Data Science Academy - Python Fundamentos - Capítulo 8

Download: http://github.com/dsacademybr (http://github.com/dsacademybr)

Análise de Dados em tempo Real

NumPy

Computação científica matemática

Numerical Python

Para importar numpy, utilize: import numpy as np

Você também pode utilizar: from numpy import * . Isso evitará a utilização de np., mas este comando importará todos os módulos do NumPy.

Para atualizar o NumPy, abra o prompt de comando e digite: pip install numpy -U

numeric + Numarray = => Numpy

In [1]:

```
# Importando o NumPy
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.14.3'
```

Criando Arrays

In [3]:

```
# Help
help(np.array)
```

```
Help on built-in function array in module numpy.core.multiarray:

array(...)
    array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

    Create an array.

    Parameters
    ----------
    object : array_like
        An array, any object exposing the array interface, an object whose
        __array__ method returns an array, or any (nested) sequence.
    dtype : data-type, optional
        The desired data-type for the array.  If not given, then the type wi
ll
        be determined as the minimum type required to hold the objects in th
e
        sequence.  This argument can only be used to 'upcast' the array.  Fo
r
        downcasting, use the .astype(t) method.
    copy : bool, optional
        If true (default), then the object is copied.  Otherwise, a copy wil
l
        only be made if __array__ returns a copy, if obj is a nested sequenc
e,
        or if a copy is needed to satisfy any of the other requirements
        (`dtype`, `order`, etc.).
    order : {'K', 'A', 'C', 'F'}, optional
        Specify the memory layout of the array. If object is not an array, t
he
        newly created array will be in C order (row major) unless 'F' is
        specified, in which case it will be in Fortran order (column major).
        If object is an array the following holds.

        ===== ========= ===================================================
        order  no copy                     copy=True
        ===== ========= ===================================================
        'K'   unchanged F & C order preserved, otherwise most similar order
        'A'   unchanged F order if input is F and not C, otherwise C order
        'C'   C order   C order
        'F'   F order   F order
        ===== ========= ===================================================

        When ``copy=False`` and a copy is made for other reasons, the result
is
        the same as if ``copy=True``, with some exceptions for `A`, see the
        Notes section. The default order is 'K'.
    subok : bool, optional
        If True, then sub-classes will be passed-through, otherwise
        the returned array will be forced to be a base-class array (defaul
t).
    ndmin : int, optional
        Specifies the minimum number of dimensions that the resulting
        array should have.  Ones will be pre-pended to the shape as
```

```
        needed to meet this requirement.

    Returns
    -------
    out : ndarray
        An array object satisfying the specified requirements.

    See Also
    --------
    empty, empty_like, zeros, zeros_like, ones, ones_like, full, full_like

    Notes
    -----
    When order is 'A' and `object` is an array in neither 'C' nor 'F' order,
    and a copy is forced by a change in dtype, then the order of the result
is
    not necessarily 'C' as expected. This is likely a bug.

    Examples
    --------
    >>> np.array([1, 2, 3])
    array([1, 2, 3])

    Upcasting:

    >>> np.array([1, 2, 3.0])
    array([ 1.,  2.,  3.])

    More than one dimension:

    >>> np.array([[1, 2], [3, 4]])
    array([[1, 2],
           [3, 4]])

    Minimum dimensions 2:

    >>> np.array([1, 2, 3], ndmin=2)
    array([[1, 2, 3]])

    Type provided:

    >>> np.array([1, 2, 3], dtype=complex)
    array([ 1.+0.j,  2.+0.j,  3.+0.j])

    Data-type consisting of more than one element:

    >>> x = np.array([(1,2),(3,4)],dtype=[('a','<i4'),('b','<i4')])
    >>> x['a']
    array([1, 3])

    Creating an array from sub-classes:

    >>> np.array(np.mat('1 2; 3 4'))
    array([[1, 2],
           [3, 4]])

    >>> np.array(np.mat('1 2; 3 4'), subok=True)
    matrix([[1, 2],
```

```
[3, 4]])
```

Numpy cria a base para análise dos dados
Por outros pacotes como Pandas.

In [4]:

```
# Array criado a partir de uma lista:
vetor1 = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [5]:

```
print(vetor1)
```

```
[0 1 2 3 4 5 6 7 8]
```

In [6]:

```
# Um objeto do tipo ndarray é um recipiente multidimensional de itens do mesmo tipo e tamar
type(vetor1)
```

Out[6]:

```
numpy.ndarray
```

In [7]:

```
# Usando métodos do array NumPy
vetor1.cumsum()
```

Soma o 1º Nº com o seguinte

Out[7]:

```
array([ 0,  1,  3,  6, 10, 15, 21, 28, 36])
```

In [8]:

```
# Criando uma lista. Perceba como listas e arrays são objetos diferentes, com diferentes pr
lst = [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

In [9]:

```
lst
```

Out[9]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

In [10]:

```
type(lst)
```

Out[10]:

```
list
```

In [11]:

```
# Imprimindo na tela um elemento especifico no array
vetor1[0]
```

Out[11]:

```
0
```

In [12]:

```
# Alterando um elemento do array
vetor1[0] = 100
```

In [13]:

```
print(vetor1)
```

```
[100   1   2   3   4   5   6   7   8]
```

In [14]:

```
# Não é possível incluir elemento de outro tipo
vetor1[0] = 'Novo elemento'
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-7ecc71e33a97> in <module>()
      1 # Não é possível incluir elemento de outro tipo
----> 2 vetor1[0] = 'Novo elemento'

ValueError: invalid literal for int() with base 10: 'Novo elemento'
```

In [15]:

```
# Verificando o formato do array
print(vetor1.shape)
```

```
(9,)
```

Funções NumPy

In [16]:

```
# A função arange cria um vetor contendo uma progressão aritmética a partir de um intervalo
vetor2 = np.arange(0., 4.5, .5)
```

Parecido com função Range

Início Fim Intervalo

In [17]:

```
print(vetor2)
```

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

In [18]:

```
# Verificando o tipo do objeto
type(vetor2)
```

Out[18]:

```
numpy.ndarray
```

In [19]:

```
# Formato do array
np.shape(vetor2)
```

Out[19]:

```
(9,)
```

In [20]:

```
print (vetor2.dtype)
```

```
float64
```

de 1 a 10 crescente de 0,25

In [21]:

```
x = np.arange(1, 10, 0.25)
print(x)
```

```
[1.   1.25 1.5  1.75 2.   2.25 2.5  2.75 3.   3.25 3.5  3.75 4.   4.25
 4.5  4.75 5.   5.25 5.5  5.75 6.   6.25 6.5  6.75 7.   7.25 7.5  7.75
 8.   8.25 8.5  8.75 9.   9.25 9.5  9.75]
```

In [22]:

```
print(np.zeros(10))
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [23]:

```
# Retorna 1 nas posições em diagonal e 0 no restante
z = np.eye(3)
```

In [24]:

```
z
```

Out[24]:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [25]:

```
# Os valores passados como parâmetro, formam uma diagonal
d = np.diag(np.array([1, 2, 3, 4]))
```

In [26]:

```
d
```

Out[26]:

```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

Cria um array e joga esses dados na Diagonal

In [27]:

```
# Array de números complexos
c = np.array([1+2j, 3+4j, 5+6*1j])
```

Numpy = Ferramenta matemática

?

In [28]:

```
c
```

Out[28]:

```
array([1.+2.j, 3.+4.j, 5.+6.j])
```

In [29]:

```
# Array de valores booleanos
b = np.array([True, False, False, True])
```

In [30]:

```
b
```

Out[30]:

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

In [31]:

```
# Array de strings
s = np.array(['Python', 'R', 'Julia'])
```

In [32]:

```
s
```

Out[32]:

```
array(['Python', 'R', 'Julia'], dtype='<U6')
```

In [33]:

```
# O método linspace (linearly spaced vector) retorna um número de
# valores igualmente distribuídos no intervalo especificado
np.linspace(0, 10)
```

Out[33]:

```
array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
        1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
        2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
        3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
        4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
        5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
        6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
        7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
        8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
        9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

In [34]:

```
print(np.linspace(0, 10, 15))
```

```
[ 0.          0.71428571  1.42857143  2.14285714  2.85714286  3.57142857
  4.28571429  5.          5.71428571  6.42857143  7.14285714  7.85714286
  8.57142857  9.28571429 10.        ]
```

In [35]:

```
print(np.logspace(0, 5, 10))
```

```
[1.00000000e+00 3.59381366e+00 1.29154967e+01 4.64158883e+01
 1.66810054e+02 5.99484250e+02 2.15443469e+03 7.74263683e+03
 2.78255940e+04 1.00000000e+05]
```

Criando Matrizes

In [36]:

```
# Criando uma matriz
matriz = np.array([[1,2,3],[4,5,6]])
```

In [37]:

```
print(matriz)
```

```
[[1 2 3]
 [4 5 6]]
```

In [38]:

```
print(matriz.shape)
```

```
(2, 3)
```

2 linhas
3 Colunas

In [39]:

```
# Criando uma matriz 2x3 apenas com números "1"
matriz1 = np.ones((2,3))
```

In [40]:

```
print(matriz1)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

In [41]:

```
# Criando uma matriz a partir de uma lista de listas
lista = [[13,81,22], [0, 34, 59], [21, 48, 94]]
```

In [42]:

```
# A função matrix cria uma matria a partir de uma sequência
matriz2 = np.matrix(lista)
```

In [43]:

```
matriz2
```

converte a lista em matriz

Out[43]:

```
matrix([[13, 81, 22],
        [ 0, 34, 59],
        [21, 48, 94]])
```

In [44]:

```
type(matriz2)
```

Out[44]:

```
numpy.matrixlib.defmatrix.matrix
```

In [45]:

```
# Formato da matriz
np.shape(matriz2)
```

Out[45]:

```
(3, 3)
```

3 linhas
3 Colunas

In [46]:

```
matriz2.size
```

Out[46]:

9 = 3 x 3 = 9

In [47]:

```
print(matriz2.dtype)
```

```
int64
```

In [48]:

```
matriz2.itemsize
```

Out[48]:

```
8
```

In [49]:

```
matriz2.nbytes
```

Out[49]:

```
72
```

= 9 x 8 = 72

In [50]:

```
print(matriz2[2,1])
```

linha 2, coluna 1

slice 0 1 2

```
48
```

In [51]:

```
# Alterando um elemento da matriz
matriz2[1,0] = 100
```

In [52]:

```
matriz2
```

Out[52]:

```
matrix([[ 13,  81,  22],
        [100,  34,  59],
        [ 21,  48,  94]])
```

0 1 2

In [53]:

```
x = np.array([1, 2])  # NumPy decide o tipo dos dados
y = np.array([1.0, 2.0])  # NumPy decide o tipo dos dados
z = np.array([1, 2], dtype=np.float64)  # Forçamos um tipo de dado em particular

print (x.dtype, y.dtype, z.dtype)
```

```
int64 float64 float64
```

In [54]:

```
matriz3 = np.array([[24, 76], [35, 89]], dtype=float)
```

In [55]:
```
matriz3
```

Out[55]:
```
array([[24., 76.],
       [35., 89.]])
```

In [56]:
```
matriz3.itemsize
```

Out[56]:
```
8
```

In [57]:
```
matriz3.nbytes
```

Out[57]:
```
32
```

In [58]:
```
matriz3.ndim
```

Out[58]:
```
2
```

In [59]:
```
matriz3[1,1]
```

Out[59]:
```
89.0
```

In [60]:
```
matriz3[1,1] = 100
```

In [61]:
```
matriz3
```

Out[61]:
```
array([[ 24.,  76.],
       [ 35., 100.]])
```

Usando o Método random() do NumPy

In [62]:

```
print(np.random.rand(10))
```

```
[0.32876755 0.8059542  0.38401033 0.96814204 0.242465   0.30801586
 0.26278155 0.92680311 0.10689803 0.62445356]
```

In [63]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

Possibilita mostrar gráficos usando o Navegador

In [64]:

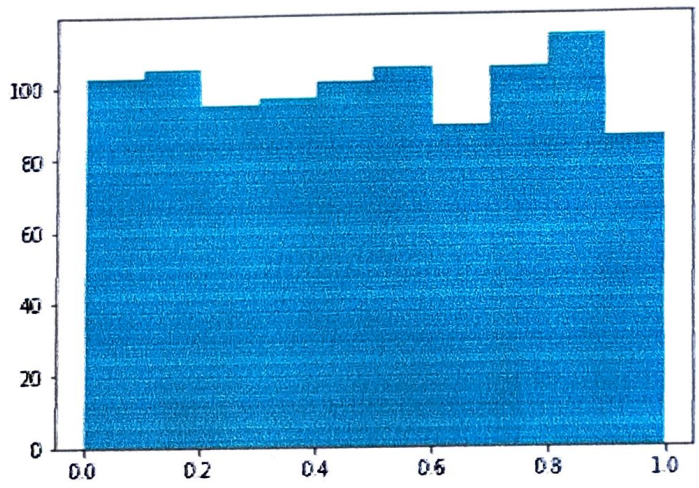
```
print(np.random.rand(10))
```

```
[0.83974807 0.55953056 0.70762925 0.96989638 0.20978129 0.4878879
 0.47990909 0.07467143 0.3638612  0.44397199]
```

In [65]:

NumPy

```
plt.show((plt.hist(np.random.rand(1000))))
```



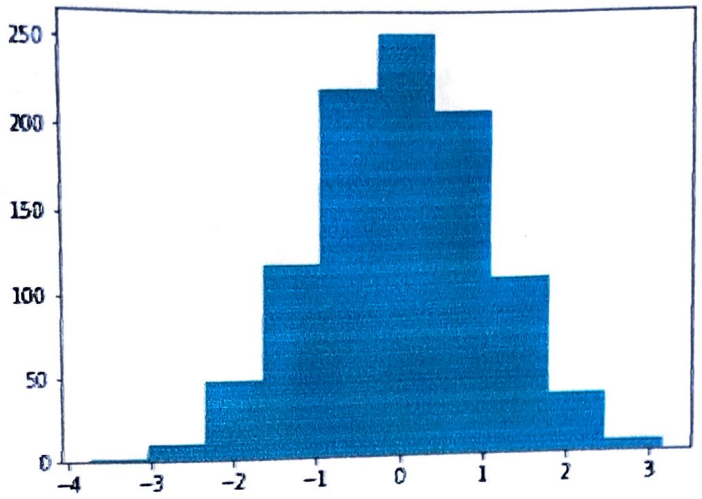
In [66]:

```
print(np.random.randn(5,5))
```

```
[[-0.22911957 -0.82305328  2.00337749  0.41001026 -0.82809627]
 [ 2.33185302 -0.13344758 -0.56499447 -0.36810252  0.3048597 ]
 [ 0.54069696  0.52974186  0.75901069  0.31220585  1.82403205]
 [ 0.30703907  0.00894393  0.46768762  1.13357679 -0.01140446]
 [-0.26418087 -0.61113705 -0.12890487 -0.47011605 -1.36438349]]
```

In [67]:

```
plt.show(plt.hist(np.random.randn(1000)))
```



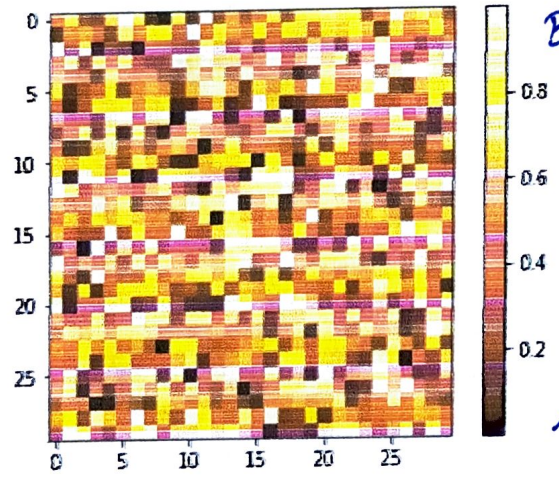
In [68]:

```
imagem = np.random.rand(30, 30)
plt.imshow(imagem, cmap = plt.cm.hot)
plt.colorbar()
```

Out[68]:

```
<matplotlib.colorbar.Colorbar at 0x110540b00>
```

Grafico de Calor
Hit Map



Baixa concentração

Alta concentração

Operações com datasets

In [69]:

```
import os
filename = os.path.join('iris.csv')
```

In [70]:

```
# No Windows use !more iris.csv. Mac ou Linux use !head iris.csv
!head iris.csv
#!more iris.csv
```

```
sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,setosa
4.9,3,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
5,3.6,1.4,0.2,setosa
5.4,3.9,1.7,0.4,setosa
4.6,3.4,1.4,0.3,setosa
5,3.4,1.5,0.2,setosa
4.4,2.9,1.4,0.2,setosa
```

In [71]:

```
# Carregando um dataset para dentro de um array
arquivo = np.loadtxt(filename, delimiter=',', usecols=(0,1,2,3), skiprows=1)
print (arquivo)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
```

In [72]:

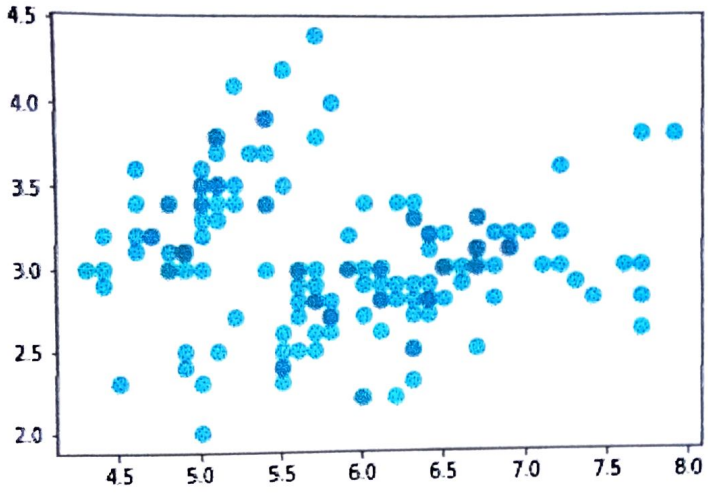
```
type(arquivo)
```

Out[72]:

numpy.ndarray

In [73]:

```
# Gerando um plot a partir de um arquivo usando o NumPy
var1, var2 = np.loadtxt(filename, delimiter=',', usecols=(0,1), skiprows=1, unpack=True)
plt.show(plt.plot(var1, var2, 'o', markersize=8, alpha=0.75))
```



Estatística

In [74]:

```
# Criando um array
A = np.array([15, 23, 63, 94, 75])
```

In [75]:

```
# Em estatística a média é o valor que aponta para onde mais se concentram os dados de uma
np.mean(A)
```

Out[75]:

```
54.0
```

In [76]:

```
# O desvio padrão mostra o quanto de variação ou "dispersão" existe em
# relação à média (ou valor esperado).
# Um baixo desvio padrão indica que os dados tendem a estar próximos da média.
# Um desvio padrão alto indica que os dados estão espalhados por uma gama de valores.
np.std(A)
```

Out[76]:

```
30.34468652004828
```

In [77]:

```
# Variância de uma variável aleatória é uma medida da sua dispersão
# estatística, indicando "o quão longe" em geral os seus valores se
# encontram do valor esperado
np.var(A)
```

Out[77]:

```
920.8
```

In [78]:

```
d = np.arange(1, 10)
```

In [79]:

```
d
```

Out[79]:

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

In [80]:

```
np.sum(d)
```

Soma

Out[80]:

```
45
```

In [81]:

```
# Retorna o produto dos elementos
np.prod(d)
```

multiplicação de todos os elementos

Out[81]:

```
362880
```

In [82]:

```
# Soma acumulada dos elementos
np.cumsum(d)
```

Out[82]:

```
array([ 1,  3,  6, 10, 15, 21, 28, 36, 45])
```

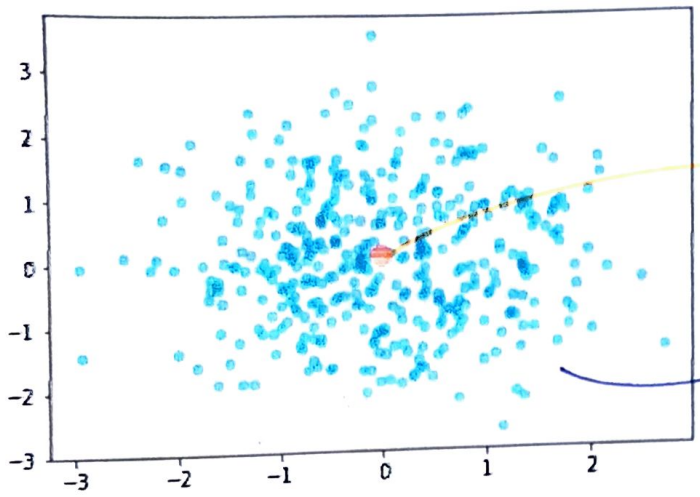
In [83]:

```
a = np.random.randn(400,2)
m = a.mean(0)
print (m, m.shape)
```

```
[-0.05316976  0.05460669] (2,)
```

In [84]:

```
plt.plot(a[:,0], a[:,1], 'o', markersize=5, alpha=0.50)
plt.plot(m[0], m[1], 'ro', markersize=10)
plt.show()
```



média

Valores dispersos

Outras Operações com Arrays

In [85]:

```
# Slicing
a = np.diag(np.arange(3))
```

In [86]:

```
a
```

Out[86]:

```
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
```

In [87]:

```
a[1, 1]
```

Out[87]:

```
1
```

In [88]:

```
a[1]
```

linha

Out[88]:

```
array([0, 1, 0])
```

In [89]:

```
b = np.arange(10)
```

In [90]:

```
b
```

Out[90]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

começa em 2
Vai até 9
de 3 em 3

Muito útil

In [91]:

```
# [start:end:step]
b[2:9:3]
```

Out[91]:

```
array([2, 5, 8])
```

In [92]:

```
# Comparação
a = np.array([1, 2, 3, 4])
b = np.array([4, 2, 2, 4])
a == b
```

Out[92]:

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

Verifica se os Arrays são iguais

In [93]:

```
np.array_equal(a, b)
```

Out[93]:

```
False
```

In [94]:

```
a.min()
```

Out[94]:

```
1
```

In [95]:

```
a.max()
```

Out[95]:

```
4
```

In [96]:

```
# Somando um elemento ao array
np.array([1, 2, 3]) + 1.5
```

Out[96]:

```
array([2.5, 3.5, 4.5])
```

Somo 1,5 com cada item

In [97]:

```
# Usando o método around
a = np.array([1.2, 1.5, 1.6, 2.5, 3.5, 4.5])
```

In [98]:

```
b = np.around(a)
```

In [99]:

```
b
```

Out[99]:

```
array([1., 2., 2., 2., 4., 4.])
```

In [100]:

```
# Criando um array
B = np.array([1, 2, 3, 4])
```

In [101]:

```
B
```

Out[101]:

```
array([1, 2, 3, 4])
```

cópia

In [102]:

```
# Copiando um array
C = B.flatten()
```

In [103]:

```
C
```

Out[103]:

```
array([1, 2, 3, 4])
```

In [104]:

```
# Criando um array
v = np.array([1, 2, 3])
```

In [105]:

```
# Adcionando uma dimensão ao array
v[:, np.newaxis], v[:,np.newaxis].shape, v[np.newaxis,:].shape
```

Out[105]:

```
(array([[1],
        [2],
        [3]]), (3, 1), (1, 3))
```

In [106]:

```
# Repetindo os elementos de um array
np.repeat(v, 3)
```

Out[106]:

```
array([1, 1, 1, 2, 2, 2, 3, 3, 3])
```

Repete o 1: 3 vezes

In [107]:

```
# Repetindo os elementos de um array
np.tile(v, 3)
```

Out[107]:

```
array([1, 2, 3, 1, 2, 3, 1, 2, 3])
```

Repete cada 1 por vez

In [108]:

```
# Criando um array
w = np.array([5, 6])
```

In [109]:

```
# Concatenando
np.concatenate((v, w), axis=0)
```

[1,2,3] [5,6]

Out[109]:

```
array([1, 2, 3, 5, 6])
```

In [110]:

```
# Copiando arrays
r = np.copy(v)
```

In [111]:

```
r
```

Out[111]:

```
array([1, 2, 3])
```

mais cobiçados do mercado de análise de dados. Clique no link abaixo, faça sua inscrição, comece hoje mesmo e aumente sua empregabilidade:

https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados
(https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados)

Fim

Obrigado - Data Science Academy - facebook.com/dsacademybr (http://facebook.com/dsacademybr)

Data Science Academy - Python Fundamentos - Capítulo 8

Download: http://github.com/dsacademybr (http://github.com/dsacademybr)

Pandas = Python Data Analysis Library

Para poder trabalhar com Pandas, você deve conhecer bem estas duas estruturas: Series e DataFrame.

Para importar o pandas, utilize: import pandas as pd

Você também pode utilizar: from pandas import Series, DataFrame

Para atualizar o Pandas, abra o prompt de comando ou terminal e digite: pip install pandas -U

Series

Series é um array unidimensional que contém um array de dados e um array de labels, chamado índice.

