IMMUNE CODING INSTITUTE

Programming and frameworks for ML

Python for data analyis

About Me

Big Data Consultant at Indra / Big Data Lecturer

- More than 20 years of experience in different environments, technologies, customers, countries ...
- Passionate data and technology
- Enthusiastic Big Data world and NoSQL

Daniel Villanueva Jiménez

BigData Developer / Lecturer

INDRA • Universidad Pontificia de Salamanca

daniel.villanueva@immune.institute

NumPy

- **Numpy** (Numerical Python) is probably the most important library in the entire Python language (beyond those included in the standard library).
- At the same time, it is the best known and most powerful library of linear algebra available today.

```
import numpy as np
```

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

IMMUNE CODING INSTITUTE

Python lists (review)

- Python lists are flexible, easy to use and very versatile. In a list, we can include different types of objects:

```
mi_lista = [1, 3.14, "hola", 0x3b] # 0x3b es el número hexadecimal 3b, que Python
                                   # traduce automáticamente a su equivalente
                                   # decimal, que resulta ser el número 59
```

Exercise 1

- Create a list with 3 items of different type
- Add an element
- Print the list
- Delete the last item on the list
- Get the last item on the list by removing it from the list
- Create a function, called "multiply_by_3", which, given a list of integers, returns another list, where each number has been multiplied by 3

```
[1, 'Hola', True, 5]
[1, 'Hola', True]
Ultimo elemento:  True
[1, 'Hola']
[3, 6, 9, 12, 15, 18, 21, 24, 27]
```

Exercise 1 - Solution

```
# Crea una lista con 3 elementos de distinto tipo

lista = [1, "Hola", True]

# Añade un elemento
lista.append(5)

# Imprime la lista
print(lista)

# Elimina el último elemento de la lista
del lista[-1]
print(lista)

# Crea una nueva función, denominada <<multiplica_por_3>>, que dada una lista de números enteros
# devuelve otra lista, donde cada número haya sido multiplicado por 3
def multiplica_por_3(lista):
  return [elemento * 3 for elemento in lista]

print (multiplica_por_3(list(range(1,10))))
```

IMMUNE CODING INSTITUTE

Exercise 2

- Create a list that stores the following matrix (list of lists):

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

- Create a function called "multiply_matrix_by_2" that multiplies each number of the matrix by 2

```
print(matriz)
print(multiplica_matriz_por_2(matriz))
```

```
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
[[2, 4, 6, 8], [10, 12, 14, 16], [18, 20, 22, 24]]
```

Exercise 2 - Solution

```
matriz = [
          [1, 2, 3, 4],
          [5, 6, 7, 8],
          [9, 10, 11, 12]
]

def multiplica_matriz_por_2(matriz):
  matriz_resultado = []

  for lista in matriz:
    matriz_resultado.append([elemento * 2 for elemento in lista])

  return matriz_resultado

print(multiplica_matriz_por_2(matriz))
```

NumPy to the rescue

- Numpy has been designed to make these types of calculations much easier and faster
- It offers us a series of new objects. The most important is the **array**
- An **array** is similar to Python lists and is constructed with the function **numpy.array**()
- Mathematical operations on an array are performed at the same time on all elements!

```
lista = [1,2,3,4]
array = np.array(lista)
print(array)
print(array * 2)
```

```
[1 2 3 4]
[2 4 6 8]
```

IMMUNE CODING INSTITUTE

Matrices

- An array in NumPy is a 2-dimensional array and works exactly like the

```
matriz = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print(matriz)
print("\n")
print(matriz * 3)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]


[[ 3  6  9 12]
 [15 18 21 24]
 [27 30 33 36]]
```

Matrices

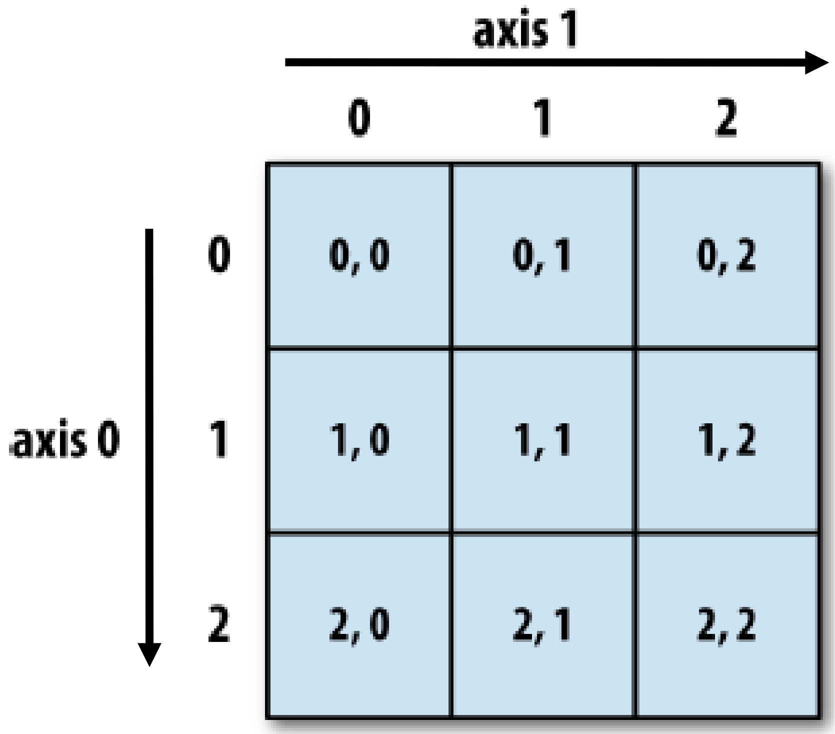
- The **shape** property returns a tuple with the dimensions of the array

```
mi_matriz = np.array([[1,2], [3,4], [5,6]])
print(mi_matriz)

filas , columnas = mi_matriz.shape

print(filas, columnas)

print("\nLas dimensiones son %s: %d filas y %d columnas" % (mi_matriz.shape, filas, columnas))
```

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

Las dimensiones son (3, 2): 3 filas y 2 columnas
```

Matrices

- In a matrix the 0 axis corresponds to the **rows** and the 1 axis corresponds to the **columns**



Exercise 3

- Create a function called "np_multiply_by_3" that accepts a list of numbers and returns this list multiplied by the number 3
- Create a function called "np_multiply_matrix_by_2" that accepts an array of numbers (lists of lists) and returns this array multiplied by the number 2

```
lista = list(range(1,10))
print(np_multiplica_por_3(lista))

matriz = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12]
]
print(np_multiplica_matriz_por_2(matriz))
```

```
[ 3  6  9 12 15 18 21 24 27]
[[ 2  4  6  8]
 [10 12 14 16]
 [18 20 22 24]]
```

Exercise 3 - Solution

```
import numpy as np

def np_multiplica_por_3(lista):
  return np.array(lista) * 3

def np_multiplica_matriz_por_2(matriz):
  return np.array(matriz) * 2

lista = list(range(1, 10))
print(np_multiplica_por_3(lista))

matriz = [
          [1, 2, 3, 4],
          [5, 6, 7, 8],
          [9, 10, 11, 12]
]

print(np_multiplica_matriz_por_2(matriz))
```

IMMUNE CODING INSTITUTE

Time Comparison

```python
import time

lista = list(range(1, 1000000))
matriz = [list(range(1, 10000000)),
          list(range(1, 10000000)),
          list(range(1, 10000000))]

t1 = time.clock()
multiplica_por_3(lista)
print("multiplica_por_3: %2.5f segundos" % (time.clock() - t1 ))

t1 = time.clock()
np_multiplica_por_3(lista)
print("np_multiplica_por_3: %2.5f segundos" % (time.clock() - t1 ))

t1 = time.clock()
multiplica_matriz_por_2(matriz)
print("multiplica_matriz_por_2: %2.5f segundos" % (time.clock() - t1 ))

t1 = time.clock()
np_multiplica_matriz_por_2(matriz)
print("np_multiplica_matriz_por_2: %2.5f segundos" % (time.clock() - t1 ))
```

```
multiplica_por_3: 0.11313 segundos
np_multiplica_por_3: 0.07376 segundos
multiplica_matriz_por_2: 3.51703 segundos
np_multiplica_matriz_por_2: 2.04833 segundos
```

Creating Matrices

- In addition to the np.array function, we can create specialized versions of arrays
  - An array with only **zeros** is created with the function **np.zeros((rows, columns))**

```
np.zeros((5,5))
```

```
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.]])
```

Creating Matrices

- In addition to the **np.array** function, we can create specialized versions of arrays
  - A matrix with only ones is created with the function **np.ones((rows, columns))**
  - An identity matrix is created with **np.identity**(dimension)

```
np.identity(3)
```

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

```
np.ones((5, 5))
```

```
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.]])
```

Creating Matrices

- The **arange**(start, end [, step]) method allows you to create arrays and fill them with number sequences
- The **repeat** method (element, n) allows to repeat a number n times

```
np.arange(3,14)
```

```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
```

```
np.arange(3,14, 2)
```

```
array([ 3,  5,  7,  9, 11, 13])
```

```
np.arange(0,15)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
np.repeat(5,20)
```

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

Creating Matrices

- Another way to create arrays is to **reshape** a one-dimensional array using the **reshape**(dimension) method

```
array = np.arange(1, 16)
array
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
array.reshape((5, 3))
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15]])
```

```
array.reshape((3, 5))
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

Creating Matrices

- A matrix can also be resized as a one-dimensional array!

```
matriz = np.arange(1, 16).reshape((5,3))
matriz
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15]])
```

```
matriz.reshape(15)
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

IMMUNE CODING INSTITUTE

Creating Matrices

- NumPy offers properties that are very useful for working with matrices, such as the transposition of a matrix (**T** property)

```
matriz = np.arange(1, 16).reshape((3,5))
matriz
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

```
matriz.T
```

```
array([[ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14],
       [ 5, 10, 15]])
```

Types of data

- Within an **array,** all elements have the same type of data.
- The data types in Numpy are called **dtypes** and the most important ones are int64, float64, bool, object and string

```
verdaderos_y_falsos = np.array([True, False, False, True])
verdaderos_y_falsos
```

```
array([ True, False, False,  True], dtype=bool)
```

```
objetos = np.array([(1,2,3), (3,4), (5,6)])
objetos
```

```
array([(1, 2, 3), (3, 4), (5, 6)], dtype=object)
```

Types of data

- The **dtype** property allows to find out the type of data of an array

```
print(np.array(["Hola", "Mundo"]).dtype)
print(np.array(["uno", "dos", "tres", "cuatro", "diecisiete"]).dtype)
```

```
<U5
<U10
```

Types of data

- The array method **astype**(type) allows you to change the data type of a NumPy

```
np.array([True, False, False, True]).astype(np.int64)
```

```
array([1, 0, 0, 1], dtype=int64)
```

Exercise 4

- Create a 7 x 7 all filled with ones matrix (**np.ones** method)
- Subtract it with the 7 x 7 identity matrix (**np.identity** method)
- Create a 20 number array, from 0 to 19
- Convert it into a 5 x 4 matrix (**reshape** method)
- Displays the data type of the previous matrix
- Change it to a float data type (np.float64)
- Create another matrix with numbers from 20 to 1 of 4 x 5
- Multiply it by itself
- Shows the transposed matrix (**T** property)

Exercise 4 - Solution

```
# Crea una matriz  7x7 rellena de unos
matriz = np.ones((7,7))
print(matriz)

# Restala de una matriz identidad de 7x7
print(matriz - np.identity(7))

# Crea un array de 20 números, de 0 a 19
array = np.arange(0, 20)
print(array)

# Conviertelo en una matriz de 5x4
print(array.reshape((5,4)))

# Muestra el tipo de datos de la matriz anterior
print(array.dtype)

# Cambialo a un tipo de datos de tipo float
print(array.astype(np.float64))

# Crea otra matriz de números de 20 a 1 de 5x4
matriz = np.arange(20, 0, -1).reshape((5,4))
print(matriz)

# Mutliplicala por si misma
print(matriz * matriz)

# Muestra la matriz traspuesta
print(matriz.T)
```

Accessing the elements of an array

- Access to the elements of an array is done in a similar way to those of a list

```
mi_lista = ["perro", "gato", "loro", "lince", "python", "oso"]
mi_array = np.array(mi_lista)
```

```
mi_array[2]
```

```
'loro'
```

```
mi_array[3:6]
```

```
array(['lince', 'python', 'oso'], dtype='<U6')
```

```
mi_array[::-1]
```

```
array(['oso', 'python', 'lince', 'loro', 'gato', 'perro'], dtype='<U6')
```

```
mi_array[-1:]
```

```
array(['oso'], dtype='<U6')
```

Modifying the elements of an array

- The elements in an array are modified in a similar way to those in a list
- **But NumPy allows you to write several items at once**

```
mi_array[0] ="caballo"
mi_array
```

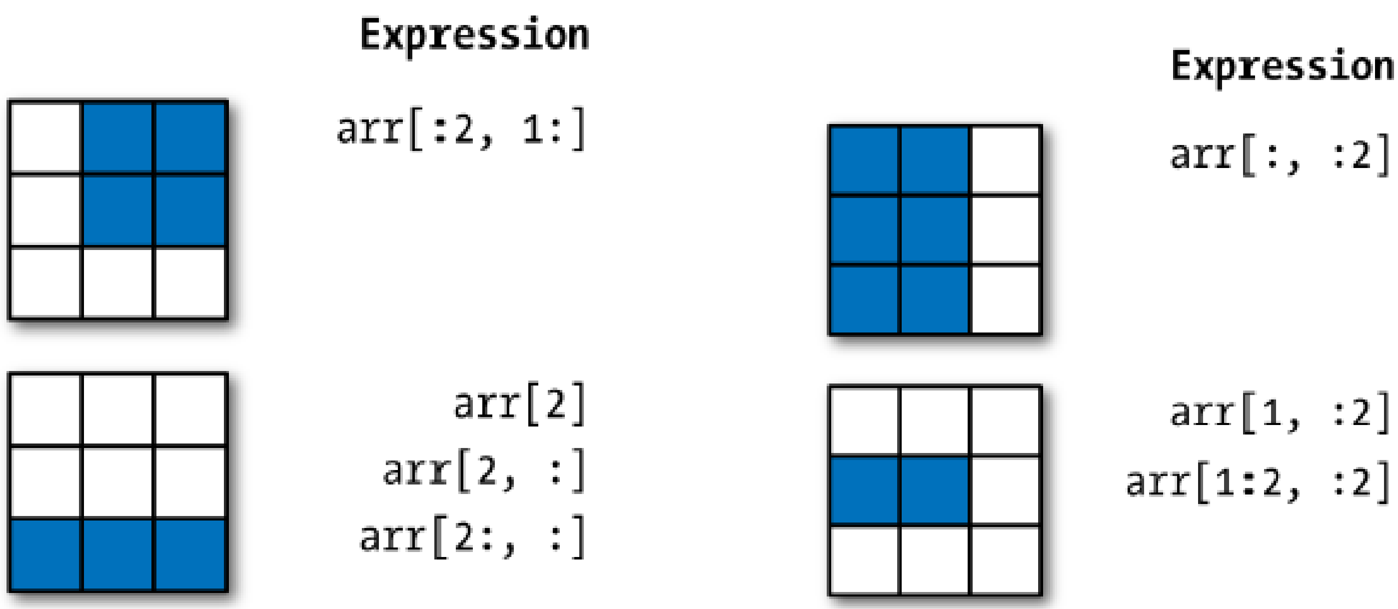
```
array(['caball', 'gato', 'loro', 'lince', 'python', 'oso'], dtype='<U6')
```

```
mi_array[3:5] ="tigre"
mi_array
```

```
array(['caball', 'gato', 'loro', 'tigre', 'tigre', 'oso'], dtype='<U6')
```

Accessing the elements of a matrix

- The access to the elements of a matrix is done with double indexing, one for each dimension: **matrix [rows, columns]**



Accessing the elements of a matrix

- The access to the elements of a matrix is done with double indexing, one for each dimension: **matrix [rows, columns]**

```
data = np.arange(1, 10).reshape(3,3)
data
```

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

```
data[1, 1]
```

```
5
```

```
data[1][1]
```

```
5
```

```
data[:2, 1:]
```

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

```
data[2]
```

```
array([7, 8, 9])
```

```
data[:,2]
```

```
array([3, 6, 9])
```

Exercise 5

- Create a 5 x 2 matrix with numbers from 1 to 10
- Print the 2nd column
- Print the 3rd row
- Print the number that is in the 3rd row, 1st column
- Print the first 2 rows (all columns)
- Print the last 2 rows (all columns)

```
[[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]]
[ 2  4  6  8 10]
[5 6]
5
[[1 2]
 [3 4]]
[[ 7  8]
 [ 9 10]]
```

Exercise 5 - Solution

```python
# Crea una matriz de 5 x 2 con números del 1 al 10
matriz = np.arange(1,11).reshape((5,2))
print(matriz)

# Imprime la segunda columna
print(matriz[:,1])

# Imprime la tercera fila
print(matriz[2,:])

# Imprime el número que está en la tercera fila y en la primera columna
print(matriz[2,0])

# Imprime las primeras 2 filas
print(matriz[:2])

# Imprime las últimas 2 filas
print(matriz[-2:])
```

Boolean indexing

- Indexing in Numpy is a much more powerful tool than in standard Python.
- We can use it to filter the content of any array according to a condition

```
un_array = np.array(["Julio", "Jose", "Alberto", "Julio", "Nuria", "Daniel"])
un_array
```

```
array(['Julio', 'Jose', 'Alberto', 'Julio', 'Nuria', 'Daniel'],
      dtype='<U7')
```

```
un_array == "Julio"
```

```
array([ True, False, False,  True, False, False], dtype=bool)
```

Boolean indexing

- Indexing in Numpy is a much more powerful tool than in standard Python
- We can use it to filter the content of any array according to a condition

```
un_array
```

```
array(['Julio', 'Jose', 'Alberto', 'Julio', 'Nuria', 'Daniel'],
      dtype='<U7')
```

```
un_array[ [ True, False, False,  True, False, False] ]
```

```
array(['Julio', 'Julio'], dtype='<U7')
```

```
un_array[ un_array == "Julio" ]
```

```
array(['Julio', 'Julio'], dtype='<U7')
```

IMMUNE CODING INSTITUTE

Boolean indexing

- NumPy allows us to join more than one condition using Boolean algebra operations

```
un_array[ (un_array == "Julio") | (un_array == "Nuria" ) ]
```

```
array(['Julio', 'Julio', 'Nuria'],
      dtype='<U7')
```

```
un_array[ (un_array == "Julio") & ~ (un_array == "Nuria" ) ]
```

```
array(['Julio', 'Julio'],
      dtype='<U7')
```

operador	equivalencia
and	&
or	|
not	~

IMMUNE CODING INSTITUTE

Boolean indexing

- The **where** function allows update an array based on a condition

```
arr = np.array([72, 23,  5, 61, 54, 53, 80, 90, 28, 80])
arr
```

```
array([72, 23,  5, 61, 54, 53, 80, 90, 28, 80])
```

```
np.where(arr > 50, -1, arr)
```

```
array([-1, 23,  5, -1, -1, -1, -1, -1, 28, -1])
```

Exercise 6

- Create an array of numbers from 20 to 49
- Filter out numbers under 31.5
- Filter out numbers greater than 31 and less than 40
- Create a 5x6 matrix with numbers from 15 to -14
- Assign the value 0 to negative numbers

```
[20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
 44 45 46 47 48 49]
[20 21 22 23 24 25 26 27 28 29 30 31]
[32 33 34 35 36 37 38 39]
[[ 15  14  13  12  11  10]
 [  9   8   7   6   5   4]
 [  3   2   1   0  -1  -2]
 [ -3  -4  -5  -6  -7  -8]
 [ -9 -10 -11 -12 -13 -14]]
[[15 14 13 12 11 10]
 [ 9  8  7  6  5  4]
 [ 3  2  1  0  0  0]
 [ 0  0  0  0  0  0]
 [ 0  0  0  0  0  0]]
```

