APUNTES DE PROGRAMACIÓN EN PYTHON

Elaborado por César Chávez

1 MANEJANDO DATOS EN PYTHON

Antes de entrar al análisis econémtrico y financiero, empezaremos por lo esencial, que es la creación, manipulación y estructuración de datos. Partamos por el hecho de que hay algunas palabras que no pueden ser puestos a las variables, palabras como for, import y entre otras. Si colocamos for = 1 y ejecutamos, nos saldrá error:

```
In [144]: for = 1
          and = 1000
          import = 12


            File "<ipython-input-144-412254606233>", line 1
          for = 1
              ^
      SyntaxError: invalid syntax
```

Otros casos en los que hay error, es por ejemplo:

```
In [145]: x: = 1.0
          1X = 1
          X-1 = 1


            File "<ipython-input-145-ce6a57ea6879>", line 1
          x: = 1.0
             ^
      SyntaxError: invalid syntax
```

1.1 TIPOS DE DATOS

Como en cualquier lenguaje de programación hay distintos clases de datos, están los númericos que se dividen en float, integer y complex. Luego están los string que básicamente son textos. También hay otros tipos de datos como boolean, lists, tuples y dictionaries. Empecemos, primero por los númericos:

1.1.1 Númericos

Como ya lo hemos mencionado, se divide en float, integer y complejos.

1.1.1.1 Clase float

Es el más importante tipo de dato scalar para el análisis número. Para ingresar un tipo de dato número de la clase float debemos incluir primero un "."(punto) en la expresión y para saber si una variable "x" es float se le pone type(x):

```
In [146]: x = 1.2
          type(x)

Out[146]: float

In [147]: y = 1
          type(y)

Out[147]: int

In [148]: y = float(1)
          type(y)

Out[148]: float
```

1.1.1.2 Clase integer

Almacena números usando una representación exacta, de modo que no se necesita una aproximación. El costo de la representación exacta es que el tipo de datos enteros no puede expresar nada que no sea un entero, lo que hace que los enteros tengan un uso limitado en la mayoría de los trabajos numéricos.

```
In [149]: x = 1
          type(x)

Out[149]: int

In [150]: y = 1.2
          type(y)

Out[150]: float

In [151]: y = int(1)
          type(y)

Out[151]: int
```

1.1.1.3 Clase complex

Complex son creados en Python usando j o la función complex().

```
In [152]: x = 1j
          type(x)

Out[152]: complex

In [153]:  x = complex(1)
           x

Out[153]: (1+0j)
```

Note que a+bj es lo mismo que complex(a ,b), mientras complex(a) es lo mismo que a +0j.

1.1.2 Tipo de dato String

Strings son delimitados usando citas (”) o comillas (" ") pero no usando la combinación de ellas, es decir, (’") en un único string, excepto cuando se usa para expresar una citación.

```
In [154]:  x = 'abc'
           type(x)

Out[154]: str

In [155]: y = '"Es una cita!"'
          print(y)

"Es una cita!"
```

1.1.2.1 Slicing string

Substrings dentro de un string puede ser accedidos usand slicing. Slicing usa [] para contener los indices de los caracters en un string,
donde el primer indice es 0, y el último es n-1, asumiendo que el string tiene n letras.

```
In [156]: text = 'Python strings are sliceable.'
          len(text)

Out[156]: 29

In [157]: text[0]

Out[157]: 'P'

In [158]: #String entero
          text[:]

Out[158]: 'Python strings are sliceable.'

In [159]: # text[i] : Caracter en la posición i
          text[5]

Out[159]: 'n'

In [160]: # text[i:] : Caracter en la posición i hasta el final
          text[6:]

Out[160]: ' strings are sliceable.'

In [161]: # text[:i] : Caracter en la posición 0 hasta i
          text[:6]

Out[161]: 'Python'

In [162]: # text[j:i] : Caracter en la posición j  hasta i
          text[1:6]
```

```
Out[162]: 'ython'

In [163]: # text[j:i:m] : Caracter j, j+m,..j+ m|(i-m-j)/m|
          text[1:10:2]

Out[163]: 'yhnsr'

In [164]: # text[-i] : Caracter n-i
          text[-5]

Out[164]: 'a'

In [165]: # text[-i:] : Caracter n-i,..n-1
          text[-5:]

Out[165]: 'able.'

In [166]: # text[:-i] : Caracter 0,..., n-i-1
          text[:-5]

Out[166]: 'Python strings are slice'
```

1.1.3 Tipo de dato Boolean

Son usado para representar verdadero o falso, usando las palabras reservadas True y False.

```
In [168]: x = True
          type(x)

Out[168]: bool

In [169]: x = bool(1)
          x

Out[169]: True

In [170]: x = bool(0)
          x

Out[170]: False
```

1.1.4 Tipo de dato Lists

Un list es una recolección de otros objetos(floats, integers, complex, strings e incluso otras listas) Son esenciales para programar en Python y son usados para almacenar colecciones de otros valores. Los Lists apoyan los slicings para recuperar uno o más elementos. Son construidos con "[]" y valores son separados por comas ",".

```
In [172]: x=[1,2,3,4]
          type(x)
```

```
Out[172]: list

In [173]: x

Out[173]: [1, 2, 3, 4]

In [174]: # Listas de listas
          x = [[1,2,3,4], [5,6,7,8]]
          x

Out[174]: [[1, 2, 3, 4], [5, 6, 7, 8]]

In [175]: #Listas no rectangulares
          x = [[1,2,3,4] , [5,6,7]]

In [176]: #Listas mezcladas
          x = [1,1.0,1+0j,'one',None,True]
```

1.1.4.1 Slicing lists

Se puede extraer valores de listas, o listas de listas.

Slice	Retorna	Slice	Retorna
$x[:]$	Todos los valores de la lista x	$x[-i]$	El valor de x_{n-i} excepto cuando i = -0
$x[i]$	El valor de la posición i de la lista x, (x_i)	$x[-i:]$	Los valores desde $x_{n-i}, ..., x_{n-1}$
$x[i:]$	Los valores desde $x_i, .., x_{n-1}$	$x[:-i]$	Los valores desde $x_0, .., x_{n-i-1}$
$x[:i]$	Los valores desde $x_0, ..., x_{i-1}$	$x[-j:-i]$	Los valores desde $x_{n-j}, .., x_{n-i-1}$
$x[i:j]$	Los valores desde $x_i, .., x_{j-1}$	$x[-j:-i:m]$	Los valores desde $x_{n-j}, .., x_{n-j+m\lvert\frac{j-i-1}{m}\rvert}$
$x[i:j:m]$	Los valores desde $x_i, .., x_{i+m\lvert\frac{j-i-1}{m}\rvert}$		

```
In [177]: x = [0,1,2,3,4,5,6,7,8,9]

In [178]: x[0]

Out[178]: 0

In [179]: x[5]
```

```
Out[179]: 5

In [180]: x[4:]

Out[180]: [4, 5, 6, 7, 8, 9]

In [181]: x[:4]

Out[181]: [0, 1, 2, 3]

In [182]: x[1:4]

Out[182]: [1, 2, 3]

In [183]: x[-1]

Out[183]: 9

In [184]: x[-10:-1]

Out[184]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

List pueden ser multidimensional, y slicing puede ser hecho directamente en dimensiones mayores a 1. Por ejemplo, consideremos el siguiente list x = [[1,2,3,4], [5,6,7,8]]. Si un solo indice es usado, x[0] retornará la primera lista, y x[1] retornará la segunda lista. Ya que la lista retornada por x[0] es sliceable (dividible), Podemos dividirla directamente usando, por ejemplo, x[0][0] o x[0][1:4].

```
In [185]: x = [[1,2,3,4], [5,6,7,8]]
          x[0]

Out[185]: [1, 2, 3, 4]

In [186]: x[1]

Out[186]: [5, 6, 7, 8]

In [187]: x[0][0]

Out[187]: 1

In [188]: x[0][1:4]

Out[188]: [2, 3, 4]

In [189]: x[1][-4:-1]

Out[189]: [5, 6, 7]
```

1.1.4.2 Tipo de dato Lists

Un número de funciones son disponibles para manipular listas. Las más útiles son presentadas en la tabla de abajo

Function	Method	Description
list.append(x,value)	x.append(value)	Anexa valor al final de la lista.
len(x)	–	Devuelve el número de elementos en la lista.
list.extend(x,list)	x.extend(list)	Anexa los valores en lista a la lista existente.
list.pop(x,index)	x.pop(index)	Elimina el valor en el índice de posición y devuelve el valor
list.remove(x,value)	x.remove(value)	Elimina la primera aparición de valor de la lista.
list.count(x,value)	x.count(value)	Cuenta el número de ocurrencias de valor en la lista.
del x[slice]		Borra los elementos en la rebanada.

```
In [235]:  x = [0,1,2,3,4,5,6,7,8,9]

In [236]: x.append(0)
          x

Out[236]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]

In [237]: len(x)

Out[237]: 11

In [238]: x.extend([11,12,13])
          x

Out[238]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 11, 12, 13]

In [239]: x.pop(1)
          x

Out[239]: [0, 2, 3, 4, 5, 6, 7, 8, 9, 0, 11, 12, 13]

In [240]: x.remove(0)
          x

Out[240]: [2, 3, 4, 5, 6, 7, 8, 9, 0, 11, 12, 13]
```

Los elementos pueden ser borrados de la lista usando del en combinación con un slice.

```
In [241]: x = [0,1,2,3,4,5,6,7,8,9]
          del x[0]
          x

Out[241]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [242]: del x[:3]
          x

Out[242]: [4, 5, 6, 7, 8, 9]
```

1.1.5 Tipo de datos Tuple

Un tuple es identico a un list, salvo con una diferencia, que un tuple no puede ser cambiado una vez que se crea. No es posible agregar, remover o reemplazar elementos en un tiple. Pero, si un tuple contiene un tipo de dato mutable, por ejemplo, un list, ese tipo de dato puede ser alterado.Tuples son construidos usando parentesis "()".

