],
              [10. , 10.5, 11. , 11.5, 12. ],
              [12.5, 13. , 13.5, 14. , 14.5],
              [15. , 15.5, 16. , 16.5, 17. ],
              [17.5, 18. , 18.5, 19. , 19.5]])
```

Podemos aplicar os métodos em todos os elementos ou em um eixo específico.

```
[120]: b.mean() # Média de b
```

```
[120]: 9.75
```

```
[121]: b.mean(axis=0) # Média das colunas
```

```
[121]: array([ 8.75,  9.25,  9.75, 10.25, 10.75])
```

```
[122]: b.mean(axis=1) # Média das linhas
```

```
[122]: array([ 1. ,  3.5,  6. ,  8.5, 11. , 13.5, 16. , 18.5])
```

Por padrão a função `numpy.std` calcula desvio padrão populacional conforme fórmula abaixo.

$$\sigma = \sqrt{\frac{1}{N} \sum (x_i - \mu)^2}$$

```
[123]: b.std(axis=1) # Desvio padrão populacional de cada linha
```

```
[123]: array([0.70710678, 0.70710678, 0.70710678, 0.70710678, 0.70710678,
            0.70710678, 0.70710678, 0.70710678])
```

Para calcular desvio padrão amostral (padrão do MS Excel, R, Julia e `python.statistics`) utilize atributo `ddof=1` :

$$s_x = \sqrt{\frac{1}{n-1} \sum (x_i - \bar{x})^2}$$

```
[124]: b.std(1, ddof=1) # Desvio padrão amostral de cada linha
```

```
[124]: array([0.79056942, 0.79056942, 0.79056942, 0.79056942, 0.79056942,
            0.79056942, 0.79056942, 0.79056942])
```

```
[125]: print(b, '\n')
print(b.cumsum(axis=0)) # Soma acumulada das colunas
```

```
[[ 0.   0.5  1.   1.5  2. ]
 [ 2.5  3.   3.5  4.   4.5]
 [ 5.   5.5  6.   6.5  7. ]
 [ 7.5  8.   8.5  9.   9.5]
 [10.  10.5 11.  11.5 12. ]
 [12.5 13.  13.5 14.  14.5]
 [15.  15.5 16.  16.5 17. ]
 [17.5 18.  18.5 19.  19.5]]
```

```
[[ 0.   0.5  1.   1.5  2. ]
 [ 2.5  3.5  4.5  5.5  6.5]
 [ 7.5  9.  10.5 12.  13.5]
 [15.  17.  19.  21.  23. ]
 [25.  27.5 30.  32.5 35. ]
 [37.5 40.5 43.5 46.5 49.5]
 [52.5 56.  59.5 63.  66.5]
 [70.  74.  78.  82.  86. ]]
```

4.2 Indexing / Slicing

A seleção de elementos do `ndarray` funciona diferente do que vimos para o objeto `list` natural do Python.

```
[126]: b
```

```
[126]: array([[ 0. ,  0.5,  1. ,  1.5,  2. ],
             [ 2.5,  3. ,  3.5,  4. ,  4.5],
             [ 5. ,  5.5,  6. ,  6.5,  7. ],
             [ 7.5,  8. ,  8.5,  9. ,  9.5],
             [10. , 10.5, 11. , 11.5, 12. ],
             [12.5, 13. , 13.5, 14. , 14.5],
             [15. , 15.5, 16. , 16.5, 17. ],
             [17.5, 18. , 18.5, 19. , 19.5]])
```

```
[127]: b[0,:] # Primeira linha
```

```
[127]: array([0. , 0.5, 1. , 1.5, 2. ])
```

```
[128]: b[:,1] # Segunda coluna
```

```
[128]: array([ 0.5,  3. ,  5.5,  8. , 10.5, 13. , 15.5, 18. ])
```

```
[129]: b[1:3,1:3] # Elementos b22, b23, b32 e b33
```

```
[129]: array([[3. , 3.5],
             [5.5, 6. ]])
```

```
[130]: b[:,[[1,4]]] # Segunda e Quinta colunas
```

```
[130]: array([[[ 0.5,  2. ]],
             [[ 3. ,  4.5]],
             [[ 5.5,  7. ]],
             [[ 8. ,  9.5]],
             [[10.5, 12. ]],
             [[13. , 14.5]],
             [[15.5, 17. ]],
             [[18. , 19.5]])])
```

```
[131]: b[[0,-1]] # Primeira e última linha
```

```
[131]: array([[ 0. ,  0.5,  1. ,  1.5,  2. ],
              [17.5, 18. , 18.5, 19. , 19.5]])
```

Além de realizar seleção por indexação, podemos usar regras lógicas para realizar seleção de elementos.

```
[132]: b>15
```

```
[132]: array([[False, False, False, False, False],
              [False, False, False, False, False],
              [False, False, False, False, False],
              [False, False, False, False, False],
              [False, False, False, False, False],
              [False, False, False, False, False],
              [False, True,  True,  True,  True],
              [ True,  True,  True,  True,  True]])
```

```
[133]: b[b>15]  # Elementos de b tal que elemento maior que 15
```

```
[133]: array([15.5, 16. , 16.5, 17. , 17.5, 18. , 18.5, 19. , 19.5])
```

```
[134]: print(b, '\n')
print(b[(b>1) & (b<10)], '\n')
print(b[(b>1) & (b<10)].sum())  # Soma dos números pertencentes a (1,10)
```

```
[[ 0.   0.5  1.   1.5  2. ]
 [ 2.5  3.   3.5  4.   4.5]
 [ 5.   5.5  6.   6.5  7. ]
 [ 7.5  8.   8.5  9.   9.5]
 [10.  10.5 11.  11.5 12. ]
 [12.5 13.  13.5 14.  14.5]
 [15.  15.5 16.  16.5 17. ]
 [17.5 18.  18.5 19.  19.5]]
```

```
[1.5 2.   2.5 3.   3.5 4.   4.5 5.   5.5 6.   6.5 7.   7.5 8.   8.5 9.   9.5]
```

```
93.5
```

Assim como fizemos com `list`, podemos não apenas selecionar como modificar e filtrar os valores de elementos específicos do `ndarray`

```
[135]: b[:,2] = b[:,2]**2  # Terceira coluna ao quadrado
b
```

```
[135]: array([[ 0.   ,  0.5 ,  1.   ,  1.5 ,  2.   ],
              [ 2.5 ,  3.   , 12.25,  4.   ,  4.5 ],
              [ 5.   ,  5.5 , 36.   ,  6.5 ,  7.   ],
              [ 7.5 ,  8.   , 72.25,  9.   ,  9.5 ],
              [10.   , 10.5 , 121.   , 11.5 , 12.   ],
```

```
[ 12.5 , 13. , 182.25, 14. , 14.5 ],
[ 15. , 15.5 , 256. , 16.5 , 17. ],
[ 17.5 , 18. , 342.25, 19. , 19.5 ]])
```

```
[136]: b[b>50] = 0
      b
```

```
[136]: array([[ 0. ,  0.5 ,  1. ,  1.5 ,  2. ],
              [ 2.5 ,  3. , 12.25,  4. ,  4.5 ],
              [ 5. ,  5.5 , 36. ,  6.5 ,  7. ],
              [ 7.5 ,  8. ,  0. ,  9. ,  9.5 ],
              [10. , 10.5 ,  0. , 11.5 , 12. ],
              [12.5 , 13. ,  0. , 14. , 14.5 ],
              [15. , 15.5 ,  0. , 16.5 , 17. ],
              [17.5 , 18. ,  0. , 19. , 19.5 ]])
```

4.3 Broadcasting

Broadcasting é o comportamento de trabalharmos com um tensor como se estivéssemos trabalhando com escalares, evitando termos que utilizar *loopings* para modificar cada elemento de um tensor.

```
[137]: a
```

```
[137]: array([[10,  2,  1],
              [ 1,  5,  1],
              [ 2,  3, 10]])
```

```
[138]: 5*a + a**2
```

```
[138]: array([[150,  14,  6],
              [ 6,  50,  6],
              [ 14,  24, 150]])
```

```
[139]: a + 5
```

```
[139]: array([[15,  7,  6],
              [ 6, 10,  6],
              [ 7,  8, 15]])
```

Também é possível operação de um tensor no espaço \mathbb{R}^n por outro tensor no espaço \mathbb{R}^{n-i} . No exemplo a seguir realizamos a multiplicação de uma matriz em \mathbb{R}^2 por um vetor em \mathbb{R} .

```
[140]: a * array([1.2,2.3, 3.4]) # Array (3,3) multiplicado por Array (1,3)
```

```
[140]: array([[12. ,  4.6,  3.4],
              [ 1.2, 11.5,  3.4],
              [ 2.4,  6.9, 34. ]])
```



```
[141]: print(a * a, '\n') # Multiplicação elemento a elemento
print(a @ a, '\n') # Multiplicação de matriz a por matriz a
print(a.dot(a)) # Equivalente ao código da linha anterior
```

```
[[100  4  1]
 [ 1 25  1]
 [ 4  9 100]]
```

```
[[104 33 22]
 [ 17 30 16]
 [ 43 49 105]]
```

```
[[104 33 22]
 [ 17 30 16]
 [ 43 49 105]]
```

```
[142]: print(type(a))
print(type(np.sin)) # Universal function
print(np.sin(a), '\n')
print(np.rad2deg(np.sin(a)), '\n')
```

```
<class 'numpy.ndarray'>
<class 'numpy.ufunc'>
[[-0.54402111  0.90929743  0.84147098]
 [ 0.84147098 -0.95892427  0.84147098]
 [ 0.90929743  0.14112001 -0.54402111]]

[[-31.17011362  52.09890488  48.21273601]
 [ 48.21273601 -54.94231381  48.21273601]
 [ 52.09890488   8.08558087 -31.17011362]]
```

O NumPy conta com muitas [Funções Matemáticas](#) como `ufunc`. Para mais informações verificar a documentação do NumPy disponível neste texto.

4.4 Mais Rotinas

```
[143]: f = array([1,2,3,5,6,4,5,6,7,4,5,6,7,8,5,3,5,6,8,9,9,5,4,4])
```

```
[144]: unique(f)
```

```
[144]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
[145]: histogram(f, bins=6)
```

```
[145]: (array([ 2,  2,  4, 10,  2,  4], dtype=int64),
array([1.          , 2.33333333, 3.66666667, 5.          , 6.33333333,
7.66666667, 9.          ]))
```

```
[146]: roots([ 1, -9, 26, -24]) # Raízes de x**3 - 9*x*2 + 26*x - 24
```

```
[146]: array([4., 3., 2.])
```

```
[147]: # Regressão Linear
```

```
x = [ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
```

```
y = [ 10, 12, 22, 24, 32, 39, 41, 55, 56, 60, 70, 72, 81, 87, 90]
```

```
polyfit(x,y, 1) # Regressão em polinômio de ordem 1 (regressão linear)
```

```
[147]: array([2.97857143, 5.38809524])
```

Abaixo temos o exemplo de uma regressão polinomial de ordem 2.

```
[148]: polyfit(x,y, 2) # Regressão em polinômio de ordem 2
```

```
[148]: array([-1.97963801e-03, 3.03796057e+00, 5.09048966e+00])
```

4.5 Resolução de sistemas lineares com `numpy.linalg`

$$\begin{cases} 10x_1 + 0,5x_2 + 0,6x_3 + 3x_4 + 2x_5 + 3x_6 = 48,05 \\ 3x_1 + 1x_2 + 13x_3 + 5x_4 + 2x_5 + x_6 = 55 \\ x_1 + 10x_2 + 0,8x_3 + 2x_4 + 3x_5 + x_6 = 101 \\ 4x_1 + 2x_2 + x_3 + 15x_4 + 3x_5 + 4x_6 = 105 \\ x_1 + 0,5x_2 + 0,6x_3 + 0,3x_4 + 9x_5 + 5x_6 = 54,7 \\ 3x_1 + 2x_2 + 3x_3 + x_4 + 4x_5 + 15x_6 = 126 \end{cases}$$

$$A \times x = B$$

$$\begin{bmatrix} 10 & 0,5 & 0,6 & 3 & 2 & 3 \\ 1 & 10 & 0,8 & 2 & 3 & 1 \\ 3 & 1 & 13 & 5 & 2 & 1 \\ 4 & 2 & 1 & 15 & 3 & 4 \\ 1 & 0,5 & 0,6 & 0,3 & 9 & 5 \\ 3 & 2 & 3 & 1 & 4 & 15 \end{bmatrix} \times x = \begin{bmatrix} 48,5 \\ 55 \\ 101 \\ 105 \\ 54,7 \\ 126 \end{bmatrix}$$

```
[149]: A = array([[10, 0.5, 0.6, 3, 2, 3],
                 [1, 10, 0.8, 2, 3, 1],
                 [3, 1, 13, 5, 2, 1],
                 [4, 2, 1, 15, 3, 4],
                 [1, 0.5, 0.6, 0.3, 9, 5],
                 [3, 2, 3, 1, 4, 15]])
```

```
B = array([ 48.5, 55 , 101 , 105 , 54.7, 126 ])
```

```
solve(A, B)
```

```
[149]: array([1., 3., 5., 4., 2., 6.])
```

Salientamos que usamos diretamente da função `solve` por termos importado a função em `from pylab import *`. Se for utilizado a importação por `import numpy as np` o correto seria utilizada `np.linalg.solve(A, B)`.⁴

$$x = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 4 \\ 2 \\ 6 \end{bmatrix}$$

```
[150]: # Utilizando @ o Python entendendo que é multiplicação de Matrizes com os ndarray
inv(A) @ B
```

```
[150]: array([1., 3., 5., 4., 2., 6.])
```

`ndarray` não tem métodos de matrizes. Devem ser utilizadas funções do NumPy para realizar operações de matrizes com este tipo de array.

```
[151]: det(A) # Determinante da Matriz
```

```
[151]: 1850722.16000000008
```

```
[152]: inv(A) # Inversa da matriz
```

```
[152]: array([[ 0.11370042,  0.0018882 , -0.00015139, -0.02183456, -0.01292944,
              -0.01272351],
             [-0.00331326,  0.10298361, -0.00565853, -0.011062 , -0.03212843,
               0.00783366],
             [-0.01525415, -0.00346535,  0.0783854 , -0.02291181, -0.00832108,
               0.00693967],
             [-0.02403062, -0.01087867, -0.00098783,  0.07417396, -0.01084161,
              -0.01056863],
             [-0.00097238,  0.00201067,  0.00396237, -0.00111338,  0.12925506,
              -0.04299184],
             [-0.01738614, -0.01322665, -0.01588311,  0.00577618, -0.02521135,
               0.07894801]])
```

```
[153]: pinv(A) # Moore-Penrose pseudo-inversa da matriz
```

```
[153]: array([[ 0.11370042,  0.0018882 , -0.00015139, -0.02183456, -0.01292944,
              -0.01272351],
             [-0.00331326,  0.10298361, -0.00565853, -0.011062 , -0.03212843,
               0.00783366],
             [-0.01525415, -0.00346535,  0.0783854 , -0.02291181, -0.00832108,
               0.00693967],
```

```
[-0.02403062, -0.01087867, -0.00098783,  0.07417396, -0.01084161,
 -0.01056863],
[-0.00097238,  0.00201067,  0.00396237, -0.00111338,  0.12925506,
 -0.04299184],
[-0.01738614, -0.01322665, -0.01588311,  0.00577618, -0.02521135,
 0.07894801]])
```

4.6 Matplotlib

Uma das bibliotecas mais utilizadas no Python para criar visualizações de dados de forma estática, animada e interativa. Não temos intenção de esgotar todas as possibilidades do `Matplotlib`, mas passar pelo básico para plotar gráfico. Para aprofundamento a documentação oficial da ferramenta deve ser consultada.