Exercise 6 - Solution

```
# Crea una un array de números del 20 al 49
arr = np.arange(20, 50)
print(arr)

# Filtra los números por debajo de 31.5
print(arr[ arr < 31.5 ])

# Filtra los números mayores a 31 y menores que 40
print(arr[ (arr > 31) & (arr < 40) ])

# Crea una matriz de 5x6 with números del 15 al -14
matriz = np.arange(15,-15, -1).reshape((5,6))
print(matriz)

# Asigna el valor 0 a los números negativos
matriz = np.where(matriz < 0, 0, matriz)
print(matriz)
```

Universal functions

- A universal function is a function that performs operations on all elements of an array
- An example is the functions **np.sqrt**() or **np.exp**()

```
array = np.arange(10)
print(array)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```
np.sqrt(array)
```

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

```
np.exp(array)
```

```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

```
np.maximum(array, 3)
```

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

https://docs.scipy.org/doc/numpy/reference/ufuncs.html#available-ufuncs

Statistical methods with Numpy

- When the array is numerical, Numpy offers a number of simple methods for running statistical functions
- Numpy allows you to use both universal functions and methods available in the array

```
mi_array = np.array([ 0,  1,  12,  3,  4,  15, 18,  9, 10, -1])
```

```
print(mi_array)
print(mi_array.mean()) # Media
print(mi_array.sum()) # Suma
print(mi_array.var()) # Varianza
print(mi_array.std()) # Desviación típica
print(mi_array.min()) # Mínimo
print(mi_array.max()) # Máximo
print(np.median(mi_array)) # Mediana
```

```
[ 0  1 12  3  4 15 18  9 10 -1]
7.1
71
39.690000000000005
6.300000000000001
-1
18
6.5
```

IMMUNE CODING INSTITUTE

Statistical methods with Numpy

- Numpy offers other functions that allow you to accumulate intermediate results such as **cumsum**() or **cumprod**():

```
mi_array = np.array([ 5,  1,  12,  3,  4,  15,  6,  7, 2,  9, 10, -1])
```

```
print(mi_array)
print(mi_array.cumsum()) # Acumulación de la suma
print(mi_array.cumprod()) # Acumulación del producto
```

```
[ 5  1 12  3  4 15  6  7  2  9 10 -1]
[ 5  6 18 21 25 40 46 53 55 64 74 73]
[        5         5        60       180       720     10800     64800
    453600    907200   8164800  81648000 -81648000]
```

Statistical methods with Numpy

- When using statistical functions on logical values, false values are automatically converted to 0 and true values to 1
- This makes it easy to obtain percentages based on conditions

```
mi_array = np.array([0, 1, 12, 3, 4, 15, 6, 7, 18, 9, 10, -1])
print(mi_array)

print(mi_array > 10)
print(np.where(mi_array > 10, 1, 0))

print((mi_array > 10).sum())
print(len(mi_array))
(mi_array > 10).sum() / len(mi_array)
```

```
[ 0  1 12  3  4 15  6  7 18  9 10 -1]
[False False  True False False  True False False  True False False False]
[0 0 1 0 0 1 0 0 1 0 0 0]
3
12
0.25
```

Exercise 7 (1/2)

- Create a vector named "x" with the following values: 36, 28, 19, 22, 27, 28, 30, 31, 38, 46, 40, 29, 21, 28, 39, 46, 43, 27, 30 and 54
- Calculate the size of the vector
- Calculates its average without using the array.mean() function
- Calculates its range (maximum value minus the minimum)
- Calculates its variance without using the array.var() function

$$Var(X) = \frac{\sum_1^n (x_i - \bar{X})^2}{n}$$

Exercise 7 (2/2)

- Based on the above calculation it prints out the standard deviation (square root of the variance)
- Calculates the median without using np.median(array)
- Calculate mode without using statistics.mode. You can use the most_common() method of the Counter class

```
[36 28 19 22 27 28 30 31 38 46 40 29 21 28 39 46 43 27 30 54]
Len 20
Mean 33.1
Range 35
Varianza 82.18999999999998
Desviación Típica 9.065870063044141
Mediana 30.0
Moda 28
```

https://pymotw.com/3/collections/counter.html

Exercise 7 - Solution

```
# Crea un vector llamado x
x = np.array([36, 28, 19, 22, 27, 28, 30, 31, 38, 46, 40, 29, 21, 28, 39, 46, 43, 27, 30, 54])
print("Array", x)

# Calcula su tamaño
print("Tamaño", len(x))

# Calcula la media
print("Media", x.sum() / len(x))

# Rango
print("Rango", x.max() - x.min())

# Varianza
print("Varianza", (((x - x.mean()) ** 2).sum() / len(x)))

# Sd
import math
print("Desviación Típica", (math.sqrt(((x - x.mean()) ** 2).sum() / len(x))))

# Mediana
print("Mediana", np.sort(x)[[int(len(x) / 2) - 1, int(len(x) / 2)]].mean())

# Mode
from collections import Counter
print("Moda", Counter(x).most_common()[0][0])
```

Set operations

- Numpy offers a set of basic operations for one-dimensional arrays
- The most common is **np.unique()** which returns the unique values of an array

```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe', 'Alex'])
```

```
np.unique(names)
```

```
array(['Alex', 'Bob', 'Joe', 'Will'],
      dtype='<U4')
```

Linear Algebra with NumPy

- Linear algebra is an essential part of the implementation of Machine Learning algorithms
- The most important methods are:
- Transposing a matrix: matrix.T

```
matriz = np.arange(1,11).reshape((2,5))
matriz
```

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

```
matriz.T
```

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

Linear Algebra with NumPy

- Matrix multiplication: matrix1.dot(matrix2)

```
matriz1 = np.arange(1,11).reshape((2,5))
matriz1
```

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

```
matriz2 = np.arange(11,1,-1).reshape((5,2))
matriz2
```

```
array([[11, 10],
       [ 9,  8],
       [ 7,  6],
       [ 5,  4],
       [ 3,  2]])
```

```
matriz1.dot(matriz2)
```

```
array([[ 85,  70],
       [260, 220]])
```

Linear Algebra with NumPy

- Inverse of a matrix: np.linalg.inv(matrix)

```
matriz = np.array([[1., 2.], [3., 4.]])
matriz
```

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

```
np.linalg.inv(matriz)
```

```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

Linear Algebra with NumPy

- Concatenate matrices (by rows)

```
matriz = np.arange(1, 11).reshape((2,5))
matriz
```

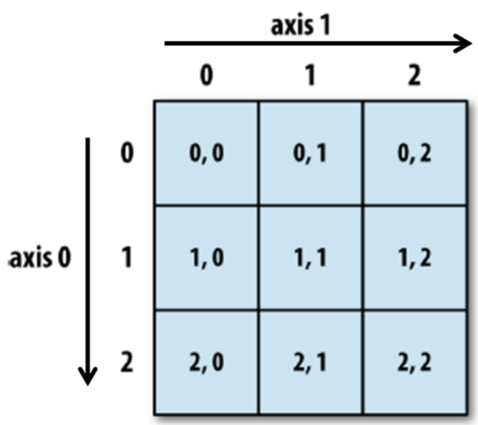
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

```
np.concatenate((matriz, matriz), axis = 0)
```

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

```
np.concatenate((matriz, matriz), axis = 1)
```

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



IMMUNE CODING INSTITUTE

Linear Algebra with NumPy

- The function **concatenate** on two arrays returns another array where its elements have been concatenated (it never returns a matrix)

```
array = np.arange(1, 6)
array
```

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

```
np.concatenate((array, array))
```

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

```
np.concatenate((array, array), axis = 1)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-11-432ab28d3493> in <module>()
----> 1 np.concatenate((array, array), axis = 1)

IndexError: axis 1 out of bounds [0, 1)
```

Exercise 8 (1/2)

- Given the following matrices:

$$A = \begin{bmatrix} 2 & 1 & 3 & 3 \\ 4 & 4 & 6 & 2 \\ 8 & 9 & 1 & 11 \end{bmatrix} \qquad B = \begin{bmatrix} 4 & 41 & 3 \\ 4 & 24 & 3 \\ 6 & 12 & 1 \\ 1 & 22 & 32 \end{bmatrix}$$

- Calculate the transposition of A
- Multiply A and B. Does that give the same result as B multiplied by A (np.array_equal)?
- Add the array [1, 1, 1, 2] to matrix B so that it becomes the 4th column
- Calculates the inverse of matrix B

Exercise 8 (2/2)

```
A
 [[ 2  1  3  3]
 [ 4  4  6  2]
 [ 8  9  1 11]]
B
 [[ 4 41  3]
 [ 4 24  3]
 [ 6 12  1]
 [ 1 22 32]]
Traspuesta de A
 [[ 2  4  8]
 [ 1  4  9]
 [ 3  6  1]
 [ 3  2 11]]
A multiplicado por B
 [[ 33 208 108]
 [ 70 376  94]
 [ 85 798 404]]
```

```
B multiplicado por A
 [[196 195 261 127]
 [128 127 159  93]
 [ 68  63  91  53]
 [346 377 167 399]]
¿Da el mismo resultado?
 False
Añadimos una columna a B
 [[ 4 41  3  1]
 [ 4 24  3  1]
 [ 6 12  1  1]
 [ 1 22 32  2]]
Inversa de B
 [[ 5.63467492e-01 -1.35294118e+00  6.84210526e-01  5.26315789e-02]
 [ 5.88235294e-02 -5.88235294e-02  2.95115105e-19 -3.06181921e-19]
 [ 2.10526316e-01 -5.00000000e-01  1.84210526e-01  5.26315789e-02]
 [-4.29721362e+00  9.32352941e+00 -3.28947368e+00 -3.68421053e-01]]
```

Exercise 8 - Solution

```
# Dadas las matrices A y B
A = np.array([[2, 1, 3, 3], [4, 4, 6, 2], [8, 9, 1, 11]])
print("a", a)

B = np.array([[4, 41, 3], [4, 24, 3],[6, 11, 1], [1, 22, 32]])
print("b", b)

# Transposición de a
print("Transpuesta de a", A.T)

# A multiplicada por B
print("A * B", A.dot(B))

# B multiplicada por A
print("A * B", B.dot(A))

# ¿Da el mismo resultado?
print ("¿Mismo Resultado?", np.array_equal(A.dot(B), B.dot(A)))

# Añadimos una cuarta columna a B
B = np.concatenate((B, np.array([1, 1, 1, 2]).reshape(4, 1)), axis = 1)
print("Añadimos una 4° columna a B", B)

# Inversa de B
print("Inversa de B", np.linalg.inv(B))
```

Exercise 9

- Create a function that accepts a numerical array and returns the same ordered array using the QuickSort algorithm

```
function quicksort(array):

    si el array está vacio salir y devolver un array vacio

    pivots = elementos del array iguales al primer elemento
    lesser = elementos del array menores al primer elemento
    greatter = elementos del array mayores al primer elemento

    devolver quicksort(lesser) + pivots + quicksort(greater)
```

Exercise 9 - Solution

```
# Quicksort

def quicksort(array):
  a = np.array(array)
  if not a.size:
    return []

  pivots = a[ a == a[0]]
  lesser = a[ a < a[0]]
  greater = a[ a > a[0]]

  return np.concatenate((quicksort(lesser), pivots, quicksort(greater)))


quicksort( [-44, 0, 2, -34, 3, 44, -1] )
```

```
array([-44., -34.,  -1.,   0.,   2.,   3.,  44.])
```

Random numbers with NumPy

- Numpy has its own module to generate random and pseudo random numbers through np.random
- For example, we can :
  - Set the random seed with **np.random.seed**

```
np.random.seed(10)
print(np.random.randint(1, 10, 5))
print(np.random.randint(1, 10, 15))
```

```
[5 1 2 1 2]
[9 1 9 7 5 4 1 5 7 9 2 9 5 2 4]
```

```
print(np.random.randint(1, 10, 5))
print(np.random.randint(1, 10, 15))
```

```
[7 6 4 7 2]
[5 3 7 8 9 9 3 1 7 8 9 2 8 2 5]
```

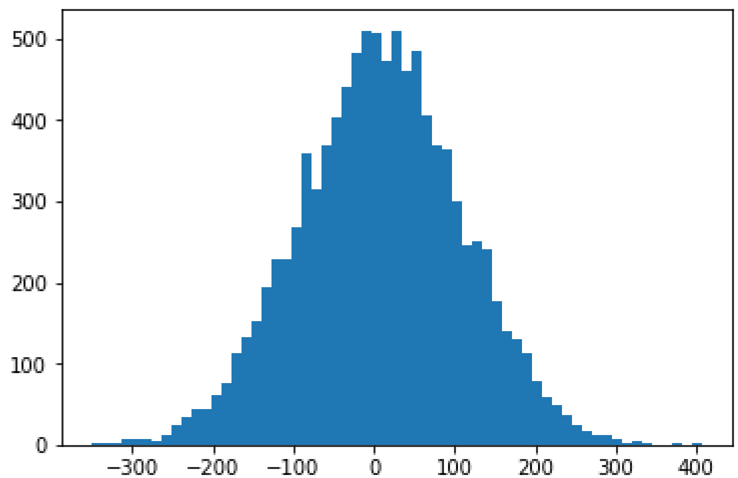
```
np.random.seed(10)
print(np.random.randint(1, 10, 5))
print(np.random.randint(1, 10, 15))
```

```
[5 1 2 1 2]
[9 1 9 7 5 4 1 5 7 9 2 9 5 2 4]
```

Random numbers with NumPy

- Generate random numbers according to the normal distribution
  np.random.normal (mean, standard deviation, (rows, columns))

```
import matplotlib.pyplot as plt

mu, sigma = 10, 100
arr = np.random.normal(mu, sigma, size = 10000)

plt.hist(arr, bins = 'auto')
plt.show()
```



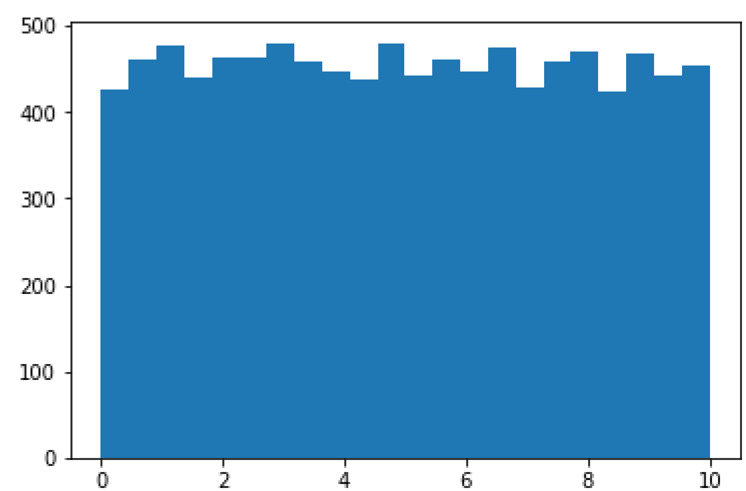
Random numbers with NumPy

- Generate integers based on the uniform distribution: **np.random.randint**(minimum, maximum, (rows, columns))
- Generate decimal numbers according to the uniform distribution **np.random.uniform**(minimum, maximum, (rows, columns)

```
np.random.randint(0, 10, 10)
```

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

```
arr = np.random.uniform(0, 10, 10000)

plt.hist(arr, bins = 'auto')
plt.show()
```



Random numbers with NumPy

- Generate random numbers based on an array
  np.random.choice(array, size=(rows,columns), replace, probabilities)

```
np.random.choice(["A", "B", "C"], size = 2)
```

```
array(['C', 'B'],
      dtype='<U1')
```

```
np.random.choice(["A", "B", "C"], size = (2,2), replace = True)
```

```
array([['B', 'A'],
       ['B', 'B']],
      dtype='<U1')
```

```
np.random.choice(["A", "B", "C"], size = (2,2), replace = True, p = [0.8, 0.1, 0.1])
```

```
array([['C', 'A'],
       ['A', 'B']],
      dtype='<U1')
```

Exercise 10 (1/2)

- Set the random seed to 5
- Generate an array of 10,000 positions according to the normal distribution (mean = 5 and standard deviation of 10)
- Print the first 10 values of the array
- Check that the mean and standard deviation match the parameters
- Generate a 2x2 matrix with integers between 1 and 99
- Generate an array of 10 positions according to the uniform distribution between 1 and 10

Exercise 10 (2/2)

- Generate an array of 10 positions with the letters A, B and C, with the following probabilities: A = 60%, B = 30%

```
[ 9.41227487  1.69129848 29.30771187  2.4790787   6.09609842 20.82481117
 -4.09232405 -0.91636658  6.87603226  1.70130042]
media =  4.997066946970099
sd =  10.008499791458087
[[25 55]
 [78 61]]
[90.03418563 34.60973983 19.09929278 11.56579953 71.67677291 24.63382647
 52.05741326 90.23565038 15.5590007  30.5917763 ]
['A' 'A' 'A' 'A' 'A' 'B' 'A' 'A' 'B' 'A']
```

Exercise 10 - Solution

```
# Establece la semilla aleatoria a 10
np.random.seed(5)

# Genera 10.000 números según la distribución normal (media 5, sd = 10)
a = np.random.normal(5, 10, 10000)
print(a[:10])

# Comprueba la media y la desviación típica corresponden a los parámetros
print("Mean", a.mean())
print("Sd", a.std())

# Genera una matriz de 2x2 de enteros entre 1 y 99
print(np.random.randint(0, 100, (2, 2)))

# Genera un array de 10 números entre 1 y 99 según la distribución uniforme
print(np.random.uniform(0, 100, 10))

# Genera un array de 10 posiciones con las letras "A", "B" y "C", Probabilidades A: 60%, B: 30%
print(np.random.choice(["A","B","C"], 10, p = [.6, 0.3, 0.1]))
```

Loading and Saving Data with NumPy

- Numpy offers an **np.save()** and **np.load()** method to enable you to write and recover data from disk

```
values = np.array([6, 0, 0, 3, 2, 5, 6])
values
```

```
array([6, 0, 0, 3, 2, 5, 6])
```

```
np.save('some_array', values)
```

```
values2 = np.load('some_array.npy')
```

```
values2
```

```
array([6, 0, 0, 3, 2, 5, 6])
```

Index

- NumPy
- Pandas
- Dataframes
- Reading / Writing data
- Exploring a DataFrame
- Operations on a DataFrame

Pandas

- [Pandas](https://pandas.pydata.org) is the most popular Python library for cleaning, exploring, and manipulating data.

model	mpg	cyl	disp	hp	drat
Mazda RX4	21	6	160	110	3.9
Mazda RX4 Wag	21	6	160	110	3.9
Datsun 710	22.8	4	108	93	3.85
Hornet 4 Drive	21.4	6	258	110	3.08
Hornet Sportabout	18.7	8	360	175	3.15
Valiant	18.1	6	225	105	2.76
Duster 360	14.3	8	360	245	3.21
Merc 240D	24.4	4	146.7	62	3.69
Merc 230	22.8	4	140.8	95	3.92
Merc 280	19.2	6	167.6	123	3.92
Merc 280C	17.8	6	167.6	123	3.92
Merc 450SE	16.4	8	275.8	180	3.07
Merc 450SL	17.3	8	275.8	180	3.07

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

Pandas

pandas documentation

Date: Feb 26, 2020 **Version**: 1.1.0.dev0+609.g52a63ab42

Download documentation: PDF Version | Zipped HTML

Useful links: Binary Installers | Source Repository | Issues & Ideas | Q&A Support | Mailing List

`pandas` is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Getting started

New to *pandas*? Check out the getting started guides. They contain an introduction to *pandas'* main concepts and links to additional tutorials.

To the getting started guides

User guide

The user guide provides in-depth information on the key concepts of pandas with useful background information and explanation.

To the user guide

API reference

The reference guide contains a detailed description of the pandas API. The reference describes how the methods work and which parameters can be used. It assumes that you have an understanding of the key concepts.

To the reference guide

Developer guide

Saw a typo in the documentation? Want to improve existing functionalities? The contributing guidelines will guide you through the process of improving pandas.