```
In [243]: x =(0,1,2,3,4,5,6,7,8,9)
          type(x)

Out[243]: tuple

In [244]: x[-10:-5]

Out[244]: (0, 1, 2, 3, 4)

In [245]: x = list(x)
          type(x)

Out[245]: list

In [246]: x = tuple(x)
          type(x)

Out[246]: tuple

In [247]: #El contenido puede ser cambiado
          x= ([1,2],[3,4])
          x[0][1] = -10
          x

Out[247]: ([1, -10], [3, 4])
```

1.1.5.1 Funciones de Tuple

Tuples solo tiene los metodos index y count.

1.1.6 Tipo de dato Dict

Los Dicts son usados para funciones como optimizers. Están compuestos de keys (palabras) y values(definiciones). Los keys deben ser tipo de datos inmutable y único(por ejemplo, strings, integers, tuples que contengan tipos inmutables) y values pueden contener cualquier tipo de data válido en Python.

```
In [248]: data = {'age': 34, 'children' : [1,2], 1: 'apple'}
          type(data)

Out[248]: dict

In [249]:  data['age']

Out[249]: 34

In [250]: #Podemos agregar nuevos keys y values.
          data['name'] = 'abc'

In [251]: #Podemos cambiar
          data['age'] = 'ade'

In [252]: del data['age']
          data

Out[252]: {'children': [1, 2], 1: 'apple', 'name': 'abc'}
```

1.1.7 Sets (set, frozenset)

Sets son colecciones que contienen todos los elementos únicos de una colección. Set y forzenset solo differen en que el último es inmutable, y set es similar a una lista única mientras que frozenset es simular a un tuple único. Mientras los sets no son importante en el análisis númerico, pueden ser útiles con datos desordenados. Por ejemplo, encontrar el set de tickets únicos en una lista larga de tickets.

1.1.7.1 Funciones de set

Los más útiles son.

Función	Método	Descripción
set.add(x,element)	x.add(element)	Anexa elementos al set.
len(x)	–	Devuelve el número de elementos en la lista.
set.difference(x,set)	x.difference(set)	Retorna los elementos en x que no están en el set.
set.intersection(x,set)	x.intersection(set))	Retorna los elementos de x que están en el set.

Función	Método	Descripción
set.remove(x,element)	x.remove(element)	Remueve los elementos del set.
set.union(x,set)	x.union(set)	Retorna el set que contiene todos los elementos de x y set.

```
In [253]: x = set(['MSFT','GOOG','AAPL','HPQ','MSFT'])

In [254]: x

Out[254]: {'AAPL', 'GOOG', 'HPQ', 'MSFT'}

In [255]: x.add('CSCO')
          x

Out[255]: {'AAPL', 'CSCO', 'GOOG', 'HPQ', 'MSFT'}

In [256]: y = set(['XOM', 'GOOG'])
          x.intersection(y)

Out[256]: {'GOOG'}

In [257]: x = x.union(y)
          x

Out[257]: {'AAPL', 'CSCO', 'GOOG', 'HPQ', 'MSFT', 'XOM'}

In [258]: x.remove('XOM')
```

2 Arrays y Matrices

La diferencia entre estos dos tipos de datos son:

Arrays puede tener 1, 2, 3 más dimensiones, y las matrices siempre tienen 2 dimensiones. Esto significa que un vector 1xn almacenado como un array tiene dimensión 1 y n elementos, mientras que el mismo vector almacenado como una matriz tiene 2 dimensiones donde el tamaño de las dimensiones son 1 y n.

Los operadores matemáticos standards sobre arrays operan elemento por elemento. No es el caso para matrces, donde la multiplicación (*) sigue las reglas del algebra lineal. Arrays pueden ser multiplcados usando el simbolo @.

Arrays son más comunes que las matrices, y todas las funciones son testeado con arrays. La misma función deberia funcionar con matrices pero hay la posibilidad de un bug cuando uses matrices.

Arrays son más comunes que las matrices, y todas las funciones son testeado con arrays. La misma función deberia funcionar con matrices pero hay la posibilidad de un bug cuando uses matrices.

2.1 Arrays

Arrays están dentro de la librería Numpy, son similar a lists o tuples. Todos contienen una colección de elementos. Arrays son iniciados desde lists o tuples usando array. Arrays de dos dimensiones son iniciados usando lists de lists, o tuples de lists o lists de tuples y etc. Y arrays de dimensiones mayore a uno pueden ser iniciados con lists o tuples anidadas.

```
In [6]: import numpy as np
        from numpy import array
        from pylab import *
        x = [0.0, 1, 2, 3, 4]
        y = array(x)
        y

Out[6]: array([0., 1., 2., 3., 4.])

In [7]: type(y)

Out[7]: numpy.ndarray

In [8]: #De dimensiones mayores
        y = array([[0.0, 1, 2, 3, 4], [5, 6, 7, 8, 9]])
        y

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

In [9]: shape(y)

Out[9]: (2, 5)

In [10]: y = array([[[1,2],[3,4]],[[5,6],[7,8]]])
         shape(y)

Out[10]: (2, 2, 2)
```

2.2 Matrices

Matrices son un subconjunto de arrays y se comporta de manera identica a ellos. Las dos importante diferencias son:

Arrays puede tener 1, 2, 3 más dimensiones, y las matrices siempre tienen 2 dimensiones. Esto significa que un vector 1xn almacenado como un array tiene dimensión 1 y n elementos, mientras que el mismo vector almacenado como una matriz tiene 2 dimensiones donde el tamaño de las dimensiones son 1 y n.

Matrices siempre tienen dos dimensiones.

Matrices sigue las reglas del algebra lineal por *.

Arrays de 1 y 2 dimensiones pueden ser copiado a una matriz llamando la matriz sobre un array. Alternativamente, mat o asmatrix provee un método más rápido para coercionar un array a comportarse como una matriz sin copiar cualquier dato.

```
In [11]: x = [0.0, 1, 2, 3, 4] # Cualquier float hace otro float
         y = array(x)
         type(y)

Out[11]: numpy.ndarray

In [12]: y * y # Elemento-por-elemento
         type(y)

Out[12]: numpy.ndarray

In [13]: z = np.asmatrix(x)
         type(z)

Out[13]: numpy.matrixlib.defmatrix.matrix

In [14]: z * z # Error


         ---------------------------------------------------------------------------

         ValueError                                Traceback (most recent call last)

         <ipython-input-14-729fd3bfa7a3> in <module>
     ----> 1 z * z # Error


         ~\Anaconda3\lib\site-packages\numpy\matrixlib\defmatrix.py in __mul__(self, other)
         213         if isinstance(other, (N.ndarray, list, tuple)) :
         214             # This promotes 1-D vectors to row vectors
     --> 215             return N.dot(self, asmatrix(other))
         216         if isscalar(other) or not hasattr(other, '__rmul__') :
         217             return N.dot(self, other)


         ValueError: shapes (1,5) and (1,5) not aligned: 5 (dim 1) != 1 (dim 0)
```

2.3 Arrays de una dimension

Un vector:

```
x = [1 2 3 4 5]
```

Es ingresado como un array de una dimensión usando:

```
In [15]:  x=array([1.0, 2.0, 3.0, 4.0, 5.0])

In [16]: ndim(x)
```

```
Out[16]: 1
```

Una matriz es siempre dos dimensiones

```
In [17]: x=matrix([1.0,2.0,3.0,4.0,5.0])
         x

Out[17]: matrix([[1., 2., 3., 4., 5.]])
```

Note que la representación de la matriz usa listas anidadas ([[]]) para enfatizar la estructura de dos dimensiones de todas las matrices. El vector columna es:

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

Es ingresado como una matriz o un array de dos dimensiones usando un set de listas anidadas.

```
In [18]: x=matrix([[1.0],[2.0],[3.0],[4.0],[5.0]])
         x

Out[18]: matrix([[1.],
                 [2.],
                 [3.],
                 [4.],
                 [5.]])

In [19]: x = array([[1.0],[2.0],[3.0],[4.0],[5.0]])
         x

Out[19]: array([[1.],
                [2.],
                [3.],
                [4.],
                [5.]])
```

2.4 Arrays de dos dimensiones

Matrices y arrays de dos dimensiones son filas de columnas and 2-dimensional arrays are rows of columns

$$x = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 5 & & \end{pmatrix}$$

```
In [20]: x = array([[1.0,2.0,3.0],[4.0,5.0,6.0],[7.0,8.0,9.0]])
         x

Out[20]: array([[1., 2., 3.],
                [4., 5., 6.],
                [7., 8., 9.]])
```

2.5 Concatenación

Concatenación es el proceso por el que un vector o matriz está añadido a otro. Arrays y matrices pueden ser concatenados horizontalmente o verticalmente. Por ejemplo:

$$x = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$y = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

$$z = \begin{pmatrix} 9 \\ 10 \end{pmatrix}$$

Si queremos concatenar x e y como elementos de un nuevo array podemos hacerlo con la función concatenate. Los inputos para concatenar deben ser agrupados en un tuple y el argumento axis que especifica si el array es vertical (axis = 0) u horizontal(axis = 1).

```
In [21]: x = array([[1.0,2.0],[3.0,4.0]])
         y = array([[5.0,6.0],[7.0,8.0]])
         z = concatenate((x,y),axis = 0)
         z

Out[21]: array([[1., 2.],
                [3., 4.],
                [5., 6.],
                [7., 8.]])

In [22]: z = concatenate((x,y),axis = 1)
         z

Out[22]: array([[1., 2., 5., 6.],
                [3., 4., 7., 8.]])
```

Concatenar es equivalente a programar formas de bloques de matriz en algebra lineal.Alternativamente, las funciones vstack y hstack pueden ser usado para arrays verticales y horizontales, respectivamente.

```
In [23]: z = vstack((x,y)) # Lo mismo que z = concatenate((x,y),axis = 0)
         z

Out[23]: array([[1., 2.],
                [3., 4.],
                [5., 6.],
                [7., 8.]])

In [24]: z = hstack((x,y)) # Lo mismo que z = concatenate((x,y),axis = 1)
         z

Out[24]: array([[1., 2., 5., 6.],
                [3., 4., 7., 8.]])
```

2.6 Accediendo a elementos de Arrays

Cuatro métodos están disponibles para acceder a elementos contenidos dentro de un array: scalar selection, slicing, numerical indexing and logical indexing. Los primeros dos son los más simples.

2.6.1 Scalar Selection

Es implementado usando [i] para arrays de una dimensión, [i,j] para arrays de dos dimensiones y [i1,i2,..,in] para arrays de n dimensiones. Como toda indexación en Python, selección está basada en 0, por lo que [0] es el primer elemento en un array de 1 dimensión, [0,0] para 2 dimensiones y así.