```
[154]: from pylab import *

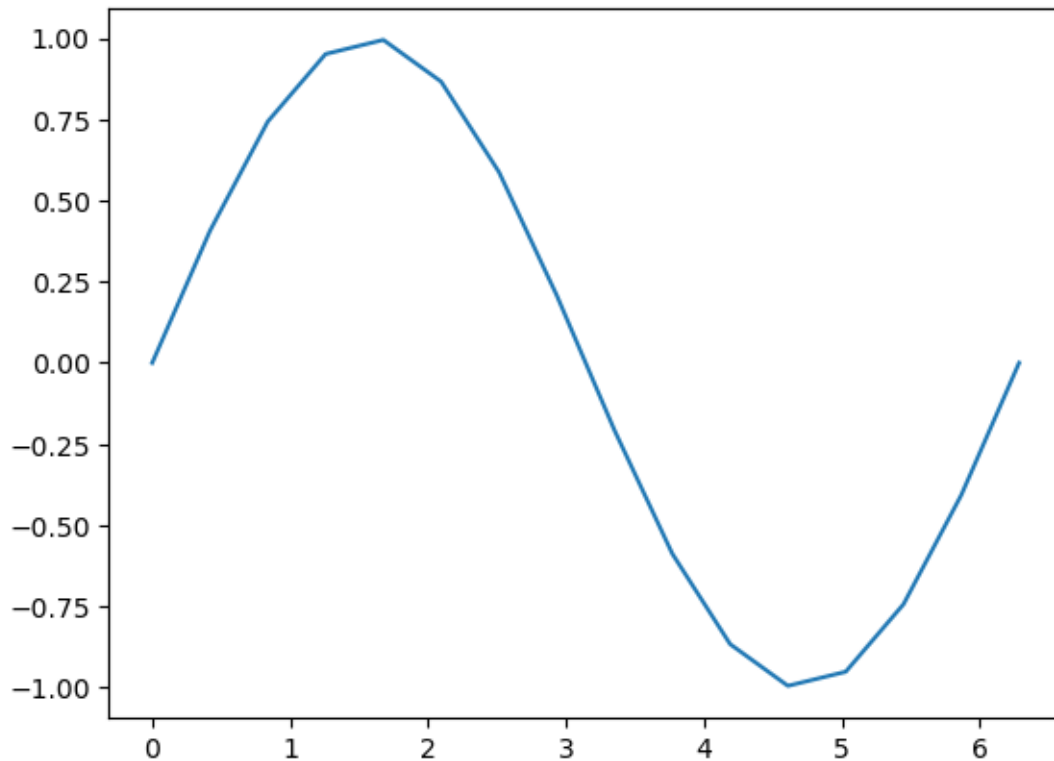
matplotlib.__version__
```

```
[154]: '3.8.4'
```

Abaixo realizamos a plotagem simples de um gráfico sendo `x` um `ndarray` com 16 pontos entre zero e 2π e `y` um `ndarray` com valores do seno de `x`.

```
[155]: x = linspace(0, 2*pi, 16) # Array no intervalo [0,2pi) dividido em 16 amostras
y = sin(x) # calcula sen de todo o ndarray x

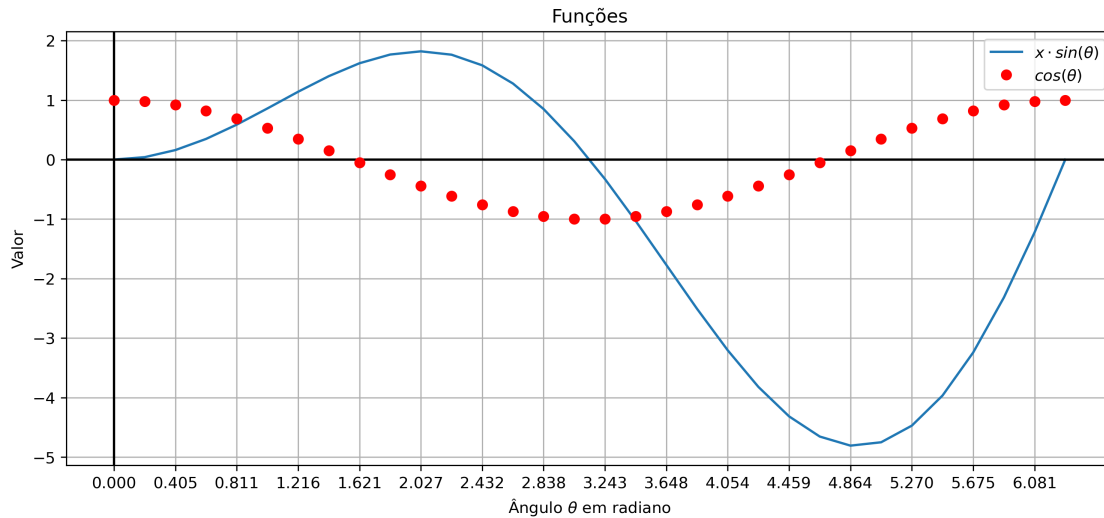
plot(x,y) # Plotar curva do sen
show() # Mostrar gráfico
```



Vamos colocar mais algumas opções no gráfico.

```
[156]: θ = linspace(0, 2*pi, 32) # Array no intervalo [0,2π) dividido em 32 amostras
ys = θ * sin(θ) # calcula seno de todo o darray, multiplicado pelo valor de θ
yc = cos(θ) # calcula cosseno de todo o darray θ

figure(figsize=(12,5), dpi=300) # Redimensionar figura e aumentar resolução
plot(θ,ys, label='$x \cdot \sin(\theta)$') # Plotar curva do sen com legenda
    ↳ em LaTeX
axhline(0, color='black') # Plotar eixo x na cor preta
axvline(0, color='black') # Plotar eixo y na cor preta
plot(θ,yc, 'ro', label='$\cos(\theta)$') # Plotar curva do cos com legenda
    ↳ "cos" utilizando marcador de ponto na cor vermelha
title('Funções') # Definir título do gráfico
xticks(θ[::2]) # Colocar de 2 em 2 valores de θ no eixo x
xlabel('Ângulo $θ$ em radiano') # Definir legenda do eixo x
ylabel('Valor') # Definir legenda do eixo y
legend() # Ativar legenda
grid() # Ativar grid
show() # Mostrar gráfico
```



No exemplo acima, se quiser salvar a figura no lugar de mostrar gráfico em tela, deve-se substituir a linha `show()` por `savefig('Gráfico ds funções.png')`, que salvaria o gráfico em formato PNG com nome “Gráfico ds funções.png” no mesmo diretório onde está o script está rodando.

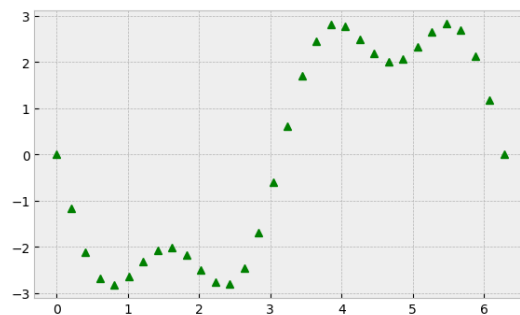
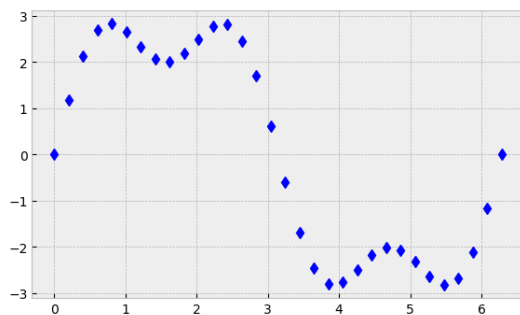
Podem ser definidos [estilos](#) distintos para mostrar o gráfico no matplotlib.

```
[157]: plt.style.use('bmh')

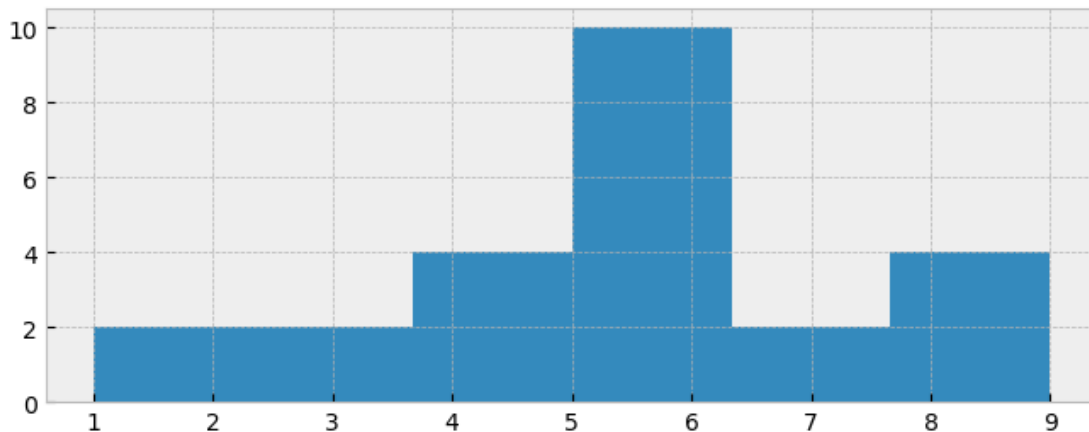
x = linspace(0, 2*pi, 32)  # Array no intervalo [0,2pi) dividido em 32 amostras

def f(x): # definindo função a ser plotada
    return 3*np.sin(x) + np.sin(3*x)

# Criando dois gráficos em uma linha e duas colunas
fig, axs = subplots(1, 2, figsize=(15,4))
# Plotar função cor azul em formato de diamante
axs[0].plot(x,f(x), 'bd')
# Plotar função negativa cor azul em formato de triângulo
axs[1].plot(x,-f(x), 'g^')
show()
```



```
[158]: f = array([1,2,3,5,6,4,5,6,7,4,5,6,7,8,5,3,5,6,8,9,9,5,4,4])
figure(figsize=(8,3))
hist(f, bins=6)
show()
```



4.7 Distribuições probabilísticas em `numpy.random`

```
[159]: randint(100, size=(3, 5)) # Array com inteiros de 0 a 100 com dimensão 3x5b
```

```
[159]: array([[47, 82, 94, 98, 74],
             [50,  8, 79,  7, 48],
             [21, 36, 39, 61, 72]])
```

Lembrando que a utilização direta da função `randint` é possível por termos usado a importação por `from pylab import *`. Se for utilizado a importação por `import numpy as np` o correto seria utilizada `np.random.randint(100, size=(3, 5))`.

```
[160]: rand(10) # Array com 10 números aleatórios entre 0 e 1
```

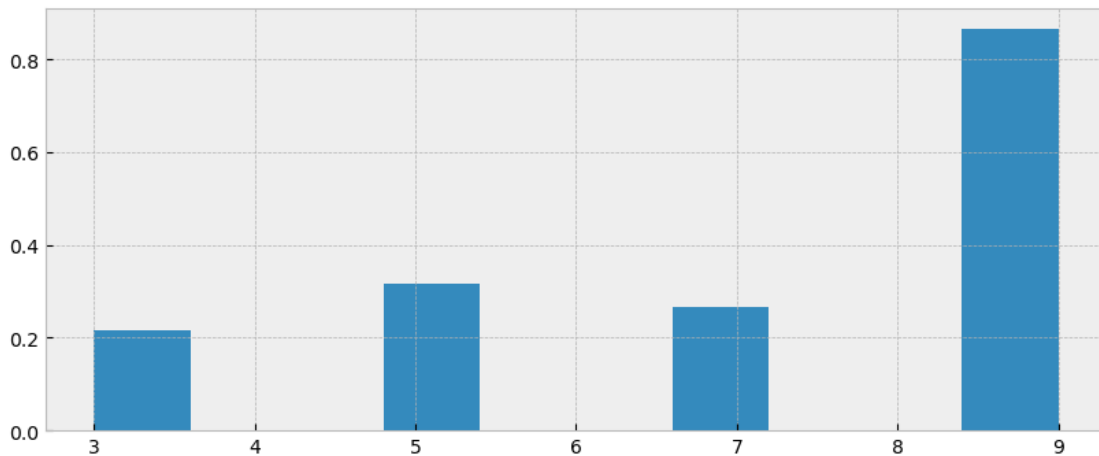
```
[160]: array([0.59332502, 0.18714174, 0.05679409, 0.53293795, 0.34700725,
             0.61232837, 0.18431783, 0.62776264, 0.08057247, 0.13561054])
```

```
[161]: # Escolha com probabilidade definida por elemento

# Dados de 100 números seguindo probailidade definida
# Números 3, 5 e 7 com probabilidade 0.15 e número 9 com prob. de 0.55
dados = np.random.choice([3, 5, 7, 9], p=(.15,.15,.15,.55), size=100)

plt.figure(figsize=(10,4)) # Definir tamanho do gráfico
```

```
plt.hist(dados, density=True) # Plotar histograma
plt.show()
```



O maior objetivo do exemplo abaixo é mostrar possibilidade de geração de distribuições probabilísticas, além de mais um exemplo de como podemos plotar gráficos usando `matplotlib`.

