numpy

February 9, 2021

1 Numpy Arrays

Las listas de Python son elementos fundamentales y muy útiles pero hay ciertas operaciones que no podemos realizar con ellas.

Los numpy arrays son conocidos por ser estructuras de datos muy fáciles de emplear al almacenar datos dentro de Python.

Se les suelen atribuir dos características importantes: - Todos los datos dentro de un numpy array deben ser de un mismo tipo. Si se intenta ingresar un dato de otro tipo, se forzará a ese dato a convertirse en el tipo de dato predominante. - Pueden ser estructuras de una dimensión, de dos dimensiones, incluso de 7 dimensiones. Sin embargo, al menos en análisis de datos, siempre es mejor tratar con información en en forma matricial (dos dimensiones).

1.1 Reglas básicas de los Numpy Arrays

Solo pueden contener datos de un mismo tipo: Si intentamos crear un arreglo de Numpy con datos de distintos tipos, Numpy "forzará" a todos los datos a ser de uno mismo.

[50]:
```
import numpy as np

np.array([1.0, 'is', True])
```

[50]:
```
array(['1.0', 'is', 'True'], dtype='<U32')
```

El operador + tiene un comportamiento diferente: Si lo empleamos en listas, las concatenará. Si lo empleamos con arreglos, sumará los valores contenidos en ellos.

[5]:
```
python_list = [1, 2, 3] # Lista
numpy_array = np.array([1, 2]) # Arreglo Numpy
```

[6]:
```
# Concatenando listas
python_list + python_list
```

[6]:
```
[1, 2, 3, 1, 2, 3]
```

[7]:
```
# Sumando arreglos
numpy_array + numpy_array
```

[7]:
```
array([2, 4, 6])
```

1.2 Problema: Calcular el BMI de un grupo de personas

Estamos realizando un estudio sobre la nutrición en un grupo de personas de determinada edad.

Formulando las primeras preguntas, nos hemos dado cuenta que necesitamos conocer el BMI de las personas involucradas. Desafortunadamente, no contamos con ese dato, pero podemos calcularlo a partir de los pesos y estaturas que hallamos en los registros del grupo.

Intentemos resolver este problema con listas:

[8]:
```
# Generamos una nueva lista con las estaturas de las personas en el grupo
height = [1.73, 1.68, 1.71, 1.89, 1.79]
height
```

[8]:
```
[1.73, 1.68, 1.71, 1.89, 1.79]
```

[9]:
```
# Generamos una nueva lista con los pesos de las personas en el grupo
weight = [65.4, 59.2, 63.6, 88.4, 68.7]
weight
```

[9]:
```
[65.4, 59.2, 63.6, 88.4, 68.7]
```

Para calcular el BMI, la fórmula es dividir el peso con la estatura, elevada al cuadrado:

[10]:
```
# Sin embargo, no es posible realizar operaciones sobre los
# elementos de la lista de esta manera
weight / height ** 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-667a3c803194> in <module>
      1 # Sin embargo, no es posible realizar operaciones sobre los
      2 # elementos de la lista de esta manera
----> 3 weight / height ** 2

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

1.2.1 Solución: Vamos a convertir nuestras listas en arreglos de Numpy

[11]:
```
# Importar el paquete numpy como np
import numpy as np

# Crear 2 array numpy desde height y weight
np_height = np.array(height)
np_weight = np.array(weight)
```

Intentemos calcular el BMI de nuevo:

[12]:
```
bmi = np_weight / np_height ** 2
bmi
```

[12]:
```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

1.3 Muestreo (subsetting)

Tener nuestros datos en arrays de Numpy nos facilita las cosas al momento de consultar datos bajo ciertos criterios