In [1]:
```
from pandas import Series
```
→ importa apenas o pacote Series

In [2]:
```
import pandas as pd
```
→ importa todo o Pandas

In [3]:
```
pd.__version__
```

Out[3]:
```
'0.22.0'
```

In [4]:
```
# Criando uma série sem especificar os índices
Obj = Series([67, 78, -56, 13])
```

In [5]:
```
Obj
```

Out[5]:
```
0    67
1    78
2   -56
3    13
dtype: int64
```

In [6]:
```
type(Obj)
```

Out[6]:
```
pandas.core.series.Series
```

In [7]:
```
Obj.values
```

Out[7]:
```
array([ 67,  78, -56,  13])
```

In [8]:
```
Obj.index
```

Out[8]:
```
RangeIndex(start=0, stop=4, step=1)
```

início → o 4 é exclusivo ou seja ele vai até o 3 e para
↳ de 1 em 1

In [9]:
```
# Criando uma série e especificando os índices
Obj2 = Series([67, 78, -56, 13], index = ['a', 'b', 'c', 'd'])
```

In [10]:
```
Obj2
```

Out[10]:
```
a    67
b    78
c   -56
d    13
dtype: int64
```

In [11]:
```
Obj2.values
```

Out[11]:
```
array([ 67,  78, -56,  13])
```

In [12]:
```
Obj2.index
```

Out[12]:
```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

In [13]:
```
Obj2[Obj2 > 3]
```

Legal

Out[13]:
```
a    67
b    78
d    13
dtype: int64
```

→ mostra os valores dentro do Obj2 que são maiores que 3

In [14]:
```
Obj2['b']
```

Out[14]:
```
78
```

In [15]:
```
'd' in Obj2
```

Verifica se 'd' está em Obj2

Out[15]:
```
True
```

In [16]:
```
# Criando uma série de dados passando um dicionário como parâmetro
dict = {'Futebol':5200, 'Tenis': 120, 'Natação':698, 'Volleyball':1550}
```

In [17]:
```
# Criando uma série a partir de um dicionário
Obj3 = Series(dict)
```

In [18]:

```
Obj3
```

Out[18]:

```
Futebol      5200
Natação       698
Tenis         120
Volleyball   1550
dtype: int64
```

In [19]:

```
type(Obj3)
```

Out[19]:

```
pandas.core.series.Series
```

In [20]:

```
# Criando uma lista
esportes = ['Futebol', 'Tenis', 'Natação', 'Basktetball']
```

In [21]:

```
# Criando uma serie e usando uma lista como índice
Obj4 = Series(dict, index=esportes)
```

In [22]:

```
Obj4
```

Out[22]:

```
Futebol        5200.0
Tenis           120.0
Natação         698.0
Basktetball       NaN
dtype: float64
```

Foi casado com os índices de esporte e Obj3

Não encontrou casado com Basktetball

In [23]:

```
pd.isnull(Obj4)
```

Out[23]:

```
Futebol       False
Tenis         False
Natação       False
Basktetball    True
dtype: bool
```

In [24]:

```
pd.notnull(Obj4)
```

Não Nulo => método Pandas

Out[24]:

```
Futebol       True
Tenis         True
Natação       True
Basktetball  False
dtype: bool
```

In [25]:

```
Obj4.isnull()
```

método desse objeto

Out[25]:

```
Futebol       False
Tenis         False
Natação       False
Basktetball    True
dtype: bool
```

In [26]:

```
# Concatenando Series
Obj3 + Obj4
```

Out[26]:

```
Basktetball       NaN
Futebol       10400.0
Natação        1396.0
Tenis           240.0
Volleyball        NaN
dtype: float64
```

Se ele encontra relação entre os índices ele Soma os Valores
Se não coloca NaN

In [27]:

```
Obj4.name = 'população'
```

In [28]:

```
Obj4.index.name = 'esporte'
```

In [29]:

```
Obj4
```

Out[29]:

```
esporte
Futebol       5200.0
Tenis          120.0
Natação        698.0
Basktetball      NaN
Name: população, dtype: float64
```

Dataframes

Dataframes representam uma estrutura tabular semelhante a estrutura de uma planilha do Excel, contendo uma coleção de colunas em que cada uma pode ser um diferente tipo de valor (número, string, etc...). Os Dataframes possuem index e linhas e esta estrutura é muito semelhante a um dataframe em R. Os dados de um dataframe são armazenados e um ou mais blocos bidimensionais, ao invés de listas, dicionários ou alguma outra estrutura de array.

In [30]:

```
from pandas import DataFrame
```

In [31]:

```
data = {'Estado': ['Santa Catarina', 'Paraná', 'Goiás', 'Bahia', 'Minas Gerais'],
        'Ano': [2002, 2003, 2004, 2005, 2006],
        'População': [1.5, 1.7, 3.6, 2.4, 2.9]}
```

In [32]:

```
frame = DataFrame(data)
```

In [33]:

```
frame
```

Out[33]:

	Ano	Estado	População
0	2002	Santa Catarina	1.5
1	2003	Paraná	1.7
2	2004	Goiás	3.6
3	2005	Bahia	2.4
4	2006	Minas Gerais	2.9

In [34]:

```
type(frame)
```

Out[34]:

```
pandas.core.frame.DataFrame
```

In [35]:

```
DataFrame(data, columns=['Ano', 'Estado', 'População'])
```

Out[35]:

	Ano	Estado	População
0	2002	Santa Catarina	1.5
1	2003	Paraná	1.7
2	2004	Goiás	3.6
3	2005	Bahia	2.4
4	2006	Minas Gerais	2.9

Dicionário Criado Podendo vir de um arquivo excel Numpy CSV

In [36]:

```
# Criando outro dataframe com os mesmo dados anteriores mas adicionando uma coluna
frame2 = DataFrame(data, columns = ['Ano', 'Estado', 'População', 'Débito'],
                   index = ['um', 'dois', 'três', 'quatro', 'cinco'])
```

In [37]:

```
# Imprimindo o Dataframe
frame2
```

Out[37]:

	Ano	Estado	População	Débito
um	2002	Santa Catarina	1.5	NaN
dois	2003	Paraná	1.7	NaN
três	2004	Goiás	3.6	NaN
quatro	2005	Bahia	2.4	NaN
cinco	2006	Minas Gerais	2.9	NaN

Débito não constava no dicionário

In [38]:

```
# Imprimindo apenas uma coluna do Dataframe
frame2['Estado']
```

Out[38]:

```
um          Santa Catarina
dois                Paraná
três                 Goiás
quatro               Bahia
cinco        Minas Gerais
Name: Estado, dtype: object
```

In [39]:

```
type(frame2)
```

Out[39]:

```
pandas.core.frame.DataFrame
```

In [40]:

```
frame2.index
```

Out[40]:

```
Index(['um', 'dois', 'três', 'quatro', 'cinco'], dtype='object')
```

In [41]:

```
frame2.columns
```

Out[41]:

```
Index(['Ano', 'Estado', 'População', 'Débito'], dtype='object')
```

In [42]:

```
frame2.values
```

Out[42]:

```
array([[2002, 'Santa Catarina', 1.5, nan],
       [2003, 'Paraná', 1.7, nan],
       [2004, 'Goiás', 3.6, nan],
       [2005, 'Bahia', 2.4, nan],
       [2006, 'Minas Gerais', 2.9, nan]], dtype=object)
```

In [43]:

```
frame2.dtypes
```

Out[43]:

```
Ano            int64
Estado        object
População    float64
Débito        object
dtype: object
```

In [44]:

```
frame2['Ano']
```

Out[44]:

```
um        2002
dois      2003
três      2004
quatro    2005
cinco     2006
Name: Ano, dtype: int64
```

In [45]:

```
frame2.Ano
```

mesmo resultado

Out[45]:

```
um        2002
dois      2003
três      2004
quatro    2005
cinco     2006
Name: Ano, dtype: int64
```

In [46]:

```
frame2[:2]
```

→ todas as colunas – 2 linhas

Out[46]:

		Ano	Estado	População	Débito
0	um	2002	Santa Catarina	1.5	NaN
1	dois	2003	Paraná	1.7	NaN

Usando NumPy e Pandas

In [47]:

```
# Importando o NumPy
import numpy as np
```

In [48]:

```
# Usando o NumPy para alimentar uma das colunas do dataframe
frame2['Débito'] = np.arange(5.)
```

exclusiva

Varia de 0 a 4

In [49]:

```
frame2
```

Out[49]:

	Ano	Estado	População	Débito
um	2002	Santa Catarina	1.5	0.0
dois	2003	Paraná	1.7	1.0
três	2004	Goiás	3.6	2.0
quatro	2005	Bahia	2.4	3.0
cinco	2006	Minas Gerais	2.9	4.0

In [50]:

```
frame2.values
```

Out[50]:

```
array([[2002, 'Santa Catarina', 1.5, 0.0],
       [2003, 'Paraná', 1.7, 1.0],
       [2004, 'Goiás', 3.6, 2.0],
       [2005, 'Bahia', 2.4, 3.0],
       [2006, 'Minas Gerais', 2.9, 4.0]], dtype=object)
```

In [51]:

```
# Resumo do Dataframe
frame2.describe()
```

→ Gera um relatório estatístico

Out[51]:

	Ano	População	Débito
count	5.000000	5.000000	5.000000
mean	2004.000000	2.420000	2.000000
std	1.581139	0.864292	1.581139
min	2002.000000	1.500000	0.000000
25%	2003.000000	1.700000	1.000000
50%	2004.000000	2.400000	2.000000
75%	2005.000000	2.900000	3.000000
max	2006.000000	3.600000	4.000000

In [52]:

```
frame2['dois':'quatro']
```

Out[52]:

	Ano	Estado	População	Débito
dois	2003	Paraná	1.7	1.0
três	2004	Goiás	3.6	2.0
quatro	2005	Bahia	2.4	3.0

→ o de dois até quatro

In [53]:

```
frame2 < 5
```

Out[53]:

	Ano	Estado	População	Débito	
um	False	True	True	True	0
dois	False	True	True	True	1
três	False	True	True	True	2
quatro	False	True	True	True	3
cinco	False	True	True	True	4

índice

Localizando Registros Dentro do Dataframe

In [54]:

```
frame2.loc['quatro']
```

localização

Out[54]:

```
Ano           2005
Estado       Bahia
População      2.4
Débito           3
Name: quatro, dtype: object
```

In [55]:

```
frame2.iloc[2]
```

→ index location

Out[55]:

```
Ano           2004
Estado       Goiás
População      3.6
Débito           2
Name: três, dtype: object
```

Invertendo as Colunas e Índices

In [56]:

```
# Criando um dicionário
web_stats = {'Dias':[1, 2, 3, 4, 5, 6, 7],
             'Visitantes':[45, 23, 67, 78, 23, 12, 14],
             'Taxas':[11, 22, 33, 44, 55, 66, 77]}
```

In [57]:

```
df = pd.DataFrame(web_stats)
```

In [58]:

```
print(df)
```

```
   Dias  Taxas  Visitantes
0     1     11          45
1     2     22          23
2     3     33          67
3     4     44          78
4     5     55          23
5     6     66          12
6     7     77          14
```

In [59]:

```
# Visualizando uma coluna  index
print(df.set_index('Dias'))
```

```
      Taxas  Visitantes
Dias
1        11          45
2        22          23
3        33          67
4        44          78
5        55          23
6        66          12
7        77          14
```

! Determina a coluna Dias como index

Não modifica

In [60]:

```
print(df.head())
```

```
   Dias  Taxas  Visitantes
0     1     11          45
1     2     22          23
2     3     33          67
3     4     44          78
4     5     55          23
```

Original

In [61]:

```
print(df['Visitantes'])
```

```
0    45
1    23
2    67
3    78
4    23
5    12
6    14
Name: Visitantes, dtype: int64
```

In [62]:

```
print(df[['Visitantes', 'Taxas']])
```

```
   Visitantes  Taxas
0          45     11
1          23     22
2          67     33
3          78     44
4          23     55
5          12     66
6          14     77
```

Dataframes e Arquivos csv

In [63]:

```
# Usando o método read_csv
df = pd.read_csv('salarios.csv')
```

→ considero que o separador é ',' Por vírgulas

In [64]:

```
df
```

Out[64]:

	Name	Position Title	Department	Employee Annual Salary
0	AARON, ELVIA J	WATER RATE TAKER	WATER MGMNT	$88968.00
1	AARON, JEFFERY M	POLICE OFFICER	POLICE	$80778.00
2	AARON, KARINA	POLICE OFFICER	POLICE	$80778.00
3	AARON, KIMBERLEI R	CHIEF CONTRACT EXPEDITER	GENERAL SERVICES	$84780.00
4	ABAD JR, VICENTE M	CIVIL ENGINEER IV	WATER MGMNT	$104736.00
5	ABARCA, ANABEL	ASST TO THE ALDERMAN	CITY COUNCIL	$70764.00
6	ABARCA, EMMANUEL	GENERAL LABORER - DSS	STREETS & SAN	$40560.00
7	ABBATACOLA, ROBERT J	ELECTRICAL MECHANIC	AVIATION	$91520.00
8	ABBATEMARCO, JAMES J	FIRE ENGINEER	FIRE	$90456.00
9	ABBATE, TERRY M	POLICE OFFICER	POLICE	$86520.00
10	ABBOTT, BETTY L	FOSTER GRANDPARENT	FAMILY & SUPPORT	$2756.00
11	ABBOTT, LYNISE M	CLERK III	POLICE	$43920.00
12	ABBRUZZESE, WILLIAM J	INVESTIGATOR - IPRA II	IPRA	$72468.00
13	ABDALLAH, ZAID	POLICE OFFICER	POLICE	$69684.00
14	ABDELHADI, ABDALMAHD	POLICE OFFICER	POLICE	$80778.00
15	ABDELLATIF, AREF R	FIREFIGHTER (PER ARBITRATORS AWARD)-PARAMEDIC	FIRE	$98244.00
16	ABDELMAJEID, AZIZ	POLICE OFFICER	POLICE	$80778.00
17	ABDOLLAHZADEH, ALI	FIREFIGHTER/PARAMEDIC	FIRE	$87720.00
18	ABDUL-KARIM, MUHAMMAD A	ENGINEERING TECHNICIAN VI	WATER MGMNT	$106104.00
19	ABDULLAH, DANIEL N	FIREFIGHTER-EMT	FIRE	$91764.00
20	ABDULLAH, KEVIN	LIEUTENANT	FIRE	$110370.00
21	ABDULLAH, LAKENYA N	CROSSING GUARD	POLICE	$16692.00
22	ABDULLAH, RASHAD J	ELECTRICAL MECHANIC-AUTO-POLICE MTR MNT	GENERAL SERVICES	$91520.00

	Name	Position Title	Department	Employee Annual Salary
32171	ZWIESLER, MATTHEW	AIRPORT OPERATIONS SUPVSR I	AVIATION	$69840.00
32172	ZWIT, JEFFREY J	POLICE OFFICER	POLICE	$83616.00
32173	ZWOLFER, MATTHEW W	FIREFIGHTER-EMT	FIRE	$91764.00
32174	ZYDEK, BRYAN	POLICE OFFICER	POLICE	$80778.00
32175	ZYGADLO, JOHN P	MACHINIST (AUTOMOTIVE)	GENERAL SERVICES	$92248.00
32176	ZYGADLO, MICHAEL J	FRM OF MACHINISTS - AUTOMOTIVE	GENERAL SERVICES	$97448.00
32177	ZYGOWICZ, PETER J	POLICE OFFICER	POLICE	$86520.00
32178	ZYMANTAS, MARK E	POLICE OFFICER	POLICE	$83616.00
32179	ZYRKOWSKI, CARLO E	POLICE OFFICER	POLICE	$86520.00
32180	ZYSKOWSKI, DARIUSZ	CHIEF DATA BASE ANALYST	DoIT	$110352.00
32181	NaN	NaN	NaN	NaN

32182 rows × 4 columns

In [65]:

```
# Usando o método read_table
df = pd.read_table('salarios.csv', sep = ',')
```

	Name	Position Title	Department	Employee Annual Salary
32171	ZWIESLER, MATTHEW	AIRPORT OPERATIONS SUPVSR I	AVIATION	$69840.00
32172	ZWIT, JEFFREY J	POLICE OFFICER	POLICE	$83616.00
32173	ZWOLFER, MATTHEW W	FIREFIGHTER-EMT	FIRE	$91764.00
32174	ZYDEK, BRYAN	POLICE OFFICER	POLICE	$80778.00
32175	ZYGADLO, JOHN P	MACHINIST (AUTOMOTIVE)	GENERAL SERVICES	$92248.00
32176	ZYGADLO, MICHAEL J	FRM OF MACHINISTS - AUTOMOTIVE	GENERAL SERVICES	$97448.00
32177	ZYGOWICZ, PETER J	POLICE OFFICER	POLICE	$86520.00
32178	ZYMANTAS, MARK E	POLICE OFFICER	POLICE	$83616.00
32179	ZYRKOWSKI, CARLO E	POLICE OFFICER	POLICE	$86520.00
32180	ZYSKOWSKI, DARIUSZ	CHIEF DATA BASE ANALYST	DoIT	$110352.00
32181	NaN	NaN	NaN	NaN

32182 rows × 4 columns

In [67]:

```
# No Windows use !type. No Mac ou Linux use !head
!head salarios.csv
# !type salarios.csv
```

```
Name,Position Title,Department,Employee Annual Salary
"AARON,  ELVIA J",WATER RATE TAKER,WATER MGMNT,$88968.00
"AARON,  JEFFERY M",POLICE OFFICER,POLICE,$80778.00
"AARON,  KARINA",POLICE OFFICER,POLICE,$80778.00
"AARON,  KIMBERLEI R",CHIEF CONTRACT EXPEDITER,GENERAL SERVICES,$84780.00
"ABAD JR,  VICENTE M",CIVIL ENGINEER IV,WATER MGMNT,$104736.00
"ABARCA,  ANABEL",ASST TO THE ALDERMAN,CITY COUNCIL,$70764.00
"ABARCA,  EMMANUEL",GENERAL LABORER - DSS,STREETS & SAN,$40560.00
"ABBATACOLA,  ROBERT J",ELECTRICAL MECHANIC,AVIATION,$91520.00
"ABBATEMARCO,  JAMES J",FIRE ENGINEER,FIRE,$90456.00
```

In [68]:

```
# Alterando o título das colunas
df = pd.read_csv('salarios.csv', names = ['a', 'b', 'c', 'd'])
```

Se quisen alterar a coluna (títulos)

	a	b	c	d
32173	ZWIT, JEFFREY J	POLICE OFFICER	POLICE	$83616.00
32174	ZWOLFER, MATTHEW W	FIREFIGHTER-EMT	FIRE	$91764.00
32175	ZYDEK, BRYAN	POLICE OFFICER	POLICE	$80778.00
32176	ZYGADLO, JOHN P	MACHINIST (AUTOMOTIVE)	GENERAL SERVICES	$92248.00
32177	ZYGADLO, MICHAEL J	FRM OF MACHINISTS - AUTOMOTIVE	GENERAL SERVICES	$97448.00
32178	ZYGOWICZ, PETER J	POLICE OFFICER	POLICE	$86520.00
32179	ZYMANTAS, MARK E	POLICE OFFICER	POLICE	$83616.00
32180	ZYRKOWSKI, CARLO E	POLICE OFFICER	POLICE	$86520.00
32181	ZYSKOWSKI, DARIUSZ	CHIEF DATA BASE ANALYST	DoIT	$110352.00
32182	NaN	NaN	NaN	NaN

32183 rows × 4 columns

In [70]:

```
import sys
```

In [71]:

```
data = pd.read_csv('salarios.csv')
```

In [72]:

```
data.to_csv(sys.stdout, sep = '|')
```

```
|Name|Position Title|Department|Employee Annual Salary
0|AARON,  ELVIA J|WATER RATE TAKER|WATER MGMNT|$88968.00
1|AARON,  JEFFERY M|POLICE OFFICER|POLICE|$80778.00
2|AARON,  KARINA|POLICE OFFICER|POLICE|$80778.00
3|AARON,  KIMBERLEI R|CHIEF CONTRACT EXPEDITER|GENERAL SERVICES|$84780.00
4|ABAD JR,  VICENTE M|CIVIL ENGINEER IV|WATER MGMNT|$104736.00
5|ABARCA,  ANABEL|ASST TO THE ALDERMAN|CITY COUNCIL|$70764.00
6|ABARCA,  EMMANUEL|GENERAL LABORER - DSS|STREETS & SAN|$40560.00
7|ABBATACOLA,  ROBERT J|ELECTRICAL MECHANIC|AVIATION|$91520.00
8|ABBATEMARCO,  JAMES J|FIRE ENGINEER|FIRE|$90456.00
9|ABBATE,  TERRY M|POLICE OFFICER|POLICE|$86520.00
10|ABBOTT,  BETTY L|FOSTER GRANDPARENT|FAMILY & SUPPORT|$2756.00
11|ABBOTT,  LYNISE M|CLERK III|POLICE|$43920.00
12|ABBRUZZESE,  WILLIAM J|INVESTIGATOR - IPRA II|IPRA|$72468.00
13|ABDALLAH,  ZAID|POLICE OFFICER|POLICE|$69684.00
14|ABDELHADI,  ABDALMAHD|POLICE OFFICER|POLICE|$80778.00
15|ABDELLATIF,  AREF R|FIREFIGHTER (PER ARBITRATORS AWARD)-PARAMEDIC|FIRE|
$98244.00
16|ABDELMAJEID,  AZIZ|POLICE OFFICER|POLICE|$80778.00
```

In [73]:

```
# Criando um Dataframe
dates = pd.date_range('20180101', periods = 10)
df = pd.DataFrame(np.random.randn(10,4), index = dates, columns = list('ABCD'))
```

cria lista de datas com dez dias

cria lista 'A', 'B', 'C', 'D'

Matriz 10 por 4

In [74]:

```
df
```

Out[74]:

	A	B	C	D
2018-01-01	1.206918	1.888850	0.734489	-0.561248
2018-01-02	-0.384682	-0.197144	-0.525458	-0.098813
2018-01-03	0.568625	2.902776	1.798679	-0.171690
2018-01-04	-0.130032	0.979133	0.008807	-1.996278
2018-01-05	-0.730827	-0.323303	0.529557	0.044718
2018-01-06	0.282634	-1.239240	0.124268	-0.208659
2018-01-07	0.165682	0.269816	1.464278	0.575116
2018-01-08	-0.858474	-0.001004	0.767940	-0.108990
2018-01-09	-0.566800	0.258109	0.967017	1.050102
2018-01-10	-0.574292	0.663247	0.738917	0.100237

In [75]:

```
df.head()
```

Out[75]:

	A	B	C	D
2018-01-01	1.206918	1.888850	0.734489	-0.561248
2018-01-02	-0.384682	-0.197144	-0.525458	-0.098813
2018-01-03	0.568625	2.902776	1.798679	-0.171690
2018-01-04	-0.130032	0.979133	0.008807	-1.996278
2018-01-05	-0.730827	-0.323303	0.529557	0.044718

In [76]:

```
# quick data summary
df.describe()
```

Out[76]:

	A	B	C	D
count	10.000000	10.000000	10.000000	10.000000
mean	-0.102125	0.520124	0.660849	-0.137551
std	0.655750	1.178780	0.682684	0.792402
min	-0.858474	-1.239240	-0.525458	-1.996278
25%	-0.572419	-0.148109	0.225590	-0.199417
50%	-0.257357	0.263962	0.736703	-0.103902
75%	0.253396	0.900162	0.917247	0.086357
max	1.206918	2.902776	1.798679	1.050102

In [77]:

```
# Calculando a média
df.mean()
```

Out[77]:

```
A   -0.102125
B    0.520124
C    0.660849
D   -0.137551
dtype: float64
```

calculo média por coluna

In [78]:

```
# Pivot e cálculo da média
df.mean(1)
```

Out[78]:

```
2018-01-01    0.817252
2018-01-02   -0.301524
2018-01-03    1.274597
2018-01-04   -0.284592
2018-01-05   -0.119964
2018-01-06   -0.260249
2018-01-07    0.618723
2018-01-08   -0.050132
2018-01-09    0.427107
2018-01-10    0.232027
Freq: D, dtype: float64
```

—> Pivo = ~~calcular~~ trocar linha por coluna

calculo a média por linha

In [79]:

```
# Usando métodos
df.apply(np.cumsum)
```

Posso criar minha própria função. Soma acumulada função do Numpy

Out[79]:

	A	B	C	D
2018-01-01	1.206918	1.888850	0.734489	-0.561248
2018-01-02	0.822236	1.691706	0.209030	-0.660061
2018-01-03	1.390861	4.594481	2.007709	-0.831751
2018-01-04	1.260828	5.573615	2.016516	-2.828030
2018-01-05	0.530001	5.250312	2.546073	-2.783312
2018-01-06	0.812635	4.011071	2.670341	-2.991971
2018-01-07	0.978317	4.280887	4.134619	-2.416855
2018-01-08	0.119843	4.279883	4.902559	-2.525845
2018-01-09	-0.446957	4.537992	5.869576	-1.475743
2018-01-10	-1.021249	5.201239	6.608493	-1.375507

In [80]:

```
# Merge de Dataframes
left = pd.DataFrame({'chave': ['chave1', 'chave2'], 'coluna1': [1, 2]})
right = pd.DataFrame({'chave': ['chave1', 'chave2'], 'coluna2': [4, 5]})
pd.merge(left, right, on='chave')
```

usa a chave p/ fazer a união

Out[80]:

	chave	coluna1	coluna2
0	chave1	1	4
1	chave2	2	5

In [81]:

```
# Adicionando um elemento ao Dataframe
df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
```

In [82]:

```
df
```

Out[82]:

	A	B	C	D
0	-1.331739	0.466996	1.055676	0.588182
1	1.186476	-0.901904	-0.735868	-0.345473
2	-0.112531	1.193238	-1.244090	0.938319
3	-0.442347	-0.311708	1.164455	0.601853
4	-0.640157	-0.008553	-1.798673	-0.406582
5	0.141861	1.804597	0.590099	-0.350188
6	-0.500411	0.345327	-2.071235	-1.171509
7	-0.009930	1.184951	0.847882	0.863164

Data Frame → Numpy → Matplotlib

Plotagem completa como se fosse excel

In [83]:

```
s = df.iloc[3]
```

localiza indice 3

In [84]:

```
# Adicionando um elemento ao Dataframe
df.append(s, ignore_index=True)
```

ignora o indice

Out[84]:

	A	B	C	D
0	-1.331739	0.466996	1.055676	0.588182
1	1.186476	-0.901904	-0.735868	-0.345473
2	-0.112531	1.193238	-1.244090	0.938319
3	-0.442347	-0.311708	1.164455	0.601853
4	-0.640157	-0.008553	-1.798673	-0.406582
5	0.141861	1.804597	0.590099	-0.350188
6	-0.500411	0.345327	-2.071235	-1.171509
7	-0.009930	1.184951	0.847882	0.863164
8	-0.442347	-0.311708	1.164455	0.601853

linha adicionada

Time Series

In [85]:

```
# Criando um range de datas com frequência de segundos
rng = pd.date_range('1/1/2018', periods = 50, freq = 'S')
```

seconds

início

50 segundos

In [86]:

```
ts = pd.Series(np.random.randint(0, 500, len(rng)), index = rng)
```

Série temporal

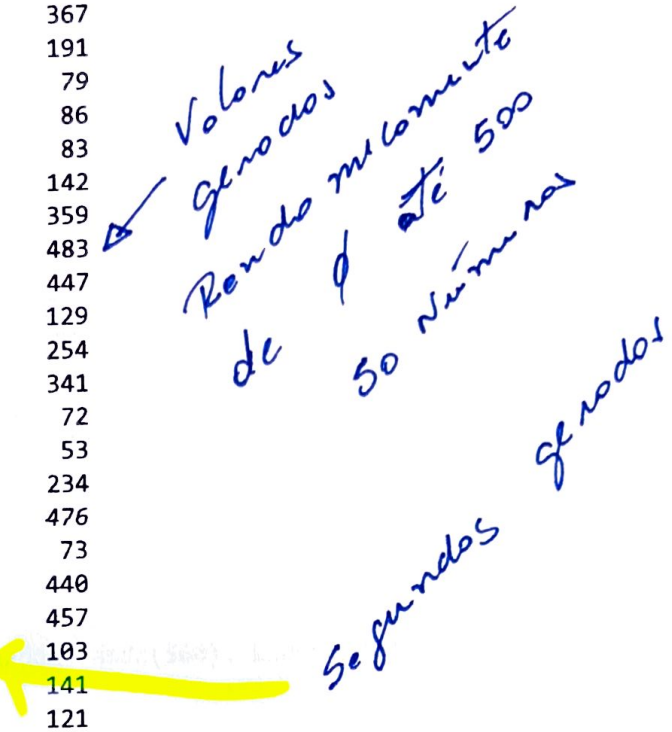
Valores até o limite rng

In [87]:

```
ts
```

Out[87]:

```
2018-01-01 00:00:00    385
2018-01-01 00:00:01    499
2018-01-01 00:00:02    263
2018-01-01 00:00:03    308
2018-01-01 00:00:04    367
2018-01-01 00:00:05    191
2018-01-01 00:00:06     79
2018-01-01 00:00:07     86
2018-01-01 00:00:08     83
2018-01-01 00:00:09    142
2018-01-01 00:00:10    359
2018-01-01 00:00:11    483
2018-01-01 00:00:12    447
2018-01-01 00:00:13    129
2018-01-01 00:00:14    254
2018-01-01 00:00:15    341
2018-01-01 00:00:16     72
2018-01-01 00:00:17     53
2018-01-01 00:00:18    234
2018-01-01 00:00:19    476
2018-01-01 00:00:20     73
2018-01-01 00:00:21    440
2018-01-01 00:00:22    457
2018-01-01 00:00:23    103
2018-01-01 00:00:24    141
2018-01-01 00:00:25    121
2018-01-01 00:00:26    280
2018-01-01 00:00:27    377
2018-01-01 00:00:28    413
2018-01-01 00:00:29     35
2018-01-01 00:00:30    479
2018-01-01 00:00:31    485
2018-01-01 00:00:32    148
2018-01-01 00:00:33    230
2018-01-01 00:00:34    276
2018-01-01 00:00:35    283
2018-01-01 00:00:36    453
2018-01-01 00:00:37    122
2018-01-01 00:00:38    176
2018-01-01 00:00:39    175
2018-01-01 00:00:40    449
2018-01-01 00:00:41    425
2018-01-01 00:00:42    236
2018-01-01 00:00:43    133
2018-01-01 00:00:44    271
2018-01-01 00:00:45    137
2018-01-01 00:00:46    419
2018-01-01 00:00:47    196
2018-01-01 00:00:48    135
2018-01-01 00:00:49    299
Freq: S, dtype: int64
```



In [88]:

```
# Criando um range de datas com frequência de meses
rng = pd.date_range('1/1/2016', periods=5, freq='M')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

Out[88]:

```
2016-01-31    2.667950
2016-02-29   -0.299170
2016-03-31    1.342942
2016-04-30    0.594104
2016-05-31    0.991905
Freq: M, dtype: float64
```

5 meses

Plotting

In [89]:

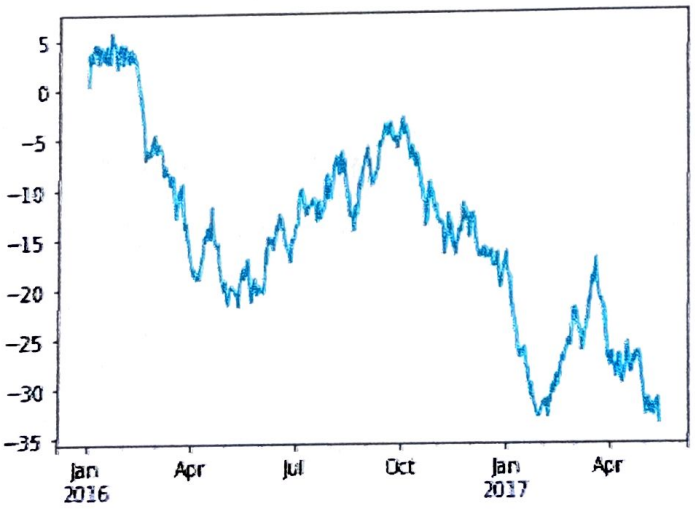
```
import matplotlib.pyplot as plt
from matplotlib import style
%matplotlib inline
```

In [90]:

```
# Time Series Plot
ts = pd.Series(np.random.randn(500), index=pd.date_range('1/1/2016', periods=500))
ts = ts.cumsum()
ts.plot()
```

Out[90]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10853ecf8>
```

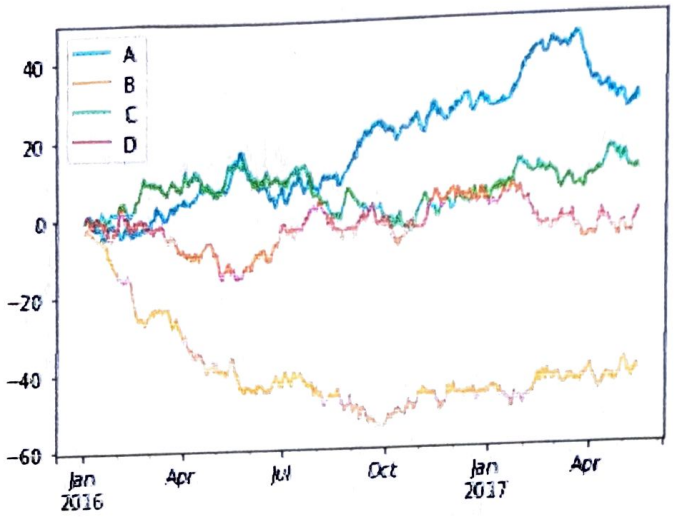


In [91]:

```
# DataFrame Plot
df = pd.DataFrame(np.random.randn(500, 4), index = ts.index, columns = ['A', 'B', 'C', 'D'])
df = df.cumsum()
plt.figure(); df.plot(); plt.legend(loc = 'best')
```

Out[91]:

```
<matplotlib.legend.Legend at 0x1085d7fd0>

<matplotlib.figure.Figure at 0x1085ed860>
```



Outuput

In [92]:

```
# Import
import os
```

In [93]:

```
# Verificando se o arquivo existe. No Windows use !type teste-df-output.xlsx
!head teste-df-output.xlsx
```

```
head: teste-df-output.xlsx: No such file or directory
```

In [94]:

```
# Gerando um arquivo excel a partir de um Dataframe
df.to_excel('teste-df-output.xlsx', sheet_name='Sheet1')
```

In [95]:

```
# Verificando se o arquivo existe. No Windows use !type teste-df-output.xlsx
!head teste-df-output.xlsx
```

```
QBW��s�└���|�������_└�?~�����?�������W���w_��└������
�����?�_��o ��Ɔ��  ��/��������?��������_�����_��+
=��  ��_�������������w��'~���/��'�O��/  ���  ��������
_□□┘└□_□□□□┘└+□□□┘└□└□□□┘┘└□□□.┘┘w◆<�����_��������?��w��
���┘��o�������└g����/���└�������7�������?��└└□□└□□
□  ����?����{□□_□
□□□]m~W}R□□}□□ᓚ□ϖ□□□□Ƭ□Y□□□□i□□□{OE□d□□qI□G□s□□jC□W□□5□
□□□gT>F□f□□□□_□□□i□ў└└a□□□i□□n□□5□□└□□□<□c□z□□□□q!□□/f
□□w□9N[□Ӣ□t□u□┐□□ ¯□9□□ ¯X□┐□□□(&□□□;□□□□:□¯YKZ□□□□@□□
_□□w□9N[□Ӣ□t□u□┐□□ ¯□9□□ ¯X□┐□□□(&□□□;□□□□:□¯YKZ□□□□@□□
ꚍ□PsM□ỹ□y~□□_[□□-5ꚍ□□R]O□┌□□a□aq3l{□□x□□□O□j□□|□□□□□S□□┐j
*9□□}□y□,N·□¯k1□□J□□e□□ᓰ^hj2+□Z?[□>Z□6~□wO_3□¯]Ų□=□g|□K□□Cp)□
□j□□Þ□□□>□□=□|□ov□□YS□>□□K+□□U□□Uy□j{s□□□\M□□}┐□O)□Yy□□
|□□u┘r_k□Ds]?┘w_|└□n=□5□□□c□D□E \□□g□□G□□f□□[□□zw)xY□□□Jt□
□y□i□k>□□|)M□□□□eN□└|□-EW□+i└□□k□□U□4□C□/□□?□V?_V7õEq□□a□e
□0f□~K-□au┌□□□k□□ƀɔau□□\s□UKK 0□□□U□]□□□□□□□□g□□G□□f□□
□□□□%];A)□□V□>□6a□m□5&k□□>□,□n□|Z□□i□f
□□□□□□□z□A□□y□□v□□□□i□□K<□□f□Hõc□□'□¯□o□O5□5□k□k□□□Se
□□□□□□□z□A□□y□□v□□□□i□□K<□□f□Hõc□□'□¯□o□O5□5□k□k□□□Se
□T□6S□□;□□□□$□□□□□□^rɬ□□?□□□□□□□□DY2K┌□~□n]□□eVS3□)₪/
□T□6S□□;□□□□$□□□□□□^rɬ□□?□□□□□□□□DY2K┌□~□n]□□eVS3□)₪/
>?□F/□No□ð□k_ų5.□□□b□9ǔ?□□Y□□M□□□□□4□□□S+□□□□□□y□¯□□□bo
e□n□□□□□3□□A□X□¯□5□□□Gfb□□□□□p□□□□□□o□d□¯□w□□y□□}□~Q□□□□□缄□.□□t
□□Nm□□}y□□R□ɹ□□¯□□l-k□"R'□□E□;□□;□Y□□M□□]□□2□ꞵ□¯;□□\□□n]□□I□
6□□┘└□[□D└/Zf)□□□□└□□□□□qU N[□^□ir~□)~□)m□)□N□□□□ hII<F□□□qE
_%4ie'iQ□Q□b□V¯ṂU□q.□□¯□]E'h□V= ¦□Y□□M□□K□□□□□%au□□od□h□□Ƶ□
□&□ᴅ□□z□q□□□□□N□□□□D)□□□□/□□,r□d`□□□Yz□□□□b□di□d□K□□□k
ȷ1^□(ᵽyM□8K_K□Ꞡ1□B□□USW
□=□Mc7□□C]Sa□"Q□F<|□□Y□□M□□]□□%F□ᗅ□J-□□□}□qY₹TT□nW□□□~*\□□
□S□□□{□M□□□[□Y₹□□ᗰ8□N□□Y□□M□□]□□□□□□M□VXP□ɹ□□U□□l□|□bt□3,%□□
k|o□□┘z□}□9P+□s□□,□□□I□□C□dS,N□RD)└□┘Z-□g7□´V┘┘F^□$ή0a)  □□/
□R□=□□□3□`□^□□}□□□K;□z□d  □d□d  q□|□□6□ƈt□□□□□□
Ł□Q┘□□□_□□o_Q4└t□x□'?□┘[^s`b_□o□W□□□□□,Y□&Y□□□□□r%□"□□□
□5□c□Ẋ□>D□[□□g>□□"□+□□□□□@^ai □*□□□□?(□□Y□□M□□]□□□ i□jB
□□w□□2ON,2CgY□□j~^□□ɔ□V□□v'Gn7n) □-.┌□47}8}□Y%l□□□%□□□□:□4□□
qs□
<G-□□□□S□7□9□□2□Vm□┘E□KS□H□i2H=□Ќi□Rh-ɣ□□Z□□p□□□□_□□,R□□□
□EJz□□□□d□□□□g?□8□n)5┘□'□>□Ww□Æ□□.└.5□C□□□□□□E□tx□D□□□O□
□LI□L□LI+□!□_:'y□□□U□C2N□|\□逍:k5X[□;□□□Y□□M□□]□□%└□%□YJj□g
□}k}□qQ□3□□□□3□"□ǵ$□)w┐¯□[1□□Y┐U□^□¯p┌□□□┐+q¯+}□+┐¯┌A□_>□6M-
□=□□□□d]┌□7!□.□□Yw□ꚍ□□□N□i□□□□□W□&□□^□D□!□□□q□□J□□'□
□q┐X□;K5□□□r□o□]□□□□┌□□:?□□2□□df□2□}□□□□┌□/□|□□i□?□□¯┌X
□44¯┌□□□□¯RW  s□□j□□□□h□□.L□□)J□□]□j3a□□□□|□^□┌A"□
□¯V<□□□□:s¦□□□□d^F□□d□□□'□□□"□  [□□□□k□□□□□□t¯□35□]
\□L□□□□W6□□□?□¯'□ش□v□¯□UN□N%3 4□□1□□8/騅□□S□&□p□Kr_□
vᵩ□I□□□t□□G¯┌1□┐□{NWY□J□y3nW)q□¯'□□kr□z□A□,□□□□□□□□'□5.
□j□□(□□J□□□-□□□mA□□vL□□/□d□g□¯□f□.Q□[,□□%,u:□=□□n□□q□U
b.Pe6W□□2□□y□□□:□$1□R┌□□□□B□┌;D□□g□┌□f□.Qb□□I□¯Lч□□□ □c□
□E{<□U□□□□□¯'78□ɔ□jq□□□ □yI□1Zm9□mj.□□I□Y□Ķ Y□k□□□v□□□□□□
˅]□□qLKM□□e□□o[□/□p□□D□□□j□□WHEr@□OK□□□z□□F□}□n□(□Q□e□□□e
□□o□□]□□W□□□H□5N□□□pg□□□□□□□W□R&□□□M□□cU□□X□,Q□□%JZ□
!□K□□2□□□,□□3□□i□□y□zV□□V□□%SP;□□□ʌ□□□G□>□□□□□□□□□T
□□◦□L□;┘-O)□□□H□□□$r□□□&□o|□}□g□□6□□□□□{z□□□□□9□!o□□ □□
□_YL□v┘□□f@□□S□qu9□l□□□|┘_\("□:h$r□□□□r%mre□□□□┘□Hz□!x□
(□┘□□6□ś t□□g$A□□□=□q□&□□C_\□□0_o□^□□□;□□VN□}□+i□+~□+□
□_└□□□e□□┘Q□Â□sG(□□_S□|□~_W□□□□□□F□v
□Y□□□K□*:-□□□□□□□i□:1□□□ZεZ8□=*{□ى6□□□^}□□ىYW□  □□□□7j
```

Tentando ler o arquivo excel
Não é possível obter no Jupyter

```
[illegible]
```

In [96]:

```
# Lendo o arquivo excel para um Dataframe
newDf2 = pd.read_excel('teste-df-output.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
```

In [97]:

```
newDf2.head()
```

Out[97]:

	A	B	C	D
2016-01-01	-0.761369	-3.342568	-0.313439	0.578510
2016-01-02	0.617142	-2.462747	-0.461796	0.778261
2016-01-03	1.302244	-3.243220	0.130391	1.374311
2016-01-04	0.777429	-3.201828	0.099690	0.199357
2016-01-05	1.421156	-3.081203	-0.621358	-1.700826

Quando encontrar valores NA preencher com NA

In [98]:

```
os.remove('teste-df-output.xlsx')
```

In [99]:

```
# Verificando se o arquivo existe. No Windows use !type teste-df-output.xlsx
!head teste-df-output.xlsx
```

```
head: teste-df-output.xlsx: No such file or directory
```


Fim

Data Science Academy - Python Fundamentos - Capítulo 8

Download: http://github.com/dsacademybr (http://github.com/dsacademybr)

inspirado no matlab

Matplotlib

Para atualizar o Matplotlib abra o prompt de comando ou terminal e digite: pip install matplotlib -U update

Construindo Plots

www.matplotlib.org
Lo exemplo de códigos.

In [1]:

```
# O matplotlib.pyplot é uma coleção de funções e estilos que fazem com que o Matplotlib fun
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
```

Lo sinal do jupyter → apenas para jupyter. notebook

menos é mais
não constrúa um Gráfico
muito poluído

In [2]:

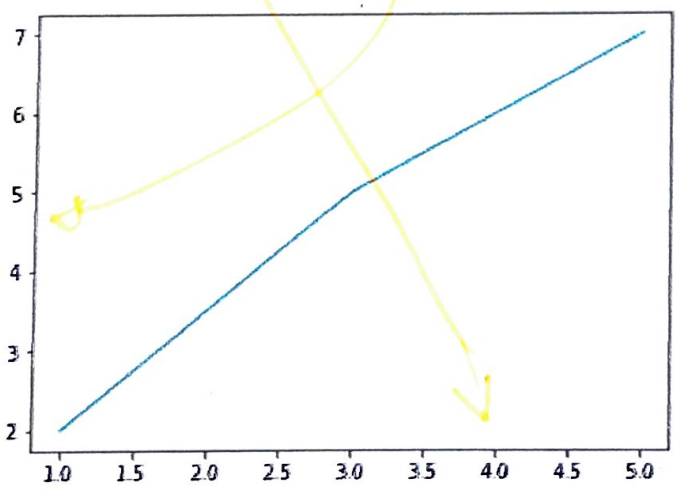
```
mpl.__version__
```

Out[2]:

```
'2.2.2'
```

In [3]:

```
# O método plot() define os eixos do gráfico
plt.plot([1, 3, 5], [2, 5, 7])
plt.show()
```



In [4]:

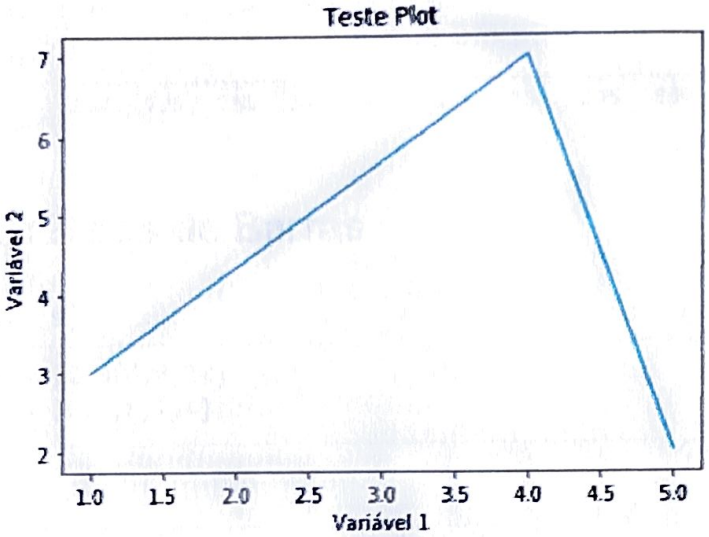
```
x = [1, 4, 5]
```

In [5]:

```
y = [3, 7, 2]
```

In [6]:

```
plt.plot(x, y)
plt.xlabel('Variável 1')
plt.ylabel('Variável 2')
plt.title('Teste Plot')
plt.show()
```

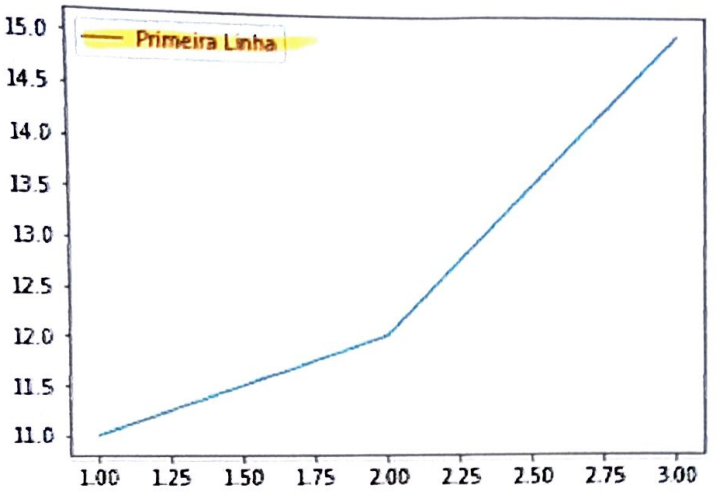


In [7]:

```
x2 = [1, 2, 3]
y2 = [11, 12, 15]
```

In [8]:

```
plt.plot(x2, y2, label = 'Primeira Linha')
plt.legend()
plt.show()
```



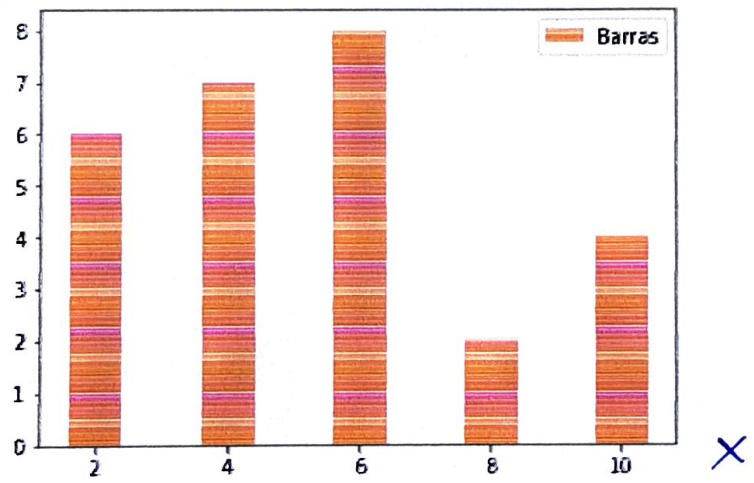
Gráficos de Barras

In [9]:

```
x = [2,4,6,8,10]
y = [6,7,8,2,4]
```

In [10]:

```
plt.bar(x, y, label = 'Barras', color = 'r')
plt.legend()
plt.show()
```



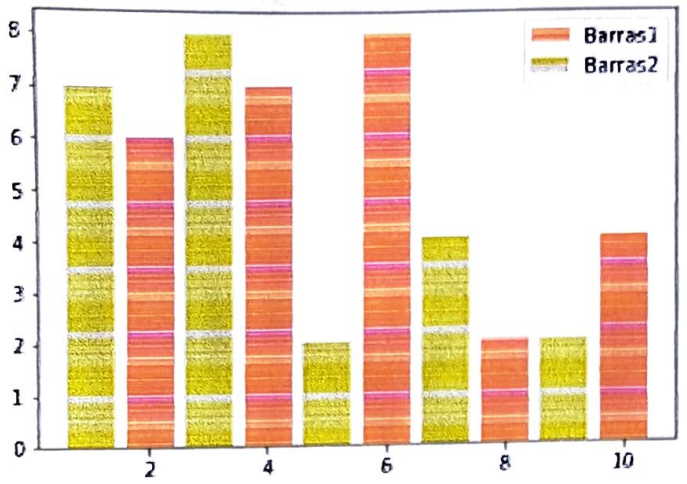
In [11]:

```
x2 = [1,3,5,7,9]
y2 = [7,8,2,4,2]
```

In [12]:

```
plt.bar(x, y, label = 'Barras1', color = 'r')
plt.bar(x2, y2, label = 'Barras2', color = 'y')
plt.legend()
plt.show()
```

Red yellow



In [13]:

```
idades = [22,65,45,55,21,22,34,42,41,4,99,101,120,122,130,111,115,80,75,54,44,64,13,18,48]
```

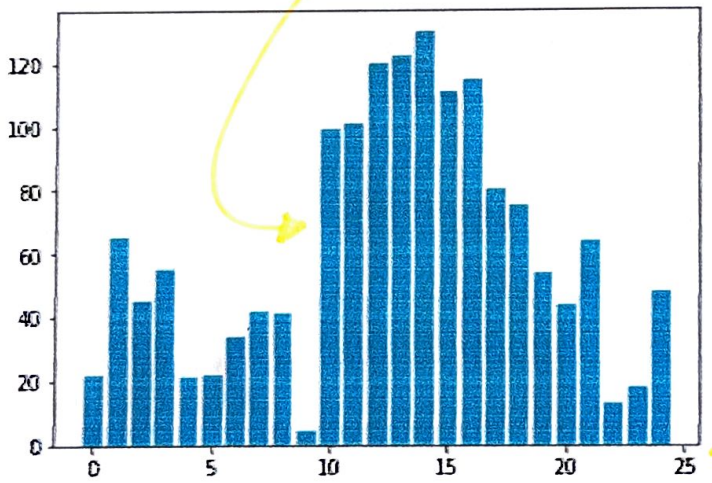
In [14]:

```
ids = [x for x in range(len(idades))]
```

list comprehension

In [15]:

```
plt.bar(ids, idades)
plt.show()
```

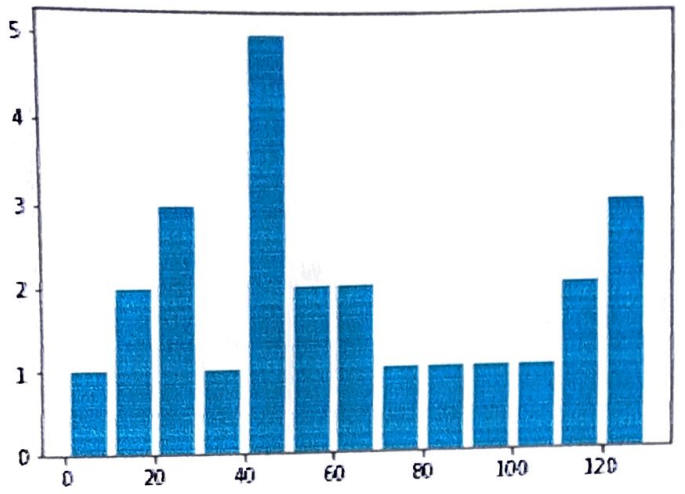


In [16]:

```
bins = [0,10,20,30,40,50,60,70,80,90,100,110,120,130]
```

In [17]:

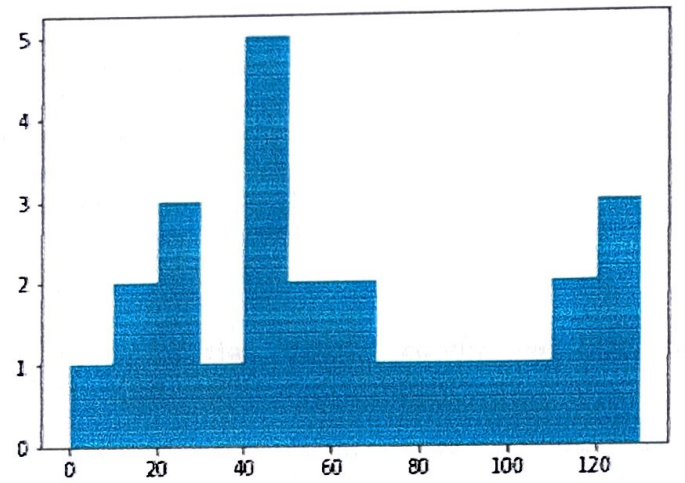
```
plt.hist(idades, bins, histtype = 'bar', rwidth = 0.8)
plt.show()
```



Histogram em barras

In [18]:

```
plt.hist(idades, bins, histtype = 'stepfilled', rwidth = 0.8)
plt.show()
```



Histogram

Scatterplot

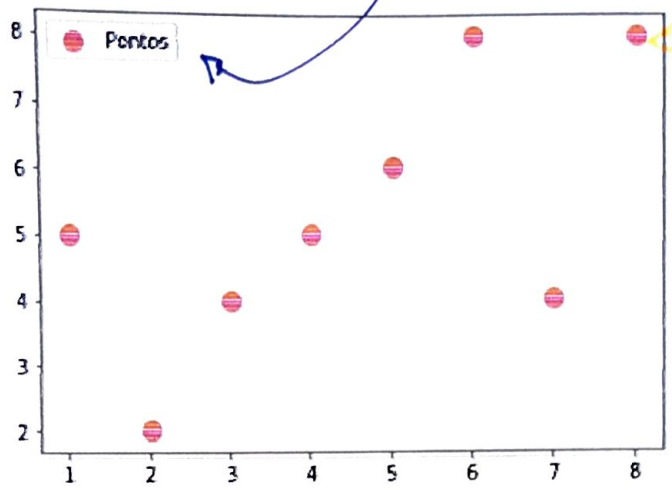
In [19]:

```
x = [1,2,3,4,5,6,7,8]
y = [5,2,4,5,6,8,4,8]
```

In [20]:

```
plt.scatter(x, y, label = 'Pontos', color = 'r', marker = 'o', s = 100)
plt.legend()
plt.show()
```

Size ?



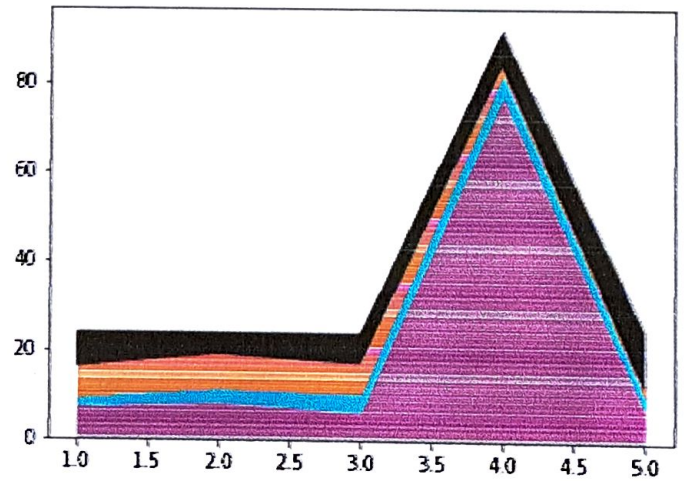
Stack Plots

In [21]:

```
dias = [1,2,3,4,5]
dormir = [7,8,6,77,7]
comer = [2,3,4,5,3]
trabalhar = [7,8,7,2,2]
passear = [8,5,7,8,13]
```

In [22]:

```
plt.stackplot(dias, dormir, comer, trabalhar, passear, colors = ['m','c','r','k','b'])
plt.show()
```



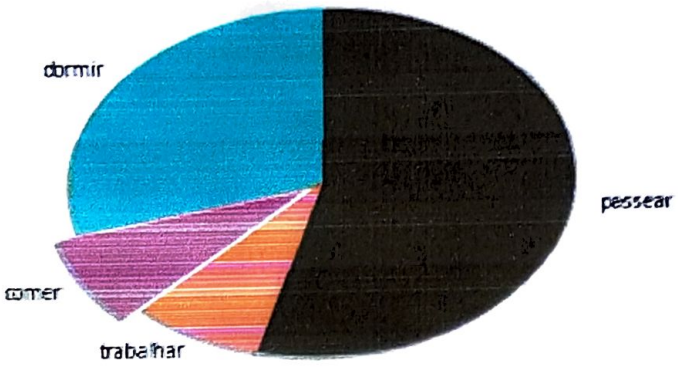
Pie Chart

In [23]:

```
fatias = [7, 2, 2, 13]
atividades = ['dormir','comer','trabalhar','passear']
colunas = ['c','m','r','k']
```

In [24]:

```
plt.pie(fatias, labels = atividades, colors = colunas, startangle = 90, shadow = True, expl
plt.show()
```



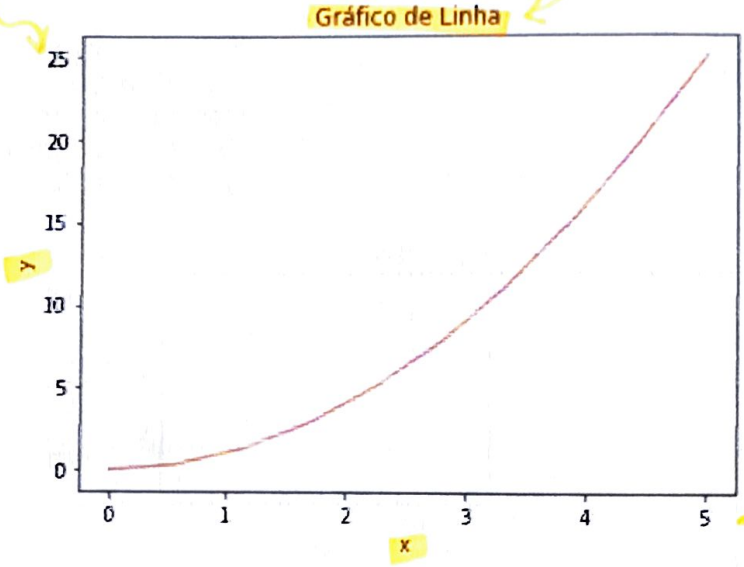
Pylab

In [25]:

```
# Visualizando os gráfico dentro do Jupyter Notebook
# O Pylab combina funcionalidades do pyplot com funcionalidades do Numpy
from pylab import *
%matplotlib inline
```

In [26]:

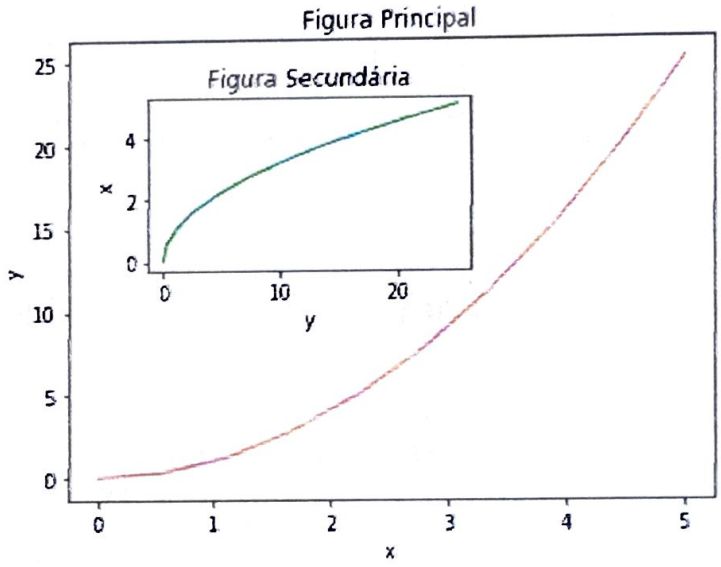
```
x = linspace(0, 5, 10)
y = x ** 2

fig = plt.figure()

# Definindo os eixos
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])

axes.plot(x, y, 'r')

axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('Gráfico de Linha');
```



In [27]:

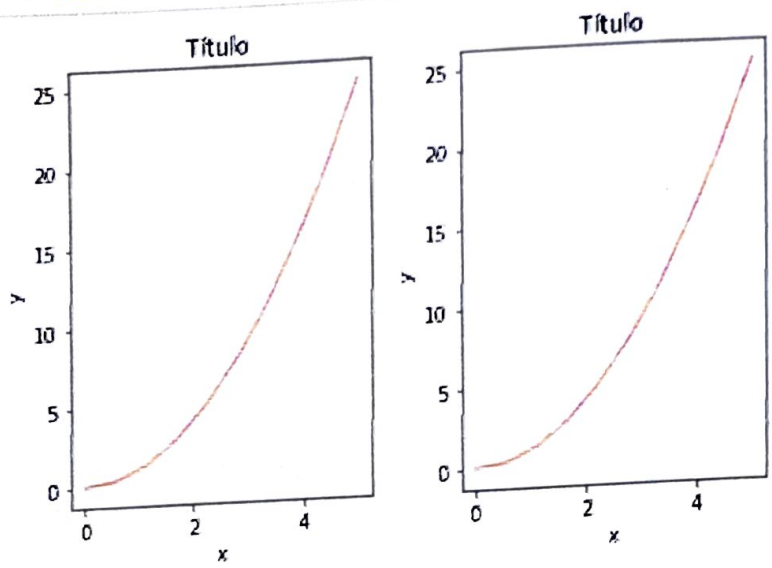
```
# Gráficos com 2 figuras
x = linspace(0, 5, 10)
y = x ** 2

fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # eixos da figura principal
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # eixos da figura secundária

# Figura principal
axes1.plot(x, y, 'r') Red
axes1.set_xlabel('x')
axes1.set_ylabel('y')
axes1.set_title('Figura Principal')

# Figura secundária
axes2.plot(y, x, 'g') green
axes2.set_xlabel('y')
axes2.set_ylabel('x')
axes2.set_title('Figura Secundária');
```

a sequencia determina qual é Principal e qual é secundário



In [28]:

```
# Gráficos em Paralelo
fig, axes = plt.subplots(nrows = 1, ncols = 2)

for ax in axes:
    ax.plot(x, y, 'r')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('Título')

fig.tight_layout()
```



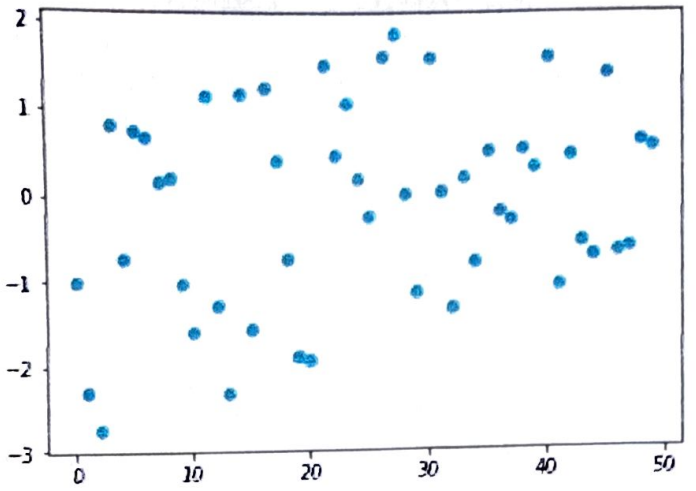
Gráficos a partir do NumPy

In [29]:

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

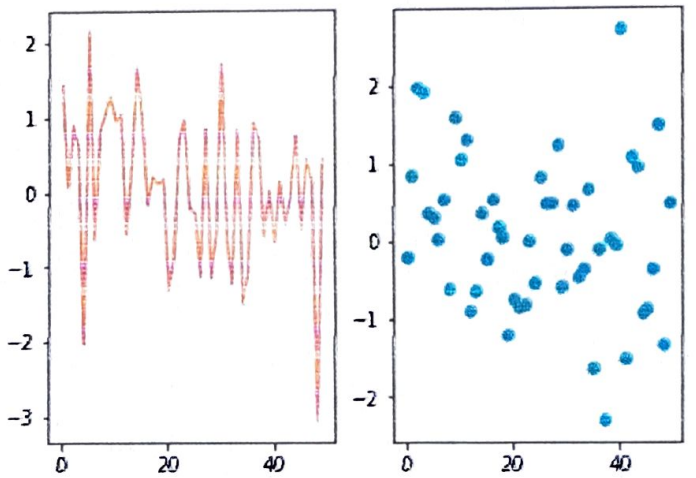
In [30]:

```
plt.scatter(np.arange(50), np.random.randn(50))
plt.show()
```



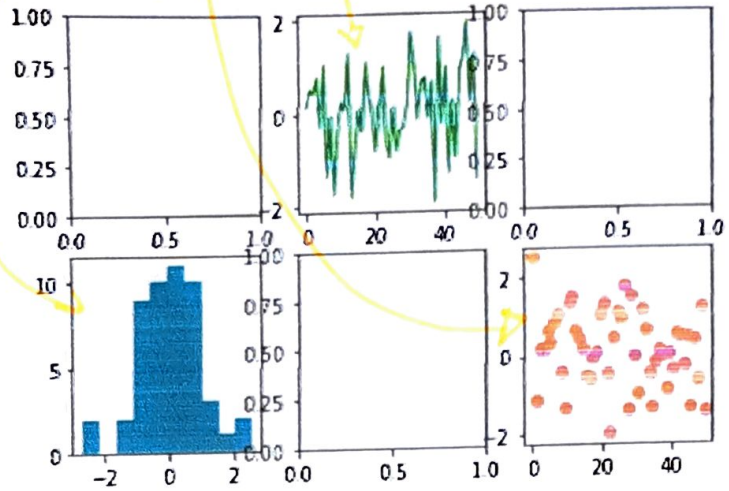
In [31]:

```
# Plot e Scatter
fig = plt.figure()

ax1 = fig.add_subplot(1,2,1)
ax1.plot(np.random.randn(50), color='red')

ax2 = fig.add_subplot(1,2,2)
ax2.scatter(np.arange(50), np.random.randn(50))
plt.show()
```



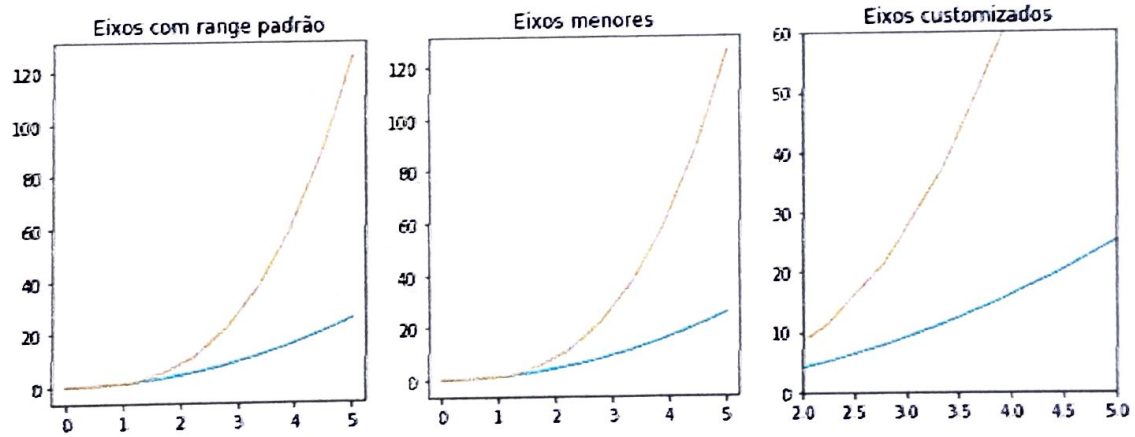
In [32]:

```
# Plots diversos
_, ax = plt.subplots(2,3)

ax[0,1].plot(np.random.randn(50), color = 'green', linestyle = '-')
ax[1,0].hist(np.random.randn(50))
ax[1,2].scatter(np.arange(50), np.random.randn(50), color = 'red')
plt.show()
```



In [33]:

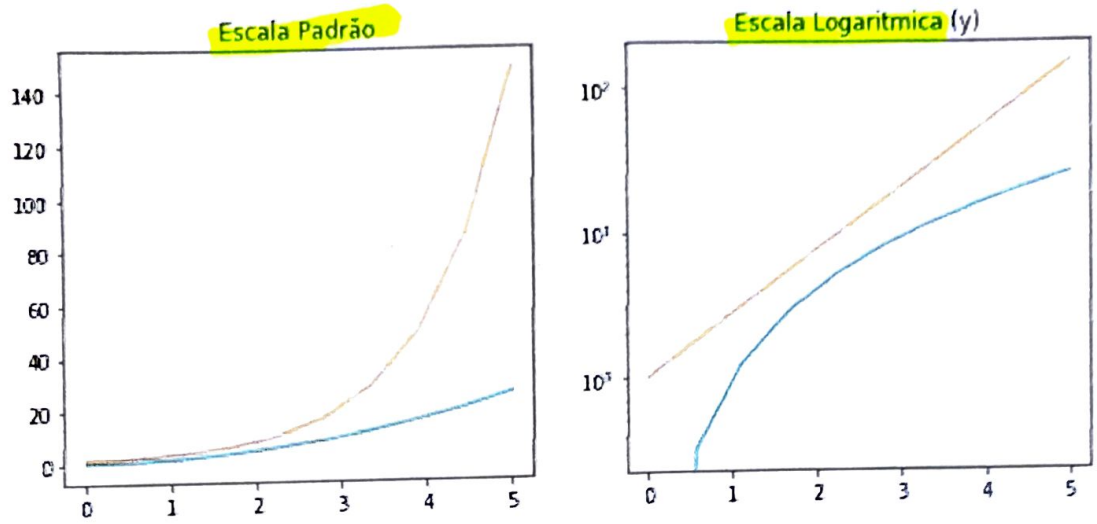
```
# Controle dos eixos
fig, axes = plt.subplots(1, 3, figsize = (12, 4))

axes[0].plot(x, x**2, x, x**3)
axes[0].set_title("Eixos com range padrão")

axes[1].plot(x, x**2, x, x**3)
axes[1].axis('tight')
axes[1].set_title("Eixos menores")

axes[2].plot(x, x**2, x, x**3)
axes[2].set_ylim([0, 60])
axes[2].set_xlim([2, 5])
axes[2].set_title("Eixos customizados");
```



In [34]:

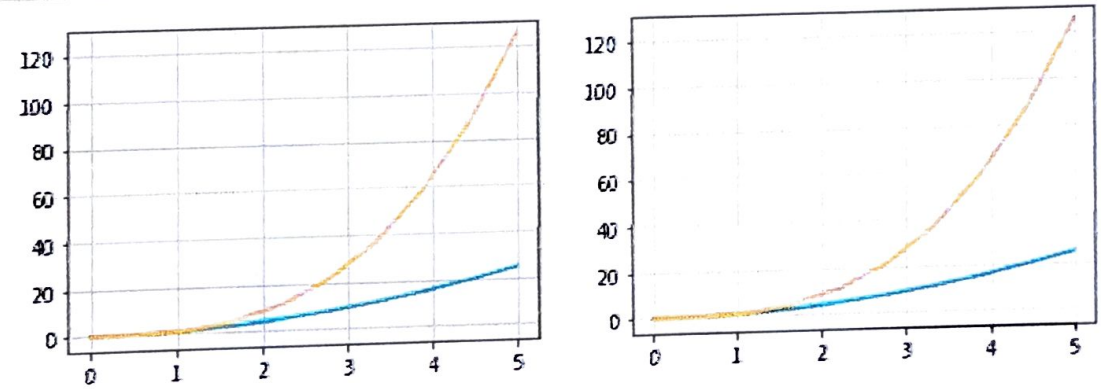
```
# Escala
fig, axes = plt.subplots(1, 2, figsize=(10,4))

axes[0].plot(x, x**2, x, exp(x))
axes[0].set_title("Escala Padrão")

axes[1].plot(x, x**2, x, exp(x))
axes[1].set_yscale("log")
axes[1].set_title("Escala Logaritmica (y)");
```

Cuidado com a escala dos Gráficos



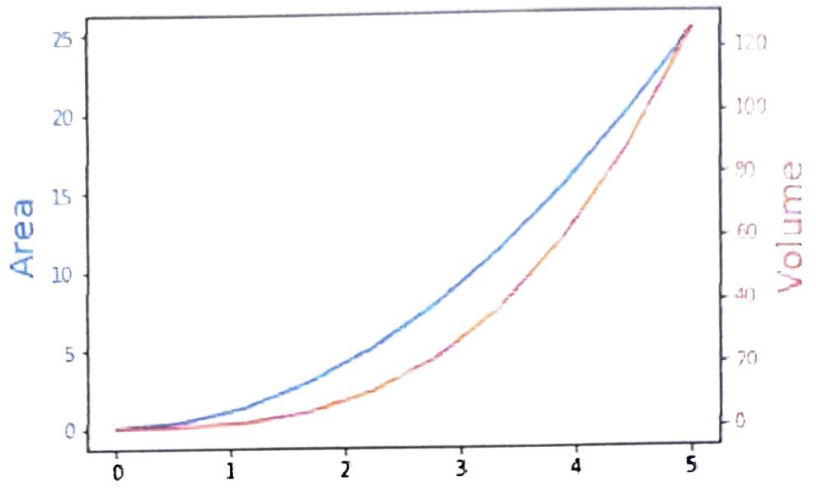
In [35]:

```
# Grid
fig, axes = plt.subplots(1, 2, figsize=(10,3))