```
[162]: bins = 50
media = 10
n = 1000 # Número de amostra

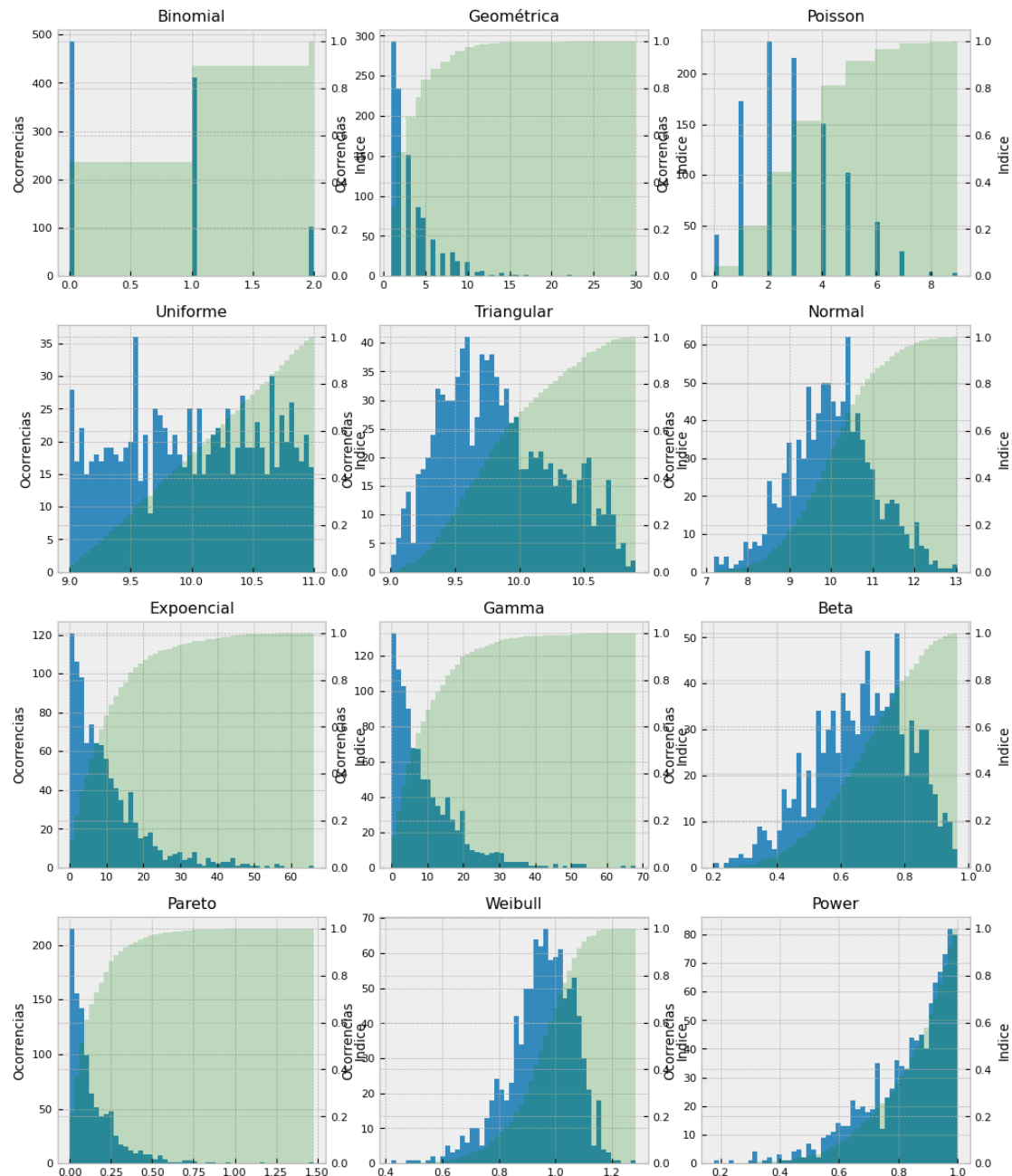
dc = dict()
dc['Binomial'] = binomial(2, 0.3, n) # (tamanho, probabilidade, tamanho)
dc['Geométrica'] = geometric(0.3, n) # (propabilidade, tamanho)
dc['Poisson'] = poisson(3, n) # (lambda, tamanho)
dc['Uniforme'] = uniform(9,11,n) # (mínimo, máximo, tamanho)
dc['Triangular'] = triangular(9,9.5,11,n) # (mínimo, moda, máximo, tamanho)
dc['Normal'] = normal(media,1,n) # (média, desvio padrão, tamanho)
dc['Exponencial'] = exponential(media,n) # (média, tamanho)
dc['Gamma'] = gamma(1,media,n) # (alpha, beta, tamanho)
dc['Beta'] = beta(6,3,n) # (alpha, beta, tamanho)
dc['Pareto'] = pareto(8,n) # (forma, tamanho)
dc['Weibull'] = weibull(10,n) # (alpha, tamanho)
dc['Power'] = power(5,n) # (formato, tamanho)

fig = figure(figsize=(12,15))
rcParams['font.size'] = 8 # Definindo tamanho da fonte
for i, label in enumerate(dc.keys()):
    ax1 = fig.add_subplot(4, 3, i+1) # 4 linha, 3 colunas, sequencial
    ax1.hist(dc[label], bins, label=label)
    ax1.set_ylabel('Ocorrencias')
    ax2 = ax1.twinx() # Criar segundo eixo de ordenada
```



```
ax2.hist(dc[label], bins, density=True, cumulative=True,
        alpha=0.2, label='Acumulado', color='g') # Histograma Cumulativo
ax2.set_ylabel('Indice')
title(label)
```

```
show()
```



5 Programação Orientada a Objeto

Não temos intenção de nos aprofundarmos em teoria de programação utilizando paradigma de Orientação a Objeto, mas sim explorar como podemos utilizar esse paradigma em Python para escrever códigos. Podemos usar paradigma funcional, procedural ou orientação a objeto em Python, mas todas as variáveis criadas são sempre um objeto com instância de alguma classe pré-definida com seus atributos e métodos. Quando criamos uma variável com texto esta variável é uma instância da classe `str` com todos seus métodos e atributos.

```
[163]: texto = 'Python'
       print(type(texto))
```

```
<class 'str'>
```

A ferramenta utilizada em Python para implantar objetos é `class`. Abaixo vamos criar uma classe `Esfera` com atributos declarados `r` (raio) e `cor` (que terá a cor preta como pré-definida) e com métodos `area` (cálculo de área) e `volume` (cálculo de volume), sendo:

$$area = 4\pi r^2$$

$$volume = \frac{4}{3}\pi r^3$$

Para que os atributos calculados ou declarados estejam disponíveis para serem utilizados em qualquer parte da instância ou da classe o nome da variável sempre deve ser precedida de `self`. e em todos os métodos da classe o `self` sempre deve ser o primeiro atributo a ser declarado.

```
[164]: from pylab import *

class Esfera:
    def __init__(self, r, cor='preta'): # Método Dunder Construtor
        self.r = r # Atributo declarado
        self.cor = cor # Atributo declarado

    def area(self): # Método
        return 4 * pi * self.r**2

    def volume(self): # Método
        return 4/3 * pi * self.r**3
```

Sempre que uma instância é criada o método `__init__` é executado. Este é o método em Python utilizado como **construtor** da classe.

```
[165]: esfera1 = Esfera(4)

       print(f'Área: {esfera1.area()}')
       print(f'Volume: {esfera1.volume()}')
       print(f'Cor: {esfera1.cor}')
```

Área: 201.06192982974676
Volume: 268.082573106329
Cor: preta

```
[166]: esfera2 = Esfera(2, 'branca')

print(f'Área: {esfera2.area()}')
print(f'Volume: {esfera2.volume()}')
print(f'Cor: {esfera2.cor}')
```

Área: 50.26548245743669
Volume: 33.510321638291124
Cor: branca

Utilizando a função `dir` podemos listar todos os métodos e atributos de um objeto.

```
[167]: dir(esfera2)
```

```
[167]: ['__class__',
        '__delattr__',
        '__dict__',
        '__dir__',
        '__doc__',
        '__eq__',
        '__format__',
        '__ge__',
        '__getattribute__',
        '__getstate__',
        '__gt__',
        '__hash__',
        '__init__',
        '__init_subclass__',
        '__le__',
        '__lt__',
        '__module__',
        '__ne__',
        '__new__',
        '__reduce__',
        '__reduce_ex__',
        '__repr__',
        '__setattr__',
        '__sizeof__',
        '__str__',
        '__subclasshook__',
        '__weakref__',
        'area',
        'cor',
        'r',
```

```
'volume']
```

Note que para utilizar um método sempre se abre e fecha os parênteses após o nome do método, mesmo que não haja parâmetros a serem informador. No caso de utilização de um valor de atributo, como no caso de `esfera1.cor` os parênteses não são utilizados.

Em Python existem Métodos Especial que iniciam e terminam com “__”, são os métodos Dunder, ou *Double Underscore Before and After*. Este tipo de método normalmente é utilizado com auxílio de operadores, como exemplificaremos mais à frente.

Temos mais um exemplo com classe `Impedancia` contendo atributos declarados R, L, C e f e com métodos `xc` (cálculo de impedância capacitiva), `xl` (impedância indutiva), `Z` (impedância) e `conteudo` (lista atributos e métodos, menos os especiais).

```
[168]: class Impedancia:
        def __init__(self, R, L, C, f): # Método Dunder Construtor
            self.R = R # Atributo declarado
            self.__L = L # Atributo declarado Encapsulado
            self.__C = C # Atributo declarado Encapsulado
            self.f = f # Atributo declarado
            self.ω = 2*pi*self.f # Atributo calculado

        def xc(self): # Método
            return (self.ω*self.__C)**-1

        def xl(self): # Método
            return self.ω*self.__L

        def Z(self): # Método
            X = (self.xl() - self.xc())
            return complex(self.R, X)

        def conteudo(self):
            return [i for i in dir(self) if not i.startswith('_')]

[169]: Z = Impedancia(R=10, L=0.1, C=1e-3, f=60) # Z: instância, Impedância: classe

print('Resistência: ', Z.R) # Atributo Resistência
print('Reatância Capacitiva: ', Z.xc()) # Método cálculo da reatância capacitiva
print('Reatância Indutiva: ', Z.xl()) # Método cálculo da reatância indutiva
print('Impedância: ', Z.Z()) # Método cálculo da impedância

Resistência: 10
Reatância Capacitiva: 2.6525823848649224
Reatância Indutiva: 37.69911184307752
Impedância: (10+35.046529458212596j)

[170]: Z1 = Impedancia(R=5, L=0.2, C=5e-3, f=50)
print('Resistência: ', Z1.R) # Atributo Resistência
```

```
print('Reatância Capacitiva: ', Z1.xc()) # Método cálculo da reatância capacitiva
print('Reatância Indutiva: ', Z1.xl()) # Método cálculo da reatância indutiva
print('Impedância: ', Z1.Z())
```

```
Resistência: 5
Reatância Capacitiva: 0.6366197723675814
Reatância Indutiva: 62.83185307179587
Impedância: (5+62.19523329942829j)
```

```
[171]: Z1.conteudo()
```

```
[171]: ['R', 'Z', 'conteudo', 'f', 'xc', 'xl', 'ω']
```

The Python Language Reference lista mais de 80 nomes de métodos especiais. Exemplo de alguns Dunders:

Método	Função
<code>__init__</code>	Executado no momento da criação a instância (método construtor)
<code>__contains__</code>	<code>in</code>
<code>__eq__</code>	<code>==</code>
<code>__getitem__</code>	<code>minha_instância[x]</code>
<code>__call__</code>	<code>minha_instância(x)</code>
<code>__add__</code>	<code>+</code>

```
[172]: class Teste:
        def __init__(self, a):
            self.a = a

        def __eq__(self,b):
            print('ôxe, que pergunta da gota!!!')
            return b == self.a**2

A = Teste(3)
print(A == 9)
print(A == 3)
print()
print(A.a == 9)
print(A.a == 3)
```

```
ôxe, que pergunta da gota!!!
True
ôxe, que pergunta da gota!!!
False

False
True
```

```
[173]: # Lista com indice começando de 1 e não 0
class Lista(list):
    def __init__(self, lista):
        self.lista = lista

    def __getitem__(self, posicao):
        if type(posicao)==int:
            return list(self.lista)[posicao-1]

        if type(posicao)==slice:
            return list(self.lista)[posicao.start-1:posicao.stop-1:posicao.step]

    def __call__(self, n):
        return self.lista[0: n]
```

```
[174]: B0 = [1,2,3,4,5]
B1 = Lista([1,2,3,4,5])

print(B0[4], B1[4])
print(B0[1:4:2], B1[1:4:2])
```

```
5 4
[2, 4] [1, 3]
```

```
[175]: B1(3)
```

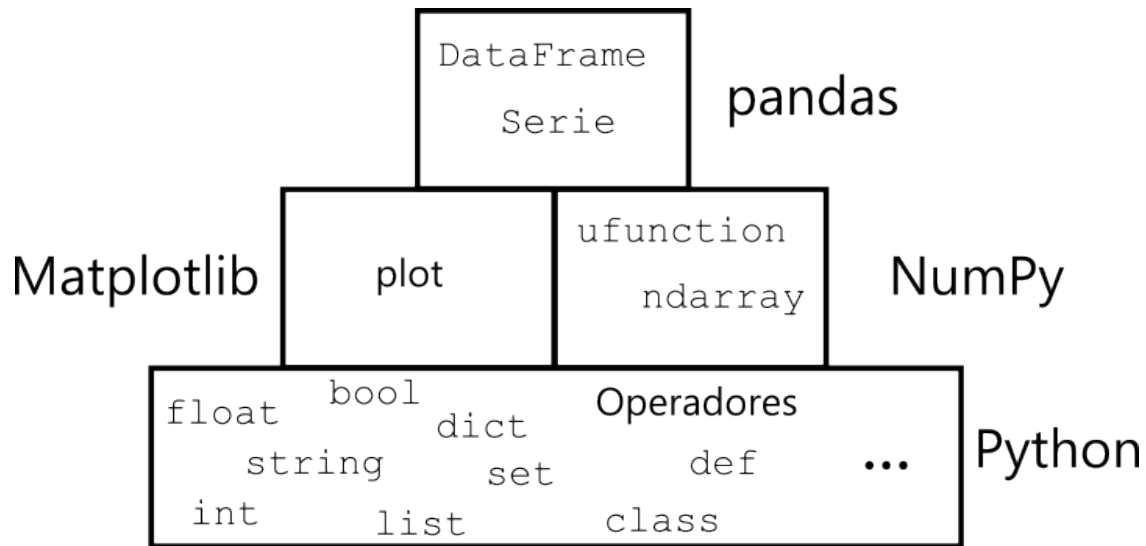
```
[175]: [1, 2, 3]
```

6 pandas

[pandas](#) é um pacote Python *open source* rápido, poderoso, flexível e fácil utilizado para análise e tratamento de dados. Assim como o `ndarray` é a base do NumPy, o `DataFrame` e `Serie` são os objetos base do pandas. Os objetos do pandas são construídos com base no Python e NumPy e tem uma forte integração com o Matplotlib para visualização de dados.

Existem pacotes especializados em geração de modelos estatísticos, como o [statsmodels](#), e outros em modelos de aprendizado de máquina, como o [scikit-learn](#), mas este assunto extrapola o objetivo de apresentação de conteúdo deste material.

Vamos fazer um entendimento geral sobre o pacote pandas, depois daremos exemplo de aquisição de dados e ferramentas para tratar e explorar informações de dados.



O pandas roda em ambiente Python, utiliza como base o pacote NumPy e tem grande integração como pacote Matplotlib para plotar os dados de um `DataFrame` ou `Serie`.

```
[176]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Limitar números de registros a ser exibido em tela
pd.options.display.max_rows = 15

pd.__version__
```

```
[176]: '2.2.2'
```

Vamos iniciar criando um `DataFrame` para entender a estrutura e algumas características básicas.