To the development guide

https://dev.pandas.io/docs/index.html

Data structures in Pandas

- In pandas there are mainly two data structures:
  - Series
  - DataFrames

Series

- A Serie is a structure composed of two elements:
  - A one-dimensional array of **values**
  - A one-dimensional array of indexes or tags called **index**

```
import numpy as np
import pandas as pd
```

```
serie = pd.Series(np.array([2, 5, 4.3, -6.4, 12]))
serie
```

```
0     2.0
1     5.0
2     4.3
3    -6.4
4    12.0
dtype: float64
```

index	valores
0	2
1	5
2	4,3
3	-6,4
4	12

Series

- Pandas provides the **index** attributes and **values** to access these elements independently

```
serie = pd.Series(np.array([2, 5, 4.3, -6.4, 12]))
serie
```

```
0     2.0
1     5.0
2     4.3
3    -6.4
4    12.0
dtype: float64
```

```
serie.index
```

```
RangeIndex(start=0, stop=5, step=1)
```

```
serie.values
```

```
array([ 2. ,  5. ,  4.3, -6.4, 12. ])
```

Series creation

- To create a serie, Pandas offers you different options:
  - A list
  - A NumPy array

```
pd.Series([1,2,3,4,5])
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

```
pd.Series(np.arange(1,6))
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int32
```

Series creation

- A Python dictionary

```
un_diccionario = {
    "David": 5.4,
    "Pablo": 128,
    "Nuria": 26,
    "Mario": -12,
    "Javier": 0
}

un_diccionario
```

```
{'David': 5.4, 'Javier': 0, 'Mario': -12, 'Nuria': 26, 'Pablo': 128}
```

```
pd.Series(un_diccionario)
```

```
David       5.4
Javier      0.0
Mario     -12.0
Nuria      26.0
Pablo     128.0
dtype: float64
```

Series creation

- Pandas allows to create a serie specifying values and indexes

```
serie = pd.Series(np.array([12, 21, 43, 11]),
                  index=["Juan", "Marta", "Paco", "Lorenzo"])
serie
```

```
Juan       12
Marta      21
Paco       43
Lorenzo    11
dtype: int32
```

Names

- It is posible assign a name, both to the series and to the index:

```
serie = pd.Series(list(range(1,4)))
serie
```

```
0    1
1    2
2    3
dtype: int64
```

```
serie.name = "Números"
serie.index.name = "Índice"
serie
```

```
Índice
0    1
1    2
2    3
Name: Números, dtype: int64
```

Empty or Null Values

- NumPy handles the concept of empty value or gap in information through the value **np.nan**

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
states = ['California', 'Ohio', 'Oregon', 'Texas']
```

```
serie = pd.Series(sdata, index=states)
serie
```

```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

Empty or Null Values

- In pandas, you can check the nulls with **isnull()** method

```
serie
```

```
California        NaN
Ohio          35000.0
Oregon         1600.0
Texas         71000.0
dtype: float64
```

```
serie.isnull()
```

```
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

Exercise 11

- Create a series with the values 4,3,2,1 and 5
- Print its index as a python list
- Without re-creating the serie, assign an index where each number is related to its letter
- Rename the series as "numbers" and the index as "letters“
- Create a new element in the serie whose index is "six" and its value is empty

```
letters
four     4.0
three    3.0
two      2.0
one      1.0
five     5.0
six      NaN
Name: numbers, dtype: float64
```

Exercise 11 - Solution

```python
# Crea una serie con los valores 4,3,2,1 y 5
serie = pd.Series([4,3,2,1,5])
print(serie)

# Imprime su índice
print(list(serie.index))

# Sin crear de nuevo la serie asigna un índice donde cada número esté relacionado con su letra
serie.index = ["four", "three", "two", "one", "five"]
print(serie)

# Renombra la serie como "numbers" y el índice como letters
serie.name = "numbers"
serie.index.name = "letters"
print(serie)

# Crea un nuevo elemento cuyo ínice es "six" y su valor es vacio
serie["six"] = np.nan
print(serie)
```

Access to the Series

- To access the contents of a Series is similar to accessing an array in NumPy
- It is possible to select elements of a series through the following elements:
  - By position or index name, by returning a single item

```
serie = pd.Series(np.array([12, 21, 43, 11]),
                  index=["Juan", "Marta", "Paco", "Lorenzo"])
serie
```

```
Juan       12
Marta      21
Paco       43
Lorenzo    11
dtype: int32
```

```
serie[0]
```

```
12
```

```
serie["Juan"]
```

```
12
```

Access to the Series

- An array of elements (position or index name), returning another series

```
serie
```

```
Juan        12
Marta       21
Paco        43
Lorenzo     11
dtype: int32
```

```
serie[[0,3]]
```

```
Juan       12
Lorenzo    11
dtype: int32
```

```
serie["Juan":"Paco"]
```

```
Juan     12
Marta    21
Paco     43
dtype: int32
```

Access to the Series

- An array of logical values, returning another set

```
serie
```

```
Juan       12
Marta      21
Paco       43
Lorenzo    11
dtype: int64
```

```
serie[(serie <40) & (serie > 20)]
```

```
Marta    21
dtype: int64
```

```
serie[serie.isnull()]
```

```
Series([], dtype: int64)
```

Removal of elements

- The **drop()** method allows to remove elements from a serie
- Does not change the serie (returns the result)

```
serie
```

```
Juan       12
Marta      21
Paco       43
Lorenzo    11
dtype: int32
```

```
serie.drop('Juan')
```

```
Marta      21
Paco       43
Lorenzo    11
dtype: int32
```

```
serie
```

```
Juan       12
Marta      21
Paco       43
Lorenzo    11
dtype: int32
```

Operations with Series

- Pandas allows to operate with Series as if it was a NumPy array

```
serie_uno = pd.Series(np.arange(0,4),
                      index =["a", "b", "c", "d"])
serie_uno
```

```
a    0
b    1
c    2
d    3
dtype: int32
```

```
serie_dos = pd.Series(np.arange(100, 104),
                      index = ["b", "c", "a",  "e"])
serie_dos
```

```
b    100
c    101
a    102
e    103
dtype: int32
```

```
serie_uno + (serie_dos * 1000)
```

```
a    102000.0
b    100001.0
c    101002.0
d         NaN
e         NaN
dtype: float64
```

Exercise 12 (1/2)

- Using the previous exercise serie
- Select even numbers
- Select empty values
- Select the items that are in positions 4 and 3 (5,1)
- Select the items "two" and "six"

```
serie = pd.Series([4, 3, 2, 1, 5, np.nan], name = "numbers",
                  index = ["four", "three", "two", "one", "five", "six"])
serie
```

```
four     4.0
three    3.0
two      2.0
one      1.0
five     5.0
six      NaN
Name: numbers, dtype: float64
```

Exercise 12 (2/2)

- Select the last item in the serie
- Select all items in reverse order (::-1)
- Multiply the series by 2
- Assign an empty value to numbers greater than 4

Exercise 12 - Solution

```
# Selecciona los valores impares
print(serie[ serie % 2 == 1])

# Selecciona los valores vacios
print(serie[ serie.isnull() ])

# Selecciona los valores que están en la posición  4 y 3
print(serie[ [4, 3]])

# Selecciona los valores "two", "six
print(serie[ ["two", "six"]])

# Selecciona el último valor
print(serie[-1:])

# Selecciona todos los valores en orden inverso
print(serie[::-1])

# Multiplica la serie por 2
print(serie * 2)

# Asigna un valor vacio a los valores mayores que 4
serie[ serie > 4] = np.nan
print(serie)
```

Counting values

- The **value_counts**() method counts the different categorical values that a serie contains, returning another serie with the result

```
serie = pd.Series(np.random.choice(["A", "B", "C"], 1000))
serie.head()
```

```
0    A
1    C
2    A
3    B
4    A
dtype: object
```

```
serie.value_counts()
```

```
C    356
A    327
B    317
dtype: int64
```

Text functions

- Pandas provides a very rich set of functions to manipulate chains. For example: lower(), upper(), len(), get(), split(), strip()

```
s = pd.Series(['AAA_23', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog', 'cat'])
```

```
list(s.str.lower())
```

```
['aaa_23', 'b', 'c', 'aaba', 'baca', nan, 'caba', 'dog', 'cat']
```

```
list(s.str.len())
```

```
[6.0, 1.0, 1.0, 4.0, 4.0, nan, 4.0, 3.0, 3.0]
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/text.html

Text functions

- Pandas provides a very rich set of functions to manipulate chains. For example: lower(), upper(), len(), get(), split(), strip()

```
s.str.split("_")
```

```
0    [AAA, 23]
1          [B]
2          [C]
3       [AaAa]
4       [Baca]
5          NaN
6       [CABA]
7        [dog]
8        [cat]
dtype: object
```

```
(s.str.split("_")).str.get(0)
```

```
0     AAA
1       B
2       C
3    AaAa
4    Baca
5     NaN
6    CABA
7     dog
8     cat
dtype: object
```

Applying functions to Series

- With **map** and **lambda** functions we can transform a list in Python

```
lista = [1, 2, -3, 5, 10]

def suma_uno(lista):
  lista_resultado = []
  for item in lista:
    lista_resultado.append( item + 1)

  return lista_resultado

suma_uno(lista)
```

```
[2, 3, -2, 6, 11]
```

Applying functions to Series

- With **map** and **lambda** functions we can transform a list in Python

```
lista = [1, 2, -3, 5, 10]

def suma_uno(elemento):
  return elemento + 1

list(map(suma_uno, lista))
```

```
[2, 3, -2, 6, 11]
```

Applying functions to Series

- With **map** and **lambda** functions we can transform a list in Python

```
lista = [1, 2, -3, 5, 10]

list(map(lambda elemento: elemento + 1, lista))
```

```
[2, 3, -2, 6, 11]
```

Applying functions to Series

- Pandas, through the **map**() function, allows to execute any function on a value of a series, so that we transform its value

```
serie = pd.Series([0,34, 34, -45])
serie
```

```
0     0
1    34
2    34
3   -45
dtype: int64
```

```
serie.map(lambda x: x + 23)
```

```
0    23
1    57
2    57
3   -22
dtype: int64
```

```
serie.map(lambda x: 5 if x > 4 else -1)
```

```
0   -1
1    5
2    5
3   -1
dtype: int64
```

Exercise 13

- Create a series containing a number range from 1 to 10
- Add 10 to the odd values without using map
- Add 10 to the even values using map
- Transform the series so that the prefix “Item" is added to each element
- Show a serie with the last 5 characters of each item

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 12, 3, 14, 5, 16, 7, 18, 9, 20]
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
['Item 11', 'Item 12', 'Item 13', 'Item 14', 'Item 15', 'Item 16', 'Item 17', 'Item 18', 'Item 19', 'Item 20']
['em 11',
 'em 12',
 'em 13',
 'em 14',
 'em 15',
 'em 16',
 'em 17',
 'em 18',
 'em 19',
 'em 20']
```

Exercise 13 - Solution

```
# Crea una serie que contenga un rango de numeros de 1 a 10
serie = pd.Series(range(1, 11))
print(list(serie))

# Añade 10 a los valores pares sin utilizar map
serie[ serie % 2 == 0 ] = serie + 10
print(list(serie))

# Añade 10 a los valores impares con map
serie = serie.map(lambda x: x + 10 if x % 2  == 1 else x)
print(list(serie))

# Añade "Item" a cada elemento de la serie
serie = serie.map(lambda x: "Item " + str(x))
print(list(serie))

# Muestra otra serie con los últimos 5 caracteres de cada elemento
list(serie.map(lambda x: x[-5:]))
```

Index

- NumPy
- Pandas
- Dataframes
- Reading / Writing data
- Exploring a DataFrame
- Operations on a DataFrame

DataFrames

- A DataFrame is a tabular structure formed by a set of series, which share the index

```
col1 = pd.Series(["Sergio", "David", "Natalia", "Daniel"])
col1
```

```
0     Sergio
1      David
2    Natalia
3     Daniel
dtype: object
```

```
col2 = pd.Series([30, 10, 11, 27])
col2
```

```
0    30
1    10
2    11
3    27
dtype: int64
```

DataFrames

- A DataFrame is a tabular structure formed by a set of series, which share the index

```
pd.DataFrame({ "nombre": pd.Series(["Sergio", "David", "Natalia", "Daniel"]),
               "edad" : pd.Series([30, 10, 11, 27])},
            columns = ["nombre", "edad"])
```

	nombre	edad
0	Sergio	30
1	David	10
2	Natalia	11
3	Daniel	27

	Columnas	
	Serie 1	Serie 2
index		
0	Sergio	30
1	David	10
2	Natalia	11
3	Daniel	27

Creating a DataFrame

- When a DataFrame is created from a series dictionary, Pandas takes into account the series indexes and can create null values

```
serie1 = pd.Series([1,2],
                   index = ["uno", "dos"])
print(serie1)
```

```
uno    1
dos    2
dtype: int64
```

```
serie2 = pd.Series([2, 3],
                   index = ["dos", "tres"])
print(serie2)
```

```
dos     2
tres    3
dtype: int64
```

Creating a DataFrame

- When a DataFrame is created from a series dictionary, Pandas takes into account the series indexes and can create null values

```
df = pd.DataFrame({"c1" : pd.Series([1,2], index = ["uno", "dos"]),
                   "c2" : pd.Series([2, 3], index = ["dos", "tres"])},
                  columns = ["c1", "c2"],
                  index = ["uno", "dos", "tres"])
df
```

	c1	c2
uno	1.0	NaN
dos	2.0	2.0
tres	NaN	3.0

Creating a DataFrame

- Usually a DataFrame is created from a dictionary of lists containing the same elements

```
diccionario = {
    "nombre": ["Julio", "Nuria", "Jose", "Luis", "Daniel"],
    "edad": [22, 26, 28, 25, 24],
    "sexo": ["M", "F", "M", "M", "M"]
}

dataframe = pd.DataFrame(diccionario, columns = ["nombre", "edad", "sexo"])
dataframe
```

	nombre	edad	sexo
0	Julio	22	M
1	Nuria	26	F
2	Jose	28	M
3	Luis	25	M
4	Daniel	24	M

Creating a DataFrame

- A DataFrame can also be created from 3 arrays: values, columns and rows

```
df = pd.DataFrame( np.arange(16).reshape(4,4),
                   columns = list("ABCD"),
                   index = ["uno", "dos", "tres", "cuatro"]
                   )
df
```

	A	B	C	D
uno	0	1	2	3
dos	4	5	6	7
tres	8	9	10	11
cuatro	12	13	14	15

Creating a DataFrame

- These arrays can be accessed through the attributes **index**, **columns** and **values**

```
df.values
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
df.index
```

```
Index(['uno', 'dos', 'tres', 'cuatro'], dtype='object')
```

```
df.columns
```

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

	A	B	C	D
uno	0	1	2	3
dos	4	5	6	7
tres	8	9	10	11
cuatro	12	13	14	15

IMMUNE CODING INSTITUTE

Exercise 14

- Create the following DataFrame, respecting the order of the columns, and assign it to the variable ‘df’
- name: 'Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'
- evolution: 'Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'
- type: 'grass', 'fire', 'water', 'bug'
- hp : 45, 39, 44, 45
- pokedex : 'yes', 'no', 'yes', 'no'

	evolution	name	hp	pokedex	type
A	Ivysaur	Bulbasaur	45	yes	grass
B	Charmeleon	Charmander	39	no	fire
C	Wartortle	Squirtle	44	yes	water
D	Metapod	Caterpie	45	no	bug

Exercise 14 - Solution

```
# Creación de un DataFrame
df = pd.DataFrame({
        "name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
        "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
        "type": ['grass', 'fire', 'water', 'bug'],
        "hp" : [45, 39, 44, 45],
        "pokedex" : ['yes', 'no', 'yes', 'no']
        },
        columns = ["evolution", "name", "hp", "pokedex", "type"],
        index = list("ABCD")
)
df
```

Transposing a DataFrame

- The **dataframe** attribute **T** allows to transpose a datraframe (for instance to change rows by columns)

```
df = pd.DataFrame({
    "c1" : np.arange(1,5),
    "c2" : np.repeat("Menor que 5", 4)
})

df
```

	c1	c2
0	1	Menor que 5
1	2	Menor que 5
2	3	Menor que 5
3	4	Menor que 5

```
df.T
```

	0	1	2	3
c1	1	2	3	4
c2	Menor que 5	Menor que 5	Menor que 5	Menor que 5

Exercise 15

- On the previous exercise's dataframe, convert rows into columns and columns into rows
- Displays values, columns and row indexes

	A	B	C	D
evolution	Ivysaur	Charmeleon	Wartortle	Metapod
name	Bulbasaur	Charmander	Squirtle	Caterpie
hp	45	39	44	45
pokedex	yes	no	yes	no
type	grass	fire	water	bug

```
[['Ivysaur' 'Bulbasaur' 45 'yes' 'grass']
 ['Charmeleon' 'Charmander' 39 'no' 'fire']
 ['Wartortle' 'Squirtle' 44 'yes' 'water']
 ['Metapod' 'Caterpie' 45 'no' 'bug']]
['evolution', 'name', 'hp', 'pokedex', 'type']
['A', 'B', 'C', 'D']
```

Exercise 15 - Solution

```
# Sobre el DataFrame del ejercicio anterior
df = pd.DataFrame({
        "name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
        "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
        "type": ['grass', 'fire', 'water', 'bug'],
        "hp" : [45, 39, 44, 45],
        "pokedex" : ['yes', 'no', 'yes', 'no']
        },
        columns = ["evolution", "name", "hp", "pokedex", "type"],
        index = list("ABCD")
)

# Convierte las columnas en filas y las filas en columnas
display(df.T)

# Muestra los valores, y los índices de fila y columna
print(df.values)
print(list(df.columns))
print(list(df.index))
```

Selecting a column

- Pandas allows access to a column in two ways:
  - Dataframe.column
  - Dataframe ["column"]
- The object it returns is always a Serie

```
diccionario = {
    "n1": np.arange(1,5),
    "n2": np.arange(50, 54)
}

dataframe = pd.DataFrame(diccionario)
dataframe
```

	n1	n2
0	1	50
1	2	51
2	3	52
3	4	53

```
dataframe.n1
```

```
0    1
1    2
2    3
3    4
Name: n1, dtype: int32
```

```
dataframe["n1"]
```

```
0    1
1    2
2    3
3    4
Name: n1, dtype: int32
```

Multi-column selection

- To select a set of columns you need to specify a list of columns
- The object it returns is a new DataFrame

```
dataframe = pd.DataFrame(np.arange(1,16).reshape(3,5),
                         columns = ["c1","c2", "c3", "c4", "c5"],
                         index = list("ABC")
)
dataframe
```

	c1	c2	c3	c4	c5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

```
dataframe[["c2", "c4"]]
```

	c2	c4
A	2	4
B	7	9
C	12	14

Exercise 16

- About last exercise's Dataframe
- Obtain the 'name' column as a serie, using two different methods
- Get the column 'name' as DataFrame

	evolution	name	hp	pokedex	type
A	Ivysaur	Bulbasaur	45	yes	grass
B	Charmeleon	Charmander	39	no	fire
C	Wartortle	Squirtle	44	yes	water
D	Metapod	Caterpie	45	no	bug

```
A     Bulbasaur
B    Charmander
C      Squirtle
D      Caterpie
Name: name, dtype: object
```

	name
A	Bulbasaur
B	Charmander
C	Squirtle
D	Caterpie

Exercise 16 - Solution

```
# Sobre el DataFrame del ejercicio anterior
df = pd.DataFrame({
        "name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
        "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
        "type": ['grass', 'fire', 'water', 'bug'],
        "hp" : [45, 39, 44, 45],
        "pokedex" : ['yes', 'no', 'yes', 'no']
        },
        columns = ["evolution", "name", "hp", "pokedex", "type"],
        index = list("ABCD")
)

# Obten la columna 'name' como una Serie a través de 2 métodos diferentes
print(df.name)
print(df["name"])

# Obten la colunmna 'name' como una DataFrame
display(df[["name"]])
```

Selecting a subset of rows

- To select a subset of rows we specify two numbers separated by a colon (similar to lists)

```
dataframe
```

	C1	C2	C3	C4	C5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

```
dataframe[0:2]
```

	C1	C2	C3	C4	C5
A	1	2	3	4	5
B	6	7	8	9	10

```
dataframe[-2:]
```

	C1	C2	C3	C4	C5
B	6	7	8	9	10
C	11	12	13	14	15

Selecting a subset of rows

- We could also use an array of logical values

```
dataframe[[False, True, False]]
```