# Grid padrão
axes[0].plot(x, x**2, x, x**3, lw = 2)
axes[0].grid(True)

# Grid customizado
axes[1].plot(x, x**2, x, x**3, lw = 2)
axes[1].grid(color = 'b', alpha = 0.5, linestyle = 'dashed', linewidth = 0.5)
```



In [36]:

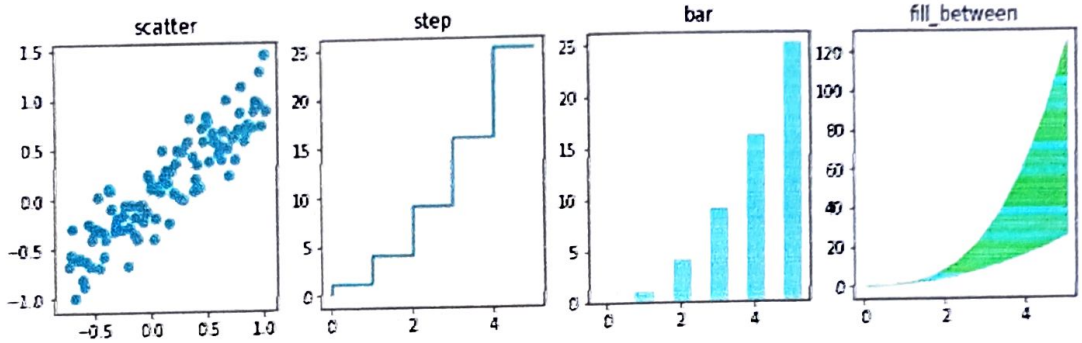
```
# Gráfico de Linhas Gêmeas
fig, ax1 = plt.subplots()

ax1.plot(x, x**2, lw=2, color="blue")
ax1.set_ylabel("Area", fontsize=18, color="blue")
for label in ax1.get_yticklabels():
    label.set_color("blue")

ax2 = ax1.twinx()
ax2.plot(x, x**3, lw=2, color="red")
ax2.set_ylabel("Volume", fontsize=18, color="red")
for label in ax2.get_yticklabels():
    label.set_color("red")
```



In [37]:

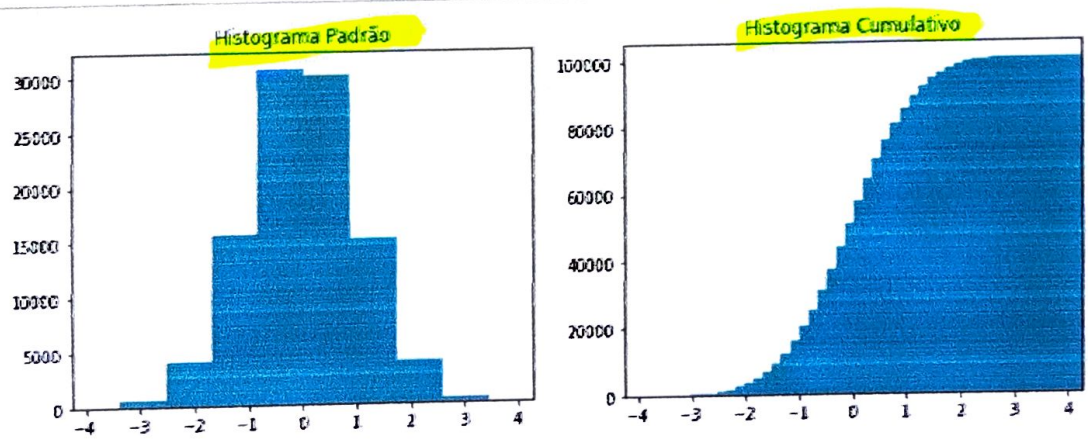
```
# Diferentes estilos de Plots
xx = np.linspace(-0.75, 1., 100)
n = np.array([0,1,2,3,4,5])

fig, axes = plt.subplots(1, 4, figsize=(12,3))

axes[0].scatter(xx, xx + 0.25*randn(len(xx)))
axes[0].set_title("scatter")

axes[1].step(n, n**2, lw=2)
axes[1].set_title("step")

axes[2].bar(n, n**2, align="center", width=0.5, alpha=0.5)
axes[2].set_title("bar")

axes[3].fill_between(x, x**2, x**3, color="green", alpha=0.5);
axes[3].set_title("fill_between");
```

mesmo conjunto de dados, diferentes estilos de dados



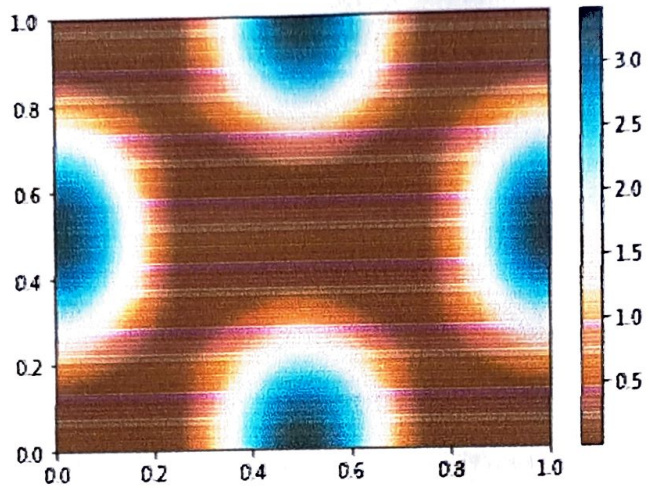
In [38]:

```
# Histogramas
n = np.random.randn(100000)
fig, axes = plt.subplots(1, 2, figsize=(12,4))

axes[0].hist(n)
axes[0].set_title("Histograma Padrão")
axes[0].set_xlim((min(n), max(n)))

axes[1].hist(n, cumulative=True, bins=50)
axes[1].set_title("Histograma Cumulativo")
axes[1].set_xlim((min(n), max(n)));
```



In [39]:

```
# Color Map
alpha = 0.7
phi_ext = 2 * np.pi * 0.5

def ColorMap(phi_m, phi_p):
    return ( + alpha - 2 * np.cos(phi_p)*cos(phi_m) - alpha * np.cos(phi_ext - 2*phi_p))

phi_m = np.linspace(0, 2*np.pi, 100)
phi_p = np.linspace(0, 2*np.pi, 100)
X,Y = np.meshgrid(phi_p, phi_m)
Z = ColorMap(X, Y).T

fig, ax = plt.subplots()

p = ax.pcolor(X/(2*np.pi), Y/(2*np.pi), Z, cmap=cm.RdBu, vmin=abs(Z).min(), vmax=abs(Z).max
cb = fig.colorbar(p, ax=ax)
```



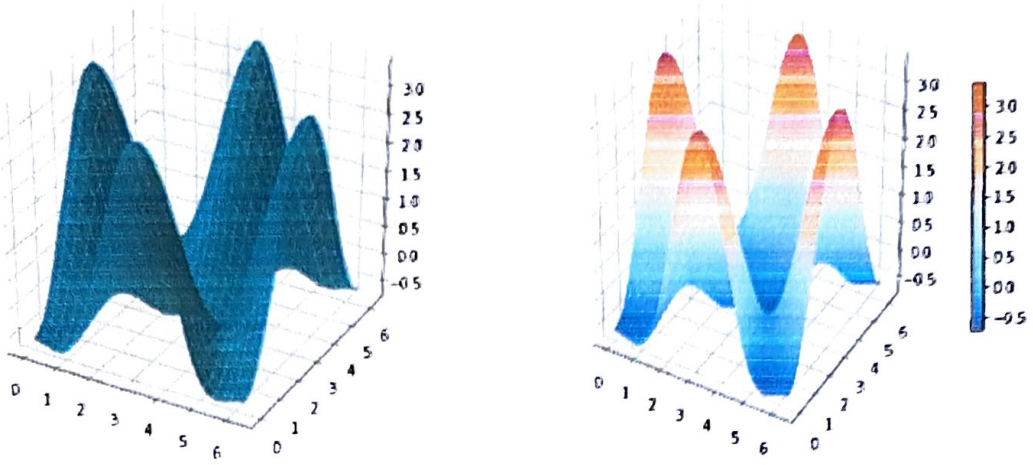
Gráficos 3D

In [40]:

```
from mpl_toolkits.mplot3d.axes3d import Axes3D
```

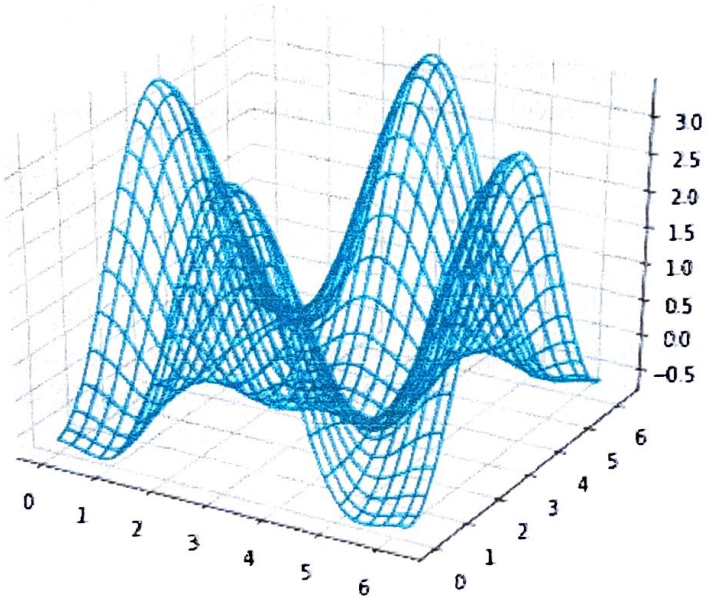
In [41]:

```
fig = plt.figure(figsize=(14,6))

ax = fig.add_subplot(1, 2, 1, projection='3d')
p = ax.plot_surface(X, Y, Z, rstride=4, cstride=4, linewidth=0)

ax = fig.add_subplot(1, 2, 2, projection='3d')
p = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm, linewidth=0, antialias
cb = fig.colorbar(p, shrink=0.5)
```



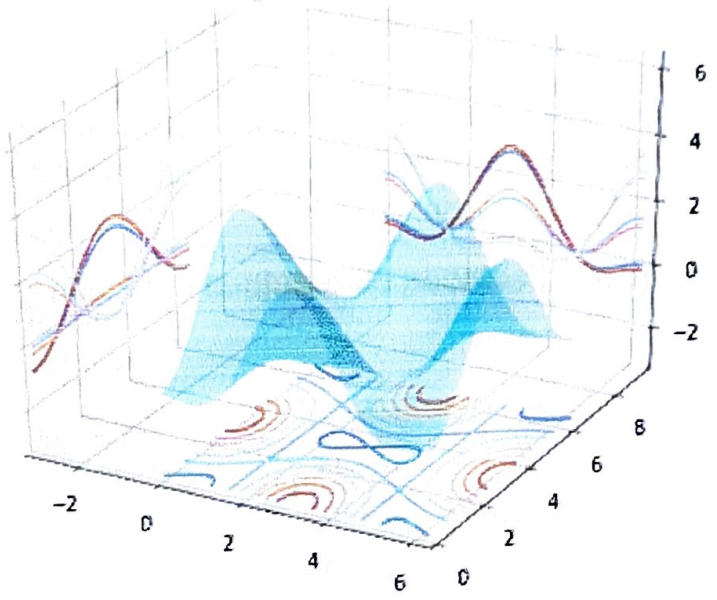
In [42]:

```
# Wire frame
fig = plt.figure(figsize=(8,6))

ax = fig.add_subplot(1, 1, 1, projection = '3d')

p = ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4)
```



In [43]:

```
# Countour Plot com projeção
fig = plt.figure(figsize=(8,6))

ax = fig.add_subplot(1,1,1, projection='3d')

ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
cset = ax.contour(X, Y, Z, zdir='z', offset=-pi, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='x', offset=-pi, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='y', offset=3*pi, cmap=cm.coolwarm)

ax.set_xlim3d(-pi, 2*pi);
ax.set_ylim3d(0, 3*pi);
ax.set_zlim3d(-pi, 2*pi);
```



Para aprender a criar visualizações de dados eficientes e profissionais, acesse:

https://www.datascienceacademy.com.br/pages/curso-visualizacao-de-dados (https://www.datascienceacademy.com.br/pages/curso-visualizacao-de-dados)

Fim

Obrigado - Data Science Academy - facebook.com/dsacademybr (http://facebook.com/dsacademybr)

Data Science Academy - Python Fundamentos - Capítulo 8

Download: http://github.com/dsacademybr (http://github.com/dsacademybr)

SciPy

Scientific Python

Para compreender o SciPy é necessário compreender conceitos avançados de Matemática e Estatística, o que está fora do escopo deste treinamento. Caso queira aprender aplicações práticas do pacote em Machine Learning e IA, consulte estes dois cursos aqui na DSA:

Machine Learning: https://www.datascienceacademy.com.br/pages/curso-machine-learning (https://www.datascienceacademy.com.br/pages/curso-machine-learning)

Programação Paralela em GPU: https://www.datascienceacademy.com.br/pages/curso-programacao-paralela-em-gpu (https://www.datascienceacademy.com.br/pages/curso-programacao-paralela-em-gpu)

O SciPy possui um conjunto de pacotes para operações matemáticas e científicas

Pacote	Descrição
`cluster`	Clustering algorithms
`constants`	Mathematical and physical constants
`fftpack`	Fourier transforms
`integrate`	*Numerical integration*
`interpolate`	Interpolation
`io`	*Input and output*
`linalg`	Linear algebra
`maxentropy`	*Maximum entropy models*
`misc`	Miscellaneous
`ndimage`	Multi-dimensional image processing
`odr`	Orthogonal distance regression
`optimize`	Optimization
`signal`	Signal processing
`sparse`	Sparse matrices
`spatial`	Spatial algorithms and data structures
`special`	Special functions
`stats`	Statistical functions
`stsci`	Image processing

In [1]:

```
from scipy import misc
misc.imread('Matplotlib-Mapa.png')

# Matplotlib tem uma função similar
import matplotlib.pyplot as plt
plt.imread('Matplotlib-Mapa.png')
```

image read

Out[1]:

```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        ...,
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        ...,
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        ...,
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       ...,

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        ...,
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        ...,
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        ...,
        [1., 1., 1., 1.],
```

Uma imagem é uma matriz de pixel

www.scipy.org

```
[1., 1., 1., 1.],
[1., 1., 1., 1.]]], dtype=float32)
```

Integração Numérica

In [2]:

```
from numpy import *
from scipy.integrate import quad, dblquad, tplquad
```

In [3]:

```
# Integração
val, abserr = quad(lambda x: exp(-x ** 2),  Inf, Inf)
val, abserr
```

Out[3]:

```
(0.0, 0.0)
```

In [4]:

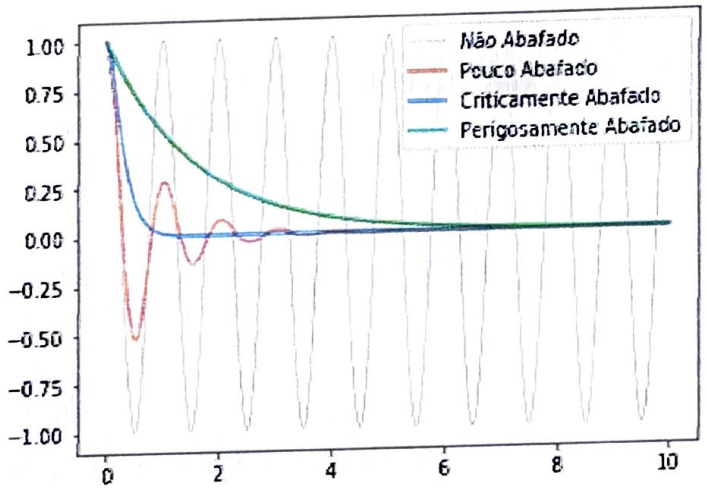
```
from scipy.integrate import odeint, ode
```

In [5]:

```
from pylab import *
%matplotlib inline
```

In [6]:

```
def dy(y, t, zeta, w0):
    x, p = y[0], y[1]

    dx = p
    dp = -2 * zeta * w0 * p - w0**2 * x

    return [dx, dp]


y0 = [1.0, 0.0]

t = linspace(0, 10, 1000)
w0 = 2*pi*1.0


y1 = odeint(dy, y0, t, args=(0.0, w0))
y2 = odeint(dy, y0, t, args=(0.2, w0))
y3 = odeint(dy, y0, t, args=(1.0, w0))
y4 = odeint(dy, y0, t, args=(5.0, w0))

fig, ax = subplots()
ax.plot(t, y1[:,0], 'k', label="Não Abafado", linewidth=0.25)
ax.plot(t, y2[:,0], 'r', label="Pouco Abafado")
ax.plot(t, y3[:,0], 'b', label="Criticamente Abafado")
ax.plot(t, y4[:,0], 'g', label="Perigosamente Abafado")
ax.legend();
```



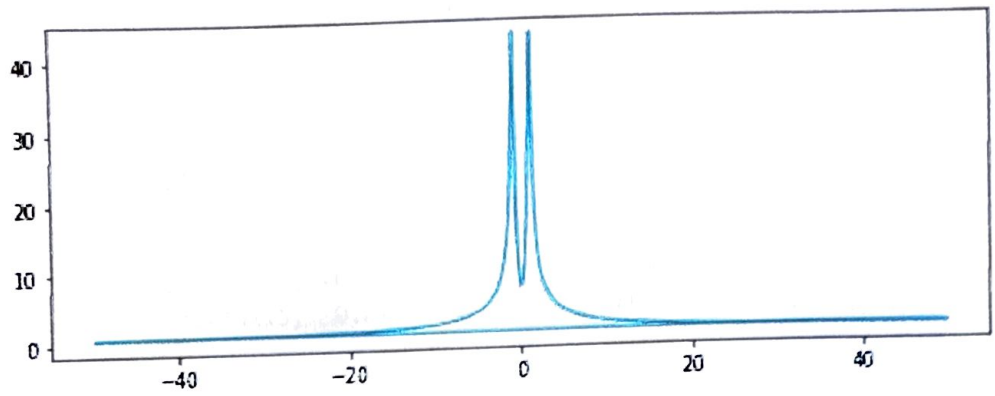
Fourier Transformation

In [7]:

```
from scipy.fftpack import *
```

In [8]:

```
# Fourier transformation
N = len(t)
dt = t[1]-t[0]

F = fft(y2[:,0])

w = fftfreq(N, dt)

fig, ax = subplots(figsize=(9,3))
ax.plot(w, abs(F));
```



Álgebra Linear

In [9]:

```
A = array([[1,2,3], [4,5,6], [7,8,9]])
b = array([1,2,3])
```

In [10]:

```
# Resolvendo um sistema de equações lineares
x = solve(A, b)
x
```

Out[10]:

```
array([-0.33333333,  0.66666667, -0.        ])
```

In [11]:

```
A = rand(3,3)
B = rand(3,3)

evals, evecs = eig(A)

evals
```

Out[11]:

```
array([1.142416  , 0.1745743 , 0.58450936])
```

In [12]:

```
evecs
```

Out[12]:

```
array([[-0.7713803 , -0.86158267,  0.93494291],
       [-0.40604163,  0.09318555, -0.28144399],
       [-0.49000268,  0.49899073,  0.21603481]])
```

In [13]:

```
svd(A)
```

Out[13]:

```
(array([[-0.6042546 ,  0.66762985, -0.43491006],
        [-0.60878132, -0.73898662, -0.2885898 ],
        [-0.51406388,  0.09038341,  0.85297665]]),
 array([1.30977886, 0.65032348, 0.13685752]),
 array([[-0.38013356, -0.80804677, -0.45006544],
        [ 0.49365081, -0.58873898,  0.64007445],
        [-0.78218115,  0.02113861,  0.62269238]]))
```

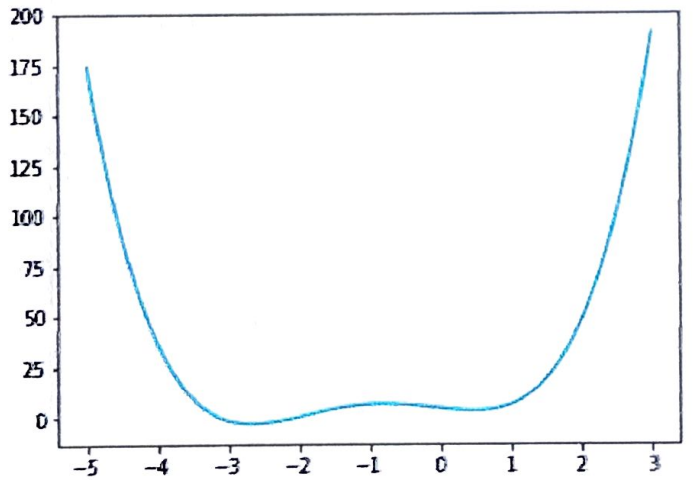
Otimização

In [14]:

```
from scipy import optimize
```

In [15]:

```
def f(x):
    return 4*x**3 + (x-2)**2 + x**4

fig, ax  = subplots()
x = linspace(-5, 3, 100)
ax.plot(x, f(x));
```



In [16]:

```
x_min = optimize.fmin_bfgs(f, -0.5)
x_min
```

```
Optimization terminated successfully.
         Current function value: 2.804988
         Iterations: 4
         Function evaluations: 18
         Gradient evaluations: 6
```

Out[16]:

```
array([0.46961743])
```

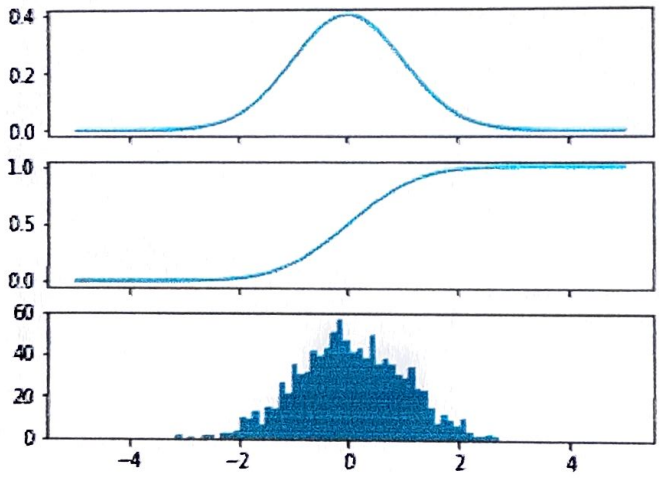
Estatística

In [17]:

```
from scipy import stats
```

In [18]:

```
Y = stats.norm()

x = linspace(-5,5,100)

fig, axes = subplots(3,1, sharex=True)

axes[0].plot(x, Y.pdf(x))

axes[1].plot(x, Y.cdf(x));

axes[2].hist(Y.rvs(size=1000), bins=50);
```



In [19]:

```
Y.mean(), Y.std(), Y.var()
```

Out[19]:

```
(0.0, 1.0, 1.0)
```

In [20]:

```
# T-test
t_statistic, p_value = stats.ttest_ind(Y.rvs(size=1000), Y.rvs(size=1000))
t_statistic, p_value
```

Out[20]:

```
(0.8414404447681616, 0.4002019308600122)
```

Fim

Obrigado - Data Science Academy - facebook.com/dsacademybr (http://facebook.com/dsacademybr)

Data Science Academy - Python Fundamentos - Capítulo 8

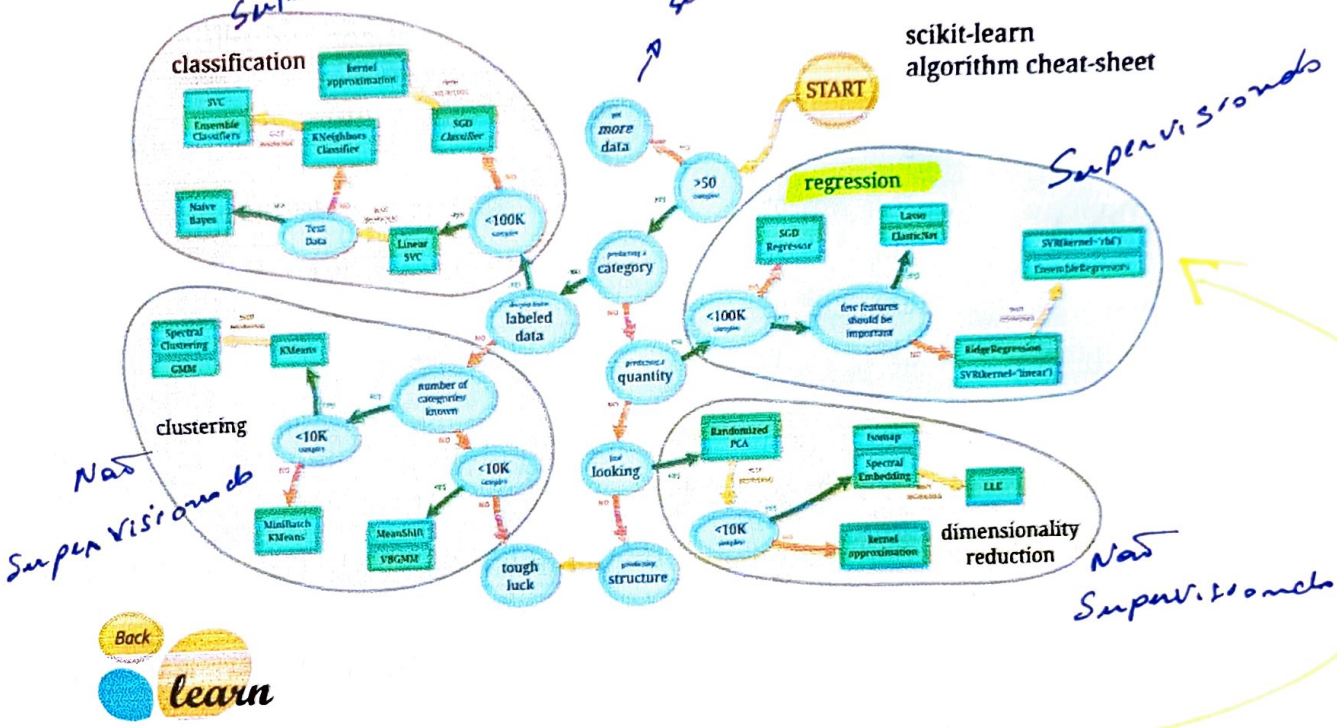
Download: http://github.com/dsacademybr (http://github.com/dsacademybr)

Scikit-learn

In [1]:

```
from IPython.display import Image
Image('ml_map.png')
```

Out[1]:



Prevendo o Preço da Pizza

Suponha que você queira prever o preço da pizza. Para isso, vamos criar um modelo de regressão linear para prever o preço da pizza, baseado em um atributo da pizza que podemos observar. Vamos modelar a relação entre o tamanho (diâmetro) de uma pizza e seu preço. Escreveremos então um programa com sckit-learn, que prevê o preço da pizza dado seu tamanho.

O conjunto de técnicas de regressão é muito provavelmente um dos mais simples modelos utilizadoa em análises de dados que procuram entender a

relação entre o comportamento de determinado fenômeno e o comportamento de uma ou mais variáveis potencialmente preditoras, sem que haja, entretanto uma obrigatória relação de causa e efeito.

É de fundamental importância que o pesquisador seja bastante cuidadoso e criterioso ao interpretar os resultados de uma modelagem de regressão. A existência de um modelo de regressão não significa que ocorra, obrigatoriamente, relação de causa e efeito entre as variáveis consideradas.

In [2]:

```
# Importando Matplotlib e Numpy
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

É necessário analisar os dados

Vamos supor que você registrou o tamanho e preço de pizzas que você comeu nos últimos meses com a sua família.