```
[177]: datas = pd.date_range('2023-01-01', periods=30)

# DataFrame com dados randômicos com 30 linhas e 4 colunas, indexador de datas e
↳ colunas de A a D
df = pd.DataFrame(np.random.randn(30, 4), index=datas, columns=['A', 'B', 'C',
↳ 'D'])

df
```

```
[177]:
```

	A	B	C	D
2023-01-01	0.407506	0.681054	-0.634441	-0.280475
2023-01-02	0.327325	0.044171	0.032811	-1.131876
2023-01-03	-1.318001	-0.808121	-0.147528	1.241311
2023-01-04	-0.091915	0.426950	0.617419	-1.600980
2023-01-05	0.639779	1.614970	-0.400563	-1.236986
...

```

2023-01-26 -0.382449 -1.383178 -0.642463 -1.596178
2023-01-27  0.712104  0.352169  0.213204  0.102629
2023-01-28  0.022960  0.120408 -0.528289  1.031777
2023-01-29  1.405593  0.038362 -0.368917 -0.002201
2023-01-30  0.936953  1.094406  1.093367  0.146714

```

[30 rows x 4 columns]

O tipo da variável `df` é `DataFrame` do pandas. Cada coluna representa uma `Series` do pandas e os valores guardados na coluna são do tipo `ndarray` no NumPy.

```
[178]: type(df)
```

```
[178]: pandas.core.frame.DataFrame
```

```
[179]: type(df.A)
```

```
[179]: pandas.core.series.Series
```

```
[180]: type(df.A.values)
```

```
[180]: numpy.ndarray
```

Existem métodos do `DataFrame` que ajudam a visualizar e entender a tabela estruturada e seus dados. Usando o método `info` são mostradas as colunas com seus respectivos nomes, valores não nulos e o tipo de dado de cada coluna. Além disso é mostrado Tamanho da tabela com intervalo do índice uso da memória pelo `DataFrame`.

```
[181]: df.info() # Informações básicas de cada coluna
```

```

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 30 entries, 2023-01-01 to 2023-01-30
Freq: D
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  -
 0    A      30 non-null      float64
 1    B      30 non-null      float64
 2    C      30 non-null      float64
 3    D      30 non-null      float64
dtypes: float64(4)
memory usage: 1.2 KB

```

Ao obter as informações básicas pelo método `info` pode-se coletar um resumo dos dados por meio de medidas estatísticas de cada coluna utilizando o método `describe`, onde é mostrado por padrão o número de valores diferentes de NaN (*Not a Number*, ou valor não informado), média, desvio padrão, valor mínimo, valor máximo e os quartis de 25%, 50% (mediana) e 75%.

```
[182]: df.describe()
```



```
[182]:
```

	A	B	C	D
count	30.000000	30.000000	30.000000	30.000000
mean	0.132575	-0.133130	-0.134527	-0.316186
std	0.940084	1.075668	0.738799	1.110633
min	-2.018219	-2.331045	-1.777541	-2.629893
25%	-0.404568	-0.759707	-0.654382	-1.210708
50%	0.283096	0.041266	-0.221828	-0.374081
75%	0.716090	0.617528	0.382426	0.400982
max	1.635504	1.614970	1.729034	1.811557

Abaixo estão demonstradas outros métodos que podem ser utilizados para coletar mais detalhes sobre os dados.

```
[183]: df.head() # Primeiras 5 linhas
```

```
[183]:
```

	A	B	C	D
2023-01-01	0.407506	0.681054	-0.634441	-0.280475
2023-01-02	0.327325	0.044171	0.032811	-1.131876
2023-01-03	-1.318001	-0.808121	-0.147528	1.241311
2023-01-04	-0.091915	0.426950	0.617419	-1.600980
2023-01-05	0.639779	1.614970	-0.400563	-1.236986

```
[184]: df.tail() # Últimas 5 linhas
```

```
[184]:
```

	A	B	C	D
2023-01-26	-0.382449	-1.383178	-0.642463	-1.596178
2023-01-27	0.712104	0.352169	0.213204	0.102629
2023-01-28	0.022960	0.120408	-0.528289	1.031777
2023-01-29	1.405593	0.038362	-0.368917	-0.002201
2023-01-30	0.936953	1.094406	1.093367	0.146714

```
[185]: df.sample(5) # Amostra aleatória de 5 linhas
```

```
[185]:
```

	A	B	C	D
2023-01-19	-0.987781	-2.010321	-0.876046	-1.305953
2023-01-27	0.712104	0.352169	0.213204	0.102629
2023-01-25	-1.370572	1.143070	1.729034	-1.089617
2023-01-24	-0.399022	-2.331045	-0.501557	0.729581
2023-01-06	0.733230	-0.105899	0.098682	-0.571183

```
[186]: df.columns # Nome das colunas
```

```
[186]: Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
[187]: df.index # Lista de indexadores do df
```

```
[187]: DatetimeIndex(['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04',
                    '2023-01-05', '2023-01-06', '2023-01-07', '2023-01-08',
```

```

'2023-01-09', '2023-01-10', '2023-01-11', '2023-01-12',
'2023-01-13', '2023-01-14', '2023-01-15', '2023-01-16',
'2023-01-17', '2023-01-18', '2023-01-19', '2023-01-20',
'2023-01-21', '2023-01-22', '2023-01-23', '2023-01-24',
'2023-01-25', '2023-01-26', '2023-01-27', '2023-01-28',
'2023-01-29', '2023-01-30'],
dtype='datetime64[ns]', freq='D')

```

```
[188]: df.values # Valores do df em um ndarray
```

```

[188]: array([[ 4.07505709e-01,  6.81054200e-01, -6.34440569e-01,
                -2.80474926e-01],
               [ 3.27325444e-01,  4.41712023e-02,  3.28114316e-02,
                -1.13187596e+00],
               [-1.31800140e+00, -8.08120674e-01, -1.47527635e-01,
                1.24131093e+00],
               [-9.19153449e-02,  4.26950031e-01,  6.17419378e-01,
                -1.60097973e+00],
               [ 6.39779442e-01,  1.61497031e+00, -4.00563045e-01,
                -1.23698571e+00],
               [ 7.33230274e-01, -1.05899474e-01,  9.86815554e-02,
                -5.71183135e-01],
               [-4.06416284e-01,  1.56419609e+00,  4.53818135e-01,
                3.57624462e-01],
               [ 2.33125160e-01,  1.75510524e-01, -8.37648279e-01,
                -2.62989331e+00],
               [ 6.73953609e-01, -8.31349954e-01,  6.21638355e-01,
                4.15434296e-01],
               [ 1.46993002e+00,  7.84019082e-01, -2.96128091e-01,
                6.44279909e-01],
               [ 2.45335164e-01, -1.92376564e+00, -6.58355618e-01,
                1.81155691e+00],
               [ 1.39178121e-02, -1.88922774e+00, -1.77754131e+00,
                -1.01158413e+00],
               [ 7.17418170e-01,  1.49747769e-01, -1.06296807e-01,
                2.93856940e-01],
               [ 5.87807594e-01, -4.20340611e-01, -9.60568556e-01,
                -1.71810475e+00],
               [-1.23509166e+00, -5.41641597e-01,  4.01750349e-01,
                3.17463624e-01],
               [ 1.63550359e+00, -4.40968112e-01, -7.91030172e-01,
                -4.67686390e-01],
               [ 1.21449527e+00,  1.07341731e+00,  4.82660903e-01,
                -7.08833463e-01],
               [-7.81031058e-01,  3.52608432e-01, -8.08971050e-01,
                -7.53633251e-01],
               [-9.87781116e-01, -2.01032121e+00, -8.76045664e-01,

```

```

-1.30595281e+00],
[ 1.39061637e+00, -3.90318810e-01, -3.87998672e-02,
-1.46736316e+00],
[-2.01821906e+00, 1.17733676e+00, 1.02714385e+00,
-1.47845300e+00],
[-7.20669619e-01, -6.14467164e-01, -7.56637515e-01,
6.80329024e-01],
[ 3.20856546e-01, -1.09564155e+00, 3.24454761e-01,
1.79285027e+00],
[-3.99022081e-01, -2.33104505e+00, -5.01557349e-01,
7.29580922e-01],
[-1.37057208e+00, 1.14307050e+00, 1.72903364e+00,
-1.08961684e+00],
[-3.82449488e-01, -1.38317788e+00, -6.42463128e-01,
-1.59617791e+00],
[ 7.12103520e-01, 3.52169372e-01, 2.13204007e-01,
1.02628663e-01],
[ 2.29595529e-02, 1.20407760e-01, -5.28288755e-01,
1.03177713e+00],
[ 1.40559281e+00, 3.83616327e-02, -3.68917189e-01,
-2.20125777e-03],
[ 9.36953399e-01, 1.09440603e+00, 1.09336680e+00,
1.46713776e-01]]))

```

Vários métodos estatístico podem ser utilizados, tanto para as colunas (deixando o parênteses vazios ou preenchendo com 0) como para cada linha (preenchendo os parênteses com o calor 1).

```
[189]: df.mean() # Média das linha mostrada por coluna
```

```

[189]: A    0.132575
      B   -0.133130
      C   -0.134527
      D   -0.316186
      dtype: float64

```

```
[190]: df.mean(1) # Média das colunas, mesmo que df.mean('columns')
```

```

[190]: 2023-01-01    0.043411
      2023-01-02   -0.181892
      2023-01-03   -0.258085
      2023-01-04   -0.162131
      2023-01-05    0.154300
      ...
      2023-01-26   -1.001067
      2023-01-27    0.345026
      2023-01-28    0.161714
      2023-01-29    0.268209
      2023-01-30    0.817860

```

Freq: D, Length: 30, dtype: float64

```
[191]: df.mean().mean() # Média das médias de cada coluna
```

```
[191]: -0.11281698758745579
```

Uma coluna pode ser criada de forma simples, com base em informação de outras colunas.

```
[192]: df = df.assign(E=df.mean(1)) # Criação de nova coluna E com valores médios
```

```
[193]: # Outra forma de criar coluna em um DataFrame
df.insert(5, 'F', np.sign(df['E']))
```

```
[194]: df.head(10) # Listar as primeiras 10 linhas
```

```
[194]:
```

	A	B	C	D	E	F
2023-01-01	0.407506	0.681054	-0.634441	-0.280475	0.043411	1.0
2023-01-02	0.327325	0.044171	0.032811	-1.131876	-0.181892	-1.0
2023-01-03	-1.318001	-0.808121	-0.147528	1.241311	-0.258085	-1.0
2023-01-04	-0.091915	0.426950	0.617419	-1.600980	-0.162131	-1.0
2023-01-05	0.639779	1.614970	-0.400563	-1.236986	0.154300	1.0
2023-01-06	0.733230	-0.105899	0.098682	-0.571183	0.038707	1.0
2023-01-07	-0.406416	1.564196	0.453818	0.357624	0.492306	1.0
2023-01-08	0.233125	0.175511	-0.837648	-2.629893	-0.764726	-1.0
2023-01-09	0.673954	-0.831350	0.621638	0.415434	0.219919	1.0
2023-01-10	1.469930	0.784019	-0.296128	0.644280	0.650525	1.0

```
[195]: df['G'] = df['A']*df['B'] + df['C'] # calcula nova coluna G = A*B + C
```

```
[196]: df.sample(6)
```

```
[196]:
```

	A	B	C	D	E	F	G
2023-01-23	0.320857	-1.095642	0.324455	1.792850	0.335630	1.0	-0.027089
2023-01-25	-1.370572	1.143070	1.729034	-1.089617	0.102979	1.0	0.162373
2023-01-15	-1.235092	-0.541642	0.401750	0.317464	-0.264380	-1.0	1.070727
2023-01-08	0.233125	0.175511	-0.837648	-2.629893	-0.764726	-1.0	-0.796732
2023-01-29	1.405593	0.038362	-0.368917	-0.002201	0.268209	1.0	-0.314996
2023-01-01	0.407506	0.681054	-0.634441	-0.280475	0.043411	1.0	-0.356907

```
[ ]: # Trocar valor -1 por Neg e 1 por Pos
df['F'].replace({-1: 'Neg', 1: 'Pos'}, inplace=True)
```

```
[198]: df.head()
```

```
[198]:
```

	A	B	C	D	E	F	G
2023-01-01	0.407506	0.681054	-0.634441	-0.280475	0.043411	Pos	-0.356907
2023-01-02	0.327325	0.044171	0.032811	-1.131876	-0.181892	Neg	0.047270
2023-01-03	-1.318001	-0.808121	-0.147528	1.241311	-0.258085	Neg	0.917577

2023-01-04	-0.091915	0.426950	0.617419	-1.600980	-0.162131	Neg	0.578176
2023-01-05	0.639779	1.614970	-0.400563	-1.236986	0.154300	Pos	0.632662

```
[199]: # Seleção por index e nome de colunas
# df.loc[<intervalo de index>, <nome de colunas>]
df.loc['2023-01-05':'2023-01-15':, ['B', 'E']]
```

```
[199]:
```

	B	E
2023-01-05	1.614970	0.154300
2023-01-06	-0.105899	0.038707
2023-01-07	1.564196	0.492306
2023-01-08	0.175511	-0.764726
2023-01-09	-0.831350	0.219919
2023-01-10	0.784019	0.650525
2023-01-11	-1.923766	-0.131307
2023-01-12	-1.889228	-1.166109
2023-01-13	0.149748	0.263682
2023-01-14	-0.420341	-0.627802
2023-01-15	-0.541642	-0.264380

```
[200]: # Seleção por índice
# df.iloc[<intervalo de linhas>, <colunas>]
df.iloc[4:15, [1,3]]
```

```
[200]:
```

	B	D
2023-01-05	1.614970	-1.236986
2023-01-06	-0.105899	-0.571183
2023-01-07	1.564196	0.357624
2023-01-08	0.175511	-2.629893
2023-01-09	-0.831350	0.415434
2023-01-10	0.784019	0.644280
2023-01-11	-1.923766	1.811557
2023-01-12	-1.889228	-1.011584
2023-01-13	0.149748	0.293857
2023-01-14	-0.420341	-1.718105
2023-01-15	-0.541642	0.317464

```
[201]: # df.loc[<critério de seleção de linha>, <nome da colunas>] = <novo valor>
df.loc[df['C']<0, 'B'] = 0
```

```
[202]: df.loc[:, ['B', 'C']].head(8)
```

```
[202]:
```

	B	C
2023-01-01	0.000000	-0.634441
2023-01-02	0.044171	0.032811
2023-01-03	0.000000	-0.147528
2023-01-04	0.426950	0.617419

```

2023-01-05  0.000000 -0.400563
2023-01-06 -0.105899  0.098682
2023-01-07  1.564196  0.453818
2023-01-08  0.000000 -0.837648

```

Uma funcionalidade muito útil é a função `pandas.DataFrame.groupby` que realiza agrupamento de acordo com valores de colunas informadas e aplica determinada função.