	C1	C2	C3	C4	C5
B	6	7	8	9	10

```
dataframe[dataframe.index == 'A']
```

	C1	C2	C3	C4	C5
A	1	2	3	4	5

```
dataframe[dataframe.C1 > 1]
```

	C1	C2	C3	C4	C5
B	6	7	8	9	10
C	11	12	13	14	15

```
dataframe
```

	C1	C2	C3	C4	C5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

Exercise 17

- About last exercise's Dataframe
- A - Select from the first 3 rows
- B - Select the last 2 rows
- C- Select odd rows (start:stop:step)
- D- Selects all rows but in reverse order (::-1)

A

	evolution	name	hp	pokedex	type
A	Ivysaur	Bulbasaur	45	yes	grass
B	Charmeleon	Charmander	39	no	fire
C	Wartortle	Squirtle	44	yes	water

B

	evolution	name	hp	pokedex	type
C	Wartortle	Squirtle	44	yes	water
D	Metapod	Caterpie	45	no	bug

C

	evolution	name	hp	pokedex	type
B	Charmeleon	Charmander	39	no	fire
D	Metapod	Caterpie	45	no	bug

D

	evolution	name	hp	pokedex	type
D	Metapod	Caterpie	45	no	bug
C	Wartortle	Squirtle	44	yes	water
B	Charmeleon	Charmander	39	no	fire
A	Ivysaur	Bulbasaur	45	yes	grass

Exercise 17 - Solution

```python
# Sobre el DataFrame del ejercicio anterior
df = pd.DataFrame({
        "name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
        "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
        "type": ['grass', 'fire', 'water', 'bug'],
        "hp" : [45, 39, 44, 45],
        "pokedex" : ['yes', 'no', 'yes', 'no']
        },
        columns = ["evolution", "name", "hp", "pokedex", "type"],
        index = list("ABCD")
)

# A - Selecciona las 3 primeras filas
display(df[0:3])

# B - Selecciona las 2 ultimas filas
display(df[-2:])

# C - Selecciona las filas impares
display(df[1::2])

# D - Selecciona las filas en orden inverso
display(df[::-1])
```

Selection of values per position

- Pandas provides the **iloc** and **iat** attributes to select values by position: iloc [rows, columns]

```
dataframe.iloc[[0,2], [0,1,4]]
```

	C1	C2	C5
A	1	2	5
C	11	12	15

```
dataframe.iloc[[True, False, False], [0,1,4]]
```

	C1	C2	C5
A	1	2	5

```
dataframe.iloc[dataframe.index == 'A', :]
```

	C1	C2	C3	C4	C5
A	1	2	3	4	5

```
dataframe
```

	C1	C2	C3	C4	C5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

Selection of values per position

- With **iat** you can specify a single row / column and always return a value instead of a Dataframe

```
dataframe
```

	C1	C2	C3	C4	C5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

```
dataframe.iat[0, 1]
```

2

```
dataframe.iat[0,-1]
```

5

Exercise 18

- On the Dataframe of the previous exercise and using positions:
- A - Select all columns, 2nd and 4th row
- B - Select the 'hp' and 'type' columns of all rows
- C - Select the column 'name' from the first and last row
- D – Value that is located in the first row, last column

A

	evolution	name	hp	pokedex	type
B	Charmeleon	Charmander	39	no	fire
D	Metapod	Caterpie	45	no	bug

C

	name
A	Bulbasaur
D	Caterpie

B

	hp	type
A	45	grass

D

```
'grass'
```

Exercise 18 - Solution

```
# Sobre el DataFrame del ejercicio anterior
df = pd.DataFrame({
        "name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
        "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
        "type": ['grass', 'fire', 'water', 'bug'],
        "hp" : [45, 39, 44, 45],
        "pokedex" : ['yes', 'no', 'yes', 'no']
        },
        columns = ["evolution", "name", "hp", "pokedex", "type"],
        index = list("ABCD")
)

# A - Selecciona todas las columnas, 2° y 4° fila
display(df.iloc[[1,3], :])

# B - Selecciona las columnas 'hp' y 'type' de la primera fila
display(df.iloc[[0] , [2, 4]])

# C - Selecciona la columna 'name', primera y ultima fila
display(df.iloc[[0, len(df) -1], [1]])

# D - Selecciona el valor de la primera fila, última columna
display(df.iat[0, len(df.columns) -1 ])
```

Selection of values per label

- Pandas provides the **loc** and **at** attributes to select values by tag: loc [rows, columns]

```
dataframe
```

	C1	C2	C3	C4	C5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

```
dataframe.loc['A':'B', ["C1", "C5"]]
```

	C1	C5
A	1	5
B	6	10

```
dataframe.loc[['A', 'C'], :]
```

	C1	C2	C3	C4	C5
A	1	2	3	4	5
C	11	12	13	14	15

```
dataframe.loc[:, 'C1':'C3']
```

	C1	C2	C3
A	1	2	3
B	6	7	8
C	11	12	13

```
dataframe.at['A', 'C2']
```

2

Exercise 19

- On the Dataframe of the previous exercise and using labels:
- A - Select all columns, 2nd and 4th row
- B - Select the 'hp' and 'type' columns of all rows
- C - Select the column 'name' from the first and last row
- D - Value that is located in the first row, last column

A

	evolution	name	hp	pokedex	type
B	Charmeleon	Charmander	39	no	fire
D	Metapod	Caterpie	45	no	bug

C

	name
A	Bulbasaur
D	Caterpie

B

	hp	type
A	45	grass

D

```
'grass'
```

Exercise 19 - Solution

```
# Sobre el DataFrame del ejercicio anterior
df = pd.DataFrame({
        "name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
        "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
        "type": ['grass', 'fire', 'water', 'bug'],
        "hp" : [45, 39, 44, 45],
        "pokedex" : ['yes', 'no', 'yes', 'no']
        },
        columns = ["evolution", "name", "hp", "pokedex", "type"],
        index = list("ABCD")
)

# A - Selecciona todas las columnas, 2° y 4° fila
display(df.loc[['B', 'D'], :])

# B - Selecciona las columnas 'hp' y 'type' de la primera fila
display(df.loc[['A'] , ['hp', 'type']])

# C - Selecciona todas la columna, primera y ultima fila
display(df.loc[['A', 'D'], ['name']])

# D - Selecciona el valor de la primera fila, última columna
display(df.at['A', 'type'])
```

Creating / Changing Columns

- To change a new column, use the syntax **dataframe["column"] = array or list**
- If the column does not exist, it will be added to the DataFrame

```
diccionario = {
    "n1": np.arange(1,5),
    "n2": np.arange(50, 54)
}

dataframe = pd.DataFrame(diccionario)
dataframe
```

	n1	n2
0	1	50
1	2	51
2	3	52
3	4	53

```
dataframe["n3"] = [10, 25, 2, -1]
dataframe
```

	n1	n2	n3
0	1	50	10
1	2	51	25
2	3	52	2
3	4	53	-1

Creating / Changing Columns

- We can use the numpy **np.where** function to create new columns

```
dataframe
```

	n1	n2	n3
0	1	50	10
1	2	51	25
2	3	52	2
3	4	53	-1

```
dataframe["n4"] = np.where(dataframe.n1 > 2, True, False)
dataframe
```

	n1	n2	n3	n4
0	1	50	10	False
1	2	51	25	False
2	3	52	2	True
3	4	53	-1	True

Creating / Changing values

- The value selection functions (iat, iloc, loc, at), also allow you to modify the content of a Daframe , or even add new elements (rows or columns)

```
df = pd.DataFrame()
df["Columna A"] = [1, 2, 3, 4]
df["Columna B"] = ["uno", "dos", "tres", "cuatro"]
df
```

	Columna A	Columna B
0	1	uno
1	2	dos
2	3	tres
3	4	cuatro

```
df.iloc[:,0] = [2,1,1,2]
df
```

	Columna A	Columna B
0	2	uno
1	1	dos
2	1	tres
3	2	cuatro

Creating / Changing values

```
df.loc['4'] = [5, 'cinco']
df
```

	Columna A	Columna B
0	2	uno
1	1	dos
2	1	tres
3	2	cuatro
4	5	cinco

```
df.iat[0,0] = 79
df
```

	Columna A	Columna B
0	79	uno
1	1	dos
2	1	tres
3	2	cuatro
4	5	cinco

```
df.loc[:, 'Columna C'] = np.random.normal(0.5, 10, 5)
df
```

	Columna A	Columna B	Columna C
0	79	uno	2.064050
1	1	dos	6.695793
2	1	tres	-1.599246
3	2	cuatro	-16.006748
4	5	cinco	9.593592

```
df.iloc[[0,1], 0] = 11
df
```

	Columna A	Columna B	Columna C
0	11	uno	2.064050
1	11	dos	6.695793
2	1	tres	-1.599246
3	2	cuatro	-16.006748
4	5	cinco	9.593592

Exercise 20

- About last exercise’s Dataframe
- Create a new row with the index 'E' and the values 'Adn', 'Alibai', 34, 'yes', 'grass'
- Creates the column 'hp45', with 45% of the values in column ‘hp’
- Add 100 to the column 'hp' if its value is less than 40

	evolution	name	hp	pokedex	type	hp45
A	Ivysaur	Bulbasaur	45.0	yes	grass	20.25
B	Charmeleon	Charmander	139.0	no	fire	17.55
C	Wartortle	Squirtle	44.0	yes	water	19.80
D	Metapod	Caterpie	45.0	no	bug	20.25
E	Adn	Alibai	134.0	yes	grass	15.30

IMMUNE CODING INSTITUTE

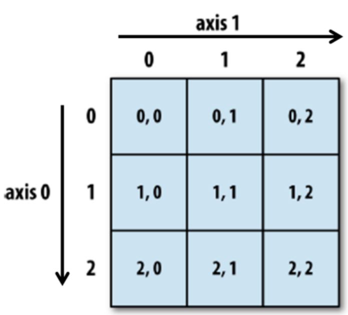
Exercise 20 - Solution

```
# Sobre el DataFrame del ejercicio anterior
df = pd.DataFrame({
        "name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
        "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
        "type": ['grass', 'fire', 'water', 'bug'],
        "hp" : [45, 39, 44, 45],
        "pokedex" : ['yes', 'no', 'yes', 'no']
        },
        columns = ["evolution", "name", "hp", "pokedex", "type"],
        index = list("ABCD")
)

# Crea una nueva fila con los valores 'Adn', 'Alibai', 34, 'yes', 'grass'
df.loc['E', :] = ['Adn', 'Alibai', 34, 'yes', 'grass']
display(df)

# Crea la columna 'hp45' con un 45% de los valores de la columna 'hp'
df['hp45'] = df.hp * 0.45
display(df)

# Añade 100  a la columna hp si si valor es menor de 40
df['hp'] = np.where(df.hp < 40, df.hp + 100, df.hp)
display(df)
```

Applying functions to DataFrames

- **df.apply**() allows you to apply a grouping function to all the values in a column or a row
- Always return a series

	c1	c2	c3	c4	c5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15




```
dataframe.apply(lambda row : row.sum(), axis = 1)
```

```
A    15
B    40
C    65
dtype: int64
```

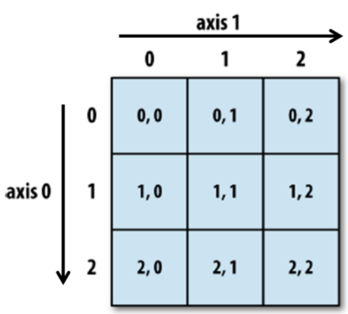
```
dataframe.apply(lambda row : row.C1, axis = 1)
```

```
A     1
B     6
C    11
dtype: int64
```

Applying functions to DataFrames

- We could select values from specific columns or rows...

	c1	c2	c3	c4	c5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15




```
dataframe.apply(lambda column : column.mean(), axis = 0)
```

```
c1     6.0
c2     7.0
c3     8.0
c4     9.0
c5    10.0
dtype: float64
```

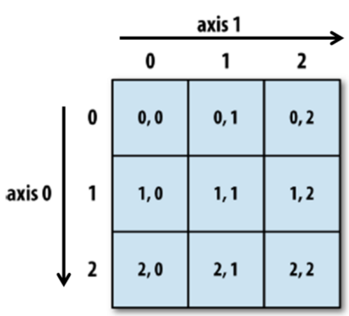
```
dataframe.apply(lambda column : column.A * column.C, axis = 0)
```

```
c1    11
c2    24
c3    39
c4    56
c5    75
dtype: int64
```

Applying functions to DataFrames

- We could use any NumPy as a grouping function

	c1	c2	c3	c4	c5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15




```
dataframe.apply(np.mean, axis = 1)
```

```
A     3.0
B     8.0
C    13.0
dtype: float64
```

```
dataframe.apply(np.max, axis = 0)
```

```
c1    11
c2    12
c3    13
c4    14
c5    15
dtype: int64
```

Applying functions to DataFrames

- It always returns a series so it is easy to assign the result to a new column

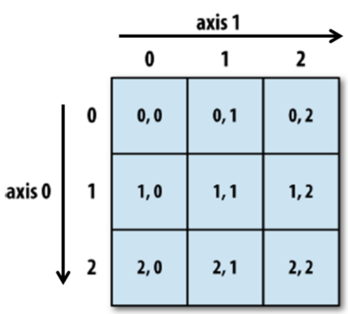
	c1	c2	c3	c4	c5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

```python
serie = dataframe.apply(lambda row : max(row) - min(row), axis = 1)
serie
```

```
A    4
B    4
C    4
dtype: int64
```

```python
dataframe["range"] = dataframe.apply(lambda row : max(row) - min(row), axis = 1)
dataframe
```

	c1	c2	c3	c4	c5	range
A	1	2	3	4	5	4
B	6	7	8	9	10	4
C	11	12	13	14	15	4



Applying functions to DataFrames

- We can execute **apply** on a specific column instead of the whole row

	C1	C2	C3	C4	C5	range
A	1	2	3	4	5	4
B	6	7	8	9	10	4
C	11	12	13	14	15	4

```
dataframe["range_str"] = dataframe.range.apply(lambda value : "Range " + str(value))
dataframe
```

	C1	C2	C3	C4	C5	range	range_str
A	1	2	3	4	5	4	Range 4
B	6	7	8	9	10	4	Range 4
C	11	12	13	14	15	4	Range 4

Applying functions to DataFrames

- The **apply**() function admits parameters that serve to set the parameters of the function being executed

	C1	C2	C3	C4	C5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

```
def multiply_by(value, num):
  return value ** num

dataframe.C1.apply(multiply_by, num = 3)
```

```
A       1
B     216
C    1331
Name: C1, dtype: int64
```

Applying functions to DataFrames

- **series.map**() allows to apply a function to each of the values of a series
- It is equivalent to the **apply**() function on a column, except that it is not allowed to specify params
- Always return a serie

	c1	c2	c3	c4	c5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

```
dataframe.C1.map(lambda value : value * 4)
```

```
A     4
B    24
C    44
Name: C1, dtype: int64
```

```
dataframe.C1.map(str)
```

```
A     1
B     6
C    11
Name: C1, dtype: object
```

```
dataframe.C1.map(lambda value: "%04d" % value)
```

```
A    0001
B    0006
C    0011
Name: C1, dtype: object
```

Applying functions to DataFrames

- **df.applymap**() allows to apply a function to each of the values of a DataFrame

	c1	c2	c3	c4	c5
A	1	2	3	4	5
B	6	7	8	9	10
C	11	12	13	14	15

```
dataframe.applymap(lambda value: value + 1)
```

	c1	c2	c3	c4	c5
A	2	3	4	5	6
B	7	8	9	10	11
C	12	13	14	15	16

```
dataframe.applymap(lambda value: value if value % 2 == 0 else value * -1)
```

	c1	c2	c3	c4	c5
A	-1	2	-3	4	-5
B	6	-7	8	-9	10
C	-11	12	-13	14	-15

Applying functions with NumPy

- If we apply a NumPy universal function to a DataFrame, that function is executed on all DataFrame values

```
np.random.seed(10)
dataframe = pd.DataFrame(
    np.random.normal(10, 100, (3,3))
)
dataframe
```

	0	1	2
0	143.158650	81.527897	-144.540029
1	9.161615	72.133597	-62.008556
2	36.551159	20.854853	10.429143

```
np.abs(dataframe)
```

	0	1	2
0	143.158650	81.527897	144.540029
1	9.161615	72.133597	62.008556
2	36.551159	20.854853	10.429143

```
np.sqrt(np.abs(dataframe))
```

	0	1	2
0	11.964892	9.029280	12.022480
1	3.026816	8.493150	7.874551
2	6.045755	4.566711	3.229418

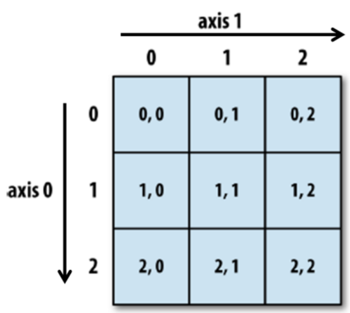
Descriptive statistics in DataFrames

- Pandas provides a number of functions that allow you to statistically describe the data

```
np.random.seed(10)
dataframe = pd.DataFrame(np.random.normal(10, 100, (5,6)),
                         columns = list("ABCDEF"))
dataframe
```

	A	B	C	D	E	F
0	143.158650	81.527897	-144.540029	9.161615	72.133597	-62.008556
1	36.551159	20.854853	10.429143	-7.460021	53.302619	130.303737
2	-86.506567	112.827408	32.863013	54.513761	-103.660221	23.513688
3	158.453700	-97.980489	-187.772828	-164.337230	36.607016	248.496733
4	122.369125	177.262221	19.914922	149.799638	-17.124799	71.320418

Descriptive statistics in DataFrames



	A	B	C	D	E	F
0	143.158650	81.527897	-144.540029	9.161615	72.133597	-62.008556
1	36.551159	20.854853	10.429143	-7.460021	53.302619	130.303737
2	-86.506567	112.827408	32.863013	54.513761	-103.660221	23.513688
3	158.453700	-97.980489	-187.772828	-164.337230	36.607016	248.496733
4	122.369125	177.262221	19.914922	149.799638	-17.124799	71.320418

```
dataframe.max(axis = 0)
```

```
A    158.453700
B    177.262221
C     32.863013
D    149.799638
E     72.133597
F    248.496733
dtype: float64
```

```
dataframe.max(axis = 1)
```

```
0    143.158650
1    130.303737
2    112.827408
3    248.496733
4    177.262221
dtype: float64
```

- 0 is the default value to the **axis** parameter

Descriptive statistics in DataFrames

	A	B	C	D	E	F
0	143.158650	81.527897	-144.540029	9.161615	72.133597	-62.008556
1	36.551159	20.854853	10.429143	-7.460021	53.302619	130.303737
2	-86.506567	112.827408	32.863013	54.513761	-103.660221	23.513688
3	158.453700	-97.980489	-187.772828	-164.337230	36.607016	248.496733
4	122.369125	177.262221	19.914922	149.799638	-17.124799	71.320418

```
dataframe.count()
```

```
A     5
B     5
C     5
D     5
E     5
F     5
dtype: int64
```

```
dataframe.max()
```

```
A    158.453700
B    177.262221
C     32.863013
D    149.799638
E     72.133597
F    248.496733
dtype: float64
```

```
dataframe.min()
```

```
A    -86.506567
B    -97.980489
C   -187.772828
D   -164.337230
E   -103.660221
F    -62.008556
dtype: float64
```

```
dataframe.sum()
```

```
A    374.026067
B    294.491891
C   -269.105780
D     41.677763
E     41.258213
F    411.626021
dtype: float64
```

IMMUNE
CODING INSTITUTE

Descriptive statistics in DataFrames

```
pd.DataFrame({
    "count": df.count(),
    "max": df.max(),
    "min": df.min(),
    "sum": df.sum(),
    "mean": df.mean(),
    "std": df.std(),
    "var": df.var(),
    "10%": df.quantile(0.10),
    "90%": df.quantile(0.90)
})
```

	count	max	min	sum	mean	std	var	10%	90%
A	5	145.451509	1.801341	324.719256	64.943851	62.807342	3944.762162	10.916830	134.382007
B	5	136.944272	-113.392774	-128.978593	-25.795719	105.757578	11184.665408	-106.868138	92.037804
C	5	125.831019	-147.058246	-198.581271	-39.716254	100.566897	10113.700716	-117.732724	57.249204
D	5	144.514859	-56.993967	206.096661	41.219332	72.366866	5236.963262	-23.592397	110.557726
E	5	55.929988	-118.293762	-87.141204	-17.428241	62.947272	3962.359007	-76.059470	32.133936
F	5	94.858954	-62.972526	45.273220	9.054644	61.190817	3744.316040	-47.909870	73.749177