Instância	Diâmetro(cm)	Preço(R$)
1	7	8
2	10	11
3	15	16
4	30	38.5
5	45	52

dados históricos

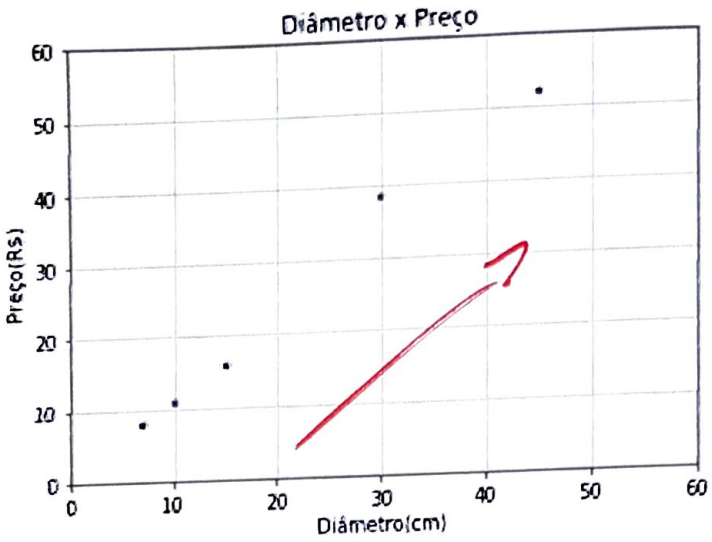
In [3]:

```
# Diâmetros (cm)
Diametros = [[7], [10], [15], [30], [45]]

# Preços (R$)
Precos = [[8], [11], [16], [38.5], [52]]
```

Vamos visualizar estes dados construindo um plot

In [4]:

```
plt.figure()
plt.xlabel('Diâmetro(cm)')
plt.ylabel('Preço(R$)')
plt.title('Diâmetro x Preço')
plt.plot(Diametros, Precos, 'k.')
plt.axis([0, 60, 0, 60])
plt.grid(True)
plt.show()
```



Relação Positiva mas não relação de causa e efeito

Pelo gráfico podemos ver que existe uma relação positiva entre diâmetro da pizza e seu preço (o que é confirmado pela experiência de comer a pizza com sua família). À medida que o diâmetro da pizza aumenta, geralmente aumenta também o preço da pizza.

Vamos agora modelar o relacionamento usando regressão linear e criar um modelo para prever o preço da Pizza.

A classe sklearn.linear_model.LinearRegression é um estimador. Um estimador prevê um valor baseado em dados observados. Em scikit-learn, todos os estimadores implementam os métodos fit() e predict(). O método fit() é usado para aprender os parâmetros de um modelo e o método predict() é usado para prever o valor de uma variável dependente em relação a uma variável explanatória usando os parâmetros aprendidos.

In [5]:

```
# Importando o módulo de Regressão Linear do scikit-learn
from sklearn.linear_model import LinearRegression
```

In [6]:

```
# Preparando os dados de treino

# Vamos chamar de X os dados de diâmetro da Pizza.
X = [[7], [10], [15], [30], [45]]

# Vamos chamar de Y os dados de preço da Pizza.
Y = [[8], [11], [16], [38.5], [52]]
```

In [7]:

```
# Criando o modelo
modelo = LinearRegression()
```

Cria uma instância (modelo)

In [8]:

```
type(modelo)
```

Out[8]:

```
sklearn.linear_model.base.LinearRegression
```

In [9]:

```
# Treinando o modelo
modelo.fit(X, Y)
```

Out[9]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [10]:

```
# Prevendo o preço de uma pizza de 20 cm de diâmetro
print("Uma pizza de 20 cm de diâmetro deve custar: R$%.2f" % modelo.predict([20][0]))
```

```
Uma pizza de 20 cm de diâmetro deve custar: R$23.41
```

O método fit() do módulo LinearRegression aprende os parâmetros do seguinte modelo de regressão linear simples:

In [11]:

```
from IPython.display import Image
Image('linear.png')
```

Out[11]:

$$y = \alpha + \beta x$$

y = ao preço = o que eu quero descobrir

x = predição = diametro

α e β é o que foi aprendido em fit()

Y – é o valor previsto da variável dependente (em nosso exemplo o preço da Pizza)

X – é a variável explanatória (em nosso exemplo o diâmetro da Pizza)

Alfa é o termo de intercepção ou coeficiente linear

Beta é o coeficiente de cada variável ou coeficiente angular

Alfa e Beta são parâmetros do modelo que são aprendidos pelo algoritmo de aprendizagem.

Construindo um Scatter Plot

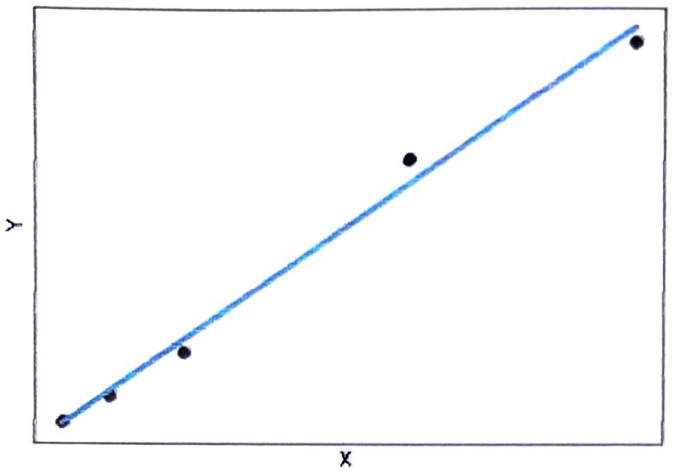
In [12]:

```
# Coeficientes
print('Coeficiente: \n', modelo.coef_)

# MSE (mean square error)
print("MSE: %.2f" % np.mean((modelo.predict(X) - Y) ** 2))

# Score de variação: 1 representa predição perfeita
print('Score de variação: %.2f' % modelo.score(X, Y))
```

```
Coeficiente:
 [[1.20422117]]
MSE: 2.74
Score de variação: 0.99
```

In [13]:

```
# Scatter Plot representando a regressão linear
plt.scatter(X, Y,  color = 'black')
plt.plot(X, modelo.predict(X), color = 'blue', linewidth = 3)
plt.xlabel('X')
plt.ylabel('Y')
plt.xticks(())
plt.yticks(())

plt.show()
```



Explorando o Dataset Boston Housing

Dataset: http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html)

In [14]:

```
# Importando os módulos necessários
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import sklearn
%matplotlib inline
```

In [15]:

```
# O dataset boston já está disponível no scikit-learn. Precisamos apenas carregá-lo.
from sklearn.datasets import load_boston
boston = load_boston()
```

In [16]:

```
# Verificando o tipo da variável boston
type(boston)
```

Out[16]:

```
sklearn.utils.Bunch
```

In [17]:

```
# Visualizando o shape do dataset, neste caso 506 instâncias (linhas) e 13 atributos (colur
boston.data.shape
```

Out[17]:

```
(506, 13)
```

In [18]:

```
# Descrição do Dataset
print(boston.DESCR)
```

```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river;
0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing (http://archive.ics.uci.edu/m
l/datasets/Housing)


This dataset was taken from the StatLib library which is maintained at Carne
gie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostic
s
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers th
at address regression
problems.
```

coluna de Previsão.

References

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
- many more! (see http://archive.ics.uci.edu/ml/datasets/Housing) (http://archive.ics.uci.edu/ml/datasets/Housing))

In [19]:

```
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [20]:

```
# Convertendo o dataset em um DataFrame pandas
df = pd.DataFrame(boston.data)
df.head()
```

Out[20]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [21]:

```
# Convertendo o título das colunas
df.columns = boston.feature_names
df.head()
```

Out[21]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

In [22]:

```
# boston.target é uma array com o preço das casas
boston.target
```

Out[22]:

```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
       36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
       32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
       20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
       20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
       22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
       21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
       19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
       32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
       18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
       16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
       13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
        7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
       12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
       27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
        8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
        9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9])
```

In [23]:

```
# Adicionando o preço da casa ao DataFrame
df['PRICE'] = boston.target
df.head()
```

Out[23]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

Prevendo o Preço das Casas em Boston

Y - variável dependente (preço das casas em Boston)

X - variáveis independentes ou explanatórias (todas as outras caracterísricas da casa)

In [24]:

```
# Importando o módulo de regressão linear
from sklearn.linear_model import LinearRegression
```

In [25]:

```
# Não queremos o preço da casa como variável dependente
X = df.drop('PRICE', axis = 1)
```

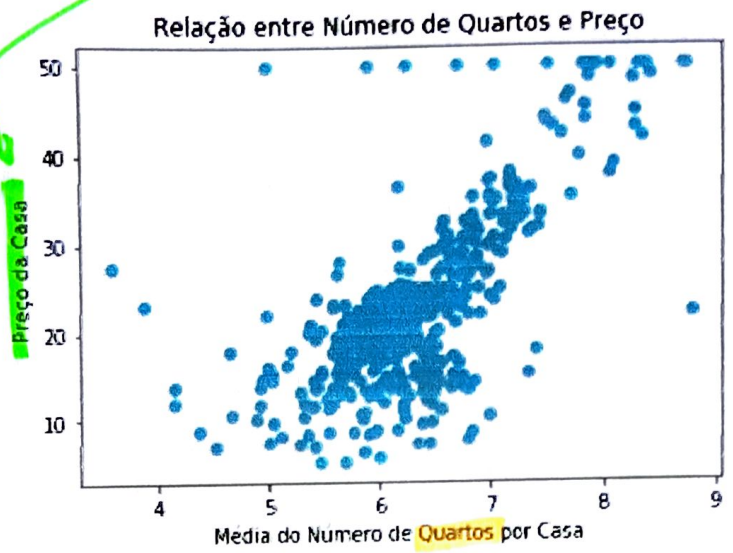
Retiro o preço das casas

In [26]:

```
# Definindo Y
Y = df.PRICE
```

In [27]:

```
plt.scatter(df.RM, Y)
plt.xlabel("Média do Número de Quartos por Casa")
plt.ylabel("Preço da Casa")
plt.title("Relação entre Número de Quartos e Preço")
plt.show()
```



In [28]:

```
# Criando o objeto de regressão linear
regr = LinearRegression()
```

In [29]:

```
# Tipo do objeto
type(regr)
```

Out[29]:

```
sklearn.linear_model.base.LinearRegression
```

fit = função do LinearRegression

In [30]:

```
# Treinando o modelo
regr.fit(X, Y)
```

Out[30]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [31]:

```
# Coeficientes
print("Coeficiente: ", regr.intercept_)
print("Número de Coeficientes: ", len(regr.coef_))
```

```
Coeficiente:  36.491103280361344
Número de Coeficientes:  13
```

In [32]:

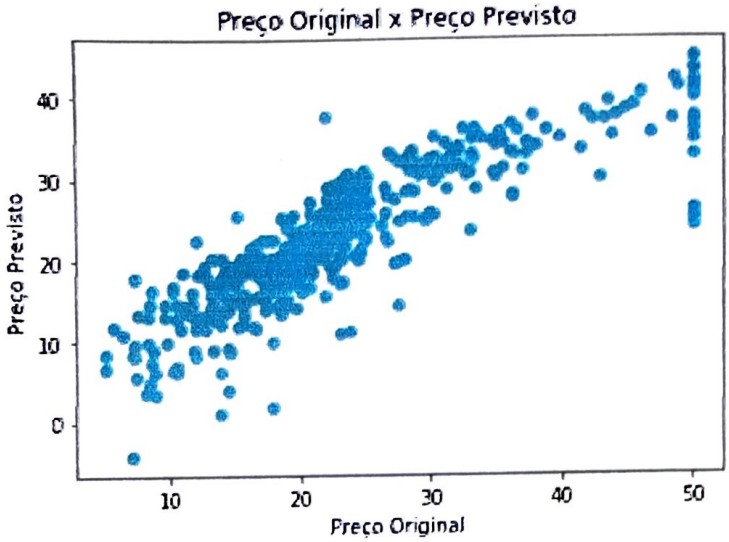
```
# Prevendo o preço da casa
regr.predict(X)
```

Out[32]:

```
array([30.00821269, 25.0298606 , 30.5702317 , 28.60814055, 27.94288232,
       25.25940048, 23.00433994, 19.5347558 , 11.51696539, 18.91981483,
       18.9958266 , 21.58970854, 20.90534851, 19.55535931, 19.2837957 ,
       19.30000174, 20.52889993, 16.9096749 , 16.17067411, 18.40781636,
       12.52040454, 17.67104565, 15.82934891, 13.80368317, 15.67708138,
       13.3791645 , 15.46258829, 14.69863607, 19.54518512, 20.87309945,
       11.44806825, 18.05900412,  8.78841666, 14.27882319, 13.69097132,
       23.81755469, 22.34216285, 23.11123204, 22.91494157, 31.35826216,
       34.21485385, 28.0207132 , 25.20646572, 24.61192851, 22.94438953,
       22.10150945, 20.42467417, 18.03614022,  9.10176198, 17.20856571,
       21.28259372, 23.97621248, 27.65853521, 24.0521088 , 15.35989132,
       31.14817003, 24.85878746, 33.11017111, 21.77458036, 21.08526739,
       17.87203538, 18.50881381, 23.9879809 , 22.54944098, 23.37068403,
       30.36557584, 25.53407332, 21.11758504, 17.42468223, 20.7893086 ,
       25.20349174, 21.74490595, 24.56275612, 24.04479519, 25.5091157 ,
       23.97076758, 22.94823519, 23.36106095, 21.26432549, 22.4345376 ,
       28.40699937, 26.99734716, 26.03807246, 25.06152125, 24.7858613 ,
       27.79291889, 22.16927073, 25.89685664, 30.67771522, 30.83225886,
       27.12127354, 27.41597825, 28.9456478 , 29.08668003, 27.04501726,
       28.62506705, 24.73038218, 35.78062378, 35.11269515, 32.25115468,
       24.57946786, 25.59386215, 19.76439137, 20.31157117, 21.4353635 ,
       18.53971968, 17.18572611, 20.74934949, 22.64791346, 19.77000977,
       20.64745349, 26.52652691, 20.77440554, 20.71546432, 25.17461484,
       20.4273652 , 23.37862521, 23.69454145, 20.33202239, 20.79378139,
       21.92024414, 22.47432006, 20.55884635, 16.36300764, 20.56342111,
       22.48570454, 14.61264839, 15.1802607 , 18.93828443, 14.0574955 ,
       20.03651959, 19.41306288, 20.06401034, 15.76005772, 13.24771577,
       17.26167729, 15.87759672, 19.36145104, 13.81270814, 16.44782934,
       13.56511101,  3.98343974, 14.59241207, 12.14503093,  8.72407108,
       12.00815659, 15.80308586,  8.50963929,  9.70965512, 14.79848067,
       20.83598096, 18.30017013, 20.12575267, 17.27585681, 22.35997992,
       20.07985184, 13.59903744, 33.26635221, 29.03938379, 25.56694529,
       32.71732164, 36.78111388, 40.56615533, 41.85122271, 24.79875684,
       25.3771545 , 37.20662185, 23.08244608, 26.40326834, 26.65647433,
       22.55412919, 24.2970948 , 22.98024802, 29.07488389, 26.52620066,
       30.72351225, 25.61835359, 29.14203283, 31.43690634, 32.9232938 ,
       34.72096487, 27.76792733, 33.88992899, 30.99725805, 22.72124288,
       24.76567683, 35.88131719, 33.42696242, 32.41513625, 34.51611818,
       30.76057666, 30.29169893, 32.92040221, 32.11459912, 31.56133385,
       40.84274603, 36.13046343, 32.66639271, 34.70558647, 30.09276228,
       30.64139724, 29.29189704, 37.07062623, 42.02879611, 43.18582722,
       22.6923888 , 23.68420569, 17.85435295, 23.49543857, 17.00872418,
       22.39535066, 17.06152243, 22.74106824, 25.21974252, 11.10601161,
       24.51300617, 26.60749026, 28.35802444, 24.91860458, 29.69254951,
       33.18492755, 23.77145523, 32.14086508, 29.74802362, 38.36605632,
       39.80716458, 37.58362546, 32.39769704, 35.45048257, 31.23446481,
       24.48478321, 33.28615723, 38.04368164, 37.15737267, 31.71297469,
       25.26658017, 30.101515  , 32.71897655, 28.42735376, 28.42999168,
       27.2913215 , 23.74446671, 24.11878941, 27.40241209, 16.32993575,
       13.39695213, 20.01655581, 19.86205904, 21.28604604, 24.07796482,
       24.20603792, 25.04201534, 24.91709097, 29.93762975, 23.97709054,
       21.69931969, 37.51051381, 43.29459357, 36.48121427, 34.99129701,
```

In [33]:

```
# Comparando preços originais x preços previstos
plt.scatter(df.PRICE, regr.predict(X))
plt.xlabel("Preço Original")
plt.ylabel("Preço Previsto")
plt.title("Preço Original x Preço Previsto")
plt.show()
```



Podemos ver que existem alguns erros na predição do preço das casas

In [34]:

```
# Vamos calcular o MSE (Mean Squared Error)
mse1 = np.mean((df.PRICE - regr.predict(X)) ** 2)
print(mse1)
```

```
21.897779217687493
```

In [35]:

```
# Aplicando regressão linear para apenas uma variável e calculando o MSE
regr = LinearRegression()
regr.fit(X[['PTRATIO']], df.PRICE)
mse2 = np.mean((df.PRICE - regr.predict(X[['PTRATIO']])) ** 2)
print(mse2)
```

```
62.65220001376927
```

O MSE aumentou, indicando que uma única característica não é um bom predictor para o preço das casas.

Na prática, você não vai implementar regressão linear em todo o dataset. Você vai dividir o dataset em datasets de treino e de teste. Assim, você treina seu modelo nos dados de treino e depois verifica como o modelo se comporta nos seus dados de teste. Vejamos:

In [36]:

```
# Dividindo X em dados de treino e de teste
X_treino = X[:-50]
X_teste = X[-50:]

# Dividindo Y em dados de treino e de teste
Y_treino = df.PRICE[:-50]
Y_teste = df.PRICE[-50:]

# Imprimindo o shape dos datasets
print(X_treino.shape, X_teste.shape, Y_treino.shape, Y_teste.shape)
```

```
(456, 13) (50, 13) (456,) (50,)
```

Podemos criar nossos datasets de treino de forma manual, mas claro este não é método correto. Vamos então dividir os datasets randomicamente. O Scikit-Learn provê uma função chamada train_test_split() para isso.

In [37]:

```
from sklearn.model_selection import train_test_split
```

Valores randomicos

In [40]:

```
# Dividindo X e Y em dados de treino e de teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, df.PRICE, test_size = 0.33, ranc
```

tamanho de teste

In [41]:

```
# Imprimindo o shape dos datasets
print(X_treino.shape, X_teste.shape, Y_treino.shape, Y_teste.shape)
```

```
(339, 13) (167, 13) (339,) (167,)
```

70% treino
30% teste

In [42]:

```
# Construindo um modelo de regressão
regr = LinearRegression()
```

In [43]:

```
# Treinando o modelo
regr.fit(X_treino, Y_treino)
```

Apenas dados de treino

Out[43]:

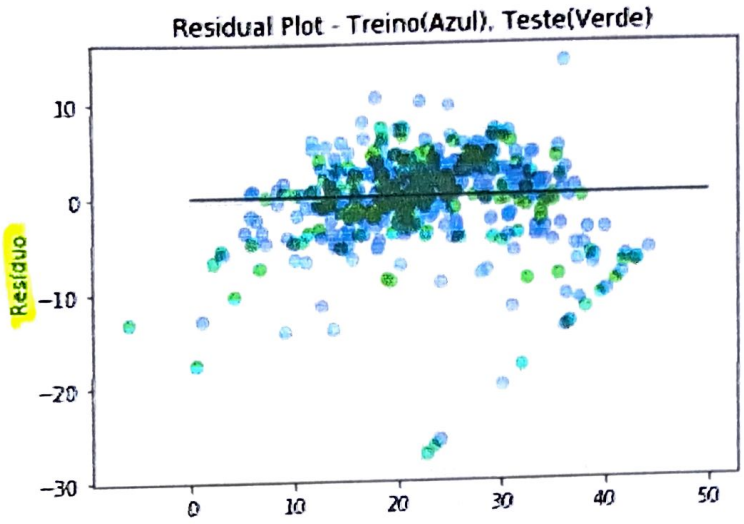
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [44]:

```
# Definindo os dados de treino e teste
pred_treino = regr.predict(X_treino)
pred_teste = regr.predict(X_teste)
```

In [45]:

```
# Comparando preços originais x preços previstos
plt.scatter(regr.predict(X_treino), regr.predict(X_treino) - Y_treino, c = 'b', s = 40, alp
plt.scatter(regr.predict(X_teste), regr.predict(X_teste) - Y_teste, c = 'g', s = 40, alpha
plt.hlines(y = 0, xmin = 0, xmax = 50)
plt.ylabel("Resíduo")
plt.title("Residual Plot - Treino(Azul), Teste(Verde)")
plt.show()
```



Fim

Obrigado - Data Science Academy - facebook.com/dsacademybr (http://facebook.com/dsacademybr)

Data Science Academy - Python Fundamentos - Capítulo 8

Download: http://github.com/dsacademybr (http://github.com/dsacademybr)

Bokeh

D3JS

Caso o Bokeh não esteja instalado, executar no prompt ou terminal: pip install bokeh

In [1]: Para rodar em Web Browser

```
# Importando o módulo Bokeh
import bokeh
from bokeh.io import show, output_notebook
from bokeh.plotting import figure, output_file
from bokeh.models import ColumnDataSource
from bokeh.transform import factor_cmap
from bokeh.palettes import Spectral6
```

Real time

In [2]:

```
# Carregando o Bokeh
output_notebook()
```

Gera Saida no Jupyter e em outra aba

(https://bokeh.pydata.org) Loading BokehJS ...

In [3]:

```
# Arquivo gerado pela visualização
output_file("Bokeh-Grafico-Interativo.html")
```

In [4]:

```
p = figure()
```

In [5]:

```
type(p)
```

Out[5]:

```
bokeh.plotting.figure.Figure
```

In [6]:

```
p.line([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], line_width = 2)
```

Out[6]:

GlyphRenderer(id = '5cae805f-3feb-4bdb-93f1-47b3e1f74d8f', ...)

In [7]:

```
show(p)
```

Gráfico de Barras

In [8]:

```
# Criando um novo gráfico
output_file("Bokeh-Grafico-Barras.html")

fruits = ['Maçãs', 'Peras', 'Tangerinas', 'Uvas', 'Melancias', 'Morangos']
counts = [5, 3, 4, 2, 4, 6]

source = ColumnDataSource(data=dict(fruits=fruits, counts=counts))

p = figure(x_range=fruits, plot_height=350, toolbar_location=None, title="Contagem de Fruta

p.vbar(x='fruits',
       top='counts',
       width=0.9,
       source=source,
       legend="fruits",
       line_color='white',
       fill_color=factor_cmap('fruits', palette=Spectral6, factors=fruits))

p.xgrid.grid_line_color = None
p.y_range.start = 0
p.y_range.end = 9
p.legend.orientation = "horizontal"
p.legend.location = "top_center"

show(p)
```

ScatterPlot

In [9]:

```
# Construindo um ScatterPlot
from bokeh.plotting import figure, show, output_file
from bokeh.sampledata.iris import flowers

colormap = {'setosa': 'red', 'versicolor': 'green', 'virginica': 'blue'}
colors = [colormap[x] for x in flowers['species']]

p = figure(title = "Iris Morphology")
p.xaxis.axis_label = 'Petal Length'
p.yaxis.axis_label = 'Petal Width'

p.circle(flowers["petal_length"], flowers["petal_width"], color=colors, fill_alpha=0.2, siz

output_file("Bokeh_grafico_Iris.html", title="iris.py example")

show(p)
```

Gráfico de Círculos

In [10]:

```
from bokeh.plotting import figure, output_file, show

# Outuput
output_file("Bokeh-Grafico-Circulos.html")

p = figure(plot_width = 400, plot_height = 400)

# Adicionando círculos ao gráfico
p.circle([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], size = 20, color = "navy", alpha = 0.5)

# Mostrando o resultado
show(p)
```

Gráfico com Dados Geofísicos

In [11]:

```
# Geojson
from bokeh.io import output_file, show
from bokeh.models import GeoJSONDataSource
from bokeh.plotting import figure
from bokeh.sampledata.sample_geojson import geojson

geo_source = GeoJSONDataSource(geojson=geojson)

p = figure()
p.circle(x = 'x', y = 'y', alpha = 0.9, source = geo_source)
output_file("Bokeh-GeoJSON.html")
show(p)
```

In [12]:

```
# Baixando o diretório de dados de exemplo do Bokeh
bokeh.sampledata.download()
```

```
Using data directory: /Users/dmpm/.bokeh/data
Downloading: CGM.csv (1589982 bytes)
   1589982 [100.00%]
Downloading: US_Counties.zip (3182088 bytes)
   3182088 [100.00%]
Unpacking: US_Counties.csv
Downloading: us_cities.json (713565 bytes)
    713565 [100.00%]
Downloading: unemployment09.csv (253301 bytes)
    253301 [100.00%]
Downloading: AAPL.csv (166698 bytes)
    166698 [100.00%]
Downloading: FB.csv (9706 bytes)
      9706 [100.00%]
Downloading: GOOG.csv (113894 bytes)
    113894 [100.00%]
Downloading: IBM.csv (165625 bytes)
    165625 [100.00%]
Downloading: MSFT.csv (161614 bytes)
    161614 [100.00%]
Downloading: WPP2012_SA_DB03_POPULATION_QUINQUENNIAL.zip (5148539 bytes)
   5148539 [100.00%]
Unpacking: WPP2012_SA_DB03_POPULATION_QUINQUENNIAL.csv
Downloading: gapminder_fertility.csv (64346 bytes)
     64346 [100.00%]
Downloading: gapminder_population.csv (94509 bytes)
     94509 [100.00%]
Downloading: gapminder_life_expectancy.csv (73243 bytes)
     73243 [100.00%]
Downloading: gapminder_regions.csv (7781 bytes)
      7781 [100.00%]
Downloading: world_cities.zip (646858 bytes)
    646858 [100.00%]
Unpacking: world_cities.csv
Downloading: airports.json (6373 bytes)
      6373 [100.00%]
Downloading: movies.db.zip (5067833 bytes)
   5067833 [100.00%]
Unpacking: movies.db
Downloading: airports.csv (203190 bytes)
    203190 [100.00%]
Downloading: routes.csv (377280 bytes)
    377280 [100.00%]
```

In [13]:

```
from bokeh.io import show
from bokeh.models import (ColumnDataSource, HoverTool, LogColorMapper)
from bokeh.palettes import Viridis6 as palette
from bokeh.plotting import figure
from bokeh.sampledata.us_counties import data as counties
from bokeh.sampledata.unemployment import data as unemployment

palette.reverse()

counties = {code: county for code, county in counties.items() if county["state"] == "tx"}

county_xs = [county["lons"] for county in counties.values()]
county_ys = [county["lats"] for county in counties.values()]

county_names = [county['name'] for county in counties.values()]
county_rates = [unemployment[county_id] for county_id in counties]
color_mapper = LogColorMapper(palette=palette)

source = ColumnDataSource(data=dict(
    x=county_xs,
    y=county_ys,
    name=county_names,
    rate=county_rates,
))

TOOLS = "pan,wheel_zoom,reset,hover,save"

p = figure(
    title="Texas Unemployment, 2009", tools=TOOLS,
    x_axis_location=None, y_axis_location=None
)
p.grid.grid_line_color = None

p.patches('x', 'y', source=source,
          fill_color={'field': 'rate', 'transform': color_mapper},
          fill_alpha=0.7, line_color="white", line_width=0.5)

hover = p.select_one(HoverTool)
hover.point_policy = "follow_mouse"
hover.tooltips = [
    ("Name", "@name"),
    ("Unemployment rate)", "@rate%"),
    ("(Long, Lat)", "($x, $y)"),
]

show(p)
```

Data Science Academy - Python Fundamentos - Capítulo 8

Download: http://github.com/dsacademybr (http://github.com/dsacademybr)

Statsmodels

Linear Regression Models

In [1]:

```
# Para visualização de gráficos
from pylab import *
%matplotlib inline
```

In [5]:

```
model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	4.020e+06
Date:	Thu, 02 Aug 2018	Prob (F-statistic):	2.83e-239
Time:	12:16:44	Log-Likelihood:	-146.51
No. Observations:	100	AIC:	299.0
Df Residuals:	97	BIC:	306.8
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.3423	0.313	4.292	0.000	0.722	1.963
x1	-0.0402	0.145	-0.278	0.781	-0.327	0.247
x2	10.0103	0.014	715.745	0.000	9.982	10.038

Omnibus:	2.042	Durbin-Watson:	2.274
Prob(Omnibus):	0.360	Jarque-Bera (JB):	1.875
Skew:	0.234	Prob(JB):	0.392
Kurtosis:	2.519	Cond. No.	14.4

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [6]:

```
print('Parameters: ', results.params)
print('R2: ', results.rsquared)
```

```
Parameters:  [ 1.34233516 -0.04024948 10.01025357]
R2:  0.9999879365025871
```

In [7]:

```
nsample = 50
sig = 0.5
x = np.linspace(0, 20, nsample)
X = np.column_stack((x, np.sin(x), (x-5)**2, np.ones(nsample)))
beta = [0.5, 0.5, -0.02, 5.]

y_true = np.dot(X, beta)
y = y_true + sig * np.random.normal(size=nsample)

res = sm.OLS(y, X).fit()
print(res.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.9
Model:	OLS	Adj. R-squared:	0.9
Method:	Least Squares	F-statistic:	21
Date:	Thu, 02 Aug 2018	Prob (F-statistic):	6.30e-
Time:	12:16:44	Log-Likelihood:	-34.4
No. Observations:	50	AIC:	76.
Df Residuals:	46	BIC:	84.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.97
x1	0.4687	0.026	17.751	0.000	0.416	0.5
x2	0.4836	0.104	4.659	0.000	0.275	0.6
x3	-0.0174	0.002	-7.507	0.000	-0.022	-0.0
const	5.2058	0.171	30.405	0.000	4.861	5.5

Omnibus:	0.655	Durbin-Watson:	2.8
Prob(Omnibus):	0.721	Jarque-Bera (JB):	0.3
Skew:	0.207	Prob(JB):	0.8
Kurtosis:	3.026	Cond. No.	22

```
1.
==============================================================================
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

In [8]:

```
print('Parameters: ', res.params)
print('Standard errors: ', res.bse)
print('Predicted values: ', res.predict())
```

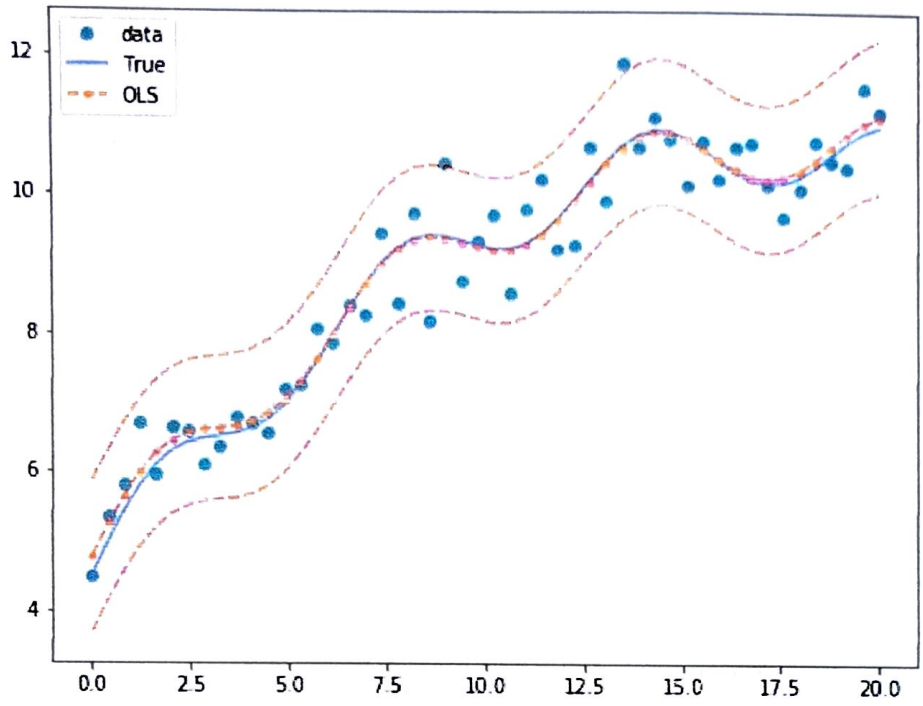
```
Parameters:  [ 0.46872448  0.48360119 -0.01740479  5.20584496]
Standard errors:  [0.02640602 0.10380518 0.00231847 0.17121765]
Predicted values:  [ 4.77072516  5.22213464  5.63620761  5.98658823  6.25643
234  6.44117491
  6.54928009  6.60085051  6.62432454  6.6518039   6.71377946  6.83412169
  7.02615877  7.29048685  7.61487206  7.97626054  8.34456611  8.68761335
  8.97642389  9.18997755  9.31866582  9.36587056  9.34740836  9.28893189
  9.22171529  9.17751587  9.1833565   9.25708583  9.40444579  9.61812821
  9.87897556 10.15912843 10.42660281 10.65054491 10.8063004  10.87946503
 10.86825119 10.78378163 10.64826203 10.49133265 10.34519853 10.23933827
 10.19566084 10.22490593 10.32487947 10.48081414 10.66779556 10.85485568
 11.01006072 11.10575781]
```

In [9]:

```
prstd, iv_l, iv_u = wls_prediction_std(res)

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(x, y, 'o', label="data")
ax.plot(x, y_true, 'b-', label="True")
ax.plot(x, res.fittedvalues, 'r--.', label="OLS")
ax.plot(x, iv_u, 'r--')
ax.plot(x, iv_l, 'r--')
ax.legend(loc='best')
```

Out[9]:

```
<matplotlib.legend.Legend at 0x1c1d6ba470>
```



Time-Series Analysis

In [10]:

```
from statsmodels.tsa.arima_process import arma_generate_sample
```

In [11]:

```
# Gerando dados
np.random.seed(12345)
arparams = np.array([.75, -.25])
maparams = np.array([.65, .35])
```

In [12]:

```
# Parâmetros
arparams = np.r_[1, -arparams]
maparam = np.r_[1, maparams]
nobs = 250
y = arma_generate_sample(arparams, maparams, nobs)
```

In [13]:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
dates = sm.tsa.datetools.dates_from_range('1980m1', length=nobs)
y = pd.Series(y, index=dates)
arma_mod = sm.tsa.ARMA(y, order=(2,2))
arma_res = arma_mod.fit(trend='nc', disp=-1)
```

```
/Users/dmpm/anaconda3/lib/python3.6/site-packages/statsmodels/tsa/base/tsa_m
odel.py:171: ValueWarning: No frequency information was provided, so inferre
d frequency M will be used.
  % freq, ValueWarning)
```

In [14]:

```
print(arma_res.summary())
```

```
                              ARMA Model Results
==============================================================================
==
Dep. Variable:                      y   No. Observations:                    2
50
Model:                     ARMA(2, 2)   Log Likelihood                -245.8
87
Method:                       css-mle   S.D. of innovations              0.6
45
Date:                Thu, 02 Aug 2018   AIC                            501.7
73
Time:                        12:16:44   BIC                            519.3
81
Sample:                    01-31-1980   HQIC                           508.8
60
                         - 10-31-2000
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
ar.L1.y        0.8411      0.403      2.089      0.038       0.052       1.6
30
ar.L2.y       -0.2693      0.247     -1.092      0.276      -0.753       0.2
14
ma.L1.y        0.5352      0.412      1.299      0.195      -0.273       1.3
43
ma.L2.y        0.0157      0.306      0.051      0.959      -0.585       0.6
16
                                    Roots
==============================================================================
=
                  Real          Imaginary           Modulus         Frequenc
y
------------------------------------------------------------------------------
-
AR.1            1.5618           -1.1289j            1.9271           -0.099
6
AR.2            1.5618           +1.1289j            1.9271            0.099
6
MA.1           -1.9835           +0.0000j            1.9835            0.500
0
MA.2          -32.1800           +0.0000j           32.1800            0.500
0
------------------------------------------------------------------------------
-
```

mais cobiçados do mercado de análise de dados. Clique no link abaixo, faça sua inscrição, comece hoje mesmo e aumente sua empregabilidade:

https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados
(https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados)

Fim

Obrigado - Data Science Academy - facebook.com/dsacademybr (http://facebook.com/dsacademybr)

Data Science Academy - Python Fundamentos - Capítulo 8

Download: http://github.com/dsacademybr (http://github.com/dsacademybr)

Seaborn

Instalar o Seaborn:

Caso o Seaborn não esteja instalado, abra o prompt de comando ou terminal e digite: pip install seaborn

In [1]:

```
import numpy as np
import pandas as pd
import random
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
import seaborn as sea
```

In [3]:

```
# Carregando um dos datasets que vem com o Seaborn
dados = sea.load_dataset("tips")
```

In [4]:

```
dados.head()
```

Out[4]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

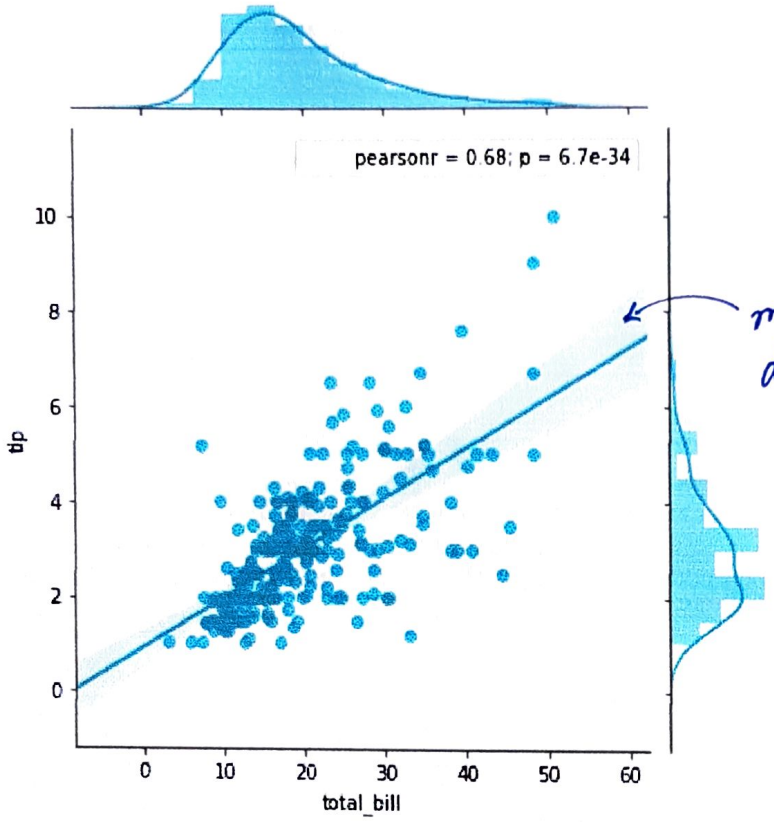
Regressão linear é uma equação para se estimar a condicional (valor esperado)

de uma variável y, dados os valores de algumas outras variáveis x.

In [5]:

```
# O método joinplot cria plot de 2 variáveis com gráficos bivariados e univariados
sea.jointplot("total_bill", "tip", dados, kind = 'reg');
```

```
/Users/dmpm/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6
462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by
the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/dmpm/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6
462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by
the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

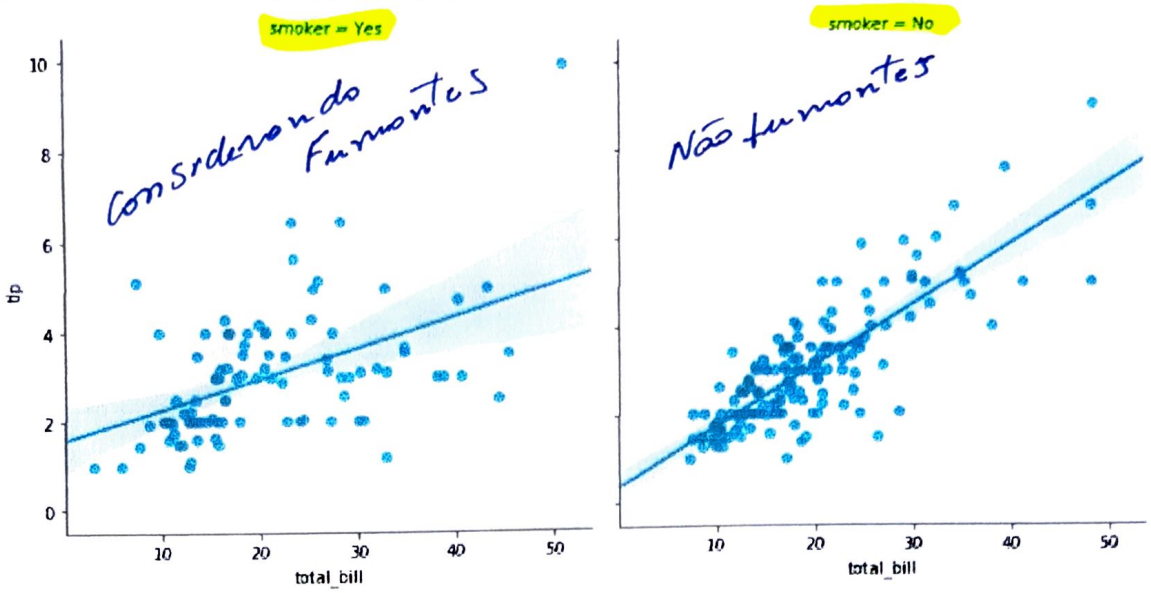


Vários Graf. cos em 1

margem de erro

In [6]:

```
# O método lmplot() cria plot com dados e modelos de regressão.
sea.lmplot("total_bill", "tip", dados, col = "smoker");
```



In [7]:

```
# Construindo um dataframe com Pandas
df = pd.DataFrame()
```

In [8]:

```
# Alimentando o Dataframe com valores aleatórios
df['a'] = random.sample(range(1, 100), 25)
df['b'] = random.sample(range(1, 100), 25)
```

In [9]:

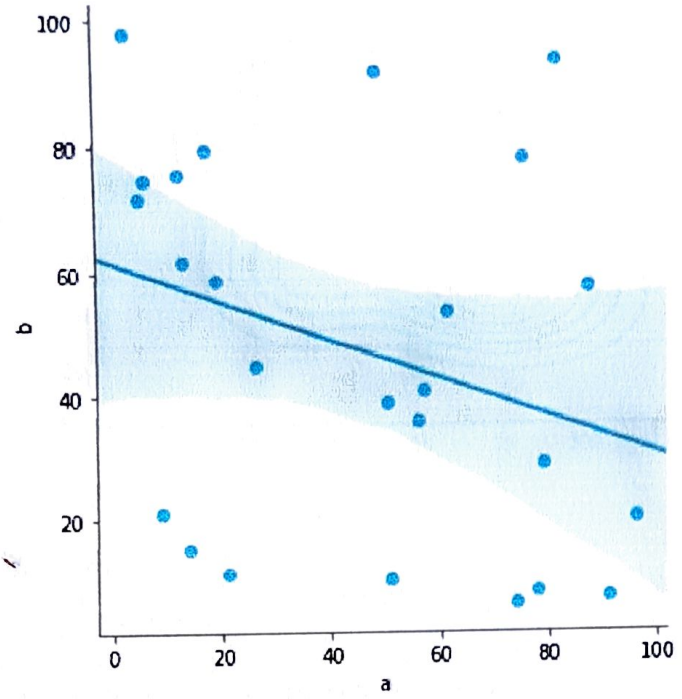
```
df.head()
```

Out[9]:

	a	b
0	78	8
1	96	20
2	17	80
3	26	45
4	87	58

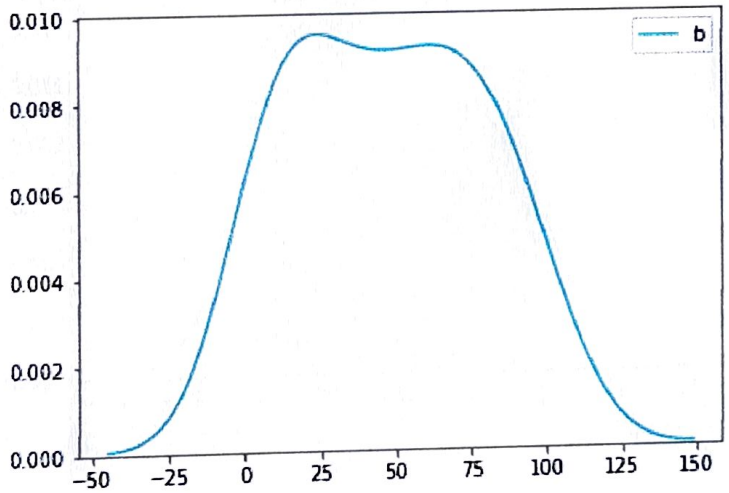
In [10]:

```
# Scatter Plot
sea.lmplot('a', 'b', data = df, fit_reg = True);
```



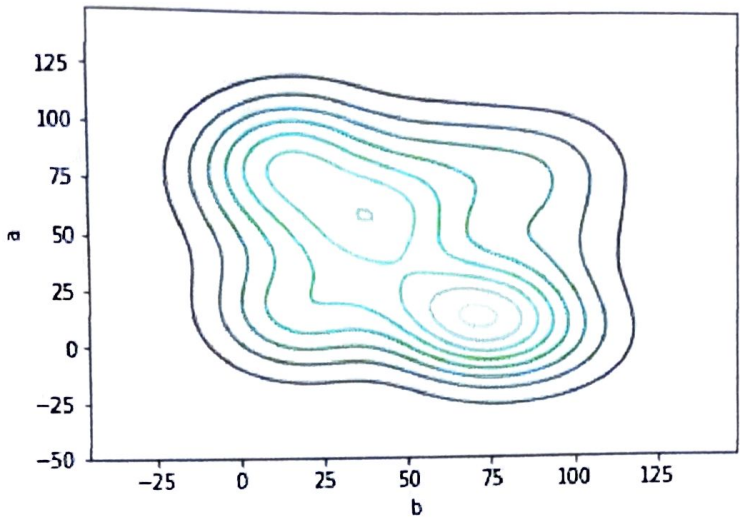
In [11]:

```
# Density Plot
sea.kdeplot(df.b);
```



In [12]:

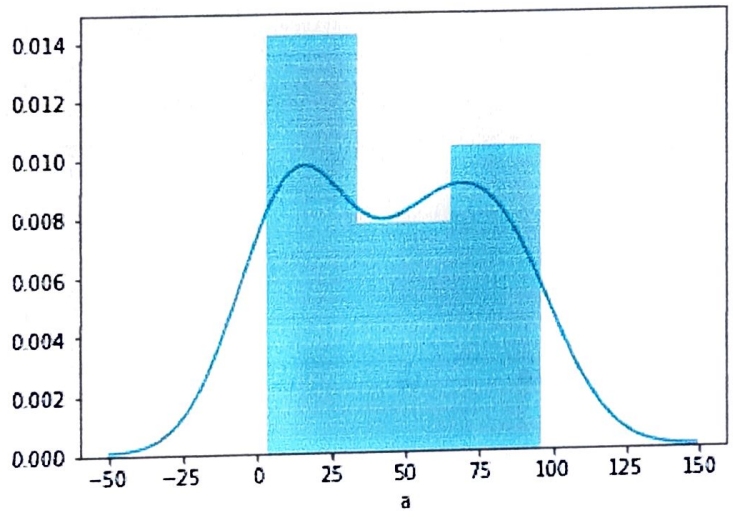
```
sea.kdeplot(df.b, df.a);
```



In [13]:

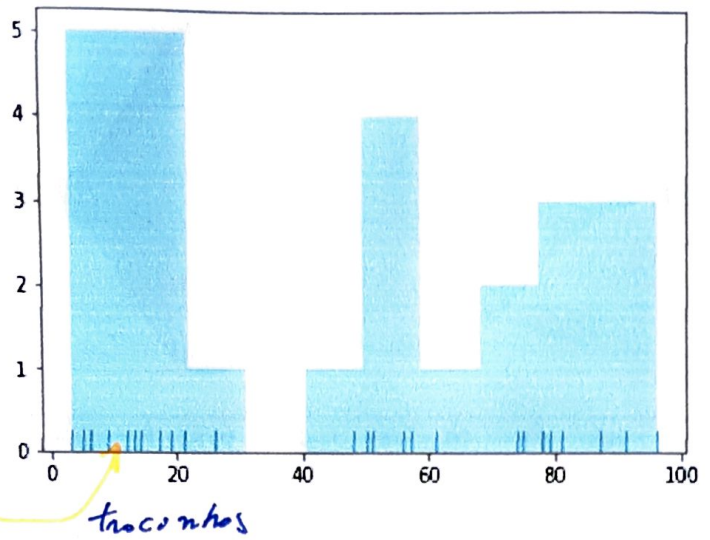
```
sea.distplot(df.a);
```

```
/Users/dmpm/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6
462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by
the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [14]:

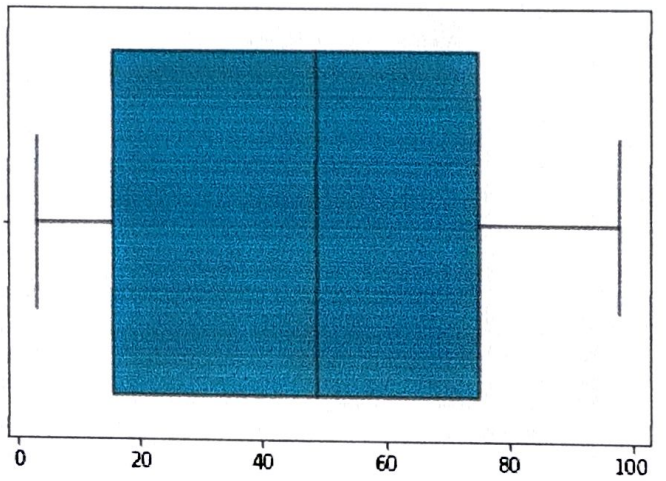
```
# Histograma
plt.hist(df.a, alpha = .3)
sea.rugplot(df.a);
```



traços

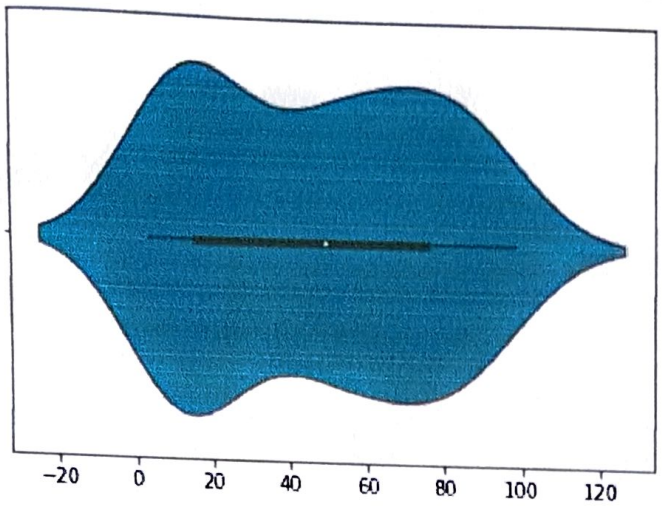
In [15]:

```
# Box Plot
sea.boxplot([df.b, df.a]);
```



In [16]:

```
# Violin Plot
sea.violinplot([df.a, df.b]);
```

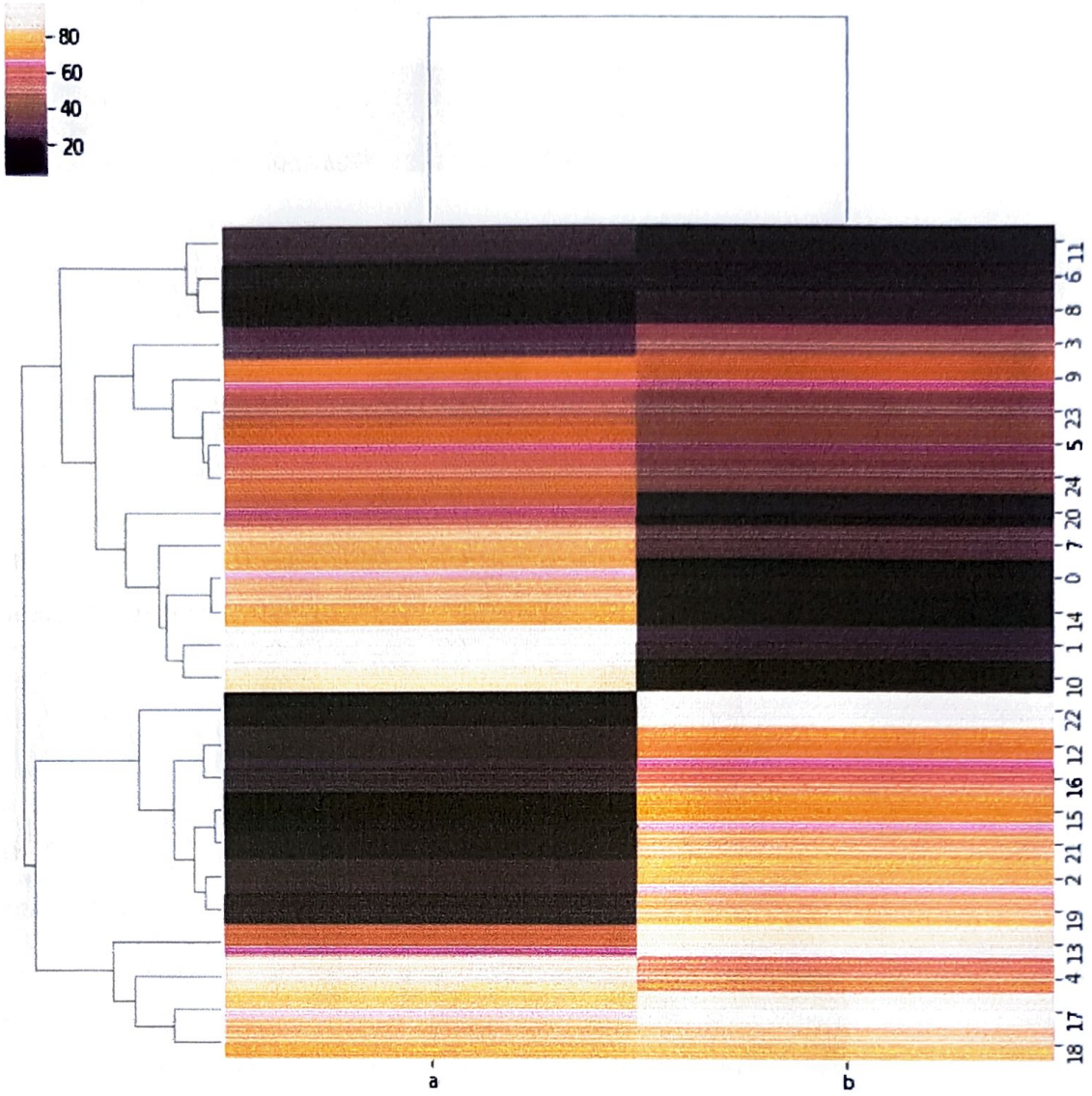


In [17]:

```
# Heatmap
sea.heatmap([df.b, df.a], annot = True, fmt = "d");
```

In [18]:

```
# Clustermap
sea.clustermap(df);
```



In [19]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sea
%matplotlib inline
```

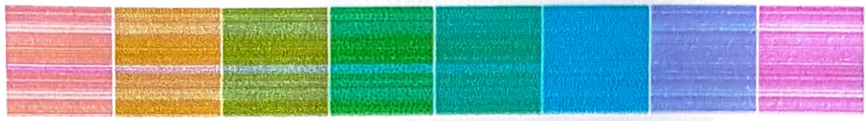
Temas

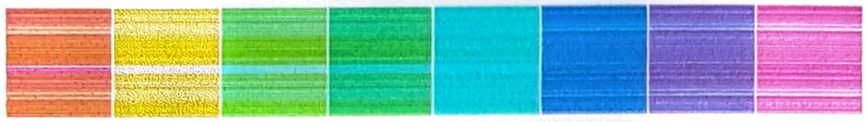
In [20]:

```
# Configurações globais para controlar estilo, tamanho de fonte, cores, etc.
sea.set(context="notebook", style="darkgrid", palette="dark")
```

In [21]:

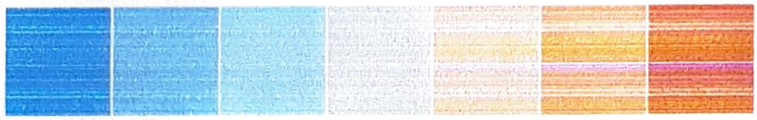
```
# Seaborn possui opções de cores variadas
sea.palplot(sea.color_palette())
sea.palplot(sea.color_palette("husl", 8))
sea.palplot(sea.color_palette("hls", 8))
```





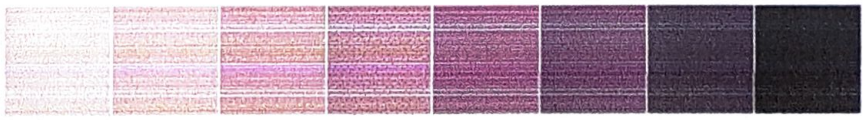
In [22]:

```
sea.palplot(sea.color_palette("coolwarm", 7))
```



In [23]:

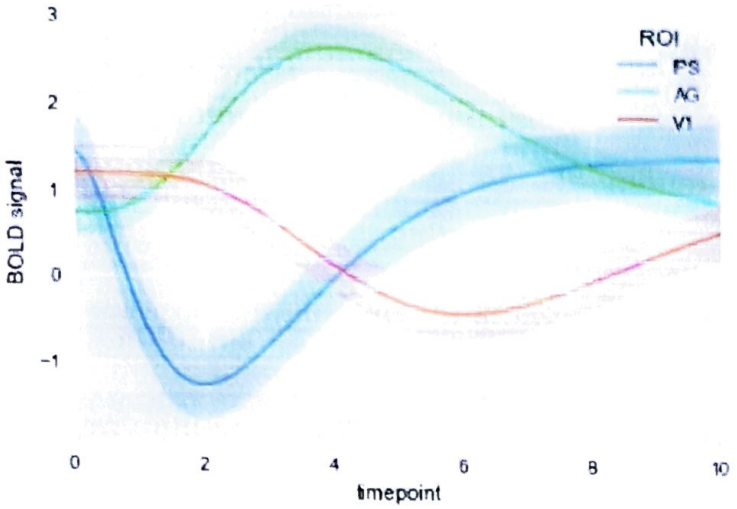
```
sea.palplot(sea.cubehelix_palette(8))
```



In [24]:

```
# O método tsplot cria plots a partir de séries temporais
gammas = sea.load_dataset("gammas")
sea.tsplot(gammas, "timepoint", "subject", "ROI", "BOLD signal", color = "muted");
```

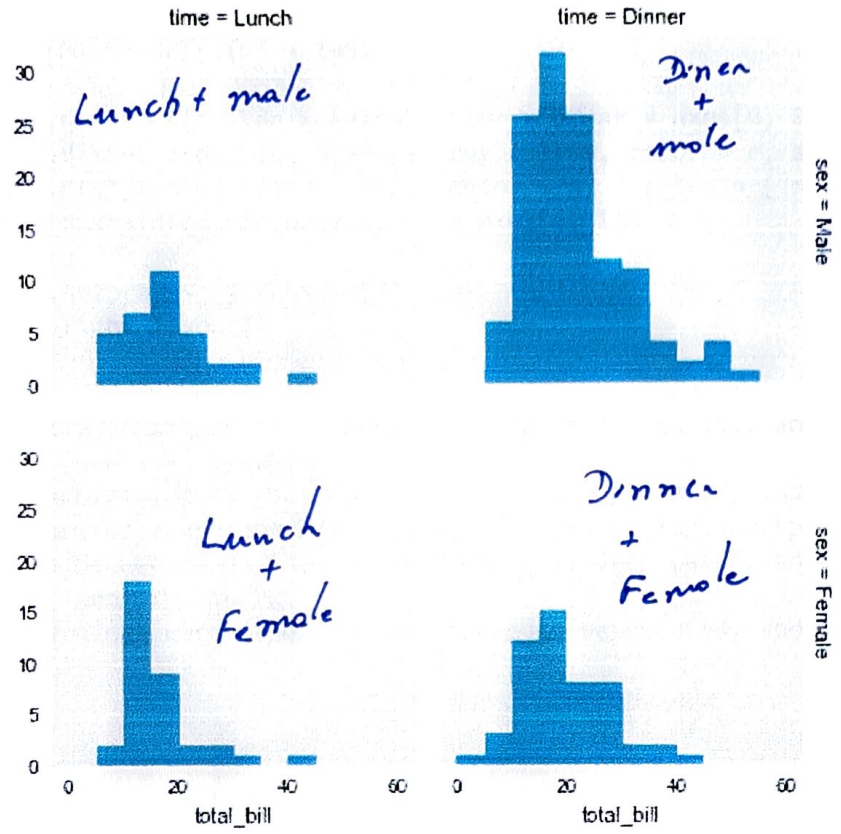
```
/Users/dmpm/anaconda3/lib/python3.6/site-packages/seaborn/timeseries.py:183:
UserWarning: The tsplot function is deprecated and will be removed or replac
ed (in a substantially altered version) in a future release.
  warnings.warn(msg, UserWarning)
```



Outros Plots

In [25]:

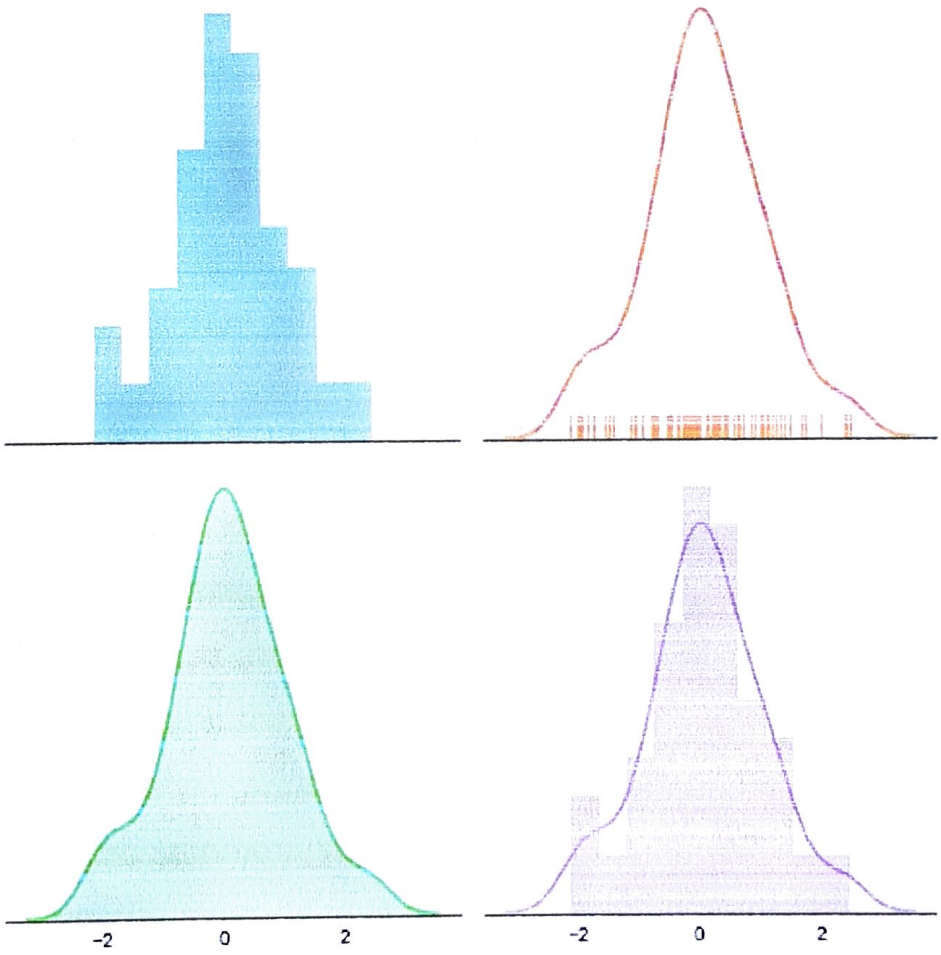
```
# Histogramas com subsets dos dados
sea.set(style = "darkgrid")

dados = sea.load_dataset("tips")
g = sea.FacetGrid(dados, row = "sex", col = "time", margin_titles = True)
bins = np.linspace(0, 60, 13)
g.map(plt.hist, "total_bill", color = "steelblue", bins = bins, lw = 0);
```



In [26]:

```
# Diversos plots simultâneos
sea.set(style = "white", palette = "muted")
f, axes = plt.subplots(2, 2, figsize = (7, 7), sharex = True)
sea.despine(left = True)

rs = np.random.RandomState(10)

b, g, r, p = sea.color_palette("muted", 4)

d = rs.normal(size = 100)

sea.distplot(d, kde = False, color = b, ax = axes[0, 0])
sea.distplot(d, hist = False, rug = True, color = r, ax = axes[0, 1])
sea.distplot(d, hist = False, color = g, kde_kws = {"shade": True}, ax = axes[1, 0])
sea.distplot(d, color = p, ax = axes[1, 1])

plt.setp(axes, yticks = [])
plt.tight_layout()
```

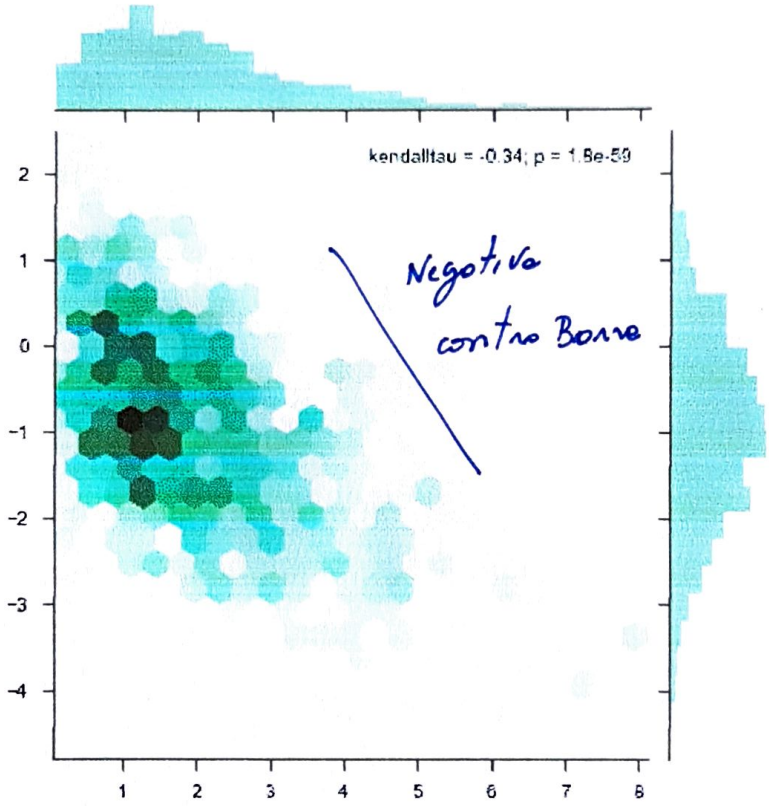
```
/Users/dmpm/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6
462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by
the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/dmpm/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6
462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by
the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [27]:

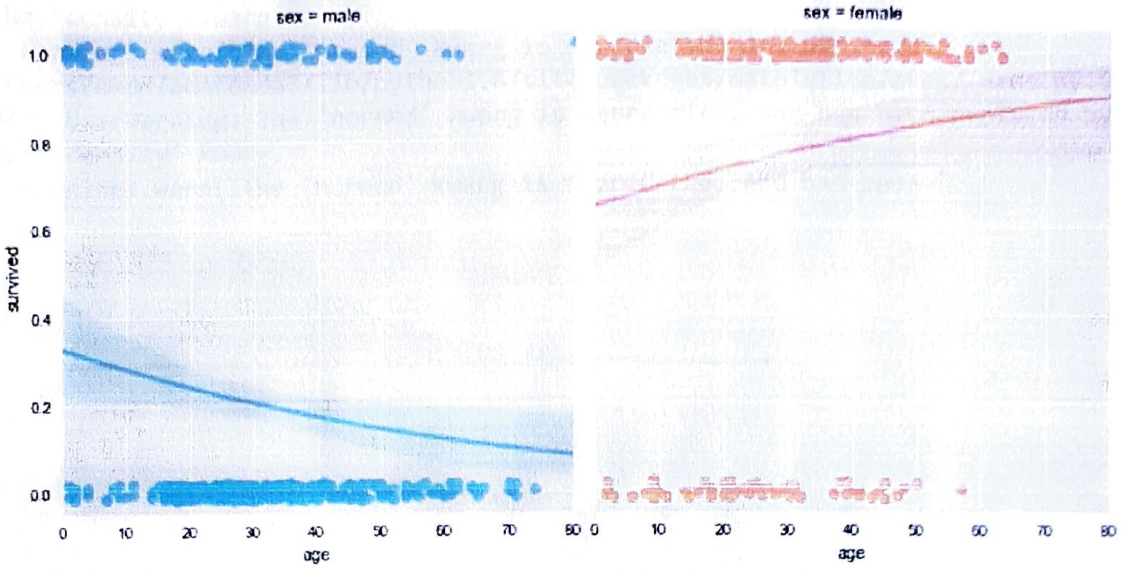
```
# Plot com distribuições marginais
from scipy.stats import kendalltau
sea.set(style="ticks")

rs = np.random.RandomState(11)
x = rs.gamma(2, size = 1000)
y = -.5 * x + rs.normal(size = 1000)
sea.jointplot(x, y, kind = "hex", stat_func = kendalltau, color = "#4CB391");
```

```
/Users/dmpm/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6
462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by
the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/dmpm/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6
462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by
the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [28]:

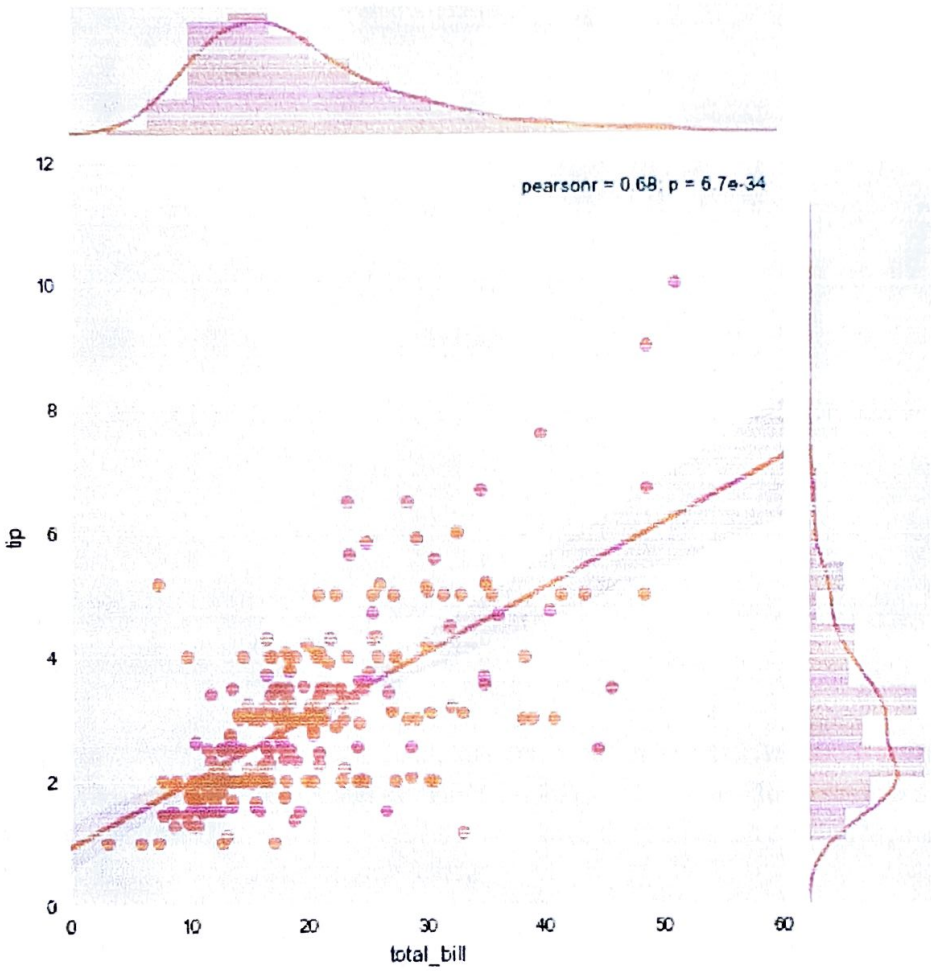
```
# Regressão Logística
sea.set(style = "darkgrid")
df = sea.load_dataset("titanic")

pal = dict(male = "#6495ED", female = "#F08080")
g = sea.lmplot("age", "survived", col = "sex", hue = "sex", data = df, palette = pal,
               y_jitter = .02, logistic = True)
g.set(xlim=(0, 80), ylim = (-.05, 1.05));
```



In [29]:

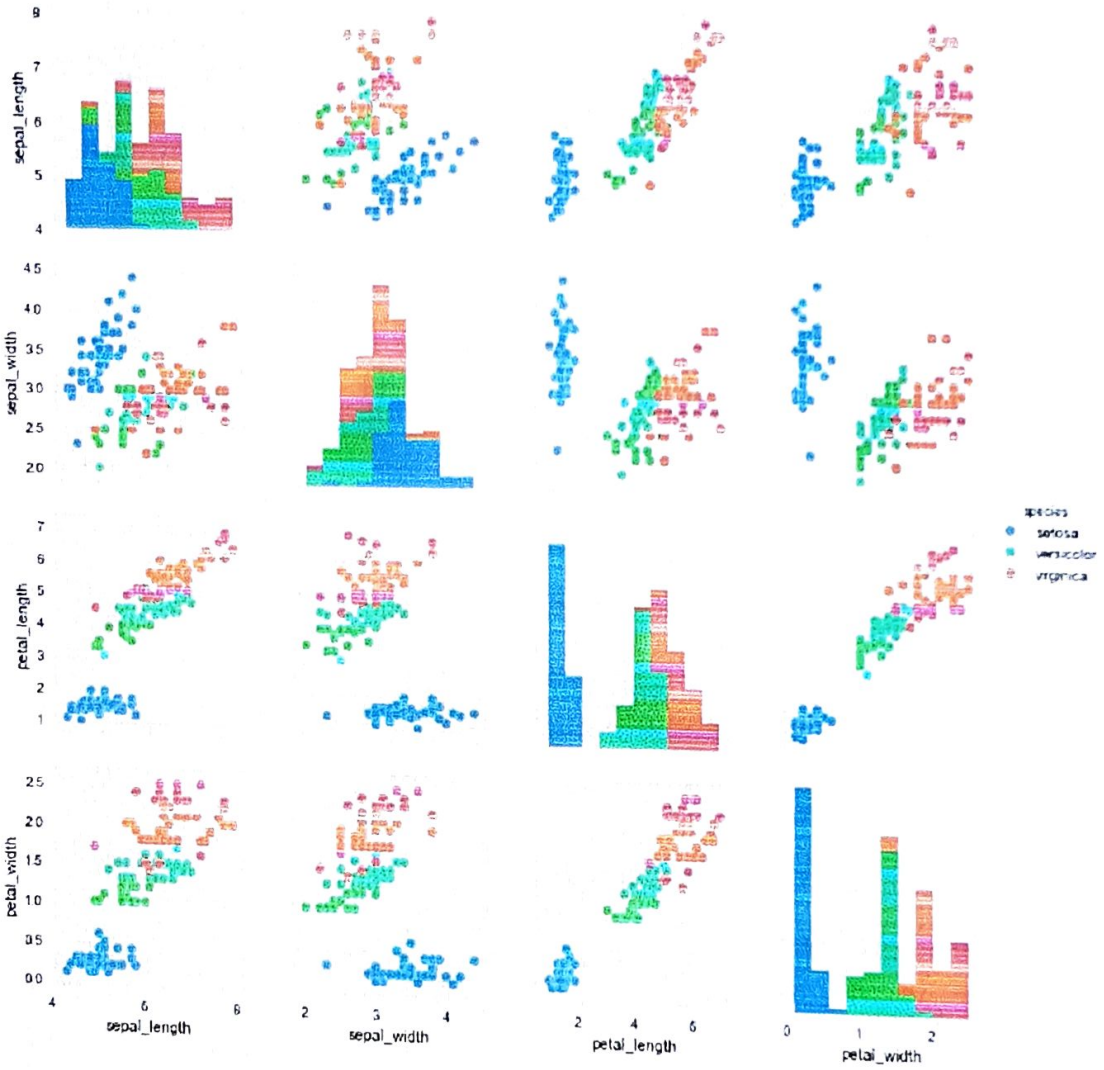
```
# Regressão Linear com Distribuições Marginais
sea.set(style = "darkgrid")
tips = sea.load_dataset("tips")
color = sea.color_palette()[2]
g = sea.jointplot("total_bill", "tip", data = tips, kind = "reg", xlim = (0, 60),
                  ylim = (0, 12), color = color, size = 7);
```

```
/Users/dmpm/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6
462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by
the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
/Users/dmpm/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6
462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by
the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [30]:

```
# Pair Plots
sea.set(style = "darkgrid")
df = sea.load_dataset("iris")
sea.pairplot(df, hue = "species", size = 2.5);
```



Fim

Obrigado - Data Science Academy - facebook.com/dsacademybr (http://facebook.com/dsacademybr)