```
[203]: df.groupby('F').mean()
```

```

[203]:           A          B          C          D          E          G
F
Neg -0.218784  0.069176 -0.419876 -0.648841 -0.487138 -0.358809
Pos  0.534128  0.228169  0.191587  0.063991  0.314979  0.335539

```

```
[204]: df.F.value_counts()
```

```

[204]: F
Neg      16
Pos      14
Name: count, dtype: int64

```

```

[205]: # Resultado em valor normalizado (entre 0 e 1)
df.F.value_counts(normalize=True)

```

```

[205]: F
Neg      0.533333
Pos      0.466667
Name: proportion, dtype: float64

```

```

[206]: # Valores divididos em 3 intervalos e contar para cada intervalo
df.A.value_counts(bins=3)

```

```

[206]: (-0.8, 0.418]          13
(0.418, 1.636]             12
(-2.0229999999999997, -0.8]  5
Name: count, dtype: int64

```

6.1 Aquisição de dados

O `pandas` conta com vários métodos de leitura de dados para criação de `DataFrame`. Abaixo temos listados os métodos, breve descrição e links para documentação oficial.

Método	Fonte do dado
<code>read_csv</code>	Arquivo CSV
<code>read_excel</code>	Microsoft Excel
<code>read_fwf</code>	Colunas de largura fixa
<code>read_table</code>	Arquivo com delimitador de coluna em geral

Método	Fonte do dado
<code>read_html</code>	Tabela HTML
<code>read_json</code>	Arquivo JSON
<code>read_xml</code>	Documento em formato XML
<code>read_clipboard</code>	MS Windows Clipboard
<code>read_gbq</code>	Google BigQuery
<code>read_hdf</code>	Arquivo HDF5
<code>read_pickle</code>	Arquivo Pickle
<code>read_sas</code>	Arquivo SAS em formato XPORT ou SAS7BDAT
<code>read_sql</code>	Base de Dados SQL
<code>read_sql_query</code>	Resultado de query string
<code>read_sql_table</code>	Tabela SQL
<code>read_stata</code>	Arquivo Stata
<code>read_orc</code>	Objeto ORC
<code>read_feather</code>	Arquivos Feather
<code>read_parquet</code>	Objeto Parquet
<code>read_spss</code>	Arquivo SPSS

Da mesma forma, existem vários métodos para exportação de dados:

<code>to_clipboard</code>	MS Windows Clipboard
<code>to_csv</code>	Arquivo csv
<code>to_dict</code>	Dicionário Python
<code>to_excel</code>	Arquivo MS Excel
<code>to_feather</code>	Formato Feather
<code>to_gbq</code>	Tabela Google BigQuery
<code>to_hdf</code>	Arquivo HDF5
<code>to_html</code>	Arquivo HTML
<code>to_json</code>	String JSON
<code>to_latex</code>	Tabela \LaTeX
<code>to_markdown</code>	Formato Markdown
<code>to_numpy</code>	<code>ndarray</code> (NumPy)
<code>to_records</code>	<code>ndarray</code> (NumPy) com mais opções
<code>to_orc</code>	Objeto ORC
<code>to_parquet</code>	Arquivo binário em formato Parquet
<code>to_period</code>	Formato PeriodIndex
<code>to_pickle</code>	Arquivo Pickle
<code>to_sql</code>	Base de dados SQL
<code>to_stata</code>	Arquivo dta formato Stata
<code>to_string</code>	Tabular console-friendly
<code>to_timestamp</code>	DatetimeArray
<code>to_xarray</code>	Objeto xarray
<code>to_xml</code>	Arquivo XML

```
[207]: df_m5 = pd.read_csv('FATOR_CAPACIDADE-2_2022_05.csv', sep=';')
df_m5.head()
```

```
[207]: id_subsistema nom_subsistema id_estado nom_estado \
0      N      Norte      MA      MARANHAO
1      NE     Nordeste     BA      BAHIA
2      NE     Nordeste     BA      BAHIA
3      NE     Nordeste     BA      BAHIA
4      NE     Nordeste     BA      BAHIA

      nom_pontoconexao nom_localizacao val_latitude val_longitude \
0      MIRANDA II500kVA      Interior      -2.727222      -42.596389
1      PINDAI II - 230 kV (A)      Interior      -14.353933      -42.575842
2      IGAPORA II - 230 kV (B)      Interior      -14.102794      -42.609369
3      U.SOBRADINHO - 500 kV (A)      Interior      -9.751812      -41.006198
4      MORRO CHAPEU2 - 230 kV (A)      Interior      -10.970000      -41.228000

      nom_modalidadeoperacao nom_tipousina      nom_usina_conjunto \
0      Conjunto de Usinas      Eólica      Conj. Paulino Neves
1      Conjunto de Usinas      Eólica      Conj. Abil I
2      Conjunto de Usinas      Eólica      Conj. Araçás
3      Conjunto de Usinas      Eólica      Conj. Arizona
4      Conjunto de Usinas      Eólica      Conj. Babilônia

      din_instante      val_geracao      val_capacidadeinstalada \
0      2022-05-01 00:00:00      1.234      426.00
1      2022-05-01 00:00:00      61.016      90.00
2      2022-05-01 00:00:00      126.185      167.70
3      2022-05-01 00:00:00      69.273      124.74
4      2022-05-01 00:00:00      116.351      136.50

      val_fatorcapacidade
0      0.002897
1      0.677956
2      0.752445
3      0.555339
4      0.852388
```

```
[208]: df_m5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120648 entries, 0 to 120647
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id_subsistema          120648 non-null object
1   nom_subsistema          120648 non-null object
2   id_estado              120648 non-null object
3   nom_estado              120648 non-null object
4   nom_pontoconexao        119904 non-null object
5   nom_localizacao         111720 non-null object
```



```

6   val_latitude           119904 non-null float64
7   val_longitude          119904 non-null float64
8   nom_modalidadeoperacao 120648 non-null object
9   nom_tipousina          120648 non-null object
10  nom_usina_conjunto      120648 non-null object
11  din_instante           120648 non-null object
12  val_geracao            120648 non-null float64
13  val_capacidadeinstalada 120648 non-null float64
14  val_fatorcapacidade     120648 non-null float64

```

dtypes: float64(5), object(10)

memory usage: 13.8+ MB

```
[209]: df_m6 = pd.read_excel('FATOR_CAPACIDADE-2_2022_06.xlsx')
df_m6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 112536 entries, 0 to 112535

Data columns (total 15 columns):

#	Column	Non-Null Count	Dtype
0	id_sistema	112536 non-null	object
1	nom_sistema	112536 non-null	object
2	id_estado	112536 non-null	object
3	nom_estado	112536 non-null	object
4	nom_pontoconexao	111864 non-null	object
5	nom_localizacao	104472 non-null	object
6	val_latitude	111744 non-null	float64
7	val_longitude	111744 non-null	float64
8	nom_modalidadeoperacao	112536 non-null	object
9	nom_tipousina	112536 non-null	object
10	nom_usina_conjunto	112536 non-null	object
11	din_instante	112536 non-null	datetime64[ns]
12	val_geracao	112536 non-null	float64
13	val_capacidadeinstalada	112536 non-null	float64
14	val_fatorcapacidade	112536 non-null	float64

dtypes: datetime64[ns](1), float64(5), object(9)

memory usage: 12.9+ MB

É comum que se tenha mais de uma fonte de dados e que seja necessário concatenar tabelas em um único Data Frame. O método `pandas.concat` pode ser usado para esta tarefa.

```
[210]: # Concatenar Data Frames
dfc = pd.concat([df_m5, df_m6])

dfc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

Index: 233184 entries, 0 to 112535

Data columns (total 15 columns):

#	Column	Non-Null Count	Dtype
0	id_subsistema	233184 non-null	object
1	nom_subsistema	233184 non-null	object
2	id_estado	233184 non-null	object
3	nom_estado	233184 non-null	object
4	nom_pontoconexao	231768 non-null	object
5	nom_localizacao	216192 non-null	object
6	val_latitude	231648 non-null	float64
7	val_longitude	231648 non-null	float64
8	nom_modalidadeoperacao	233184 non-null	object
9	nom_tipousina	233184 non-null	object
10	nom_usina_conjunto	233184 non-null	object
11	din_instante	233184 non-null	object
12	val_geracao	233184 non-null	float64
13	val_capacidadeinstalada	233184 non-null	float64
14	val_fatorcapacidade	233184 non-null	float64

dtypes: float64(5), object(10)
memory usage: 28.5+ MB

```
[211]: dfc.memory_usage(deep=True) # Uso de memória de cada coluna
```

```
[211]: Index          1865472
id_subsistema    13739448
nom_subsistema   15205416
id_estado        13757856
nom_estado       15745560
...
nom_usina_conjunto 19052328
din_instante      24474144
val_geracao       1865472
val_capacidadeinstalada 1865472
val_fatorcapacidade 1865472
Length: 16, dtype: int64
```

O limite para quantidade dados a ser tratada no pandas é definido pela memória RAM disponível. O tipo de dado das variáveis, `dtype`, é um aspecto importante para otimização de dados. Na importação de dados o pandas define de forma automática o tipo de dados de cada coluna, caso não seja declarado. A definição de `dtype` para cada coluna reduz a memória RAM utilizada pela DataFrame. No caso do exemplo abaixo, temos uma redução de utilização de memória RAM para menos da metade.

```
[212]: # Redefinir tipo de cada coluna com intuito de diminuir memória
# Esta definição poderia ser feita no momento da criação do df
dic_dtype = {'id_subsistema': 'category',
             'nom_subsistema': 'category',
             'id_estado': 'category',
             'nom_estado': 'category',
```

```

        'nom_pontoconexao': 'category',
        'nom_localizacao': 'category',
        'val_latitude': np.float32,
        'val_longitude': np.float32,
        'nom_modalidadeoperacao': str,
        'nom_tipousina': 'category',
        'nom_usina_conjunto': str,
        'din_instante': 'datetime64[ns]',
        'val_geracao': np.float32,
        'val_capacidadeinstalada': np.float32,
        'val_fatorcapacidade': np.float32}
dfc1 = dfc.astype(dic_dtype) # Redefinindo datatype das colunas
dfc1.index = dfc1.index.astype(np.int32) # Redefinindo datatype do index

```

[213]: dfc1.info()

```

<class 'pandas.core.frame.DataFrame'>
Index: 233184 entries, 0 to 112535
Data columns (total 15 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id_subsistema                        233184 non-null category
1   nom_subsistema                      233184 non-null category
2   id_estado                          233184 non-null category
3   nom_estado                         233184 non-null category
4   nom_pontoconexao                   231768 non-null category
5   nom_localizacao                    216192 non-null category
6   val_latitude                       231648 non-null float32
7   val_longitude                      231648 non-null float32
8   nom_modalidadeoperacao             233184 non-null object
9   nom_tipousina                      233184 non-null category
10  nom_usina_conjunto                  233184 non-null object
11  din_instante                       233184 non-null datetime64[ns]
12  val_geracao                        233184 non-null float32
13  val_capacidadeinstalada            233184 non-null float32
14  val_fatorcapacidade                233184 non-null float32
dtypes: category(7), datetime64[ns](1), float32(5), object(2)
memory usage: 12.2+ MB

```

[214]:

```

print('Memória dfc1 / dfc: ', end='')
print(f'{{dfc1.memory_usage().sum() / dfc.memory_usage().sum():.2%}}')
print()
print(dfc1.memory_usage(deep=True) / dfc.memory_usage(deep=True))

```

Memória dfc1 / dfc: 42.98%

```

Index                0.500000
id_subsistema        0.017001

```

```

nom_subsistema      0.015364
id_estado            0.017018
nom_estado           0.014875
...
nom_usina_conjunto   1.000000
din_instante         0.076222
val_geracao          0.500000
val_capacidadeinstalada 0.500000
val_fatorcapacidade  0.500000
Length: 16, dtype: float64

```

```
[215]: df_ansi = pd.read_html('http://engelco.com.br/tabela-ansi/')[0]
df_ansi
```

```

[215]:      0      1
0      NR      DENOMINAÇÃO
1      1      Elemento Principal
2      2      Relé de partida ou fechamento temporizado
3      3      Relé de verificação ou interbloqueio
4      4      Contator principal
...      ...
112     RIO      Dispositivo Remoto de Inputs/Outputs
113     RTU      Unidade de terminal remoto / Concentrador de D...
114     SER      Sistema de armazenamento de eventos
115     TCM      Esquema de monitoramento de Trip
116     SOTF     Fechamento sob falta

```

[117 rows x 2 columns]

Colocamos o [0] no fim da linha para trazer a primeira tabela. O retorno da função `pd.read_html` é uma lista do Python com todas as tabelas encontradas em uma página html, sendo 0 a primeira tabela encontrada, 1 a segunda tabela, e assim por diante.