[13]:
```
bmi
```

[13]:
```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

[52]:
```
bmi[3]
```

[52]:
```
24.74734749867025
```

[15]:
```
bmi > 23
```

[15]:
```
array([False, False, False,  True, False])
```

[16]:
```
bmi[bmi > 23]
```

[16]:
```
array([24.7473475])
```

1.4 Numpy arrays en 2 dimensiones

¿Cómo puedo saber si un arreglo de Numpy es de 1 o dos dimensiones?

Tables	Are	Cool
col 3 is	right-aligned	$1600

Tables	Are	Cool
col 3 is	right-aligned	1600
col 2 is	centered	12
zebra stripes	are neat	1

[17]:
```
# Este es de una sola dimensión (arreglo)
type(np_height)
```

[17]:
```
numpy.ndarray
```

Mira la principal diferencia, ahora, en lugar de utilizar un solo par de corchetes [], utilizamos pares de corchetes dentro de uno principal [[], [], [], . . .]

Cada par interno podemos decir que es una "dimensión"

[18]:
```
np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
[65.4, 59.2, 63.6, 88.4, 68.7]])

np_2d
```

[18]:
```
array([[ 1.73,  1.68,  1.71,  1.89,  1.79],
       [65.4 , 59.2 , 63.6 , 88.4 , 68.7 ]])
```

[19]:
```
np_2d.shape
```

[19]:
```
(2, 5)
```

Recuerda, si intentas introducir un valor de un tipo distinto, podrías estar alterando el tipo de todos los elementos en el arreglo

[21]:
```
np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
[65.4, 59.2, 63.6, 88.4, "68.7"]])
```

[21]:
```
array([['1.73', '1.68', '1.71', '1.89', '1.79'],
       ['65.4', '59.2', '63.6', '88.4', '68.7']], dtype='<U32')
```

1.5 Muestreo en 2 dimensiones

[22]:
```
# Una fila completa
np_2d[0]
```

[22]:
```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

[23]:
```
# El elemento en la primera fila, segunda columna
np_2d[0][2]
```

[23]:
```
1.71
```

[24]:
```
# El elemento en la primera fila, segunda columna con notación de coma
np_2d[0,2]
```

[24]:
```
1.71
```

[25]:
```
# Los elementos de todas las filas pero en las columnas 1, 2
np_2d[:,1:3]
```

[25]:
```
array([[ 1.68,  1.71],
       [59.2 , 63.6 ]])
```

[27]:
```
# Los elementos de la segunda fila pero todas las columnas (¿no te suena␣
↪familiar?)
np_2d[1,:]
```

[27]:
```
array([65.4, 59.2, 63.6, 88.4, 68.7])
```

1.6 Estadística básica con Numpy

FIFA nos ha proveído amablemente de una base de datos con todo tipo de métricas acerca de jugadores en todos los equipos del mundo. Nosotros, la hemos pedido pues tenemos una corazonada: Creemos que los jugadores que juegan en la posición de portero, tienen en general una estatura mayor que el resto de jugadores, pero nadie nos cree :(

Utilizaremos estos datos para comprobarlo:

[42]:
```
# Importamos nuestros datos
import pandas as pd
data = pd.read_csv("https://assets.datacamp.com/production/repositories/288/
 ↪datasets/026a5211b906ac118a09b1a0dbf7df48faafb379/fifa.csv")
# Mostramos las columnas con las que contamos
data.columns
```

[42]:
```
Index(['id', ' name', ' rating', ' position', ' height', ' foot', ' rare',
       ' pace', ' shooting', ' passing', ' dribbling', ' defending',
       ' heading', ' diving', ' handling', ' kicking', ' reflexes', ' speed',
       ' positioning'],
      dtype='object')
```

[55]:
```
# Convertimos las columnas a Numpy Arrays
np_positions = np.array(data[' position'])
np_heights = np.array(data[' height'])
np_heights
```

[55]:
```
array([191, 184, 185, ..., 183, 179, 179], dtype=int64)
```

[53]:
```
np_positions == ' GK'
```

[53]:
```
array([ True, False, False, ..., False, False, False])
```

[59]:
```
# Heights de los goalkeepers (porteros): gk_heights
gk_heights = np_heights[np_positions == ' GK']

# Heights de los demás jugadores: other_heights
other_heights = np_heights[np_positions != ' GK']
other_heights
```

[59]:
```
array([184, 185, 180, ..., 183, 179, 179], dtype=int64)
```

[60]:
```
# Vamos a imprimir la mediana en la estatura de los porteros. Replace 'None'
print("Median height of goalkeepers: " + str(np.median(gk_heights)))

# Print out the median height of other players. Replace 'None'
print("Median height of other players: " + str(np.median(other_heights)))
```

```
Median height of goalkeepers: 188.0
Median height of other players: 181.0
```

2 Enlaces:

Si tienes dudas sobre el uso de Markdown, puedes consultar el siguiente link