Python para Análisis de Datos

Módulo 03

Numpy

Funciones de agregación

Hemos visto que contamos con numerosas funciones para hacer operaciones elemento a elemento. Pero también contamos con lo que se conoce como funciones de agregación (o de reducción) cuya característica es que toman un conjunto de números y devuelven sólo un número. La suma de todos los números de un array es un caso típico.

Gracias a estas funciones es que vamos a poder realizar estadísticas sobre el conjunto de datos.

La mayoría de estas funciones también están implementadas como métodos, notación que muchas veces es más conveniente.

Las siguientes líneas calculan la suma de todos los elementos del array.

```
print(a)
print(np.sum(a))
print(a.sum())
```

```
[[ 6  1 13]
 [ 7 15 11]
 [19  0 11]]
83
83
```

Promedio y desviación standard

El **promedio** (o media aritmética) se puede considerar como el valor más representativo de una distribución de números. Es la suma de todos los números dividido la cantidad de números:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

La **desviación standard** es una medida de la dispersión de los datos en torno al valor promedio:

$$s_N = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (x_i - \bar{x})^2},$$

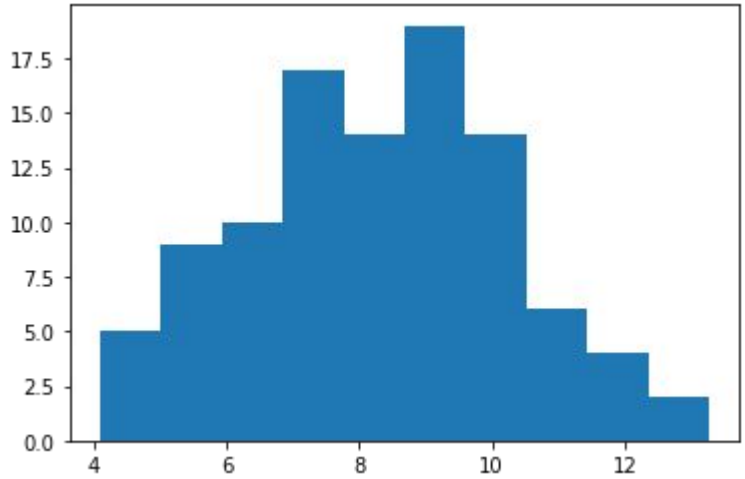
Podemos usar `mean` y `std` para calcular el promedio y la desviación standard de los números de un array.

En el siguiente ejemplo generamos **`n`** números aleatorios con distribución gaussiana. A medida que aumenta `n`, el promedio y la desviación se acercan a `mu` y `sigma.`

```
n = 100
mu = 8
sigma = 2
nums = mu + sigma*np.random.randn(n)
```

```
print(nums.mean())
print(nums.std())
```

```
8.301169639054528
1.968267286613969
```



Otras funciones

También encontramos las siguientes funciones:

varianza	`var`
mediana	`median`
mínimo	`min`
máximo	`max`
producto	`prod`
suma cumulativa	`cumsum`
producto cumulativo	`cumprod`

Missing values

¿Qué sucede con estas funciones y métodos cuando el array tiene `missing values`?

Están diseñadas para para propagar el valor al resultado final. Para ignorar los missing values se deben usar las funciones **`nansum, nanprod, nanmean, nanstd, nanvar, nanmin, nanmax,`** etc. Estas funciones no tienen métodos equivalentes.

```
a
```

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

```
a.mean(), np.max(a), a.sum()
```

```
(nan, nan, nan)
```

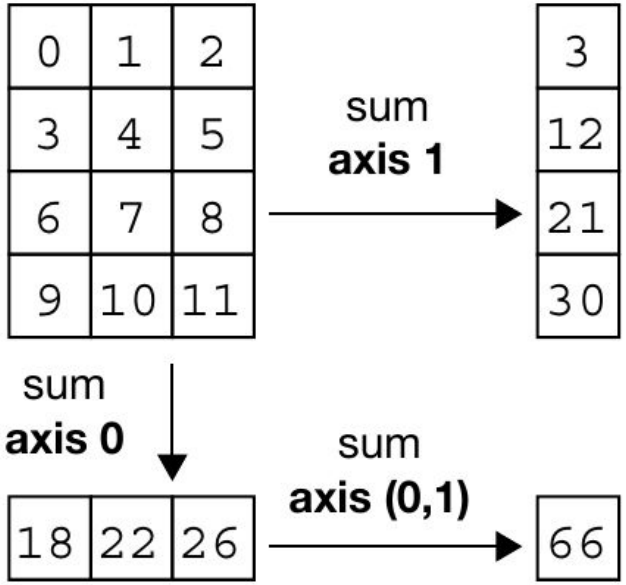
```
np.nanmean(a), np.nanmax(a), np.nansum(a)
```

```
(1.0, 1.0, 3.0)
```

Operando sobre ejes

Estas funciones, en principio, calculan el resultado con todos los elementos del array. Sin embargo, es posible controlar su comportamiento para que calculen los resultados sobre alguno de los ejes. Por ejemplo, en un array de dos dimensiones podemos calcular el promedio de todas las filas o de las columnas.

Esto se consigue usando el parámetro `axis`, que determina la dimensión a recorrer. En nuestro ejemplo de una matriz, `axis = 0` calcula el promedio de todas las columnas (recorre las filas) y `axis = 1` calcula el promedio de las filas (recorre las columnas).



```
a
```

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

```
a.mean(axis=0)
```

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

```
a.mean(axis=1)
```

```
array([0.5, 2.5, 4.5])
```

```
b
```

```
array([[22,  1, 26,  2,  4],
       [24, 18,  7, 28, 19],
       [22,  6, 20,  7,  1]])
```

```
b.min(axis=0)
```

```
array([22,  1,  7,  2,  1])
```

```
b.min(axis=1)
```

```
array([1, 7, 1])
```

Ejemplo

Supongamos que las ventas de algún producto durante cuatro semanas está en el array `ventas`. Cada fila es una semana y cada columna es un día de la semana.

```
ventas
```

```
array([[17, 16, 15, 17, 15],
       [15,  2, 11, 15, 14],
       [13,  5,  9,  3,  5],
       [ 8, 16, 12, 15,  7]])
```

```
# Promedio de ventas para cada día de la semana
ventas.mean(axis=0)
```

```
array([13.25,  9.75, 11.75, 12.5 , 10.25])
```

```
# Total de ventas de cada semana
ventas.sum(axis=1)
```

```
array([80, 57, 35, 58])
```

argmin y argmax

Si en lugar de querer encontrar el valor mínimo o máximo, queremos encontrar el índice del elemento correspondiente podemos usar **argmin** y **argmax**, pero hay que tener en cuenta que devuelve el índice del array aplanado.

```
1 a
```

```
array([[  1,   2,   4],
       [  5,   4,   3],
       [  8,   4, 999]])
```

```
1 a.argmax()
```

```
8
```

Revisión

1. Repasar el concepto de función de agregación.
2. Repasar los conceptos estadísticos y las funciones correspondientes de Numpy.
3. Repasar cómo trabajar por ejes.
4. Aplicar las funciones a distintos arrays.

¡Muchas gracias!

¡Sigamos trabajando!