Exercise 21

- About last exercise's dataframe
- Converts all dataframe values to uppercase (str.upper() and isinstance(x, str))
- Create a new column with the values of the column 'name' and 'evolution' concatenated

	evolution	name	hp	pokedex	type	New Column
A	IVYSAUR	BULBASAUR	45	YES	GRASS	BULBASAUR-IVYSAUR
B	CHARMELEON	CHARMANDER	39	NO	FIRE	CHARMANDER-CHARMELEON
C	WARTORTLE	SQUIRTLE	44	YES	WATER	SQUIRTLE-WARTORTLE
D	METAPOD	CATERPIE	45	NO	BUG	CATERPIE-METAPOD

Exercise 21 - Solution

```
# Sobre el DataFrame del ejercicio anterior
df = pd.DataFrame({
        "name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
        "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
        "type": ['grass', 'fire', 'water', 'bug'],
        "hp" : [45, 39, 44, 45],
        "pokedex" : ['yes', 'no', 'yes', 'no']
        },
        columns = ["evolution", "name", "hp", "pokedex", "type"],
        index = list("ABCD")
)

# Convierte todos los valores a mayúsculas
df = df.applymap(lambda value: value.upper() if isinstance(value, str) else value)

# Crea una nueva columna con los valores de las columnas 'name' y 'evolution' concatenados
df["New Column"] = df.apply(lambda row: row.name + "-" + row.evolution , axis = 1 )
df
```

IMMUNE
CODING INSTITUTE

Exercise 22

- About last exercise's dataframe
- Extract the first 3 characters of the evolution field and substitute the original valu
- Create the pokedex2 column with the translation of yes / no to spanish

	evolution	name	hp	pokedex	type	pokedex2
A	Ivy	Bulbasaur	45	yes	grass	si
B	Cha	Charmander	39	no	fire	no
C	War	Squirtle	44	yes	water	si
D	Met	Caterpie	45	no	bug	no

Exercise 22 - Solution

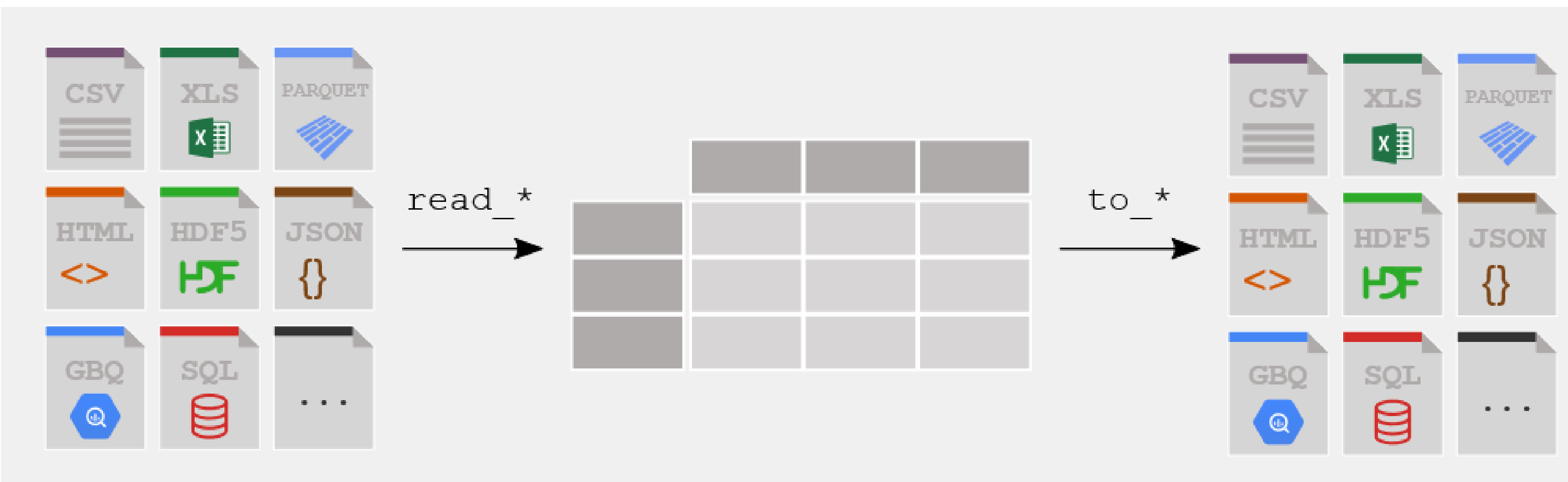
```
# Sobre el DataFrame del ejercicio anterior
df = pd.DataFrame({
        "name": ['Bulbasaur', 'Charmander', 'Squirtle', 'Caterpie'],
        "evolution": ['Ivysaur', 'Charmeleon', 'Wartortle', 'Metapod'],
        "type": ['grass', 'fire', 'water', 'bug'],
        "hp" : [45, 39, 44, 45],
        "pokedex" : ['yes', 'no', 'yes', 'no']
        },
        columns = ["evolution", "name", "hp", "pokedex", "type"],
        index = list("ABCD")
)

# Extrae los 3 primeros caracteres del campo 'evolution' y sustituye el valor original
df.evolution = df.evolution.map(lambda value: value[:3])

# Crea la columna 'pokedex2' con la traducción de 'yes/no' al castellano
df['pokedex2'] = df.pokedex.map(lambda value: 'si' if value == 'yes' else value)
df
```

Index

- NumPy
- Pandas
- Dataframes
- Reading / Writing data
- Exploring a DataFrame
- Operations on a DataFrame

Reading / Writing data

- Pandas supports the integration with many file formats or data sources out of the box (csv, excel, sql, json, parquet,…).



Reading data in text format

- The function **pd.read_csv()** allows you to read a file and store it in a DataFrame
- With the default options, files must have a header and the separator is a comma
- The file could be both on disk and on the network

```
pd.read_csv("./Sacramentorealestatetransactions.csv")
```

```
pd.read_csv("http://samplecsvs.s3.amazonaws.com/Sacramentorealestatetransactions.csv")
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Reading data in text format

- The **pd.read_table()** function allows you to set the separator using the **sep** argument

```
%%writefile input_data.txt
a|b|c|d|message
1|2|3|4|hello
5|6|7|8|world
9|10|11|12|foo
```

```
Writing input_data.txt
```

```
pd.read_csv("input_data.txt", sep = "|")
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Reading data in text format

- The **header** parameter allows you to set whether or not a header exists

```
%%writefile input_data.txt
a|b|c|d|message
1|2|3|4|hello
5|6|7|8|world
9|10|11|12|foo
```

```
Writing input_data.txt
```

```
pd.read_csv("input_data.txt", sep = "|", header = None)
```

	0	1	2	3	4
0	a	b	c	d	message
1	1	2	3	4	hello
2	5	6	7	8	world
3	9	10	11	12	foo

Reading data in text format

- The **na_values** parameter specifies the null values

```
%%writefile input_data.txt
a|b|c|d|message
1|2|3|NA|hello
5|6|7|8|world
9|NA|11|12|foo
```

```
Overwriting input_data.txt
```

```
pd.read_table("input_data.txt", sep = '|', na_values=['NA'])
```

	a	b	c	d	message
0	1	2.0	3	NaN	hello
1	5	6.0	7	8.0	world
2	9	NaN	11	12.0	foo

IMMUNE CODING INSTITUTE

Reading data in text format

- The **pd.read_fwf()** function allows you to read a file when the columns have fixed positions

```
%%writefile input_data.txt
a  b   c  d message
1  2 223  4 hello
5  6   7  8 world
9 10  11 12 foo
```

```
Overwriting input_data.txt
```

```
pd.read_fwf("input_data.txt")
```

	a	b	c	d	message
0	1	2	223	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Reading data in text format

- The **converters** parameter allows you to set conversion functions in the columns of the DataFrame

```
%%writefile input_data.txt
col1|col2|col3
one|1.232,53|a
two|2.000,00|b
```

```
Writing input_data.txt
```

```
pd.read_csv("input_data.txt", sep = "|",
            converters = {'col2': lambda value: float(value.replace('.', '').replace(',','.'))}
)
```

	col1	col2	col3
0	one	1232.53	a
1	two	2000.00	b

Reading data from Excel

- Pandas also allows you to read an Excel format file
- If we want to read several sheets of the same Excel file, it is convenient to first load the file in memory with the **pd.ExcelFile()** method

```
dataframe = pd.read_excel("FL_insurance_sample.xlsx")
dataframe = pd.read_excel("FL_insurance_sample.xlsx", 'FL_insurance_sample')
```

```
xlsx = pd.ExcelFile("FL_insurance_sample.xlsx")
dataframe = pd.read_excel(xlsx, 'FL_insurance_sample')
```

Reading data from a JSON file

- Using the pd. **read_json**() function, pandas will **read** data in JSON format and load it into a DataFrame

```
%%writefile input_data.json
[{"a": 1, "b": 2, "c": 3},
 {"a": 4, "b": 5, "c": 6},
 {"a": 7, "b": 8, "c": 9}]
```

```
Overwriting input_data.json
```

```
pd.read_json("input_data.json")
```

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

IMMUNE CODING INSTITUTE

Reading data from a JSON file

- An alternative is to use the **json** library to read the

```
%%writefile input_data.json
{
    "name": "Wes",
    "places_lived": ["United States", "Spain", "Germany"],
    "siblings": [
        {"age": 30, "name": "Scott", "pets": ["Zeus", "Zuko"]},
        {"age": 38, "name": "Katie", "pets": ["Sixes", "Stache", "Cisco"]}
    ]
}
```

```
Overwriting input_data.json
```

```
import json

with open('input_data.json') as json_data:
    result = json.load(json_data)

pd.DataFrame(result['siblings'], columns=['name', 'age'])
```

	name	age
0	Scott	30
1	Katie	38

Reading data from a Web Service

- To read the data of a web service we could use the **request** library

```python
import requests
```

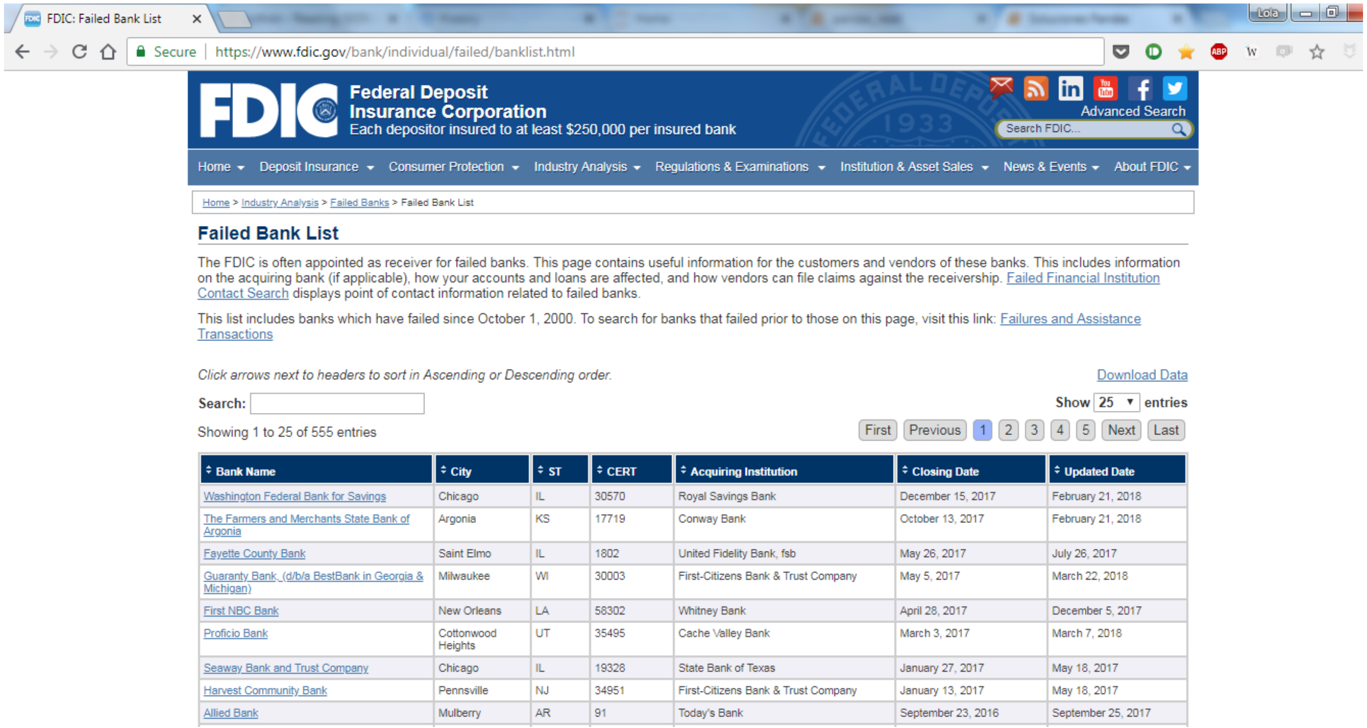
```python
url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
resp = requests.get(url)

if resp.ok:
    data = resp.json()
    dataframe = pd.DataFrame(data, columns=['number', 'title', 'labels', 'state'])

dataframe.head()
```

	number	title	labels	state
0	21649	DOC: Fix versionadded directive typos in Inter...	[{'id': 134699, 'node_id': 'MDU6TGFiZWwxMzQ2OT...	open
1	21648	API: Categorical.unique() should not drop unus...	[]	open
2	21647	addresses GH #21646	[]	open
3	21646	Test fixture datapath uses relative instead of...	[]	open
4	21645	ENH: add return_inverse to df.duplicated	[{'id': 76812, 'node_id': 'MDU6TGFiZWw3NjgxMg=...	open

https://api.github.com/repos/pandas-dev/pandas/issues

Reading data from HTML

- Pandas allows to read a file with HTML format through the **read_html**() function



FDIC: Failed Bank List

Secure | https://www.fdic.gov/bank/individual/failed/banklist.html

FDIC Federal Deposit Insurance Corporation

Each depositor insured to at least $250,000 per insured bank

Advanced Search

Home | Deposit Insurance | Consumer Protection | Industry Analysis | Regulations & Examinations | Institution & Asset Sales | News & Events | About FDIC

Home > Industry Analysis > Failed Banks > Failed Bank List

Failed Bank List

The FDIC is often appointed as receiver for failed banks. This page contains useful information for the customers and vendors of these banks. This includes information on the acquiring bank (if applicable), how your accounts and loans are affected, and how vendors can file claims against the receivership. Failed Financial Institution Contact Search displays point of contact information related to failed banks.

This list includes banks which have failed since October 1, 2000. To search for banks that failed prior to those on this page, visit this link: Failures and Assistance Transactions

Click arrows next to headers to sort in Ascending or Descending order. Download Data

Search: **Show** 25 **entries**

Showing 1 to 25 of 555 entries First Previous 1 2 3 4 5 Next Last

Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date
Washington Federal Bank for Savings	Chicago	IL	30570	Royal Savings Bank	December 15, 2017	February 21, 2018
The Farmers and Merchants State Bank of Argonia	Argonia	KS	17719	Conway Bank	October 13, 2017	February 21, 2018
Fayette County Bank	Saint Elmo	IL	1802	United Fidelity Bank, fsb	May 26, 2017	July 26, 2017
Guaranty Bank, (d/b/a BestBank in Georgia & Michigan)	Milwaukee	WI	30003	First-Citizens Bank & Trust Company	May 5, 2017	March 22, 2018
First NBC Bank	New Orleans	LA	58302	Whitney Bank	April 28, 2017	December 5, 2017
Proficio Bank	Cottonwood Heights	UT	35495	Cache Valley Bank	March 3, 2017	March 7, 2018
Seaway Bank and Trust Company	Chicago	IL	19328	State Bank of Texas	January 27, 2017	May 18, 2017
Harvest Community Bank	Pennsville	NJ	34951	First-Citizens Bank & Trust Company	January 13, 2017	May 18, 2017
Allied Bank	Mulberry	AR	91	Today's Bank	September 23, 2016	September 25, 2017

https://www.fdic.gov/bank/individual/failed/banklist.html

Reading data from HTML

- This function returns a list of dataframes (there may be several tables on the website)

```
dataframes = pd.read_html('https://www.fdic.gov/bank/individual/failed/banklist.html')
```

```
dataframes[0].head()
```

	Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date
0	Washington Federal Bank for Savings	Chicago	IL	30570	Royal Savings Bank	December 15, 2017	February 21, 2018
1	The Farmers and Merchants State Bank of Argonia	Argonia	KS	17719	Conway Bank	October 13, 2017	February 21, 2018
2	Fayette County Bank	Saint Elmo	IL	1802	United Fidelity Bank, fsb	May 26, 2017	July 26, 2017
3	Guaranty Bank, (d/b/a BestBank in Georgia & Mi...	Milwaukee	WI	30003	First-Citizens Bank & Trust Company	May 5, 2017	March 22, 2018
4	First NBC Bank	New Orleans	LA	58302	Whitney Bank	April 28, 2017	December 5, 2017

Data writing

- Once we have a DataFrame in memory, we could write it to disk with one of the following functions:
  - dataframe. **to_csv**("file.csv")
  - dataframe. **to_excel**("file.xlsx")
  - dataframe. **to_json**("file.json")

```
dataframe = pd.read_excel("FL_insurance_sample.xlsx")
dataframe = pd.read_excel("FL_insurance_sample.xlsx", 'FL_insurance_sample')
dataframe = pd.read_json("FL_insurance_sample.json")
```

```
xlsx = pd.ExcelWriter("file.xlsx")
dataframe1.to_excel(xlsx, 'Sheet1')
dataframe1.to_excel(xlsx, 'Sheet1', index = False)
xlsx.save()
```

Reading data from a database

- The **sqlalchemy** package allows you to connect to a database and load DataFrames from tables or queries

```
from sqlalchemy import create_engine
```

```
engine = create_engine('sqlite:///:memory:')
```

```
pd.read_sql("SELECT * FROM tabla;", engine)
pd_read_sql_table('tabla', engine)
```

Exercise 23

- Load the information from the following url into a Dataframe called 'df1':

https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv

- Load in a Dataframe called 'df2', the information of the following url:
  https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.user
  - The column 'user_id' must be the index of the DataFrame

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98

	age	gender	occupation	zip_code
user_id				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213

Exercise 23 - Solution

```
# Lee el fichero chipotle.tsv y guardalo en df1
df1 = pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv',
                  sep = '\t')
display(df1)

# Lee el fichero u.user y guardalo en df2
df2 = pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.user',
                  sep = '|',
                  index_col = 'user_id')
display(df2)
```

Exercise 24

- Write the dataframes from the previous exercise in an excel called "Data.xlsx"
  - df1' save it on a sheet called 'chipotle' (without the index)
  - 'df2' on another sheet called 'user'

- Recover in a different DataFrame the information from the 'user' sheet of the excel file "Data.xlsx".

	age	gender	occupation	zip_code
user_id				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213

Exercise 24 - Solution

```
# Escribe la información del ejecicio anterior en un excel llamado 'Data.xlsx'
writer = pd.ExcelWriter('Data.xlsx')
df1.to_excel(writer, 'chipotle', index = False)
df2.to_excel(writer, 'user', index = False)
writer.save()

df3 = pd.read_excel("Data.xlsx", "user")
display(df3)
```

Exercise 25

- Read the data from the following web service in a DataFrame:

https://sedeaplicaciones.minetur.gob.es/ServiciosRESTCarburantes/PreciosCarburantes/EstacionesTerrestres/