```
[216]: df_ff = pd.read_fwf('faithful.dat')
df_ff.head()
```

```

[216]:   ID  eruptions  waiting
0    1     3.600      79
1    2     1.800      54
2    3     3.333      74
3    4     2.283      62
4    5     4.533      85

```

6.2 Exemplo de Análise Exploratória de Dados

```
[217]: cod_bcb = 433 # Código para coleta do IPCA no API do Banco Central
url = f'http://api.bcb.gov.br/dados/serie/bcdata.sgs.{cod_bcb}/dados?
      ↪formato=json'
df_ipca = pd.read_json(url)
df_ipca.tail(10)
```

```
[217]:
```

	data	valor
523	01/09/2023	0.26
524	01/10/2023	0.24
525	01/11/2023	0.28
526	01/12/2023	0.56
527	01/01/2024	0.42
528	01/02/2024	0.83
529	01/03/2024	0.16
530	01/04/2024	0.38
531	01/05/2024	0.46
532	01/06/2024	0.21

```
[218]: df_ipca.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 533 entries, 0 to 532
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  -
 0   data    533 non-null       object  
 1   valor   533 non-null       float64  
dtypes: float64(1), object(1)
memory usage: 8.5+ KB
```

```
[219]: # Transformar coluna data em formato datetime
df_ipca['data'] = pd.to_datetime(df_ipca['data'], dayfirst=True)

# Defino coluna data como indice do data frame
df_ipca.set_index('data', inplace=True)

# Trocar nome da coluna "valor" para "ipca"
df_ipca.columns = ['ipca']

# Tambem se pode renomear usando função rename
df_ipca.tail()
```

```
[219]:
```

	ipca
data	
2024-02-01	0.83
2024-03-01	0.16

```
2024-04-01    0.38
2024-05-01    0.46
2024-06-01    0.21
```

```
[220]: df_ipca.info()
```

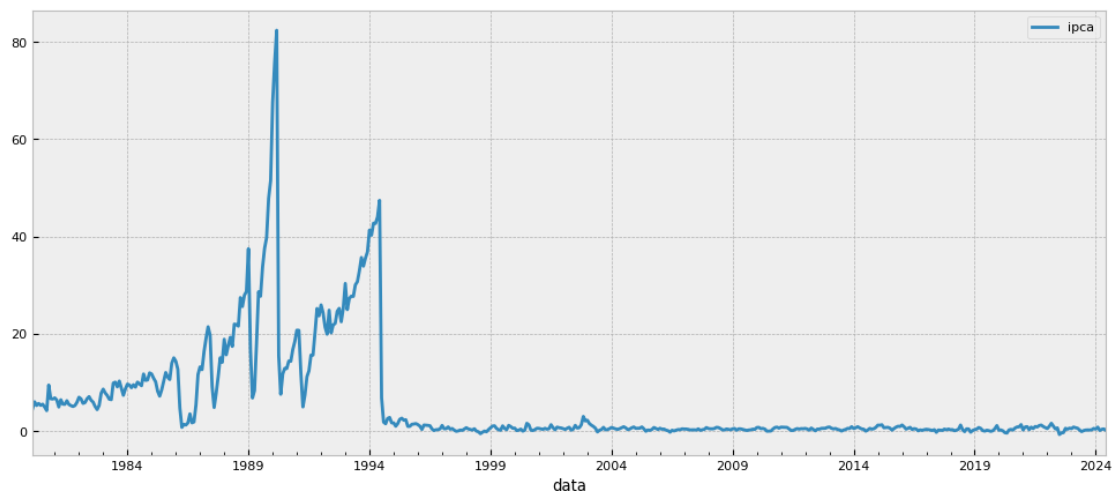
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 533 entries, 1980-02-01 to 2024-06-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  -
 0   ipca    533 non-null     float64
dtypes: float64(1)
memory usage: 8.3 KB
```

```
[221]: df_ipca.loc['2022-01-01':'2022-12-01'].describe()
```

```
[221]:          ipca
count    12.000000
mean      0.471667
std       0.649235
min      -0.680000
25%       0.235000
50%       0.565000
75%       0.755000
max       1.620000
```

O pandas conta com uma integração com a biblioteca matplotlib, conseguindo usar métodos de plotagem direto do DataFrame. Abaixo estão alguns exemplos de plotagem de gráfico de linhas, barras, histograma, diagrama de caixa (boxplot), gráfico de diferenças e de autocorrelação.

```
[222]: # Gráfico histórico do IPCA
df_ipca.plot(figsize=(12,5))
plt.show()
```



```
[223]: # Gráfico IPCA 2022
df_ipca.loc['2022-01-01':'2022-12-01'].plot(figsize=(12,5))
plt.show()
```



```
[224]: # IPCA entre janeiro de 2023 até último registro
df_ipca.loc['2023-01-01':]
```

```
[224]:          ipca
data
2023-01-01  0.53
2023-02-01  0.84
2023-03-01  0.71
2023-04-01  0.61
2023-05-01  0.23
...
2024-02-01  0.83
2024-03-01  0.16
2024-04-01  0.38
2024-05-01  0.46
2024-06-01  0.21

[18 rows x 1 columns]
```

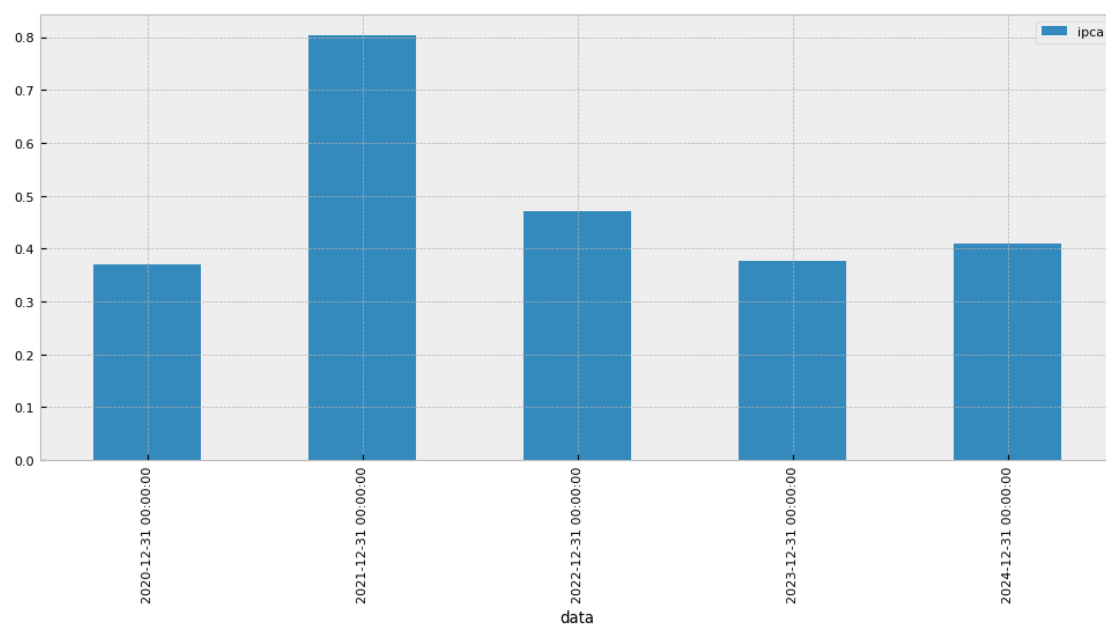
```
[225]: df_ipca_4a = df_ipca.loc['2020-01-01':]
```

```
[226]: # Refazer amostra de forma anual usando média do IPCA
df_ipca_4a.resample('YE').mean()
```

```
[226]:          ipca
data
2020-12-31  0.370000
2021-12-31  0.802500
2022-12-31  0.471667
2023-12-31  0.377500
2024-12-31  0.410000
```

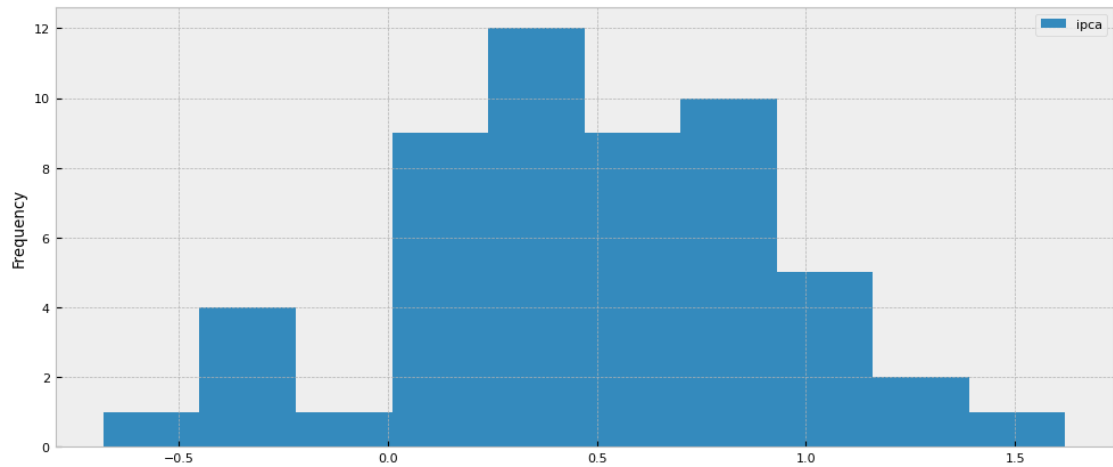
```
[227]: df_ipca_4a.resample('YE').mean().plot.bar(figsize=(12,5))
plt.plot()
```

```
[227]: []
```



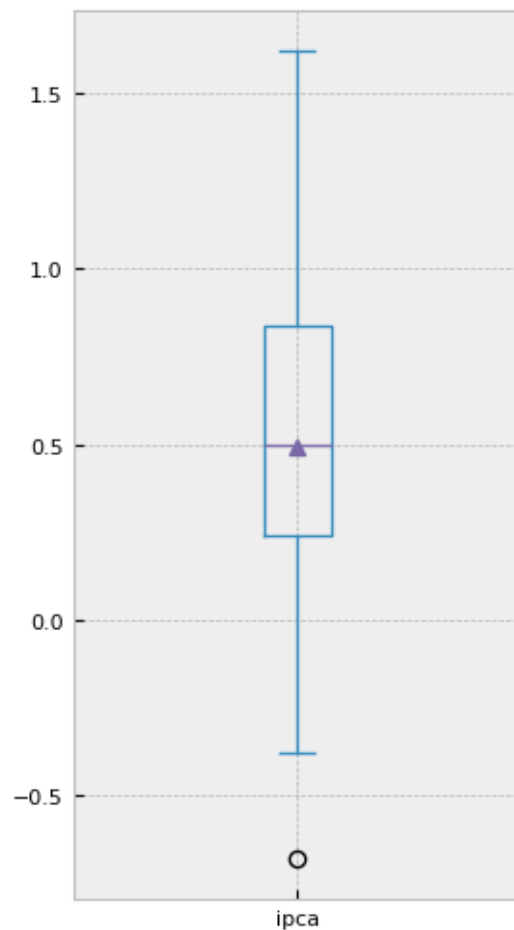
```
[228]: df_ipca_4a.plot.hist(figsize=(12,5))
plt.plot()
```

```
[228]: []
```

```
[229]: print(df_ipca_4a.describe().T)
df_ipca_4a.plot.box(showmeans=True, figsize=(3,6))
plt.show()
```

	count	mean	std	min	25%	50%	75%	max
ipca	54.0	0.494815	0.455734	-0.68	0.24	0.5	0.8375	1.62



```
[230]: # Somar unidade a cada índice e calcular inflação acumulada
print('***',2023,'***')
print((df_ipca.loc['2023-01-01:']/100 + 1).apply(np.cumprod))
print()
print('***',2022,'***')
print(((df_ipca.loc['2022-01-01':'2023-01-01']/100 + 1).apply(np.cumprod)))
```

```
*** 2023 ***
```

```
      ipca
data
2023-01-01  1.005300
2023-02-01  1.013745
2023-03-01  1.020942
2023-04-01  1.027170
2023-05-01  1.029532
...
2024-02-01  1.059325
2024-03-01  1.061020
```

```
2024-04-01  1.065052
2024-05-01  1.069951
2024-06-01  1.072198
```

```
[18 rows x 1 columns]
```

```
*** 2022 ***
```

```
                ipca
data
2022-01-01  1.005400
2022-02-01  1.015555
2022-03-01  1.032007
2022-04-01  1.042946
2022-05-01  1.047848
2022-06-01  1.054868
2022-07-01  1.047695
2022-08-01  1.043923
2022-09-01  1.040896
2022-10-01  1.047037
2022-11-01  1.051330
2022-12-01  1.057848
2023-01-01  1.063455
```

Para traçar um gráfico de médias móveis pode ser usada o método `rolling` definindo o tamanho da janela (no exemplo uma janela de 3 meses) e aplicando o método `mean`.

```
[231]: plt.style.use('ggplot')
fig, ax = plt.subplots(figsize=(12,6))
df_ipca_4a.plot(ax=ax)
df_ipca_4a.rolling(window=3).mean().plot(ax=ax, style='g:')
ax.legend(['IPCA', 'Média Móvel (3 meses)']);
```

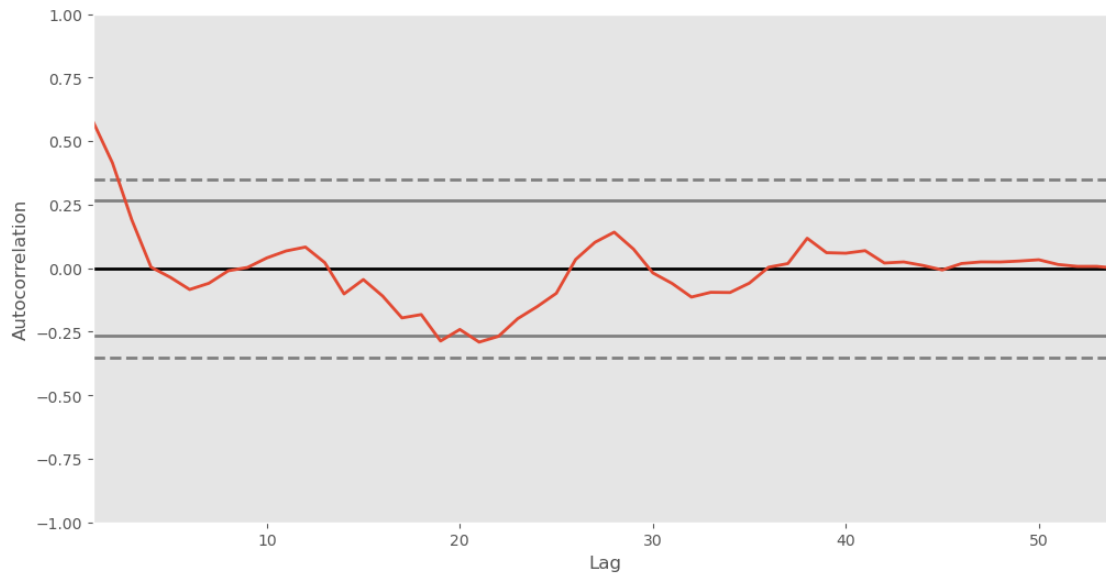


```
[232]: fig, ax = plt.subplots(figsize=(12,6))
df_ipca_4a.plot(ax=ax)
df_ipca_4a.diff().plot(ax=ax, style='g:')
ax.legend(['IPCA', 'Diferença']);
```



```
[233]: print(df_ipca_4a.ipca.autocorr())
plt.figure(figsize=(12,6))
pd.plotting.autocorrelation_plot(df_ipca_4a)
plt.show()
```

0.580598732155939



```
[234]: cod_bcb = 189 # IGP-M
url = f'http://api.bcb.gov.br/dados/serie/bcdata.sgs.{cod_bcb}/dados?
↳formato=json'
df_igpm = pd.read_json(url)
df_igpm['data'] = pd.to_datetime(df_igpm['data'], dayfirst=True)
df_igpm.set_index('data', inplace=True)
df_igpm.rename(columns={'valor': 'igpm'}, inplace=True)
df_igpm.info()
df_igpm.tail()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 420 entries, 1989-07-01 to 2024-06-01
Data columns (total 1 columns):
#   Column  Non-Null Count  Dtype
---  -
0   igpm    420 non-null         float64
dtypes: float64(1)
memory usage: 6.6 KB
```

```
[234]:          igpm
data
2024-02-01 -0.52
2024-03-01 -0.47
2024-04-01  0.31
2024-05-01  0.89
2024-06-01  0.81
```

```
[235]: cod_bcb = 191 # IPC-BR
url = f'http://api.bcb.gov.br/dados/serie/bcdata.sgs.{cod_bcb}/dados?
      ↳formato=json'
df_ipcbr = pd.read_json(url)
df_ipcbr['data'] = pd.to_datetime(df_ipcbr['data'], dayfirst=True)
df_ipcbr.set_index('data', inplace=True)
df_ipcbr.columns=['ipcbr']
df_ipcbr.info()
df_ipcbr.tail()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 413 entries, 1990-02-01 to 2024-06-01
Data columns (total 1 columns):
#   Column  Non-Null Count  Dtype
---  -
0    ipcbr    413 non-null        float64
dtypes: float64(1)
memory usage: 6.5 KB
```

```
[235]:          ipcbr
data
2024-02-01    0.55
2024-03-01    0.10
2024-04-01    0.42
2024-05-01    0.53
2024-06-01    0.22
```

```
[236]: print(f'IPCA: {df_ipca.index.min()} ',
      f'IGPM: {df_igpm.index.min()} ',
      f'IPCBR: {df_ipcbr.index.min()} ', sep='\n')
```

```
IPCA: 1980-02-01 00:00:00
IGPM: 1989-07-01 00:00:00
IPCBR: 1990-02-01 00:00:00
```