```
In [25]: x = array([1.0,2.0,3.0,4.0,5.0])
         x[0]

Out[25]: 1.0

In [26]: x = array([[1.0,2,3],[4,5,6]])
         x[1, 2]

Out[26]: 6.0
```

2.6.2 Array slicing

Arrays slicing es idéntico a list slicing. Slicing básico de arrays de 1-dimensión es idéntico a slicing un simple list.

```
In [27]: x = array([1.0,2.0,3.0,4.0,5.0])

In [31]: y = x[:]
         y

Out[31]: array([1., 2., 3., 4., 5.])

In [32]: y = x[:2]
         y

Out[32]: array([1., 2.])

In [33]: y = x[1::2]
         y

Out[33]: array([2., 4.])
```

Slicing básico de arrays de 2-dimensión es idéntico a slicing un simple list. Ña primera dimensión específica la fila o filas del slice y la segunda dimensión específica la columna o las columnas. Note que la sintaxis del slice de dos dimensiones y[a:b,c:d] es lo mismo que y[a:b,:][:,c:d] or y[a:b][:,c:d], En el caso donde solo el slicing de la fila es necesario y[a:b], es el equivalente de y[a:b,:].

```
In [35]: y = array([[0.0, 1, 2, 3, 4],[5, 6, 7, 8, 9]])
         y
```

```
Out[35]: array([[0., 1., 2., 3., 4.],
                [5., 6., 7., 8., 9.]])

In [36]: y[:1,:] # Fila 0, todas las columnas

Out[36]: array([[0., 1., 2., 3., 4.]])

In [37]: y[:1] # Lo mismo que y[:1,:]

Out[37]: array([[0., 1., 2., 3., 4.]])

In [38]: y[:,:1] # Todas las filas, columna 0

Out[38]: array([[0.],
                [5.]])

In [39]: y[:1,0:3] # Fila 0,las columnas de 0 a 2.

Out[39]: array([[0., 1., 2.]])

In [40]: y[:1][:,0:3]

Out[40]: array([[0., 1., 2.]])

In [41]: y[:,3:] # Todas las filas, columnas 3 y 4

Out[41]: array([[3., 4.],
                [8., 9.]])

In [42]:  y = array([[[1.0,2],[3,4]],[[5,6],[7,8]]])
```

2.6.3 Mixed Selection using Scalar and Slice Selectors

Cuando los arrays tienen mas que 1-dimension, es a menudo util mezclar selectores de slice y scalar para seleccionar una fila entera, columna o panel de un array de 3 dimensiones.

```
In [43]: x = array([[1.0,2],[3,4]])
         x[:1,:] # Fila 1, todas las columnas, 2-dimensiones.

Out[43]: array([[1., 2.]])

In [44]: x[0,:] # Fila 1, todas las columnas, dimensión reducida.

Out[44]: array([1., 2.])
```

El principio adoptado por Numpy es que slicing debería también preservar la dimensión del array, meintras scalar indexing regresa un scalar (o array de dimensión 0) ya que ambas selecciones son escalares.

```
In [46]: x = array([[0.0, 1, 2, 3, 4],[5, 6, 7, 8, 9]])
         x[:1,:] # fILA 0, todas las columnas, 2 dimensiones.
```

```
Out[46]: array([[0., 1., 2., 3., 4.]])

In [47]: ndim(x[:1,:])

Out[47]: 2

In [48]: x[0,:] # Fila 0, todas las columnas,

# reducción de la dimensión a un array de 1 dimensión.

Out[48]: array([0., 1., 2., 3., 4.])

In [49]: ndim(x[0,:])

Out[49]: 1

In [50]: x[0,0] # Elementos de la parte izquierda superior

 # reducción de la dimensión a scalar de 0 dimensiones.

Out[50]: 0.0

In [51]: ndim(x[0,0])

Out[51]: 0
```

2.6.4 Asignación usando Slicing

Slicing y scalar selection pueden ser usado para asignar arrays que tienen la misma dimensión como el slice.

```
In [52]: x = array([[0.0]*3]*3) # *3 repite la lista 3 veces.
         x

Out[52]: array([[0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.]])

In [53]: x[0,:] = array([1.0, 2.0, 3.0])
         x

Out[53]: array([[1., 2., 3.],
                [0., 0., 0.],
                [0., 0., 0.]])

In [54]: x[::2,::2]
         x

Out[54]: array([[1., 2., 3.],
                [0., 0., 0.],
                [0., 0., 0.]])
```

```
In [55]: x[::2,::2] = array([[-99.0,-99],[-99,-99]]) # 2 por 2
         x

Out[55]: array([[-99.,   2., -99.],
                [  0.,   0.,   0.],
                [-99.,   0., -99.]])

In [56]: x[1,1] = pi
         x

Out[56]: array([[-99.        ,   2.        , -99.        ],
                [  0.        ,   3.14159265,   0.        ],
                [-99.        ,   0.        , -99.        ]])
```

3 Matemática Básica

3.1 Operadores

1. $+$ es suma
2. $*$ es multiplicación.
3. / es división.
4. // es división entera.
5. $**$ es exponenciación.

3.2 Multiplicación de matrices(@)

@ solo puede ser usado para dos arrays y no puede ser usado para multiplicar un array y un scalar. Si y es NxM y z es LxK, y ambos son matrices no-escalares, y @ z requiere M = K. Similarmente, z @ y requiere L = N.

```
In [57]: x = array([[1.0, 2],[ 3, 2], [3, 4]])
         y = array([[9.0, 8],[7, 6]])
         x @ y

Out[57]: array([[23., 20.],
                [41., 36.],
                [55., 48.]])

In [58]: x.dot(y)

Out[58]: array([[23., 20.],
                [41., 36.],
                [55., 48.]])
```

3.3 Funciones matemáticas(@)

```
In [60]: x = randn(3,4) #Matriz aleatoria 3x4

In [61]: sum(x) # Suma de todos los elementos

Out[61]: 2.185808465272082

In [62]: sum(x, 0) #Suma de todo por columna.

Out[62]: array([-0.37780255,  1.50360181, -1.55604367,  2.61605288])

In [63]: sum(x, 1) # A través de columnas

Out[63]: array([ 2.00375335,  0.57469738, -0.39264227])

In [64]: cumsum(x,0)

Out[64]: array([[-1.09620644,  0.31167733, -0.24476973,  3.03305219],
                [ 0.2166569 ,  0.94954242, -1.10484485,  2.51709626],
                [-0.37780255,  1.50360181, -1.55604367,  2.61605288]])
```

sum y cumsum pueden ser usado como función o como métodos. Cuando es usado como método, el primer input es el axis, sum(x,0) y cuando es función x.sum(0). prod y cumprod se comportan igual salvo que es para multiplicación.

diff computa la diferncia finita de un vector, también array y retorna un elemento vector n-1 cuando es usado un elemento vectorn. diff opera en la última axis por default, y diff(x) opera entre columnas y retorna x[:,1:size(x,1)]-x[:,:size(x,1)-1] para un array de dos dimensiones. diff toma un argumento llamado axis, diff(x,axis=0) operará a través de filas. diff puede también ser usado para producir diferencias de orden mayores.

```
In [65]: x= randn(3,4)
         x

Out[65]: array([[-1.01257464,  0.25813703,  0.52899415,  1.45595481],
                [ 0.47485632, -0.73886951,  0.82294743, -0.52791372],
                [ 1.43336703, -1.13167468, -0.34470762,  0.3615943 ]])

In [66]: diff(x) # lo mismo que diff(x,1)

Out[66]: array([[ 1.27071167,  0.27085712,  0.92696066],
                [-1.21372583,  1.56181694, -1.35086115],
                [-2.56504171,  0.78696706,  0.70630191]])

In [67]: diff(x, axis=0)

Out[67]: array([[ 1.48743096, -0.99700653,  0.29395328, -1.98386852],
                [ 0.95851071, -0.39280517, -1.16765505,  0.88950801]])

In [68]: diff(x, 2, axis=0)

Out[68]: array([[-0.52892025,  0.60420136, -1.46160833,  2.87337653]])
```

3.4 Otros operadores

1. *exp* retorna el exponencial elemento-por-elemento para un array.
2. *log* retorna logaritmo natural elemento-por-elemento (ln(x)) para un array.
3. *log*10 retorna el logaritmo de base 10 elemento-por-elemento (log10(x)) para una array.
4. *sqrt* retorna la raíz cuadrada elemento-por-elemento para un array.
5. *square* retorna el cuadrado elemento-por-elemento para un array, y es equivalente que llamar $x * *2.0$ cuando x es un array.
6. *abs*, *absolute* retorna valor absoluto elemento-por elemento para un array.

3.5 Otras funciones relevantes operadores

sort

```
In [70]: x = randn(4,2)

In [71]: sort(x)

Out[71]: array([[-0.45911002,  1.04996516],
                [-0.42495815,  1.08403401],
                [-0.73332976, -0.22981448],
                [-0.40375451, -0.19316637]])
```

unique

```
In [72]: x = repeat(randn(3),(2))
         x

Out[72]: array([0.34499627, 0.34499627, 1.0152664 , 1.0152664 , 1.62197351,
                1.62197351])

In [73]: unique(x)

Out[73]: array([0.34499627, 1.0152664 , 1.62197351])
```

max, amax, argmax, min, amin, argmi

```
In [74]: x = randn(3,4)

In [75]: amax(x)

Out[75]: 1.8118939895827622

In [76]: x.max()

Out[76]: 1.8118939895827622

In [77]: x.max(0)

Out[77]: array([-0.2829011 ,  1.30644239,  1.81189399,  0.32864409])

In [78]: x = randn(4)
         y = randn(4)
         maximum(x,y)

Out[78]: array([ 0.86792572,  0.42814959, -0.56089581,  1.80662672])
```

4 Arrays especiales

ones

```
In [80]: M, N = 5, 5
         x = ones((M,N))
         x

Out[80]: 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.]])
```

zeros

```
In [81]: x = zeros((M,N))
         x

Out[81]: array([[0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.]])
```

empty

```
In [82]: x = empty((M,N))
         x

Out[82]: array([[0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.]])
```

shape
Retorna el tamaño del array o matriz.