	C.P.	Dirección	Horario	Latitud	Localidad	Longitud (WGS84)	Margen	Municipio	Precio Biodiesel	Precio Bioetanol	Precio Gas Natural Comprimido	Precio Gas Natural Licuado	Precio Gases licuados del petróleo	Precio Gasoleo A	Precio Gasoleo B	Precio Gasolina 95 Protección	Precio Gasolina 98	Precio Nuevo Gasoleo A	Provincia	Remisión
0	01240	CL MANISITU, 9	L-D: 24H	42,846028	ALEGRIA-DULANTZI	-2,509361	D	Alegría-Dulantzi	None	None	None	None	None	1,069	0,726	None	None	None	ÁLAVA	dm
1	01240	CALLE GASTEIZBIDEA, 59	L-D: 07:00-20:00	42,842917	ALEGRIA-DULANTZI	-2,519194	D	Alegría-Dulantzi	None	None	None	None	None	1,149	None	1,199	None	None	ÁLAVA	dm
2	01468	POLIGONO ZANKUETA, 0	L-D: 24H	43,044333	LARRINBE	-2,989111	D	Amurrio	None	None	None	None	None	1,019	None	1,089	1,229	1,069	ÁLAVA	dm
3	01450	CARRETERA A-624 KM. 37,8	L-V: 07:00-21:00; S-D: 08:00-20:00	43,031889	LEZAMA	-2,967611	D	Amurrio	None	None	None	None	None	1,069	None	1,144	1,269	1,129	ÁLAVA	OM
4	01120	CARRETERA A-132 VITORIA-ESTELLA KM. 23	L-D: 07:00-23:00	42,753194	MAEZTU/MAESTU	-2,477917	I	Arraia-Maeztu	None	None	None	None	None	1,145	None	1,250	1,315	None	ÁLAVA	dm

Index

- NumPy
- Pandas
- Dataframes
- Reading / Writing data
- Exploring a DataFrame
- Operations on a DataFrame

Exploring a DataFrame

- Pandas offers several functions to explore a Dataframe without printing all the content
- The **dataframe.shape** attribute shows the dimensions (number of rows and number of columns)

```
dataframe = pd.read_csv("mtcars.csv")
```

```
dataframe.shape
```

```
(32, 12)
```

Exploring a DataFrame

- The **dataframe.columns** attribute shows the columns
- The **dataframe.index** attribute shows the indexes that have the

```
dataframe.columns
```

```
Index(['Unnamed: 0', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs',
       'am', 'gear', 'carb'],
      dtype='object')
```

```
dataframe.index
```

```
RangeIndex(start=0, stop=32, step=1)
```

Exploring a DataFrame

- **df.head**() shows the first rows of the DataFrame

```
dataframe.head()
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

Exploring a DataFrame

- **df.tail**() shows the last rows of the DataFrame

```
dataframe.tail()
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

IMMUNE CODING INSTITUTE

Exploring a DataFrame

- **df.sample**() shows an example of the dataframe

```
dataframe.sample(5)
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2

Exploring a DataFrame

- **df.info**() shows summary information about the DataFrame

```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   model   32 non-null     object
 1   mpg     32 non-null     float64
 2   cyl     32 non-null     int64
 3   disp    32 non-null     float64
 4   hp      32 non-null     int64
 5   drat    32 non-null     float64
 6   wt      32 non-null     float64
 7   qsec    32 non-null     float64
 8   vs      32 non-null     int64
 9   am      32 non-null     int64
 10  gear    32 non-null     int64
 11  carb    32 non-null     int64
dtypes: float64(5), int64(6), object(1)
memory usage: 3.1+ KB
```

Exploring a DataFrame

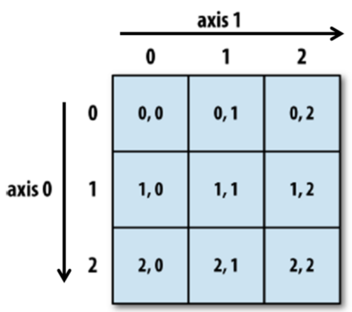
- **df.count**() returns an array with the number of non-null values for each of the columns

```
dataframe.count()
```

```
model     32
mpg       32
cyl       32
disp      32
hp        32
drat      32
wt        32
qsec      32
vs        32
am        32
gear      32
carb      32
dtype: int64
```

```
dataframe.count(axis = 1)
```

```
0      12
1      12
2      12
3      12
4      12
5      12
6      12
7      12
8      12
9      12
10     12
11     12
12     12
13     12
14     12
15     12
16     12
```



Exploring a DataFrame

- **df.describe**() returns a DataFrame with statistical information of each of the numerical columns

```
dataframe.describe()
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.0000
mean	20.090625	6.187500	230.721875	146.687500	3.596563	3.217250	17.848750	0.437500	0.406250	3.687500	2.8125
std	6.026948	1.785922	123.938694	68.562868	0.534679	0.978457	1.786943	0.504016	0.498991	0.737804	1.6152
min	10.400000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1.0000
25%	15.425000	4.000000	120.825000	96.500000	3.080000	2.581250	16.892500	0.000000	0.000000	3.000000	2.0000
50%	19.200000	6.000000	196.300000	123.000000	3.695000	3.325000	17.710000	0.000000	0.000000	4.000000	2.0000
75%	22.800000	8.000000	326.000000	180.000000	3.920000	3.610000	18.900000	1.000000	1.000000	4.000000	4.0000
max	33.900000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.000000	1.000000	5.000000	8.0000

Exploring a DataFrame

- **df.cov**() returns a DataFrame with the result of applying the covariance function in each of the numerical columns (all with all)

```
dataframe.cov()
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
mpg	36.324103	-9.172379	-633.097208	-320.732056	2.195064	-5.116685	4.509149	2.017137	1.803931	2.135685	-5.363105
cyl	-9.172379	3.189516	199.660282	101.931452	-0.668367	1.367371	-1.886855	-0.729839	-0.465726	-0.649194	1.520161
disp	-633.097208	199.660282	15360.799829	6721.158669	-47.064019	107.684204	-96.051681	-44.377621	-36.564012	-50.802621	79.068750
hp	-320.732056	101.931452	6721.158669	4700.866935	-16.451109	44.192661	-86.770081	-24.987903	-8.320565	-6.358871	83.036290
drat	2.195064	-0.668367	-47.064019	-16.451109	0.285881	-0.372721	0.087141	0.118649	0.190151	0.275988	-0.078407
wt	-5.116685	1.367371	107.684204	44.192661	-0.372721	0.957379	-0.305482	-0.273661	-0.338105	-0.421081	0.675790
qsec	4.509149	-1.886855	-96.051681	-86.770081	0.087141	-0.305482	3.193166	0.670565	-0.204960	-0.280403	-1.894113
vs	2.017137	-0.729839	-44.377621	-24.987903	0.118649	-0.273661	0.670565	0.254032	0.042339	0.076613	-0.463710
am	1.803931	-0.465726	-36.564012	-8.320565	0.190151	-0.338105	-0.204960	0.042339	0.248992	0.292339	0.046371
gear	2.135685	-0.649194	-50.802621	-6.358871	0.275988	-0.421081	-0.280403	0.076613	0.292339	0.544355	0.326613
carb	-5.363105	1.520161	79.068750	83.036290	-0.078407	0.675790	-1.894113	-0.463710	0.046371	0.326613	2.608871

Exploring a DataFrame

- **df.corr**() returns a DataFrame with the result of applying the correlation function in each of the numerical columns (all with all)

```
dataframe.corr()
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
mpg	1.000000	-0.852162	-0.847551	-0.776168	0.681172	-0.867659	0.418684	0.664039	0.599832	0.480285	-0.550925
cyl	-0.852162	1.000000	0.902033	0.832447	-0.699938	0.782496	-0.591242	-0.810812	-0.522607	-0.492687	0.526988
disp	-0.847551	0.902033	1.000000	0.790949	-0.710214	0.887980	-0.433698	-0.710416	-0.591227	-0.555569	0.394977
hp	-0.776168	0.832447	0.790949	1.000000	-0.448759	0.658748	-0.708223	-0.723097	-0.243204	-0.125704	0.749812
drat	0.681172	-0.699938	-0.710214	-0.448759	1.000000	-0.712441	0.091205	0.440278	0.712711	0.699610	-0.090790
wt	-0.867659	0.782496	0.887980	0.658748	-0.712441	1.000000	-0.174716	-0.554916	-0.692495	-0.583287	0.427606
qsec	0.418684	-0.591242	-0.433698	-0.708223	0.091205	-0.174716	1.000000	0.744535	-0.229861	-0.212682	-0.656249
vs	0.664039	-0.810812	-0.710416	-0.723097	0.440278	-0.554916	0.744535	1.000000	0.168345	0.206023	-0.569607
am	0.599832	-0.522607	-0.591227	-0.243204	0.712711	-0.692495	-0.229861	0.168345	1.000000	0.794059	0.057534
gear	0.480285	-0.492687	-0.555569	-0.125704	0.699610	-0.583287	-0.212682	0.206023	0.794059	1.000000	0.274073
carb	-0.550925	0.526988	0.394977	0.749812	-0.090790	0.427606	-0.656249	-0.569607	0.057534	0.274073	1.000000

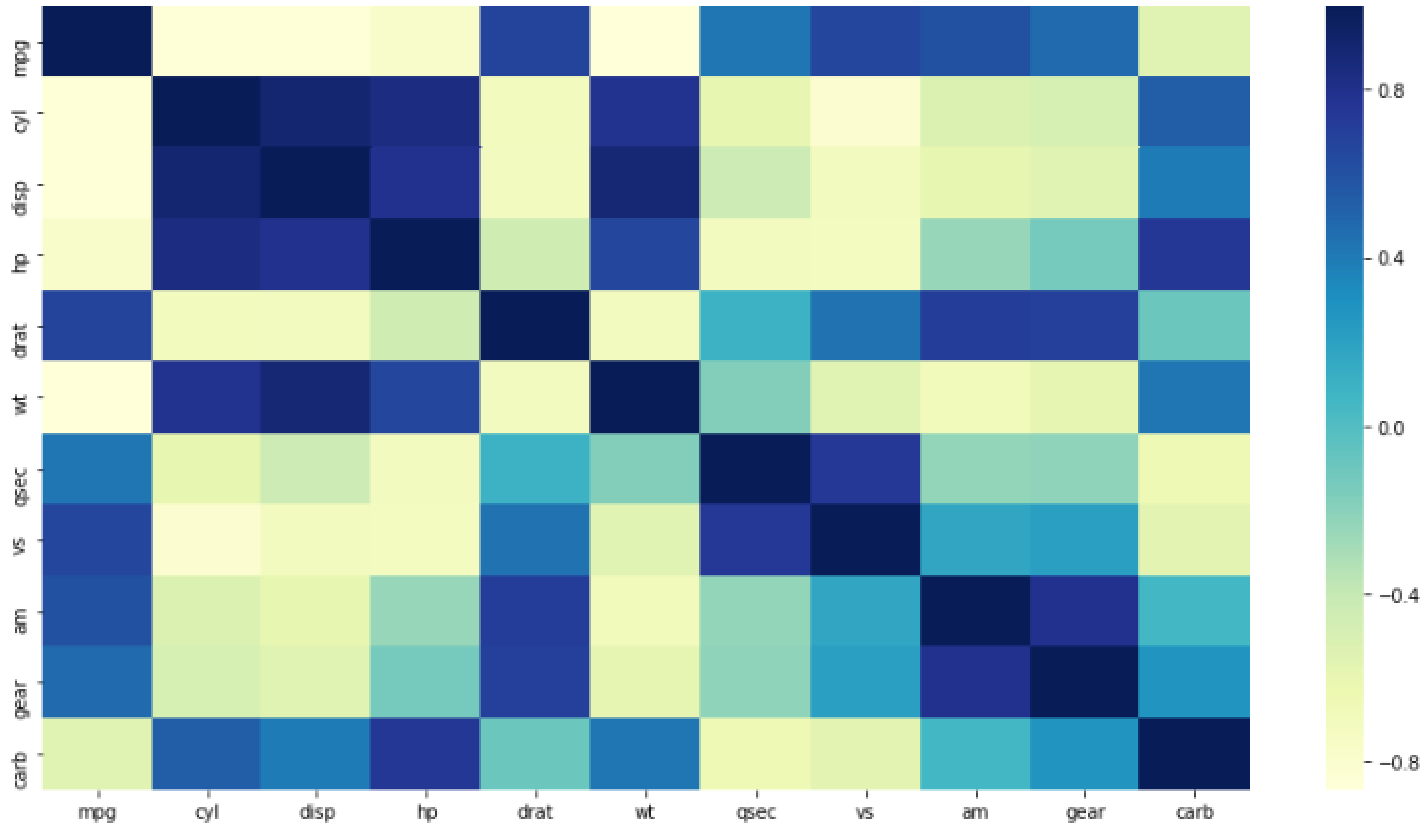
Exploring a DataFrame

- Using the Python visualization tools we could visualize the correlation matrix

```
from matplotlib import pyplot as plt
import seaborn as sns

plt.figure(figsize=(15,8))

corr = dataframe.corr()
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns,
        cmap="YlGnBu")
```

IMMUNE
CODING INSTITUTE

Exploring a DataFrame



Exercise 26 (1/2)

- Load the league data that is in an Excel file called "LigaBBVA_20170329.xlsx" and load it into a variable called ‘liga’

	#	Equipo	PJ	V	E	D	GF	GC	PTS
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0
4	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0
5	5	Villarreal	28.0	13.0	9.0	6.0	39.0	20.0	48.0
6	6	Real Sociedad	28.0	15.0	3.0	10.0	42.0	39.0	48.0
7	7	Ath. Bilbao	28.0	13.0	5.0	10.0	35.0	32.0	44.0
8	8	Eibar	28.0	11.0	8.0	9.0	44.0	39.0	41.0
9	9	RCD Espanyol	28.0	10.0	10.0	8.0	40.0	39.0	40.0

Exercise 26 (2/2)

- Show the dimensions of the DataFrame
- Print the columns and row indexes
- It shows the first 3 rows
- Shows the last 2 rows
- Displays summary information about the Dataframe
- Check if any of the columns correlate with any other
- Shows the number of values in each of the columns
- Displays statistical information about the column ‘V’

	count	mean	std	min	25%	50%	75%	max
V	20.0	10.4	5.164657	1.0	6.0	10.0	13.5	20.0

Exercise 26 - Solution

```
# Carga los datos del fichero 'LigaBBVA_20170329.xlsx'
liga = pd.read_excel('LigaBBVA_20170329.xlsx', 'Clasificación')
display(liga)

# Muestra las dimensiones del DataFrame
print(liga.shape)

# Imprime el nombre de las columnas y los índices de las filas
print(list(liga.columns))
print(list(liga.index))

# Muestra las 3 primeras filas
display(liga.head(3))

# Muestra las 3 últimas filas
display(liga.tail(3))

# Muestra información resumida del DataFrame
display(liga.info())

# Muestra el número de valores de cada una de las columnas
display(liga.count())

# Muestra información estadística sobre la columna 'V'
display( liga.describe()[['V']].T   )
```

Index

- NumPy
- Pandas
- Dataframes
- Reading / Writing data
- Exploring a DataFrame
- Operations on a DataFrame

Common operations in a Dataframe

- Pandas allows to do in an easy way operations like filtering a Dataframe, ordering it, selecting columns, renaming them, modifying them and even grouping them
- What all these functions have in common is that they do not modify the dataframe, but return another dataframe

Common operations in a Dataframe

```
dataframe = pd.DataFrame({ 'C1' : ['Afghanistan', 'Afghanistan', 'Brazil', 'Brazil', 'China', 'China'],
                           'C2' : [1999, 2000, 1999, 2000, 1999, 2000],
                           'C3' : [745, 2666, 37737, 80488, 212258, 213766],
                           'C4' : [19987071, 20595360, 172006362, 174504898, 1272915272, 1280428583]
                         }
                        )
dataframe
```

	C1	C2	C3	C4
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Rename columns

- The **rename**() function allows you to ren**ame** both the row index and the column index of a Dataframe

```
dataframe = dataframe.rename(columns = {'C1' : 'Country',
                                        'C2' : 'Year',
                                        'C3' : 'Cases',
                                        'C4' : 'Population'})
dataframe
```

	C1	C2	C3	C4
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Rename columns

- The **rename()** function could accept a function to make the modification, instead of a dictionary

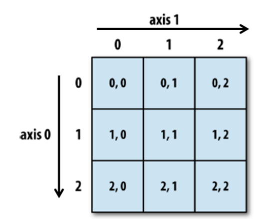
```
dataframe.rename(columns = lambda column : column.lower())
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

```
dataframe.rename(index = lambda row : 'Row ' + str(row))
```

	Country	Year	Cases	Population
Row 0	Afghanistan	1999	745	19987071
Row 1	Afghanistan	2000	2666	20595360
Row 2	Brazil	1999	37737	172006362
Row 3	Brazil	2000	80488	174504898
Row 4	China	1999	212258	1272815272
Row 5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



Rename columns

- With the parameter index **(**instead of **columns)** you would rename the index of the rows

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.rename(index = lambda x : 'Row ' + str(x))
```

	Country	Year	Cases	Population
Row 0	Afghanistan	1999	745	19987071
Row 1	Afghanistan	2000	2666	20595360
Row 2	Brazil	1999	37737	172006362
Row 3	Brazil	2000	80488	174504898
Row 4	China	1999	212258	1272915272
Row 5	China	2000	213766	1280428583

Exercise 27

- About the DataFrame “liga”:
- Rename the name of the columns making sure they are materialized in the dataset

Antes	Después
#	Puesto
PJ	PartidosJugados
V	Victorias
E	Empates
D	Derrotas
GF	GolesFavor
GC	GolesContra
PTS	Puntos

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0

Exercise 27 - Solution

```
# Sobre el Dataframe 'liga'
liga = pd.read_excel('LigaBBVA_20170329.xlsx', 'Clasificación')

# Renombra las columnas materializando el resultado sobre el dataset
liga = liga.rename( columns = { '#' : 'Puesto', 'PJ' : 'PartidosJugados', 'V': 'Victorias',
                                'E' : 'Empates', 'D' : 'Derrotas', 'GF': 'GolesFavor',
                                'GC': 'GolesContra', 'PTS' : 'Puntos'})
liga.head()
```

Exercise 28

- About the DataFrame “liga":
- Rename the columns, so that they are in capital letters (but do not materialize it in the DataFrame)

	PUESTO	EQUIPO	PARTIDOSJUGADOS	VICTORIAS	EMPATES	DERROTAS	GOLESFAVOR	GOLESCONTRA	PUNTOS
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0

Exercise 28 - Solution

```
# Renombra las columnas convirtiendolas a mayusculas (no lo materialices sobre el dataset)
liga.rename( columns = str.upper)
```

Column Selection

- There is not a specific function to select columns in Pandas, just an array of names is provided

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe[["Year", "Country", "Country"]]
```

	Year	Country	Country
0	1999	Afghanistan	Afghanistan
1	2000	Afghanistan	Afghanistan
2	1999	Brazil	Brazil
3	2000	Brazil	Brazil
4	1999	China	China
5	2000	China	China

Exercise 29

- About the DataFrame “liga":
- Select the “Puesto" and "Puntos" columns

	Puesto	Puntos
0	0	NaN
1	1	65.0

- Select all columns in alphabetical order (sorted)

	Derrotas	Empates	Equipo	GolesContra	GolesFavor	PartidosJugados	Puesto	Puntos	Victorias
0	NaN	NaN	NaN	NaN	NaN	NaN	0	NaN	NaN
1	2.0	5.0	Real Madrid	28.0	71.0	27.0	1	65.0	20.0

IMMUNE CODING INSTITUTE

Exercise 29 - Solution

```
# Seleciona las columnas 'Puesto' y 'Puntos'
liga[['Puesto', 'Puntos']]

# Selecciona todas las columnas en orden alfabético
liga[sorted(liga.columns)]
```

IMMUNE CODING INSTITUTE

Row filtering

- The **query()** function allows you to filter the rows of a Dataframe in the same way that the **WHERE** clause in a SQL statement does

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.query("Country == 'China' or Year == 1999")
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.query("Year not in (2000, 2001)")
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272

http://jose-coto.com/query-method-pandas

Row filtering

```
dataframe[dataframe.Country.str.contains('^C')]
```

	Country	Year	Cases	Population
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

```
dataframe[~dataframe.Country.isnull()]
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272815272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
help(dataframe.Country.str.contains)
```

Row filtering

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe[dataframe.Year.isin([1999])]
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272

```
dataframe.Year.value_counts().index[:1]
```

```
Int64Index([1999], dtype='int64')
```

```
dataframe[dataframe.Year.isin(dataframe.Year.value_counts().index[:1])]
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272

Exercise 30 (1/2)