O método `pandas.DataFrame.merge` serve para junção de DataFrames parecido com a funcionalidade que as funções `procv/procx` fazem no MS Excel.

```
[237]: # Pegamos como base IPCBR que é o que tem menos dados históricos
# Juntar com dados do IPCA
df_indices = pd.merge(df_ipcbr, df_ipca, how='left',
                      left_index=True, right_index=True)
df_indices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 413 entries, 1990-02-01 to 2024-06-01
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0    ipcbr    413 non-null        float64
1    ipca     413 non-null        float64
dtypes: float64(2)
memory usage: 12.9 KB
```

```

0   ipcbr   413 non-null   float64
1   ipca    413 non-null   float64
dtypes: float64(2)
memory usage: 9.7 KB

```

```

[238]: # Juntar base de IGPM com dados de IPCBR e IPCA
df_indices = df_indices.merge(df_igpm, how='left',
                              left_index=True, right_index=True)
df_indices.info()

```

```

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 413 entries, 1990-02-01 to 2024-06-01
Data columns (total 3 columns):
#   Column  Non-Null Count  Dtype
---  -
0   ipcbr   413 non-null      float64
1   ipca    413 non-null      float64
2   igpm    413 non-null      float64
dtypes: float64(3)
memory usage: 12.9 KB

```

```

[239]: df_4a = df_indices.loc['2020-01-01':]

```

```

[240]: # Correlação das colunas
df_4a.corr()

```

```

[240]:
      ipcbr      ipca      igpm
ipcbr  1.000000  0.872016  0.338258
ipca   0.872016  1.000000  0.327017
igpm   0.338258  0.327017  1.000000

```

```

[241]: df_4a.describe()

```

```

[241]:
      ipcbr      ipca      igpm
count  54.000000  54.000000  54.000000
mean    0.447963   0.494815   0.759074
std     0.467936   0.455734   1.360640
min    -1.190000  -0.680000  -1.930000
25%     0.270000   0.240000  -0.025000
50%     0.500000   0.500000   0.590000
75%     0.705000   0.837500   1.485000
max     1.430000   1.620000   4.340000

```

```

[242]: df_4a.describe(percentiles=[.10, .25, .5, .75, .9]).T

```

```

[242]:
      count      mean      std  min  10%   25%   50%   75%   90%  \
ipcbr   54.0  0.447963  0.467936 -1.19 -0.073  0.270  0.50  0.7050  0.982
ipca   54.0  0.494815  0.455734 -0.68 -0.035  0.240  0.50  0.8375  0.995

```

```
igpm      54.0  0.759074  1.360640 -1.93 -0.714 -0.025  0.59  1.4850  2.692
```

```
max
ipcbr  1.43
ipca   1.62
igpm   4.34
```

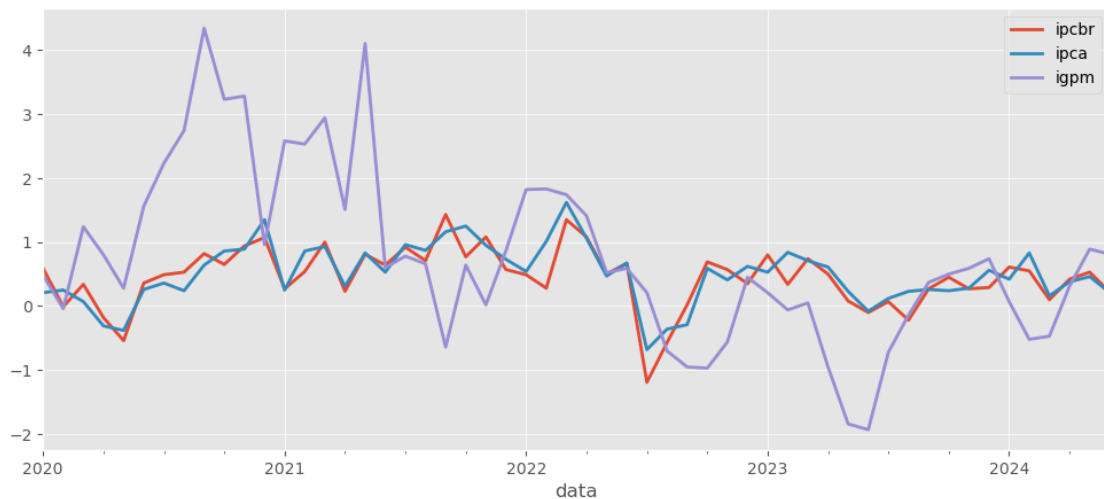
```
[243]: df_4a.query('igpm > 3')
```

```
[243]:      ipcbr  ipca  igpm
data
2020-09-01   0.82  0.64  4.34
2020-10-01   0.65  0.86  3.23
2020-11-01   0.94  0.89  3.28
2021-05-01   0.81  0.83  4.10
```

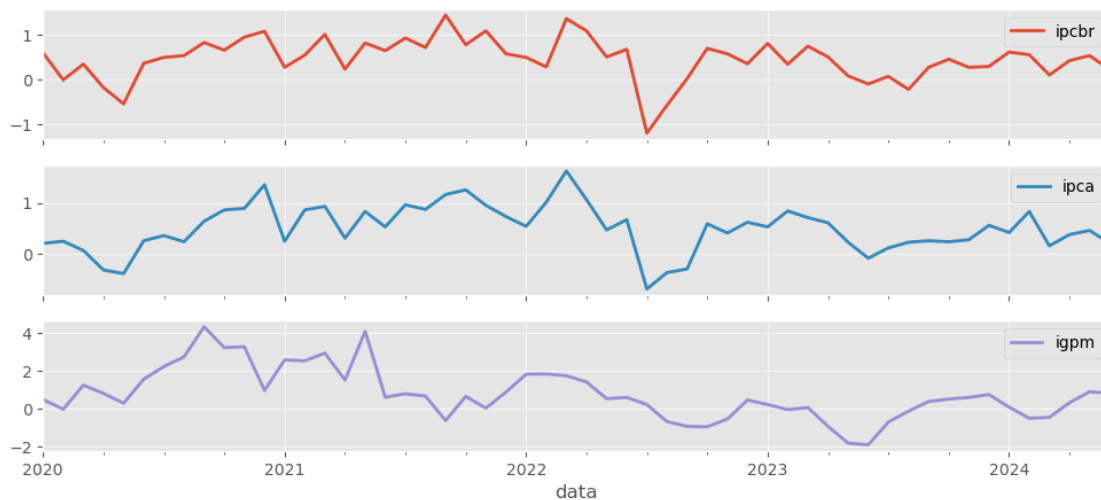
```
[244]: df_4a.query('igpm > 3 and ipca > 0.85')
```

```
[244]:      ipcbr  ipca  igpm
data
2020-10-01   0.65  0.86  3.23
2020-11-01   0.94  0.89  3.28
```

```
[245]: df_4a.plot(figsize=(12,5))
plt.show()
```

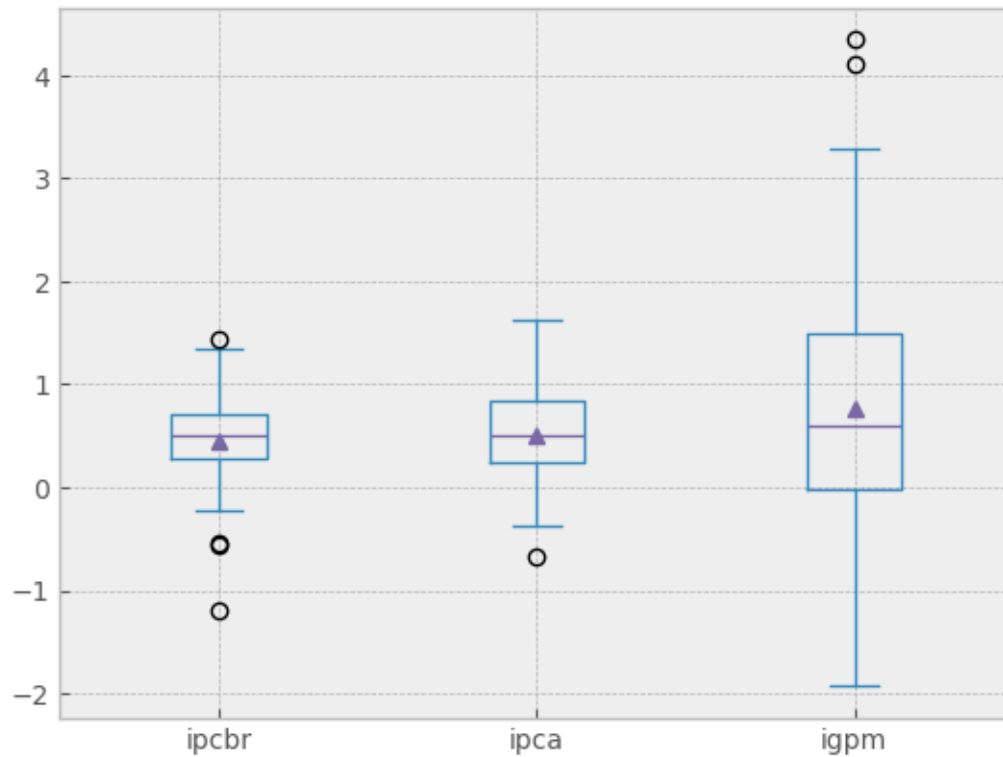


```
[246]: df_4a.plot(subplots=True, figsize=(12,5))
plt.show()
```

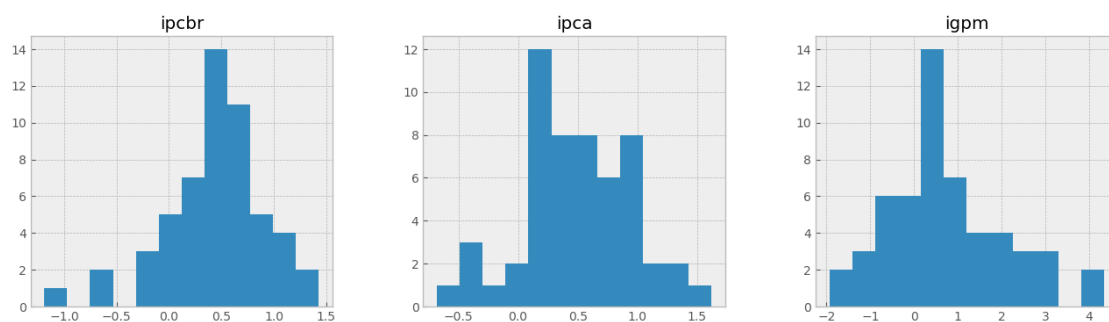



```
[247]: print(df_4a.describe().T)
plt.style.use('bmh')
df_4a.plot.box(showmeans=True)
plt.show()
```

	count	mean	std	min	25%	50%	75%	max
ipcbr	54.0	0.447963	0.467936	-1.19	0.270	0.50	0.7050	1.43
ipca	54.0	0.494815	0.455734	-0.68	0.240	0.50	0.8375	1.62
igpm	54.0	0.759074	1.360640	-1.93	-0.025	0.59	1.4850	4.34



```
[248]: df_4a.hist(bins=12, layout=(1,3), figsize=(16,4))
plt.show()
```



6.3 Seaborn

Seaborn é um pacote de visualização de dados baseado no matplotlib aplicado no Python. É bastante aplicado para visualização em análise estatística de dados.

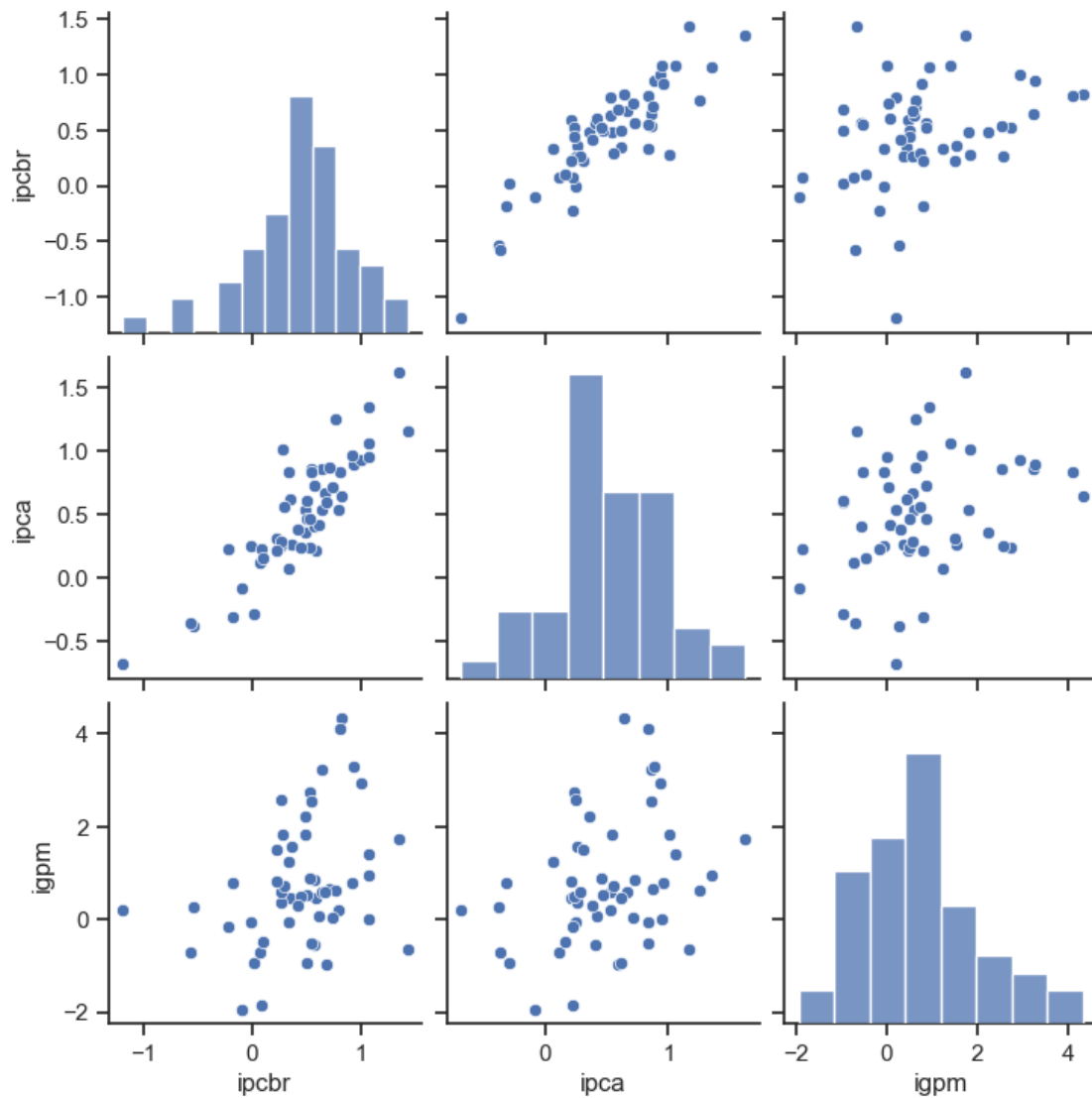
```
[249]: import seaborn as sns
sns.__version__
```