```
In [83]: x = randn(4,3)
         x.shape

Out[83]: (4, 3)
```

reshape transforma un array con un conjunto de dimensiones y a uno con un conjunto diferente, preservando el numero de elementos.

```
In [84]: x = array([[1,2],[3,4]])
         y = reshape(x,(4,1))
         y
```

```
Out[84]: array([[1],
                [2],
                [3],
                [4]])
```

size retorna el número total de elementos.

```
In [85]: x = randn(4,3)
         size(x)

Out[85]: 12
```

ravel convierte array de dimensiones mayores a 1 a 1 dimensión.

```
In [86]: x = array([[1,2],[3,4]])
         x

Out[86]: array([[1, 2],
                [3, 4]])

In [87]: x.ravel()

Out[87]: array([1, 2, 3, 4])
```

vstack, hstack vstack y hstack apilan matrices y matrices compatibles vertical y horizontalmente, respectivamente. Las matrices son compatibles con vstack si tienen el mismo número de columnas y son compatibles con hstack si tienen el mismo número de filas.

```
In [88]: x = reshape(arange(6),(2,3))
         y

Out[88]: array([[1],
                [2],
                [3],
                [4]])

In [89]: y = x
         vstack((x,y))

Out[89]: array([[0, 1, 2],
                [3, 4, 5],
                [0, 1, 2],
                [3, 4, 5]])

In [90]: hstack((x,y))

Out[90]: array([[0, 1, 2, 0, 1, 2],
                [3, 4, 5, 3, 4, 5]])
```

diag retorna la diagonal.

```
In [91]: x = array([[1,2],[3,4]])
         x

Out[91]: array([[1, 2],
                [3, 4]])

In [92]: diag(x)

Out[92]: array([1, 4])
```

triu y tril produce arrays triangulares superior e inferior, respectivamente.

```
In [93]: x = array([[1,2],[3,4]])
         triu(x)

Out[93]: array([[1, 2],
                [0, 4]])

In [94]: tril(x)

Out[94]: array([[1, 0],
                [3, 4]])
```

5 Importación de Datos

Para importar y exportar datos primero debemos llamar a la librería pandas.

```
In [73]: import pandas as pd
         import os
         os.chdir('C:/Users/CESAR/Desktop/Python_introduccion')
         #import csv
         from pandas import read_csv
         csv_data = read_csv('FTSE_1984_2012.csv')
         #import xls
         from pandas import read_excel
         excel_data = read_excel('FTSE_1984_2012.xls','FTSE_1984_2012')
         #import stata files
         from pandas import read_stata
         stata_data = read_stata('FTSE_1984_2012.dta')
         #import Matlab files
         import scipy.io as sio
         matData = sio.loadmat('FTSE_1984_2012.mat')
```

6 Tiempo y Fecha

La manipulación de fecha y hora es proporcionada por un módulo incorporado de Python llamado datetime..

```
In [75]: import datetime as dt
         yr, mo, dd = 2012, 12, 21
         dt.date(yr, mo, dd)

Out[75]: datetime.date(2012, 12, 21)

In [76]: hr, mm, ss, ms= 12, 21, 12, 21
         dt.time(hr, mm, ss, ms)

Out[76]: datetime.time(12, 21, 12, 21)

In [77]: dt.datetime(yr, mo, dd, hr, mm, ss, ms)

Out[77]: datetime.datetime(2012, 12, 21, 12, 21, 12, 21)
```

7 Gráficos

Matplotlib es una biblioteca de trazado completa capaz de gráficos de alta calidad. Matplotlib contiene funciones de alto nivel que producen tipos específicos de figuras, por ejemplo, un gráfico de líneas simple o un gráfico de barras, así como una API de bajo nivel para crear gráficos altamente personalizados.

```
In [78]: import matplotlib.pyplot as plt
         import scipy.stats as stats
```

7.1 seaborn

seaborn es un paquete de Python que proporciona una serie de gráficos avanzados de datos visualizados. También proporciona una mejora general en la apariencia predeterminada de las parcelas producidas con matplotlib, por lo que recomiendo su uso por defecto.

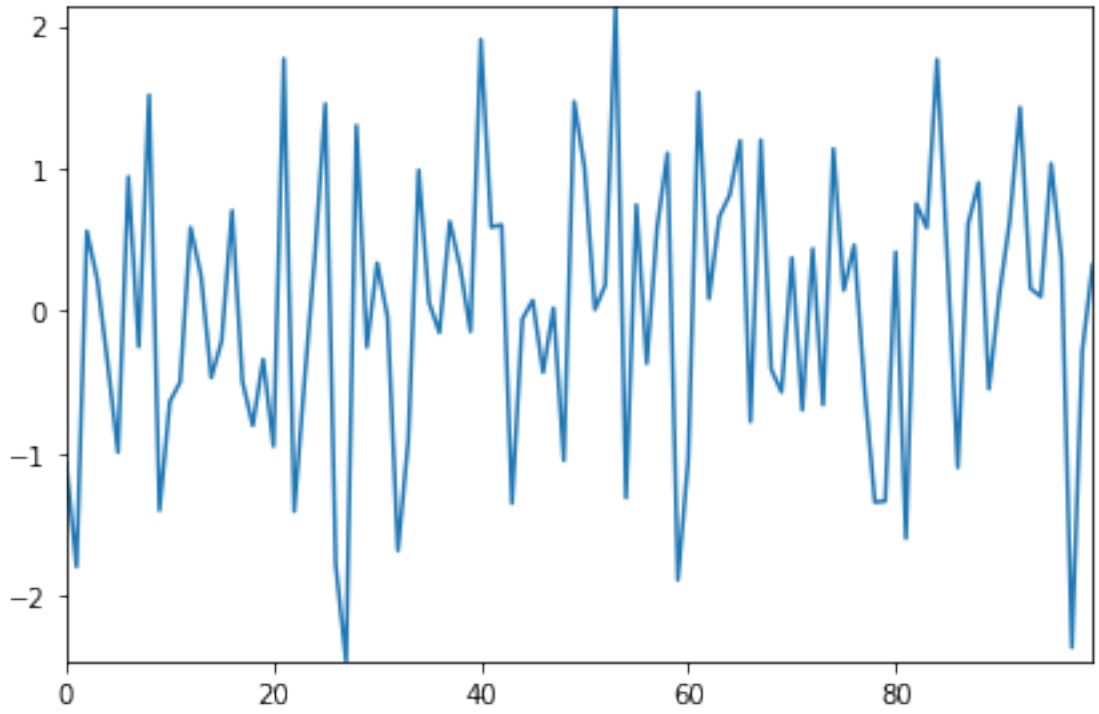
```
In [79]: import seaborn as sns
```

7.2 Gráficos en 2D

7.2.1 Gráficos de Linea

El gráfico 2D más básico y, a menudo, más útil es un trazado de líneas. Los trazados de líneas básicas se producen utilizando un trazado utilizando una sola entrada que contiene una matriz de 1 dimensión.

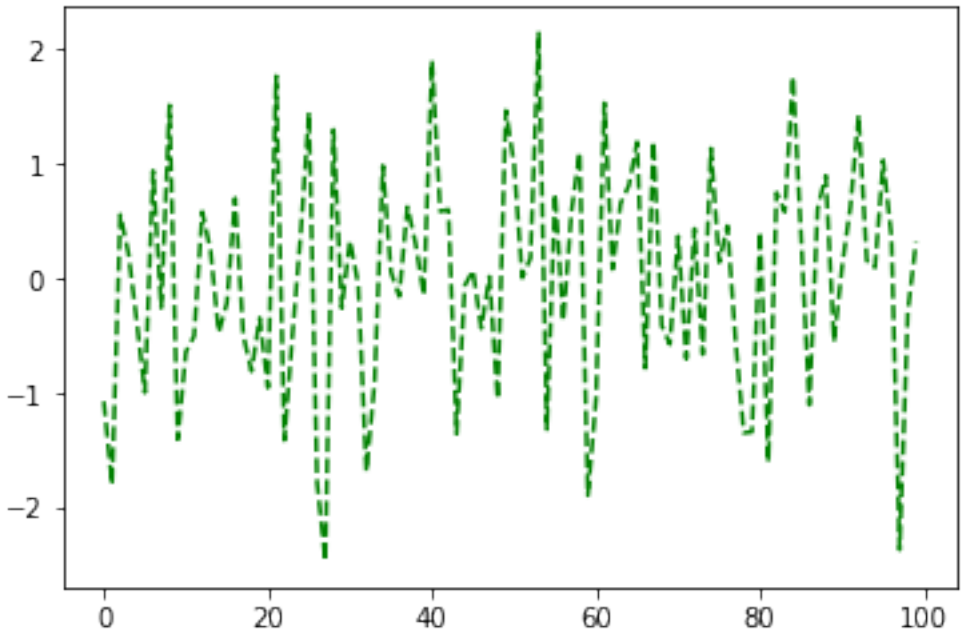
```
In [80]: y = randn(100)
         plot(y)
         autoscale(tight='x') #Establece limites.
         tight_layout() #eliminará el espacio perdido alrededor de una figura
         plt.show() #Mostrará el plot
```



7.2.2 Colores, Marcador y Estilo de Linea

Color	Marcador	Estilo de linea
Blue b	Point .	Solid -
Green g	Pixel ,	Dashed --
Red r	Circle o	Dash-dot -.
Cyan c	Square s	Dotted :
Yellow y	Thin diamond d	
Black k	Cross x	
White w	Plus +	