- About the DataFrame “liga":
- Search the rows for Real Madrid and Barcelona

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0

- Look for rows whose position is less than or equal to 2 or more than or equal to 20

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0

IMMUNE CODING INSTITUTE

Exercise 30 (2/2)

- Look for lines whose wins are greater than or equal to 18 and the goals scored are greater than 60

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0

Exercise 30 - Solution

```python
# Busca las filas correspondientes a 'Real Madrid' y 'Barcelona'
liga.query("Equipo in ('Real Madrid', 'Barcelona')")

# Busca las filas cuyo Puesto sea menor o igual a 2 o mayor o igual que 20
liga.query("Puesto <= 2 or Puesto >= 20")

# Busca las filas cuyas victorias sean mayores o iguales a 18 y los goles sean mayores que 60
liga.query("Victorias >= 18 and GolesFavor > 60")
```

Exercise 31

- Search the rows with the field 'Equipo' nullified

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- Look for teams starting with A

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
4	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0
7	7	Ath. Bilbao	28.0	13.0	5.0	10.0	35.0	32.0	44.0
10	10	Alavés	28.0	10.0	10.0	8.0	29.0	33.0	40.0

Exercise 31- Solution

```
# Busca las filas cuyo campo 'Equipo' sea nulo
liga[liga.Equipo.isnull()]

# Busca los equipos que empiecen por 'A'
liga[liga.Equipo.str.contains("^A", na=False)]
```

Ordering

- The **sort_values**() function allows you to **sort** through the values of the DataFrame

```
dataframe.sort_values("Population")
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.sort_values(["Country","Year"])
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Ordering

- The **ascending** parameter allows you to set the ascending / descending order

```
dataframe.sort_values(["Year","Country"],
                      ascending = [False, True])
```

	Country	Year	Cases	Population
1	Afghanistan	2000	2666	20595360
3	Brazil	2000	80488	174504898
5	China	2000	213766	1280428583
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272

```
dataframe.sort_values(["Country","Year"],
                      ascending = [False, False])
```

	Country	Year	Cases	Population
5	China	2000	213766	1280428583
4	China	1999	212258	1272915272
3	Brazil	2000	80488	174504898
2	Brazil	1999	37737	172006362
1	Afghanistan	2000	2666	20595360
0	Afghanistan	1999	745	19987071

Ordering

- The sorting of the index (rows or columns) is done through the **sort_index**() function

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.sort_index(ascending  = False)
```

	Country	Year	Cases	Population
5	China	2000	213766	1280428583
4	China	1999	212258	1272915272
3	Brazil	2000	80488	174504898
2	Brazil	1999	37737	172006362
1	Afghanistan	2000	2666	20595360
0	Afghanistan	1999	745	19987071

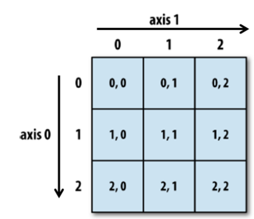
Ordering

- With the **axis** parameter we can sort the columns instead of the rows

```
dataframe.sort_index(ascending = False, axis = 1)
```

	Year	Population	Country	Cases
0	1999	19987071	Afghanistan	745
1	2000	20595360	Afghanistan	2666
2	1999	172006362	Brazil	37737
3	2000	174504898	Brazil	80488
4	1999	1272915272	China	212258
5	2000	1280428583	China	213766

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



IMMUNE CODING INSTITUTE

Exercise 32 (1/2)

- About the DataFrame "league":
- Sort the DataFrame by index (Descending)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0
19	19	Granada	28.0	4.0	7.0	17.0	25.0	58.0	19.0

- Sort the DataFrame by the ‘Puesto’ column (Descending)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0
19	19	Granada	28.0	4.0	7.0	17.0	25.0	58.0	19.0

Exercise 32 (2/2)

- Sort the DataFrame by the 'PartidosJugados' (Ascending), 'Victorias' (Descending) and 'GolesFavor' (Ascending) columns

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0
11	11	Celta de Vigo	27.0	11.0	5.0	11.0	40.0	45.0	38.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0
4	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0

Exercise 32 - Solution

```
# Ordena el DataFrame por indice (Descendente)
liga.sort_index(ascending=False)

# Ordena el DataFrame por la columna Puesto (Descendente)
liga.sort_values("Puesto", ascending=False)

# Ordena el DataFrame por las columnas:
#  - PartidosJugados (Ascendente)
#  - Victorias (Descendente)
#  - GolesFavor (Ascendente)
liga.sort_values(["PartidosJugados", "Victorias", "GolesFavor"], ascending=[True, False, True])
```

Change columns, or add new ones

- In Pandas we can create new columns by assigning a value directly to the new column

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe["Id"] = range(len(dataframe))
dataframe["Id1"] = 1
dataframe
```

	Country	Year	Cases	Population	Id	Id1
0	Afghanistan	1999	745	19987071	0	1
1	Afghanistan	2000	2666	20595360	1	1
2	Brazil	1999	37737	172006362	2	1
3	Brazil	2000	80488	174504898	3	1
4	China	1999	212258	1272815272	4	1
5	China	2000	213766	1280428583	5	1

IMMUNE CODING INSTITUTE

Change columns, or add new ones

- The **assign**() function makes easier this operation
- This función create new columns or change existing columns

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.assign(Id = range(len(dataframe)),
                 Id1 = 1)
```

	Country	Year	Cases	Population	Id	Id1
0	Afghanistan	1999	745	19987071	0	1
1	Afghanistan	2000	2666	20595360	1	1
2	Brazil	1999	37737	172006362	2	1
3	Brazil	2000	80488	174504898	3	1
4	China	1999	212258	1272915272	4	1
5	China	2000	213766	1280428583	5	1

IMMUNE CODING INSTITUTE

Change columns, or add new ones

```
dataframe.assign(Year = dataframe.Year + 100)
```

	Country	Year	Cases	Population
0	Afghanistan	2099	745	19987071
1	Afghanistan	2100	2666	20595360
2	Brazil	2099	37737	172006362
3	Brazil	2100	80488	174504898
4	China	2099	212258	1272915272
5	China	2100	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.Year + 100
```

```
0    2099
1    2100
2    2099
3    2100
4    2099
5    2100
Name: Year, dtype: int64
```

Change columns, or add new ones

```
dataframe.assign(CasesAcum = dataframe.Cases.cumsum(),
                 CasesPercent = dataframe.Cases / dataframe.Cases.sum()
)
```

	Country	Year	Cases	Population	CasesAcum	CasesPercent
0	Afghanistan	1999	745	19987071	745	0.001360
1	Afghanistan	2000	2666	20595360	3411	0.004868
2	Brazil	1999	37737	172006362	41148	0.068906
3	Brazil	2000	80488	174504898	121636	0.146967
4	China	1999	212258	1272815272	333894	0.387573
5	China	2000	213766	1280428583	547660	0.390326

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.Cases.cumsum()
```

```
0       745
1      3411
2     41148
3    121636
4    333894
5    547660
Name: Cases, dtype: int64
```

```
dataframe.Cases.sum()
```

```
547660
```

Replacing Column Values

```
dataframe.assign(Country = dataframe.Country.replace(
    {
      'Afghanistan' : 'Afg',
      'China' : 'Chin'
    }
  )
)
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afg	1999	745	19987071
1	Afg	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	Chin	1999	212258	1272815272
5	Chin	2000	213766	1280428583

Adding a unique Id

```
(codes, uniques) = pd.factorize(['b', 'b', 'a', 'c', 'b'], sort=True)
print(codes)
print(uniques)
```

```
[1 1 0 2 1]
['a' 'b' 'c']
```

```
pd.factorize(dataframe.Country, sort=True)[0]
```

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

```
dataframe.assign(
    id = pd.factorize(dataframe.Country, sort=True)[0] + 1
    )
```

	Country	Year	Cases	Population	id
0	Afghanistan	1999	745	19987071	1
1	Afghanistan	2000	2666	20595360	1
2	Brazil	1999	37737	172006362	2
3	Brazil	2000	80488	174504898	2
4	China	1999	212258	1272815272	3
5	China	2000	213766	1280428583	3

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Adding a unique Id

```
dataframe.assign(
    id = pd.factorize(
        dataframe.apply(lambda row : row.Country + str(row.Year), axis = 1)
        , sort=True)[0] + 1
    )
```

	Country	Year	Cases	Population	id
0	Afghanistan	1999	745	19987071	1
1	Afghanistan	2000	2666	20595360	2
2	Brazil	1999	37737	172006362	3
3	Brazil	2000	80488	174504898	4
4	China	1999	212258	1272815272	5
5	China	2000	213766	1280428583	6

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.apply(
    lambda row : row.Country + str(row.Year),
    axis = 1
)
```

```
0    Afghanistan1999
1    Afghanistan2000
2         Brazil1999
3         Brazil2000
4          China1999
5          China2000
dtype: object
```

Temporal Series

```
dataframe = dataframe.assign(
    Date_str = dataframe.Year.apply(lambda value : str(value) + "/01/01")
)
dataframe
```

	Country	Year	Cases	Population	Date_str
0	Afghanistan	1999	745	19987071	1999/01/01
1	Afghanistan	2000	2666	20595360	2000/01/01
2	Brazil	1999	37737	172006362	1999/01/01
3	Brazil	2000	80488	174504898	2000/01/01
4	China	1999	212258	1272815272	1999/01/01
5	China	2000	213766	1280428583	2000/01/01

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Temporal Series

```
dataframe = dataframe.assign(
    Date = pd.to_datetime(dataframe.Date_str, format = '%Y/%m/%d')
)
dataframe
```

	Country	Year	Cases	Population	Date_str	Date
0	Afghanistan	1999	745	19987071	1999/01/01	1999-01-01
1	Afghanistan	2000	2666	20595360	2000/01/01	2000-01-01
2	Brazil	1999	37737	172006362	1999/01/01	1999-01-01
3	Brazil	2000	80488	174504898	2000/01/01	2000-01-01
4	China	1999	212258	1272815272	1999/01/01	1999-01-01
5	China	2000	213766	1280428583	2000/01/01	2000-01-01

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Country     6 non-null      object
 1   Year        6 non-null      int64
 2   Cases       6 non-null      int64
 3   Population  6 non-null      int64
 4   Date_str    6 non-null      object
 5   Date        6 non-null      datetime64[ns]
dtypes: datetime64[ns](1), int64(3), object(2)
memory usage: 416.0+ bytes
```

IMMUNE CODING INSTITUTE

Categorical Series

- Pandas provides the **cut()** function to assign a group to a continuous variable, depending on a range of values

```
n = [61, 16, 21, 62, 80, 55, 32, 20, 53, 22]
n = sorted(n)
```

```
pd.DataFrame( {
    'N' : n,
    'Cut 3 Bins': pd.cut(n, bins = 3),
    'Cut 3 Label': pd.cut(n, bins = 3, labels = ['Group 1', 'Group 2', 'Group 3']),
    'Cut 3 Vector': pd.cut(n, bins = [10,32,70, 100], labels = ['10-32', '33-70', '71-100'])
})
```

	N	Cut 3 Bins	Cut 3 Label	Cut 3 Vector
0	16	(15.936, 37.333]	Group 1	10-32
1	20	(15.936, 37.333]	Group 1	10-32
2	21	(15.936, 37.333]	Group 1	10-32
3	22	(15.936, 37.333]	Group 1	10-32
4	32	(15.936, 37.333]	Group 1	10-32
5	53	(37.333, 58.667]	Group 2	33-70
6	55	(37.333, 58.667]	Group 2	33-70
7	61	(58.667, 80.0]	Group 3	33-70
8	62	(58.667, 80.0]	Group 3	33-70
9	80	(58.667, 80.0]	Group 3	71-100

Categorical Series

```
dataframe.assign(
    Type = pd.cut(dataframe.Cases,
                  bins = [0, 50000, 5000000],
                  labels= ["Type A", "Type B"])
)
```

	Country	Year	Cases	Population	Type
0	Afghanistan	1999	745	19987071	Type A
1	Afghanistan	2000	2666	20595360	Type A
2	Brazil	1999	37737	172006362	Type A
3	Brazil	2000	80488	174504898	Type B
4	China	1999	212258	1272815272	Type B
5	China	2000	213766	1280428583	Type B

Exercise 33 (1/2)

- On the DataFrame “liga”, create the following columns:
- ‘DiferenciaGoles’= ‘GolesFavor’ minus ‘GolesContra’
- ‘PorcentajeGoles’ = ‘GolesFavor’ / Sum of the ‘GolesFavor’ of all the teams
- PercentageVictorias = Victories / Matches Played
- PointsAcum = Accumulated ‘Puntos’ of all teams (cumsum)
- Materialize this new columns on the dataset

Exercise 33 – (2/2)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos	DiferenciaGoles	PorcentajeGoles	PorcentajeVictorias	PuntosAcum
0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0	43.0	0.087871	0.740741	65.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0	56.0	0.100248	0.678571	128.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0	18.0	0.064356	0.607143	185.0
4	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0	29.0	0.064356	0.571429	240.0
5	5	Villarreal	28.0	13.0	9.0	6.0	39.0	20.0	48.0	19.0	0.048267	0.464286	288.0
6	6	Real Sociedad	28.0	15.0	3.0	10.0	42.0	39.0	48.0	3.0	0.051980	0.535714	336.0
7	7	Ath. Bilbao	28.0	13.0	5.0	10.0	35.0	32.0	44.0	3.0	0.043317	0.464286	380.0
8	8	Eibar	28.0	11.0	8.0	9.0	44.0	39.0	41.0	5.0	0.054455	0.392857	421.0
9	9	RCD Espanyol	28.0	10.0	10.0	8.0	40.0	39.0	40.0	1.0	0.049505	0.357143	461.0
10	10	Alavés	28.0	10.0	10.0	8.0	29.0	33.0	40.0	-4.0	0.035891	0.357143	501.0
11	11	Celta de Vigo	27.0	11.0	5.0	11.0	40.0	45.0	38.0	-5.0	0.049505	0.407407	539.0
12	12	U. D. Las Palmas	28.0	9.0	8.0	11.0	44.0	45.0	35.0	-1.0	0.054455	0.321429	574.0
13	13	Betis	28.0	8.0	7.0	13.0	31.0	44.0	31.0	-13.0	0.038366	0.285714	605.0
14	14	Valencia C. F.	28.0	8.0	6.0	14.0	38.0	51.0	30.0	-13.0	0.047030	0.285714	635.0
15	15	Málaga	28.0	6.0	9.0	13.0	33.0	45.0	27.0	-12.0	0.040842	0.214286	662.0
16	16	Deportivo	28.0	6.0	9.0	13.0	31.0	43.0	27.0	-12.0	0.038366	0.214286	689.0
17	17	Leganés	28.0	6.0	8.0	14.0	22.0	41.0	26.0	-19.0	0.027228	0.214286	715.0
18	18	Sporting Gijón	28.0	5.0	6.0	17.0	31.0	57.0	21.0	-26.0	0.038366	0.178571	736.0
19	19	Granada	28.0	4.0	7.0	17.0	25.0	58.0	19.0	-33.0	0.030941	0.142857	755.0
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0	-39.0	0.034653	0.035714	766.0

Exercise 33 - Solution

```
# DiferenciaGoles = Goles Favor - Goles Contra
liga.assign(DiferenciaGoles =  liga.GolesFavor - liga.GolesContra)

# PorcentajeGoles = Goles Favor /  Suma de los goles de todos los equipos
liga.assign(PorcentajeGoles =  liga.GolesFavor / liga.GolesFavor.sum())

# PorcentajeVictorias = Victorias / Partidos Jugados
liga.assign(PorcentajeVictorias =  liga.Victorias / liga.PartidosJugados )

# PuntosAcum = Acumulado de los puntos de todos los equipos (cumsum)
liga.assign(PuntosAcum =  liga.Puntos.cumsum())

# Materializa el resultado en el DataFrame
liga = liga.assign(DiferenciaGoles =  liga.GolesFavor - liga.GolesContra,
                   PorcentajeGoles =  liga.GolesFavor / liga.GolesFavor.sum(),
                   PorcentajeVictorias =  liga.Victorias / liga.PartidosJugados,
                   PuntosAcum =  liga.Puntos.cumsum()
)
liga
```

Exercise 34 (1/2)

- On the DataFrame “liga", create the following columns:
- Zona = "Champions" if the team is in one of the first 4 places, “Descenso" if the team is in the last 3 places, "Normal" for the rest of the cases (pd.cut)
- DiferenciaPuntos = Difference in ‘Puntos’ between a team and the team immediately below it in the ranking (series.shift(n))
- Temporada = "2016-2017”
- Materialize this new columns on the dataset

Exercise 34 – (2/2)

	Puesto	Equipo	Puntos	Zona	DiferenciaPuntos	Temporada
0	0	NaN	NaN	NaN	NaN	2016-2017
1	1	Real Madrid	65.0	Champions	2.0	2016-2017
2	2	Barcelona	63.0	Champions	6.0	2016-2017
3	3	Sevilla	57.0	Champions	2.0	2016-2017
4	4	Atlético Madrid	55.0	Champions	7.0	2016-2017
5	5	Villarreal	48.0	Normal	0.0	2016-2017
6	6	Real Sociedad	48.0	Normal	4.0	2016-2017
7	7	Ath. Bilbao	44.0	Normal	3.0	2016-2017
8	8	Eibar	41.0	Normal	1.0	2016-2017
9	9	RCD Espanyol	40.0	Normal	0.0	2016-2017
10	10	Alavés	40.0	Normal	2.0	2016-2017
11	11	Celta de Vigo	38.0	Normal	3.0	2016-2017
12	12	U. D. Las Palmas	35.0	Normal	4.0	2016-2017
13	13	Betis	31.0	Normal	1.0	2016-2017
14	14	Valencia C. F.	30.0	Normal	3.0	2016-2017
15	15	Málaga	27.0	Normal	0.0	2016-2017
16	16	Deportivo	27.0	Normal	1.0	2016-2017
17	17	Leganés	26.0	Normal	5.0	2016-2017
18	18	Sporting Gijón	21.0	Descenso	2.0	2016-2017
19	19	Granada	19.0	Descenso	8.0	2016-2017
20	20	Osasuna	11.0	Descenso	NaN	2016-2017

Exercise 34 - Solution

```
# Zona = «Champions» si el equipo esta en uno de los 4 primeros puestos,
#        «Descenso» si el equipo es un los 3 últimos ,
#        «Normal» para el resto de casos
liga.assign(Zona = pd.cut(liga.Puesto, bins = [0, 4, 17, 20],
                          labels = ["Champions", "Normal", "Descenso"]))

# DiferenciaPuntos = Diferencia de puntos entre un equipo y el que está inmediatamente debajo en la clasificación
liga.assign(DiferenciaPuntos = liga.Puntos - liga.Puntos.shift(-1))

# Temporada = "2016-2017"
liga.assign(Temporada = "2016-2017")


# Materializa el resultado en el DataFrame
liga = liga.assign(Zona = pd.cut(liga.Puesto, bins = [0, 4, 17, 20], labels = ["Champions", "Normal", "Descenso"]),
                   DiferenciaPuntos = liga.Puntos - liga.Puntos.shift(-1),
                   Temporada = "2016-2017"
)
liga[["Puesto", "Equipo", "Puntos", "Zona", "DiferenciaPuntos", "Temporada"]]
```

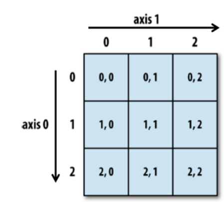
Deleting rows

- In DataFrames, the **drop()** function allows you to delete both rows and columns by specifying an array of names

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.drop([1, 3], axis = 0)
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



Deleting rows

- We can not use a condition to delete rows with **drop**()

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.drop(dataframe.Country != 'China', axis = 0)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-240-6d342841d877> in <module>()
----> 1 dataframe.drop(dataframe.Country != 'China', axis = 0)

                                  3 frames

/usr/local/lib/python3.6/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
   5015         if mask.any():
   5016             if errors != "ignore":
-> 5017                 raise KeyError(f"{labels[mask]} not found in axis")
   5018             indexer = indexer[~mask]
   5019         return self.delete(indexer)

KeyError: '[ True  True  True  True] not found in axis'
```