[249]: '0.13.2'

```
[250]: print(df_4a.corr()) # Tabela de Correlação
sns.set_theme(style="ticks")
sns.pairplot(df_4a)
```

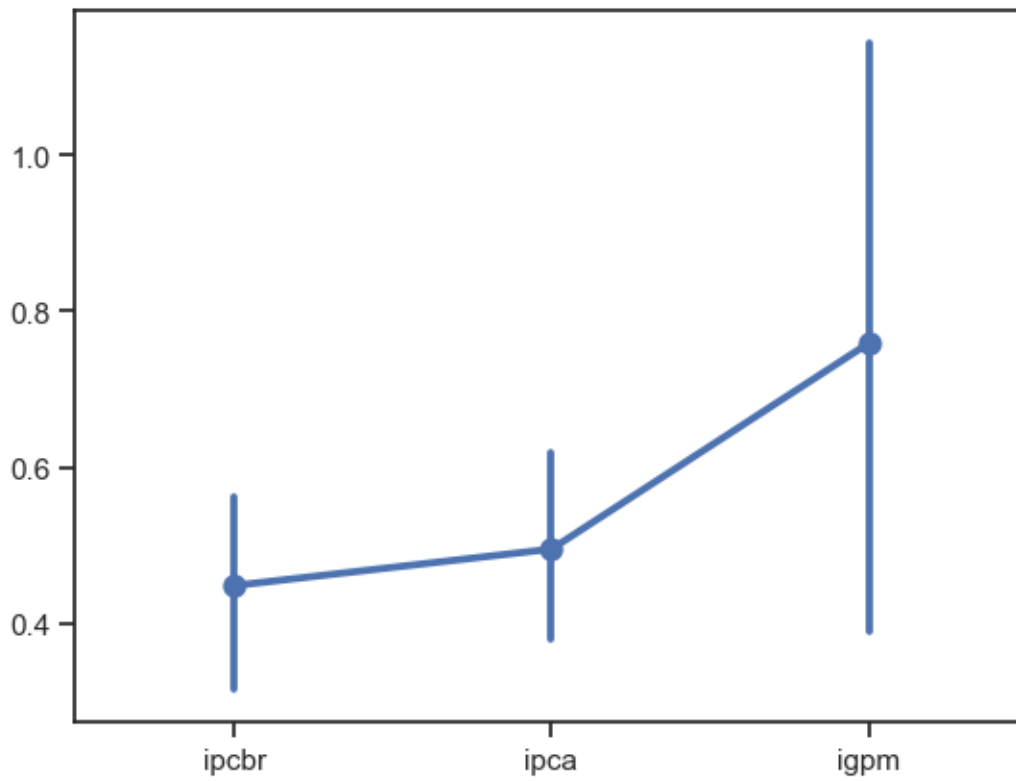
	ipcbr	ipca	igpm
ipcbr	1.000000	0.872016	0.338258
ipca	0.872016	1.000000	0.327017
igpm	0.338258	0.327017	1.000000

[250]: <seaborn.axisgrid.PairGrid at 0x2d257893990>

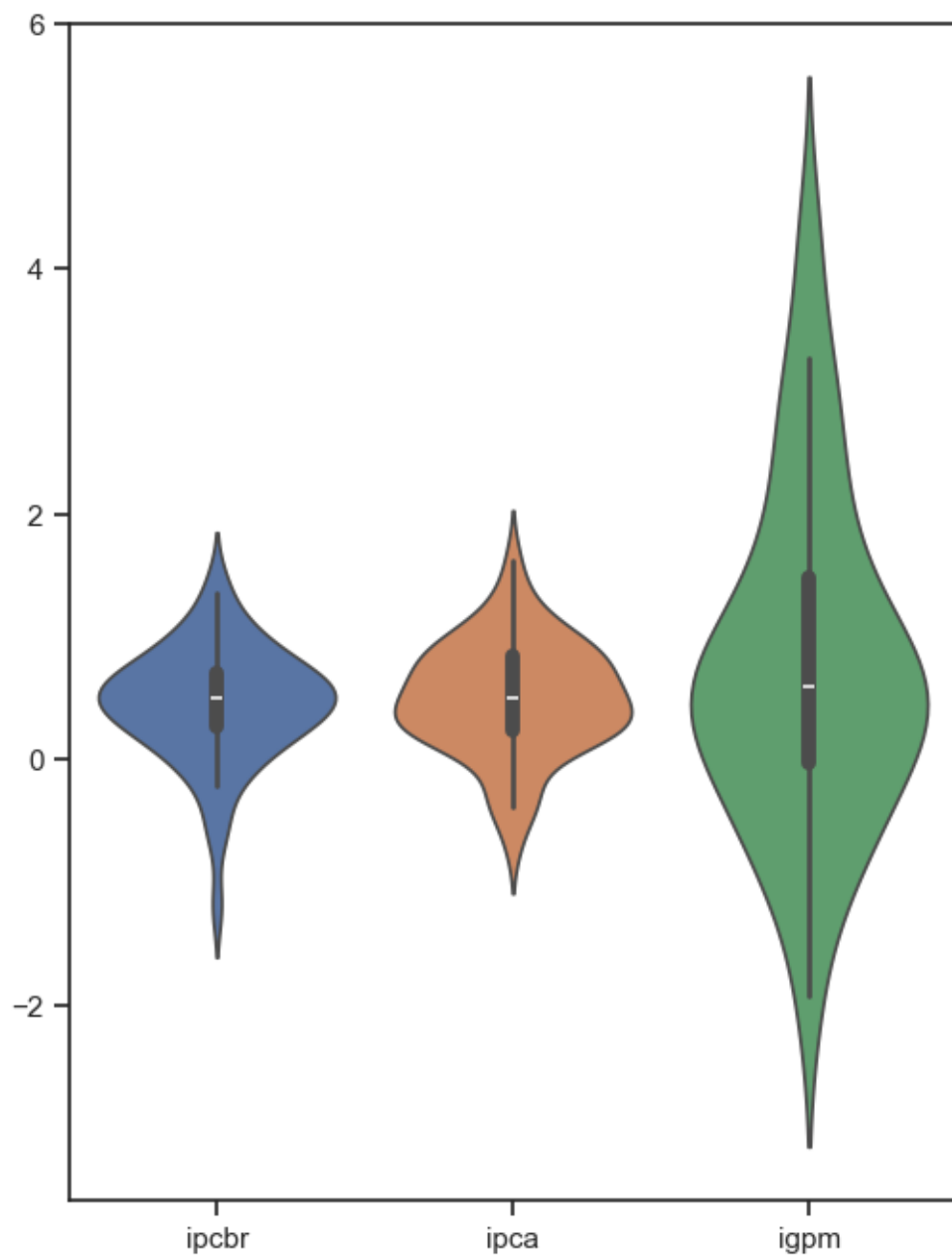


```
[251]: sns.pointplot(df_4a)
```

```
[251]: <Axes: >
```

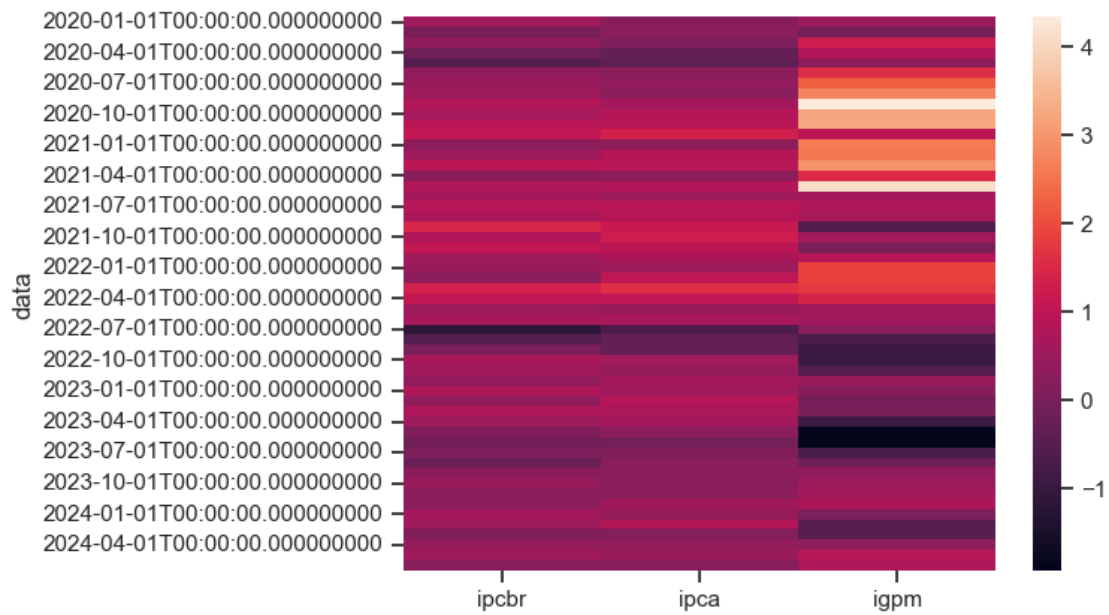


```
[252]: plt.figure(figsize=(6,8))  
sns.violinplot(data=df_4a)  
plt.show()
```



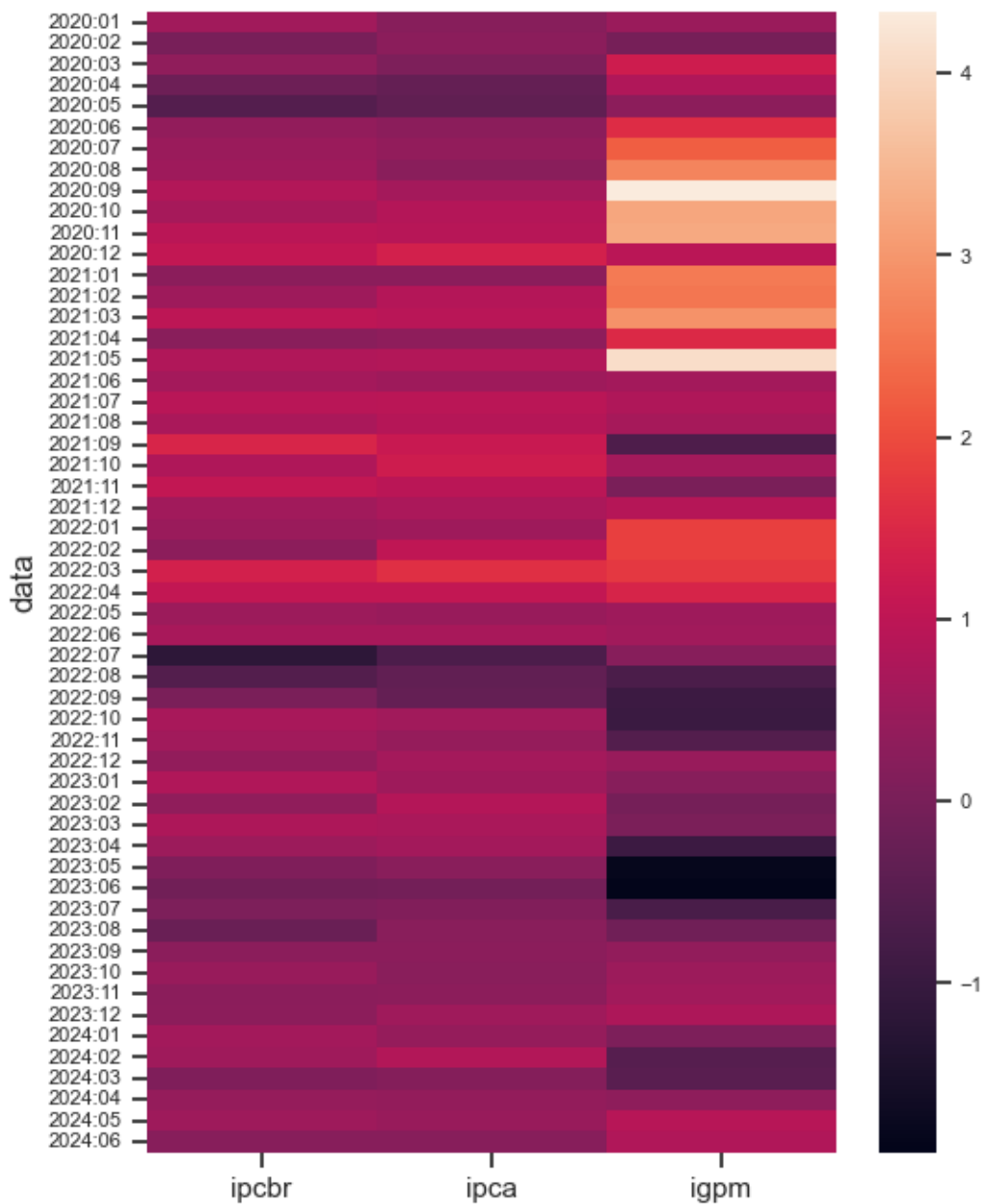
```
[253]: sns.heatmap(df_4a)
```

```
[253]: <Axes: ylabel='data'>
```



```
[254]: plt.figure(figsize=(6, 8)) # Definir tamanho da figura
plt.rc('ytick', labelsizes=8) # Definir tamanho de fonte do eixo y
sns.heatmap(df_4a, yticklabels=df_4a.index.strftime('%Y:%m'))
```

```
[254]: <Axes: ylabel='data'>
```



7 Anexos

7.1 Google Colab

- Toda conta Google tem acesso ao ambiente do Colab.
- O [Google Colab](#) entende códigos em [\$\text{\LaTeX}\$](#) , [Markdown](#) e HTML

Gerar PDF de um notebook no Google Colab

```
from IPython.display import set_matplotlib_formats
```

```
set_matplotlib_formats('pdf', 'svg')
```

```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py  
from colab_pdf import colab_pdf  
colab_pdf('Introdução ao Python.ipynb')
```

7.2 Easter egg no Python - Zen of Python

```
[255]: import this
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!