```
In [81]: ##Colores
         plot(y,'g--')
         plt.show()
```



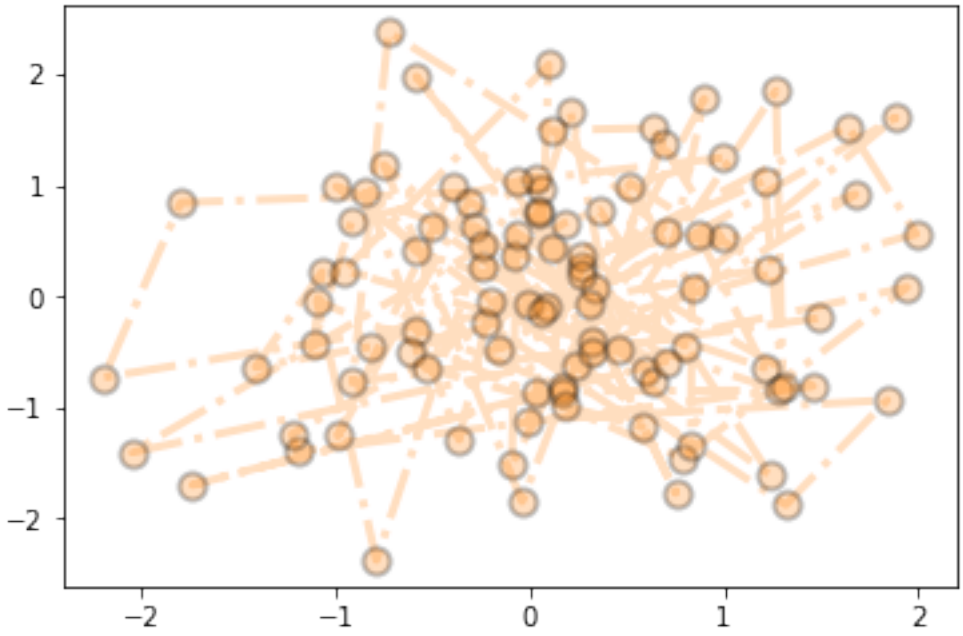
7.2.3 Argumentos para la Gráficos

Keyword	Description
alpha	Alfa (transparencia) de la trama: el valor predeterminado es 1 (sin transparencia)
color	Descripción del color de la línea.
label	Etiqueta para la línea - utilizada al crear leyendas
linestyle	Estilo de línea
linewidth	Indica el ancho de la línea
marker	Un símbolo o carácter en forma de marcador
markeredgecolor	Color del borde (una línea) alrededor del marcador.
markeredgewidth	Ancho del borde (una línea) alrededor del marcador
markerfacecolor	Color de la cara del marcador.
markersize	Un entero positivo que indica el tamaño del marcador.

Por ejemplo:

```
In [82]: from numpy.random import randn
         from pandas import *
         from matplotlib import *
```

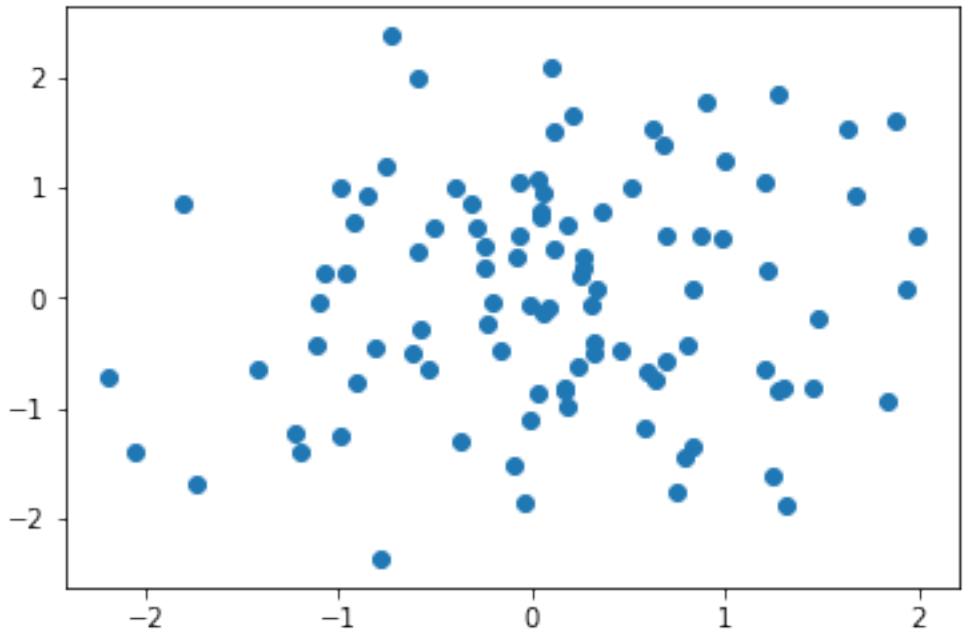
```
from matplotlib.pyplot import *
y = randn(100)
x = randn(100)
plot(x,y,alpha = 0.25, color = '#FF7F00', \
 label = 'Line Label', linestyle = '-.', \
 linewidth = 3, marker = 'o', markeredgecolor = '#000000', \
 markeredgewidth = 2, markerfacecolor = '#FF7F00', \
 markersize=10)
plt.show()
```



7.3 Scatter Plots

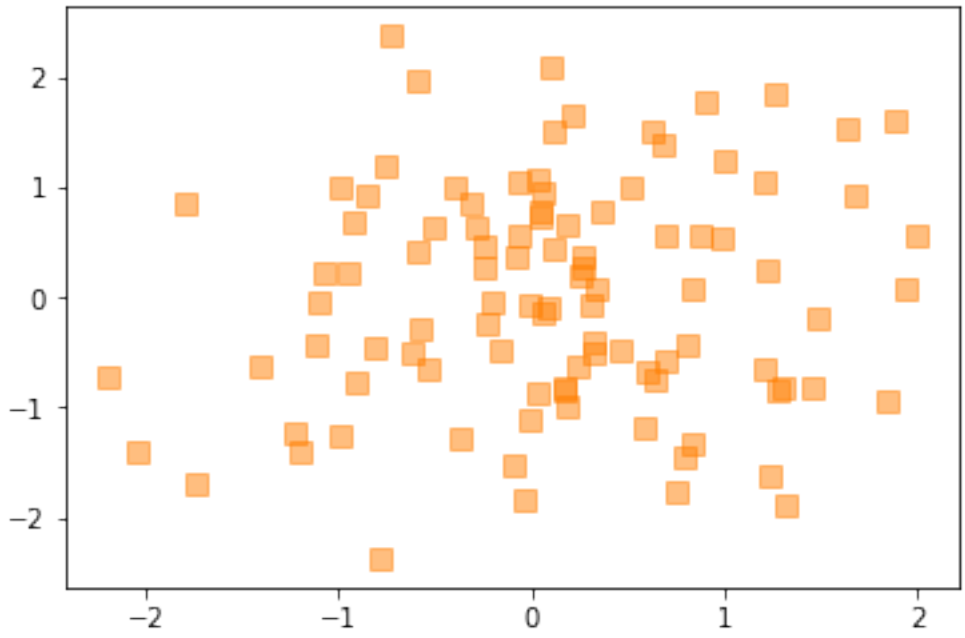
Scatter produce un diagrama de dispersión entre 2 matrices unidimensionales.<>

```
In [84]: scatter(x,y)
         plt.show()
```



Los scatter plots también pueden modificarse utilizando argumentos de palabras clave.<>

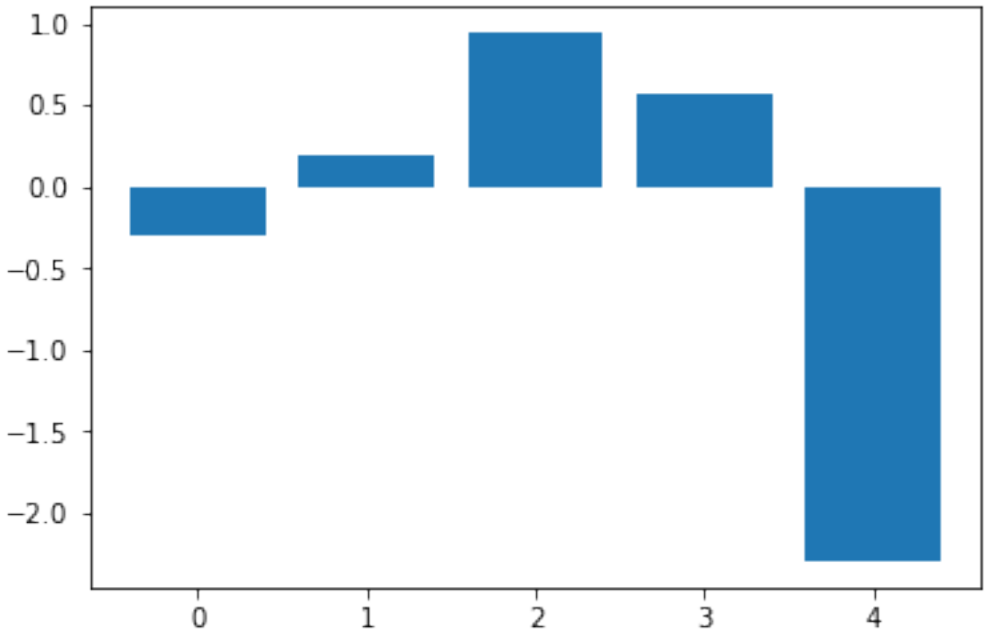
```
In [86]: scatter(x,y, s = 60, c = '#FF7F00', marker='s', \
                 alpha = .5, label = 'Scatter Data')
         plt.show()
```



7.4 Bar Charts

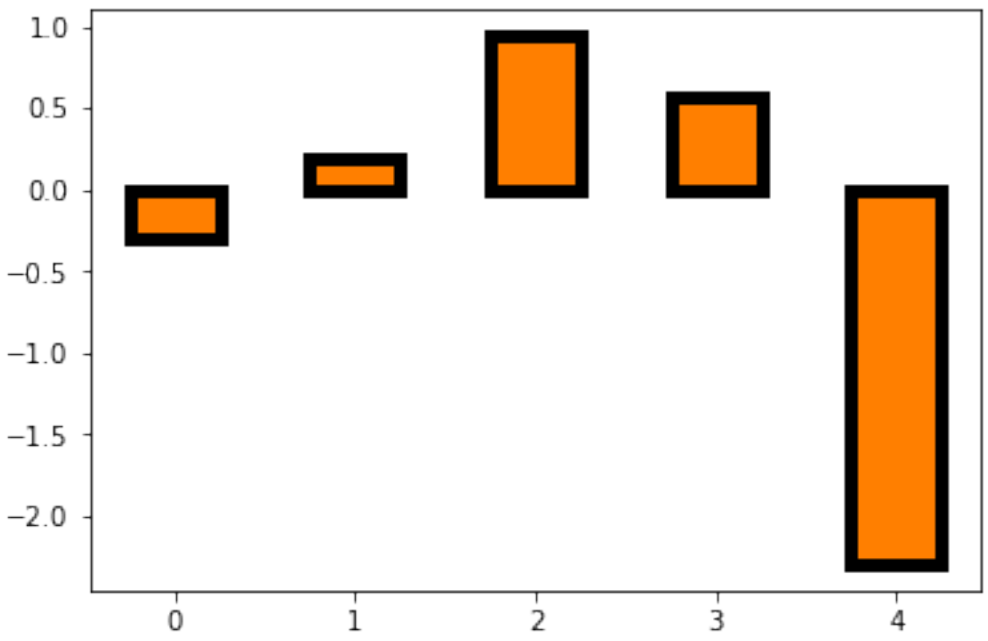
bar produce gráficos de barras utilizando dos matrices de una dimensión. El primero especifica el borde izquierdo de las barras y el segundo las alturas de la barra.<>

```
In [87]: import random as random
         y = randn(5)
         x = np.arange(5)
         bar(x,y)
         plt.show()
```



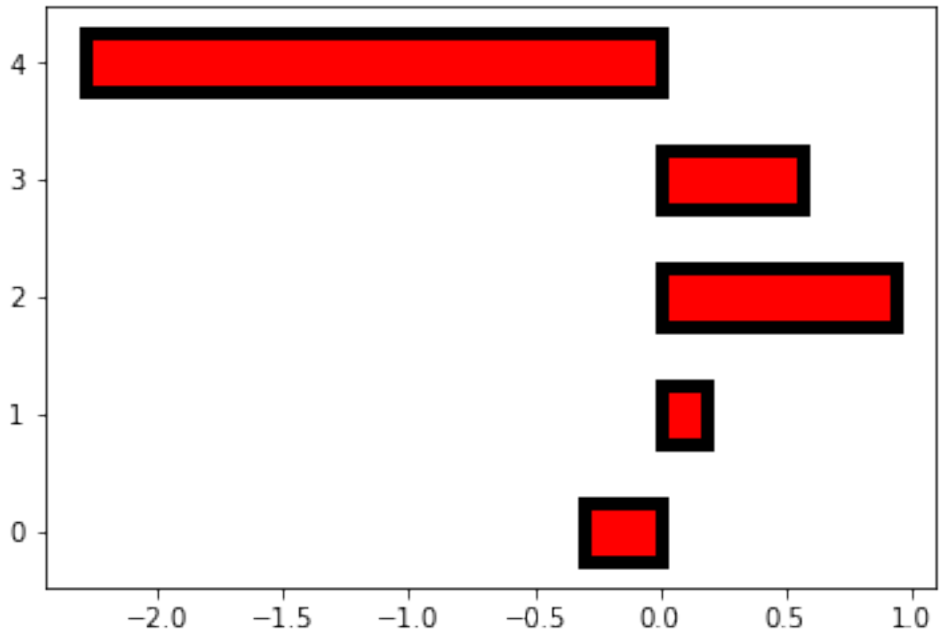
Los gráficos de barras toman argumentos de palabras clave para alterar los colores y el ancho de la barra.<>

```
In [88]: bar(x,y, width = 0.5, color = '#FF7F00', \
              edgecolor = '#000000', linewidth = 5)
         plt.show()
```



Finalmente, se puede usar barh en lugar de bar para producir un gráfico de barra horizontal.<>

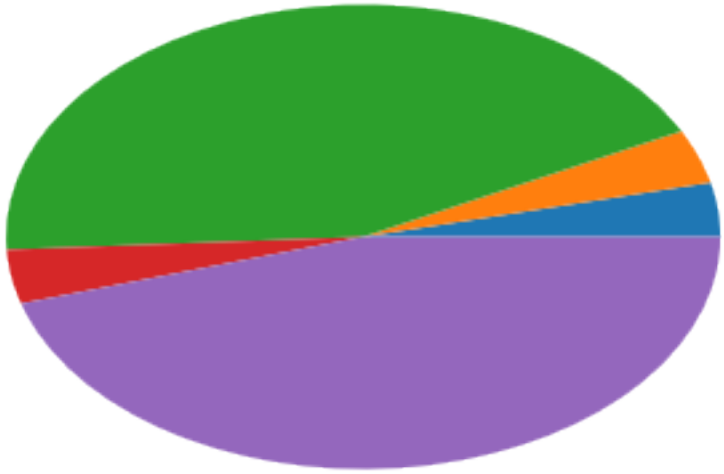
```
In [90]: barh(x, y, height = 0.5, color = 'r', \
              edgecolor = '#000000', linewidth = 5)
         plt.show()
```



7.5 Pie charts

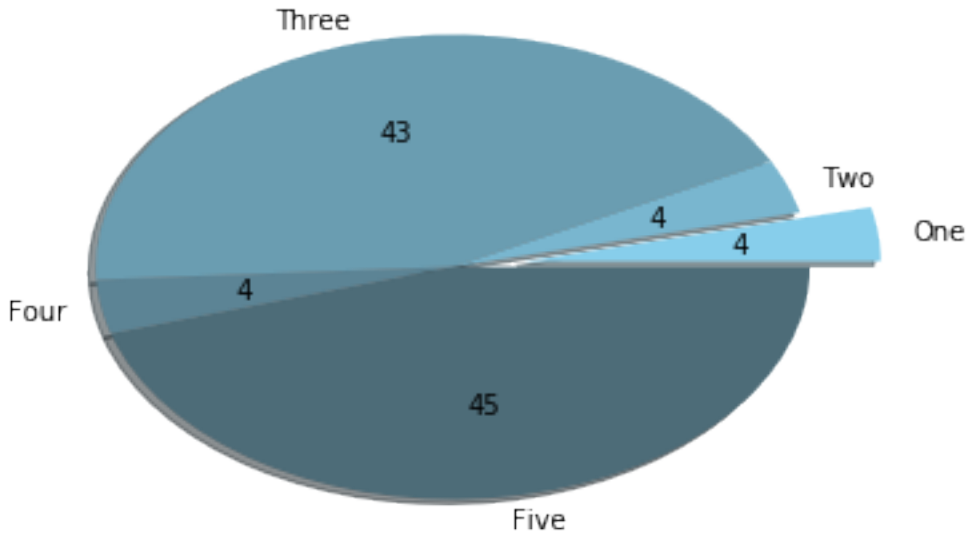
pie produce gráficos circulares usando una matriz de datos unidimensional (los datos pueden tener cualquier valor y no es necesario sumar 1)..<>

```
In [92]: y = randn(5)
         y = y/sum(y)
         y[y<.05] = .05
         pie(y)
         plt.show()
```



pie produce gráficos circulares usando una matriz de datos unidimensional (los datos pueden tener cualquier valor y no es necesario sumar 1)..<>

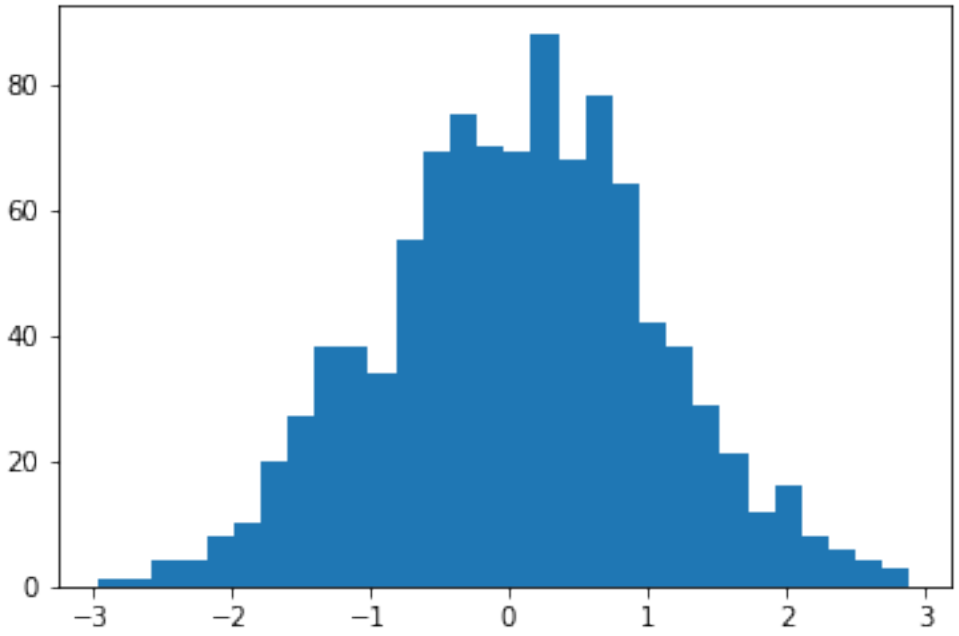
```
In [94]: explode = array([.2,0,0,0,0])
         colors = sns.dark_palette("skyblue", 8, reverse=True)
         labels = ['One', 'Two', 'Three', 'Four', 'Five']
         pie(y, explode = explode, colors = colors, \
         labels = labels, autopct = '%2.0f', shadow = True)
         plt.show()
```



7.6 Histogramas

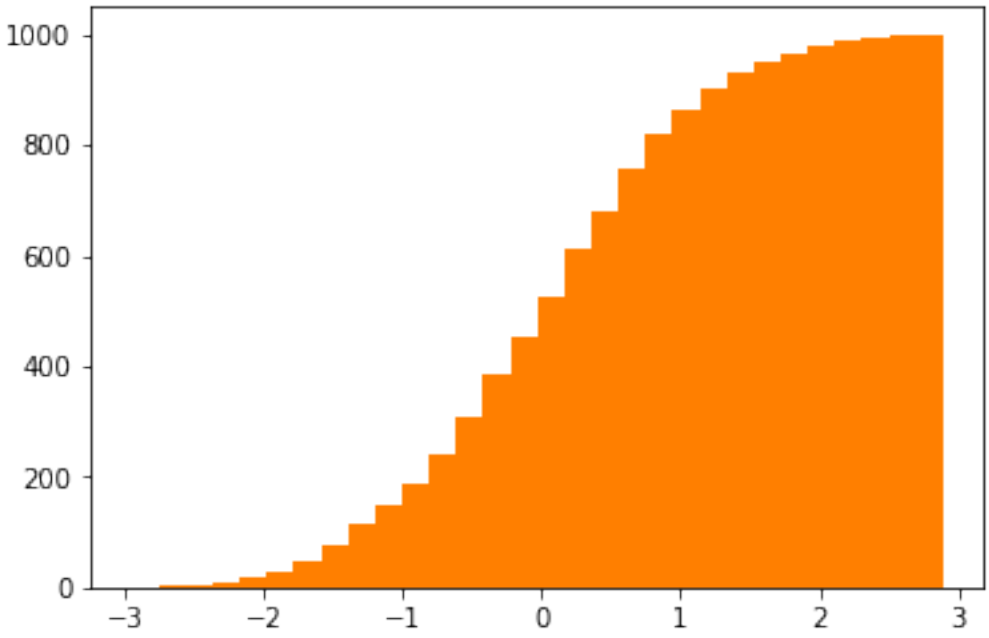
Los histogramas se pueden producir usando hist.<>

```
In [96]: x = randn(1000)
         hist(x, bins = 30)
         plt.show()
```



Los histogramas pueden modificarse aún más usando argumentos .<>

```
In [98]: hist(x, bins = 30, cumulative=True, color='#FF7F00')
         plt.show()
```

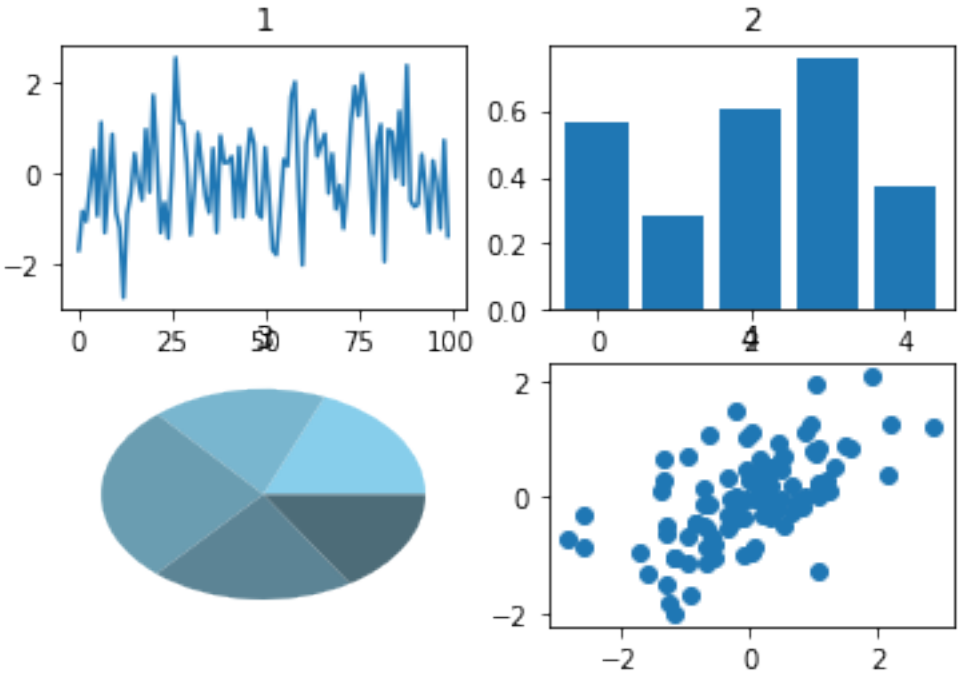


7.7 Multiple gráficos

En algunos escenarios es ventajoso tener múltiples gráficos o gráficos en una sola figura. Implementar esto es simple usando figure para inicializar la ventana de figura y luego usar add_subplot. Las subparcelas se agregan a la figura mediante una notación de cuadrícula con m filas yn columnas donde 1 es la parte superior izquierda, 2 es la derecha de 1, y así sucesivamente hasta el final de una fila, donde el siguiente elemento está debajo de 1.<>

```
In [100]: from matplotlib.pyplot import figure, plot, bar, pie, draw, scatter
          from numpy.random import randn, rand
          from numpy import sqrt, arange
          fig = figure()
          # Agregamos el subplot a la figura
          # Panel 1
          ax = fig.add_subplot(2, 2, 1)
          y = randn(100)
          plot(y)
          ax.set_title('1')
```

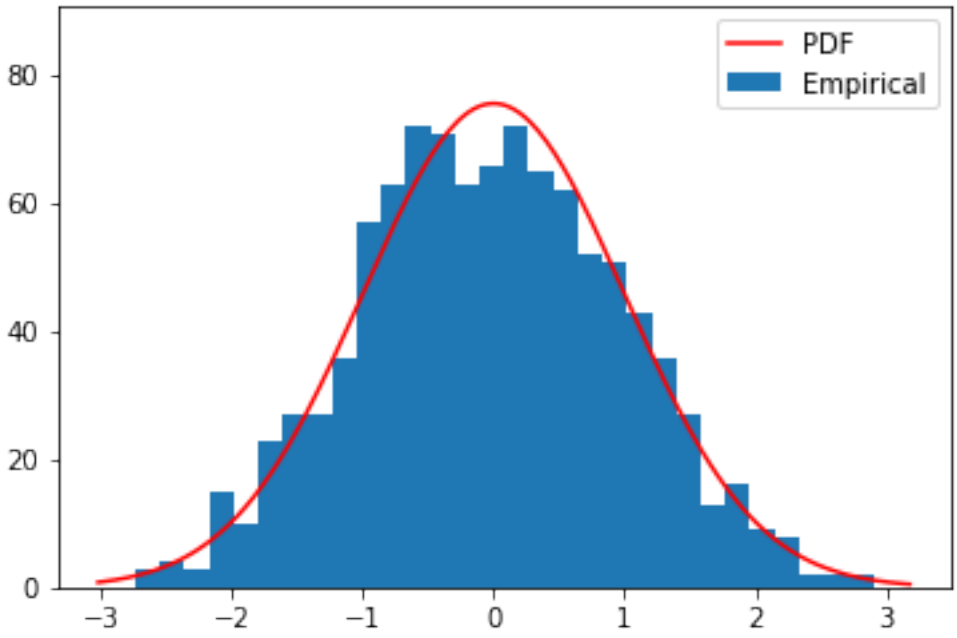
```
# Panel 2
y = rand(5)
x = arange(5)
ax = fig.add_subplot(2, 2, 2)
bar(x, y)
ax.set_title('2')

# Panel 3
y = rand(5)
y = y / sum(y)
y[y < .05] = .05
ax = fig.add_subplot(2, 2, 3)
pie(y, colors=colors)
ax.set_title('3')

# Panel 4
z = randn(100, 2)
z[:, 1] = 0.5 * z[:, 0] + sqrt(0.5) * z[:, 1]
x = z[:, 0]
y = z[:, 1]
ax = fig.add_subplot(2, 2, 4)
scatter(x, y)
ax.set_title('4')
draw()

plt.show()
```



7.8 Gráficos en un mismo axis

Ocasionalmente, se necesitan dos tipos diferentes de gráficos en los mismos ejes, por ejemplo, trazar un histograma y un PDF. Se pueden agregar múltiples gráficos a los mismos ejes al trazar el primero (por ejemplo, un histograma) y luego trazar cualquier dato restante. De forma predeterminada, si no se crea un nuevo eje, se agregarán parcelas adicionales al mismo eje. El código en el siguiente ejemplo comienza inicializando una ventana de figura y luego agregando ejes. Luego se agrega un histograma a los ejes y luego se traza un PDF normal. Se llama a legend () para producir una leyenda usando las etiquetas provistas en los comandos de macetas. get_xlim y get_ylim se utilizan para obtener los límites del eje después de agregar el histograma. Estos puntos se utilizan al calcular el PDF y, finalmente, se llama a set_ylim para aumentar la altura del eje de modo que el PDF esté contra la parte superior del gráfico.<>

```
In [101]: from matplotlib.pyplot import figure, plot, legend, draw
          from numpy import linspace
          import scipy.stats as stats
          from numpy.random import randn
          x = randn(1000)
          fig = figure()
          ax = fig.add_subplot(111)
          ax.hist(x, bins=30, label='Empirical')
          xlim = ax.get_xlim()
          ylim = ax.get_ylim()
          pdfx = linspace(xlim[0], xlim[1], 200)
          pdfy = stats.norm.pdf(pdfx)
          pdfy = pdfy / pdfy.max() * ylim[1]
          plot(pdfx, pdfy, 'r-', label='PDF')
          ax.set_ylim((ylim[0], 1.2 * ylim[1]))
          legend()
          draw()
          plt.show()
```



8 Funciones Estadísticas

Usaremos las librerías NumPy y SciPy<>

8.1 Numpy

8.1.1 Generador de números aleatorios

NumPy todos los generadores de números aleatorios se almacenan en el módulo numpy.random. Estos se pueden importar usando import numpy as np y luego llamando a np.random.rand.<>

rand, random_sample

rand y random_sample son generadores de números aleatorios uniformes que son idénticos, excepto que rand toma un número variable de entradas de enteros, una para cada dimensión, mientras que random_sample toma una tupla de n elementos.<>

```
In [106]: from numpy.random import *
          y = random_sample((3,4,5))
          y
```

```
Out[106]: array([[[0.63499192, 0.884987  , 0.03392413, 0.59104966, 0.31390193],
                  [0.20035287, 0.85506691, 0.50127389, 0.76879078, 0.14500521],
                  [0.80582565, 0.33125186, 0.3603145 , 0.5049838 , 0.16002496],
                  [0.18651736, 0.11879522, 0.81074449, 0.51152658, 0.74583846]],

                 [[0.42287751, 0.40550417, 0.69365811, 0.66322546, 0.80805777],
                  [0.38037414, 0.83461294, 0.16548812, 0.95314635, 0.31086348],
```

```
                [0.53628309, 0.57795452, 0.46914736, 0.51925924, 0.58302444],
                [0.01186444, 0.16631574, 0.06561918, 0.69607554, 0.27634662]],

               [[0.98132733, 0.29970279, 0.36898477, 0.56479549, 0.4356415 ],
                [0.98537982, 0.60984067, 0.04036182, 0.96375394, 0.36825509],
                [0.9158787 , 0.52459693, 0.46378276, 0.98133581, 0.08957659],
                [0.13332517, 0.64240699, 0.93078001, 0.47512791, 0.46498466]]])
```

```
In [107]: x = rand(3,4,5)
          x
```

```
Out[107]: array([[[0.41026918, 0.07890025, 0.06187926, 0.69149867, 0.49580121],
                  [0.67910285, 0.8891955 , 0.38715895, 0.68331861, 0.39265016],
                  [0.85321405, 0.68092206, 0.97378305, 0.12937734, 0.91275723],
                  [0.76986607, 0.30941331, 0.53705983, 0.53805036, 0.76943715]],

                 [[0.26113111, 0.04575869, 0.55918696, 0.11649218, 0.7717671 ],
                  [0.05158733, 0.85188655, 0.73062679, 0.81293434, 0.50983383],
                  [0.07021437, 0.184954  , 0.8210323 , 0.93747978, 0.19584804],
                  [0.29743406, 0.23487819, 0.1854696 , 0.40278315, 0.88278442]],

                 [[0.73752737, 0.50341107, 0.20817013, 0.18830931, 0.79546835],
                  [0.82574876, 0.72031278, 0.74280421, 0.63453871, 0.62826125],
                  [0.42315372, 0.712843  , 0.31691203, 0.71678   , 0.41333161],
                  [0.01906824, 0.47242239, 0.7766913 , 0.46938079, 0.38062244]]])
```

randn, standard_normal

randn y standard_normal son generadores de números aleatorios normales estándar. randn, como rand, toma un número variable de entradas de enteros, y standard_normal toma una tupla de n elementos. Se puede llamar a ambos sin argumentos para generar un único estándar normal (por ejemplo, randn ()).<>

```
In [136]: x = randn(3,4,5)
          x
          y = standard_normal((3,4,5))
          y
```

```
Out[136]: array([[[ 0.2772566 ,  0.27476954, -1.68322208,  0.29896668,
                    -0.46704382],
                  [ 0.009696  , -0.66486936, -0.97171868, -1.10010439,
                    -0.99035766],
                  [-3.102561  , -0.36107478,  1.11570464,  0.93429385,
                    -0.99933479],
                  [ 0.08667876, -1.36151014, -0.63652528, -0.02406181,
                     2.50002501]],

                 [[ 1.98466929,  0.57125235,  1.6994198 ,  0.87542868,
                    -1.19010259],
                  [ 1.18334691, -0.36119131,  0.42553781,  0.74243508,
```

```
        -0.23463629],
       [ 0.56676519, -0.48096798, -0.4281887 , -2.03272459,
        -1.33650373],
       [ 0.75776939,  0.35379847, -0.65720619, -1.38259852,
        -0.35680211]],

      [[-1.31677852,  0.24139734,  1.65652008, -0.54980578,
        -0.92939752],
       [ 0.42581082,  0.35346248,  0.40009012, -0.06771331,
         0.03007926],
       [ 0.3545837 , -1.00379653,  0.66718897,  0.51428505,
        -1.45849172],
       [-0.24093405,  1.73720346, -0.34452183,  1.03296033,
         1.70360131]]])
```

randint, random_integers

randint y random_integers son generadores de números aleatorios enteros uniformes que toman 3 entradas, 'Low', 'High' y 'Size'. 'Low' es el límite inferior de los enteros generados, 'Alto' es el superior y 'Size' es una tupla de n elementos. randint y random_integers difieren en que randint genera enteros exclusivos del valor en alto (como lo hacen la mayoría de las funciones de Python), mientras que random_integers incluye el valor en alto en su rango..<>

```
In [138]: x = randint(0,10,(100))
          x

Out[138]: array([3, 9, 1, 3, 8, 6, 8, 5, 6, 6, 1, 8, 3, 9, 4, 2, 8, 9, 5, 4, 3, 5,
                 1, 7, 4, 4, 9, 6, 1, 3, 7, 7, 1, 8, 8, 6, 7, 1, 2, 5, 3, 6, 6, 9,
                 2, 5, 6, 6, 2, 2, 6, 8, 5, 1, 4, 3, 3, 0, 2, 9, 1, 6, 9, 5, 7, 5,
                 5, 1, 9, 2, 3, 8, 7, 2, 2, 5, 1, 0, 2, 0, 1, 2, 7, 0, 3, 4, 2, 1,
                 9, 2, 0, 1, 7, 0, 2, 0, 2, 6, 8, 8])

In [139]: x.max()

Out[139]: 9

In [140]: y = random_integers(0,10,(100))
          y

C:\Users\CESAR\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DeprecationWarning: This fun
  """Entry point for launching an IPython kernel.


Out[140]: array([ 5,  6, 10,  6,  8,  1,  6,  3,  7,  9,  4,  2,  3,  5,  8,  3,  6,
                  0,  3,  2,  2,  3,  1,  8,  6,  7,  4,  7,  8,  0,  8, 10,  6, 10,
                  5,  3, 10,  1,  6,  1,  8,  2,  8,  1,  1,  9,  7, 10,  7,  5,  8,
                  1,  6,  8,  5, 10, 10,  5,  0,  5,  6,  3,  1,  4,  1, 10, 10, 10,
                  9,  7,  9,  6,  6,  4,  1,  4,  5,  8,  3,  5,  2,  7,  7,  2,  7,
                  7,  1,  5,  6,  6,  8,  1,  1, 10,  8,  1,  0, 10,  7,  5])
```

8.2 Generador de números aleatorios de acuerdo de su forma de distribución

1. binomial(n,p): Genera una extracción donde p es la prob de éxito y n el número de extracciones.
2. beta(a,b): Genera una extracción de una distribución beta.
3. chisquare(nv): Genera una extracción de una distribución chi-cuadrado.
4. exponential(): Genera una extracción de una distribución exponencial.
5. gamma(a): Genera una extracción de una distribución Gamma (α, 1), donde α es el parámetro de forma.
6. laplace(): Genera una extracción de una distribución de Laplace(Doble exponencial) centrado en 0 y en escala unitaria.
7. lognormal(mu, sigma): Genera una extracción de una distribución log-normal con mu como media y sigma como desviación.
8. multinomial(n, p): Genera una extracción de una distribución multinomial donde n es el número de intentos y p la probabilidad.
9. multivariate_normal(mu, Sigma): Genera una extracción de una distribución normal multivariada.
10. normal(mu, sigma): Genera una extracción de una distribución normal mu como media y sigma como desviación.
11. standard_t(nu): Genera una extracción de una t de Student con forma de parámetro v.
12. poisson(lambda): Genera una extracción de una distribución Poisson con expectativa λ.

8.3 Funciones estadísticas descriptivas

mean

mean calcula el promedio de un array.<>

```
In [141]: from numpy import *
          x = arange(10.0)
          mean(x)

Out[141]: 4.5

In [142]: x= randn(4,5)

In [143]: mean(x,0)

Out[143]: array([-0.63594048, -0.39332944, -0.26920048,  0.80373587,  0.27285858])
```

median

median calcula el valor de la mediana en un array.<>

```
In [144]: median(x)
```

```
Out[144]: 0.16053473637933713
```

7.3 Funciones estadísticas
std
std calcula el valor de la desviación standar en un array.<>

```
In [145]: std(x)

Out[145]: 1.0361766774698038
```

var
var calcula el valor de la variación en un array.<>

```
In [146]: var(x)

Out[146]: 1.0736621069323617
```

corrcoef
corrcoef calcula el coeficiente de correlación en un array.<>

```
In [147]: corrcoef(x)

Out[147]: array([[ 1.        , -0.40616736, -0.63000082,  0.76792692],
                 [-0.40616736,  1.        ,  0.15806886,  0.22672264],
                 [-0.63000082,  0.15806886,  1.        , -0.73303935],
                 [ 0.76792692,  0.22672264, -0.73303935,  1.        ]])
```

cov
cov calcula la covarianza en un array.<>

```
In [148]: cov(x)

Out[148]: array([[ 0.67198142, -0.39376762, -0.27760771,  1.00412707],
                 [-0.39376762,  1.39866177,  0.10048806,  0.42770211],
                 [-0.27760771,  0.10048806,  0.28895057, -0.62853463],
                 [ 1.00412707,  0.42770211, -0.62853463,  2.54437018]])
```

mode
mode calcula la moda en un array.<>

```
In [149]: x=randint(1,11,1000)
          stats.mode(x)

Out[149]: ModeResult(mode=array([2]), count=array([118]))
```

momento
momento calcula el momento r_{th} en un array.<>

```
In [150]: x = randn(1000)
          moment = stats.moment
          moment(x,2) - moment(x,1)**2
```

```
Out[150]: 0.9361154282250331
```

skew
skew calcula sesgadez en un array.<>

```
In [151]: x = randn(1000)
          skew = stats.skew
          skew(x)
```

```
Out[151]: 0.03260346363376576
```

kurtosis
skew calcula el exceso de kurtosis en un array.<>

```
In [153]: x = randn(1000)
          kurtosis = stats.kurtosis
          kurtosis(x)
```

```
Out[153]: 0.029321091467842564
```

pearsonr
pearsonr calcula la correlación de pearson 2 array.<>

```
In [154]: x = randn(10)
          y = x + randn(10)
          pearsonr = stats.pearsonr
          corr, pval = pearsonr(x, y)
          corr
```

```
Out[154]: 0.6215129992052246
```

```
In [156]: pval
```

```
Out[156]: 0.05508652074253713
```

linregress
linregress estima una regresión lineal de 2-arrays.<>

```
In [157]: x = randn(10)
          y = x + randn(10)
          linregress = stats.linregress
          slope, intercept, rvalue, pvalue, stderr = linregress(x,y)
          rvalue
```

```
Out[157]: 0.7729437787393605
```

7.4 test estadísticos
normaltest
normaltest() para testear la normalidad en una serie de datos.<>

```
In [158]: x = randn(20)
          normaltest = stats.normaltest
          normaltest(x)
```

```
Out[158]: NormaltestResult(statistic=0.2497145698286427, pvalue=0.8826228571932565)
```