Deleting rows

- The trick is selecting the rows that you want to mantein in the dataset

```
dataframe= dataframe.query("not Country == 'China'")
dataframe
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Deleting columns

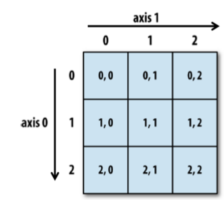
```
dataframe.drop('Year', axis = 1)
```

	Country	Cases	Population
0	Afghanistan	745	19987071
1	Afghanistan	2666	20595360
2	Brazil	37737	172006362
3	Brazil	80488	174504898
4	China	212258	1272815272
5	China	213766	1280428583

```
dataframe.drop(['Country', 'Year'], axis = 1)
```

	Cases	Population
0	745	19987071
1	2666	20595360
2	37737	172006362
3	80488	174504898
4	212258	1272815272
5	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



Deleting columns

- A different method is to select only with the columns I want to keep in the dataset

```
dataframe[['Cases', 'Population']]
```

	Cases	Population
0	745	19987071
1	2666	20595360
2	37737	172006362
3	80488	174504898
4	212258	1272815272
5	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Exercise 35 – (1/2)

- About the DataFrame "league
- Delete row 0
- Delete the “Temporada" column
- Materialize this changes on the dataset

IMMUNE
CODING INSTITUTE

Exercise 35 – (2/2)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos	DiferenciaGoles	PorcentajeGoles	PorcentajeVictorias	PuntosAcum	Zona	DiferenciaPuntos
1	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0	43.0	0.087871	0.740741	65.0	Champions	2.0
2	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0	56.0	0.100248	0.678571	128.0	Champions	6.0
3	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0	18.0	0.064356	0.607143	185.0	Champions	2.0
4	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0	29.0	0.064356	0.571429	240.0	Champions	7.0
5	5	Villarreal	28.0	13.0	9.0	6.0	39.0	20.0	48.0	19.0	0.048267	0.464286	288.0	Normal	0.0
6	6	Real Sociedad	28.0	15.0	3.0	10.0	42.0	39.0	48.0	3.0	0.051980	0.535714	336.0	Normal	4.0
7	7	Ath. Bilbao	28.0	13.0	5.0	10.0	35.0	32.0	44.0	3.0	0.043317	0.464286	380.0	Normal	3.0
8	8	Eibar	28.0	11.0	8.0	9.0	44.0	39.0	41.0	5.0	0.054455	0.392857	421.0	Normal	1.0
9	9	RCD Espanyol	28.0	10.0	10.0	8.0	40.0	39.0	40.0	1.0	0.049505	0.357143	461.0	Normal	0.0
10	10	Alavés	28.0	10.0	10.0	8.0	29.0	33.0	40.0	-4.0	0.035891	0.357143	501.0	Normal	2.0
11	11	Celta de Vigo	27.0	11.0	5.0	11.0	40.0	45.0	38.0	-5.0	0.049505	0.407407	539.0	Normal	3.0
12	12	U. D. Las Palmas	28.0	9.0	8.0	11.0	44.0	45.0	35.0	-1.0	0.054455	0.321429	574.0	Normal	4.0
13	13	Betis	28.0	8.0	7.0	13.0	31.0	44.0	31.0	-13.0	0.038366	0.285714	605.0	Normal	1.0
14	14	Valencia C. F.	28.0	8.0	6.0	14.0	38.0	51.0	30.0	-13.0	0.047030	0.285714	635.0	Normal	3.0
15	15	Málaga	28.0	6.0	9.0	13.0	33.0	45.0	27.0	-12.0	0.040842	0.214286	662.0	Normal	0.0
16	16	Deportivo	28.0	6.0	9.0	13.0	31.0	43.0	27.0	-12.0	0.038366	0.214286	689.0	Normal	1.0
17	17	Leganés	28.0	6.0	8.0	14.0	22.0	41.0	26.0	-19.0	0.027228	0.214286	715.0	Normal	5.0
18	18	Sporting Gijón	28.0	5.0	6.0	17.0	31.0	57.0	21.0	-26.0	0.038366	0.178571	736.0	Descenso	2.0
19	19	Granada	28.0	4.0	7.0	17.0	25.0	58.0	19.0	-33.0	0.030941	0.142857	755.0	Descenso	8.0
20	20	Osasuna	28.0	1.0	8.0	19.0	28.0	67.0	11.0	-39.0	0.034653	0.035714	766.0	Descenso	NaN

Exercise 35 - Solution

```
# Elimina la fila 0
liga = liga.drop(0, axis = 0)

# Elimina la columna Temporadas
liga = liga.drop("Temporada", axis = 1)

liga
```

Obtain a data sample

- The **sample()** function allows you to obtain a sample of a Dataframe, specifying both a percentage and a specific number of rows

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.sample(n=3)
```

	Country	Year	Cases	Population
1	Afghanistan	2000	2666	20595360
4	China	1999	212258	1272915272
3	Brazil	2000	80488	174504898

```
dataframe.sample(frac=.3)
```

	Country	Year	Cases	Population
1	Afghanistan	2000	2666	20595360
5	China	2000	213766	1280428583

Obtain a data sample

- It is possible to set the random seed through numpy's RandomState function

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
state = np.random.RandomState(seed = 100)
dataframe.sample(n=3, random_state=state)
```

	Country	Year	Cases	Population
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
4	China	1999	212258	1272915272

IMMUNE CODING INSTITUTE

Exercise 36

- About the DataFrame “liga”
- Sets the random seed to 201231
- Get an example of 4 rows

	Puesto	Equipo	PartidosJugados	Victoria
1	1	Real Madrid	27.0	20
5	5	Villarreal	28.0	13
0	0	NaN	NaN	Na
17	17	Leganés	28.0	6

- Get an example of the 30% of the rows

	Puesto	Equipo	PartidosJugados	Victoria
14	14	Valencia C. F.	28.0	8.
5	5	Villarreal	28.0	13.
8	8	Eibar	28.0	11.
7	7	Ath. Bilbao	28.0	13.
15	15	Málaga	28.0	6.
2	2	Barcelona	28.0	19.

Exercise 36 - Solution

```
# Establece la semilla aleatoria a 201231
state = np.random.RandomState(seed = 201231)

# Obten un ejemplo de 4 filas
liga.sample(4, random_state = state)

# Obtén el 30% de las filas
liga.sample(frac = 0.3, random_state = state)
```

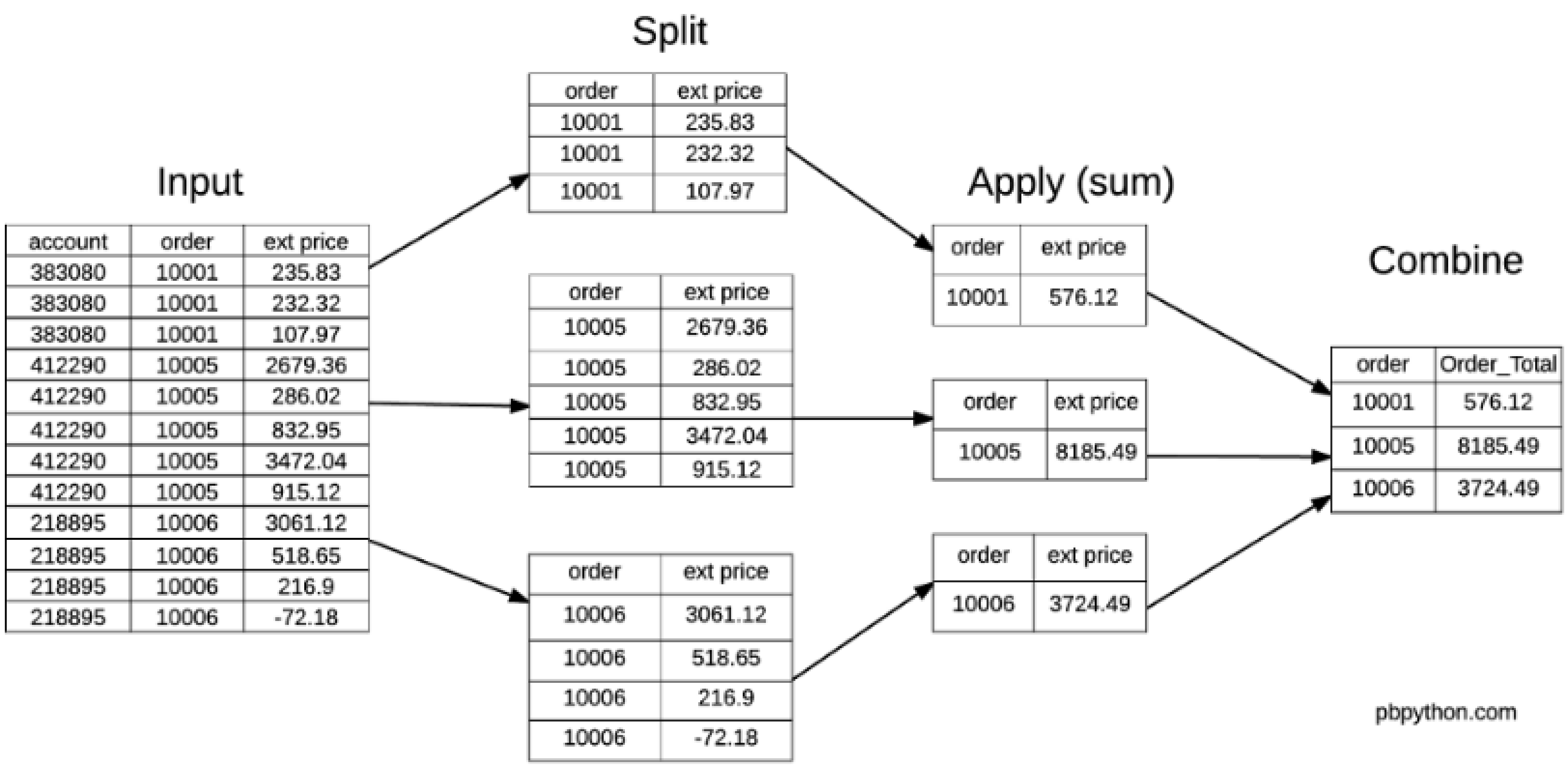
Grouping of rows

- Pandas has a series of functions that allow to obtain data from groupings

```
SELECT column1, column2, mean(column3), sum(column4)
FROM some_table
GROUP BY column1, column2
```

Grouping of rows

- Pandas has a series of functions that allow to obtain data from groupings



Grouping of rows

- The **groupby()** function allows you to group rows and generate groups

```
grouped = dataframe.groupby("Country")
grouped.groups
```

```
{'Afghanistan': Int64Index([0, 1], dtype='int64'),
 'Brazil': Int64Index([2, 3], dtype='int64'),
 'China': Int64Index([4, 5], dtype='int64')}
```

```
for name, group in grouped:
   print(name)
   print(group)
```

```
Afghanistan
       Country  Year  Cases  Population
0  Afghanistan  1999    745    19987071
1  Afghanistan  2000   2666    20595360
Brazil
  Country  Year  Cases  Population
2  Brazil  1999  37737   172006362
3  Brazil  2000  80488   174504898
China
  Country  Year   Cases  Population
4   China  1999  212258  1272815272
5   China  2000  213766  1280428583
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Grouping of rows

- Once you have a group, you can use grouping functions such as **size**(), **count**(), **sum(), mean**(), **first**(), **last(),** etc.

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.groupby("Country").count()
```

Country	Year	Cases	Population
Afghanistan	2	2	2
Brazil	2	2	2
China	2	2	2

(*) size() includes NaN values, count() does not

Grouping of rows

- Once you have a group, you can execute grouping functions such as **size**(), **count**(), **sum(), mean**(), **first**(), **last(),** etc.

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.groupby(["Country", "Year"]).count()
```

Country	Year	Cases	Population
Afghanistan	**1999**	1	1
	2000	1	1
Brazil	**1999**	1	1
	2000	1	1
China	**1999**	1	1
	2000	1	1

IMMUNE CODING INSTITUTE

Grouping of rows

```
dataframe.groupby("Country").sum()
```

Country	Year	Cases	Population
Afghanistan	3999	3411	40582431
Brazil	3999	118225	346511260
China	3999	426024	2553343855

```
dataframe.groupby("Country").mean()
```

Country	Year	Cases	Population
Afghanistan	1999.5	1705.5	2.029122e+07
Brazil	1999.5	59112.5	1.732556e+08
China	1999.5	213012.0	1.276672e+09

```
dataframe.groupby("Country").first()
```

Country	Year	Cases	Population
Afghanistan	1999	745	19987071
Brazil	1999	37737	172006362
China	1999	212258	1272915272

```
dataframe.groupby("Country").last()
```

Country	Year	Cases	Population
Afghanistan	2000	2666	20595360
Brazil	2000	80488	174504898
China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

IMMUNE CODING INSTITUTE

Grouping of rows

- The **aggregate**() function allows you to apply a set of grouping functions to all columns

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.groupby("Country").aggregate(["mean", "sum"])
```

	Year		Cases		Population	
	mean	sum	mean	sum	mean	sum
Country						
Afghanistan	1999.5	3999	1705.5	3411	20291215	40582431
Brazil	1999.5	3999	59112.5	118225	173255630	346511260
China	1999.5	3999	213012.0	426024	1276671927	2553343855

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.aggregate.html

IMMUNE CODING INSTITUTE

Grouping of rows

- We can select a specific column from the DataFrame that is generated by **aggregate**()

```
dataframe.groupby("Country").aggregate(["mean", "sum"])
```

	Year		Cases		Population	
	mean	sum	mean	sum	mean	sum
Country						
Afghanistan	1999.5	3999	1705.5	3411	20291215	40582431
Brazil	1999.5	3999	59112.5	118225	173255630	346511260
China	1999.5	3999	213012.0	426024	1276671927	2553343855

```
dataframe.groupby("Country").aggregate(["mean", "count", "sum"]).Population
```

	mean	count	sum
Country			
Afghanistan	20291215	2	40582431
Brazil	173255630	2	346511260
China	1276671927	2	2553343855

Grouping of rows

- The function **agg**() allows you to obtain new values by applying specific functions to a given column

```
dataframe.groupby("Country").agg({"Year" : "count",
                                  "Cases": np.sum,
                                  "Population" : lambda x : x.count()})
```

	Year	Cases	Population
Country			
Afghanistan	2	3411	2
Brazil	2	118225	2
China	2	426024	2

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
SELECT country,
       count(Year),
       sum(Cases),
       mean(Poputation)
FROM dataframe
GROUP BY country
```

Aggregate functions over all the data

- There are cases where we need aggregate all the data in our dataset

```
SELECT mean(column3), sum(column4)
FROM some_table
```

IMMUNE CODING INSTITUTE

Aggregate functions over all the data

- In these cases we can use functions like **agg** or **aggregate** without use **groupby** previosly

```
dataframe.aggregate("mean")
```

```
Year          1.999500e+03
Cases         9.127667e+04
Population    4.900563e+08
dtype: float64
```

```
dataframe[["Cases", "Population"]].aggregate(["mean", "sum"])
```

	Cases	Population
mean	91276.666667	4.900563e+08
sum	547660.000000	2.940338e+09

```
dataframe.agg({ "Cases": "mean", "Population" : np.sum})
```

```
Cases         9.127667e+04
Population    2.940338e+09
dtype: float64
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Transforming rows

- The **transform()** function applies a grouping function to a group, but returns an object with the same size as the original dataframe

```
dataframe.groupby("Country").Cases.sum()
```

```
Country
Afghanistan      3411
Brazil         118225
China          426024
Name: Cases, dtype: int64
```

```
dataframe.groupby("Country").Cases.transform("sum")
```

```
0      3411
1      3411
2    118225
3    118225
4    426024
5    426024
Name: Cases, dtype: int64
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Transforming rows

```
dataframe.assign(
    TotalCases = dataframe.groupby("Country").Cases.transform("sum")
)
```

	Country	Year	Cases	Population	TotalCases
0	Afghanistan	1999	745	19987071	3411
1	Afghanistan	2000	2666	20595360	3411
2	Brazil	1999	37737	172006362	118225
3	Brazil	2000	80488	174504898	118225
4	China	1999	212258	1272815272	426024
5	China	2000	213766	1280428583	426024

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Reseting the index

- The result of grouping multiple columns is a DataFrame with a MultiIndex

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.groupby(["Country", "Year"]).count()
```

		Cases	Population
Country	Year		
Afghanistan	1999	1	1
	2000	1	1
Brazil	1999	1	1
	2000	1	1
China	1999	1	1
	2000	1	1

```
dataframe.groupby(["Country", "Year"]).count().index
```

```
MultiIndex([('Afghanistan', 1999),
            ('Afghanistan', 2000),
            (     'Brazil', 1999),
            (     'Brazil', 2000),
            (      'China', 1999),
            (      'China', 2000)],
           names=['Country', 'Year'])
```

```
dataframe.groupby(["Country", "Year"]).count().columns
```

```
Index(['Cases', 'Population'], dtype='object')
```

IMMUNE CODING INSTITUTE

Reseting the index

- We can incorportate the index as regular columns with the **reset_index**() function

```
dataframe.groupby(["Country", "Year"]).count()
```

Country	Year	Cases	Population
Afghanistan	1999	1	1
	2000	1	1
Brazil	1999	1	1
	2000	1	1
China	1999	1	1
	2000	1	1

```
dataframe.groupby(["Country", "Year"]).count().reset_index()
```

	Country	Year	Cases	Population
0	Afghanistan	1999	1	1
1	Afghanistan	2000	1	1
2	Brazil	1999	1	1
3	Brazil	2000	1	1
4	China	1999	1	1
5	China	2000	1	1

IMMUNE CODING INSTITUTE

Exercise 37

- About the DataFrame "liga"
- Sum “Puntos” and “GolesFavor” for all teams
- By zones, for the fields “Puntos” and “GolesFavor”, calculate the sum and count all rows

```
Puntos        766.0
GolesFavor    808.0
dtype: float64
```

	Puntos		GolesFavor	
	sum	count	sum	count
Zona				
Champions	240.0	4	256.0	4
Normal	496.0	14	499.0	14
Descenso	30.0	2	53.0	2

Exercise 37 - Solution

```
# Suma de puntos y goles a favor para todos los equipos
liga[["Puntos", "GolesFavor"]].aggregate("sum")

# Por zonas, para los campos Puntos y GolesFavor, calcula la suma y la cuenta
liga.groupby("Zona")[["Puntos", "GolesFavor"]].aggregate(["sum", "count"])
```

Exercise 38 (1/2)

- By zones:
  - Distinct values in “PartidosJugados” (np.nunique)
  - Sum “GolesFalor”
  - Calculate the average in “Diferencia de Goles”
- Over the result of the previous step calculate the percentage of goals scored by each group (“GolesFavor”/ Total of “GolesFavor”)

Zona	PartidosJugados	GolesFavor	DiferenciaGoles	PorcentajeGoles
Champions	2.0	256.0	36.500000	0.316832
Normal	2.0	468.0	-3.692308	0.579208
Descenso	1.0	84.0	-32.666667	0.103960

Exercise 38 (2/2)

- Show all the teams in each zone (join function on an array)

	Equipo
Zona	
Champions	Real Madrid / Barcelona / Sevilla / Atlético M...
Normal	Villarreal / Real Sociedad / Ath. Bilbao / Eib...
Descenso	Granada / Osasuna

Exercise 38 - Solution

```
# Por zonas:
# Valores distintos de PartidosJugados
# Suma los Goles a Favor
# Media de diferencia de Goles
df_grupo = liga.groupby("Zona").agg({
    "PartidosJugados" : lambda x: x.nunique(),
    "GolesFavor": "sum",
    "DiferenciaGoles": "mean"
})

# Sobre el resultado del paso anterior, calcula el porcentaje de Goles marcados por cada grupo:
df_grupo.assign(PorcentajeGoles = df_grupo.GolesFavor / df_grupo.GolesFavor.sum())

# Muestra los distintos equipos que contiene cada zona
liga.groupby("Zona").agg({
    "Equipo" : lambda x: ' / '.join(x)
})
```

IMMUNE CODING INSTITUTE

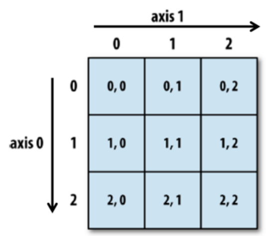
Concatenating a DataFrame

- The **concat**() function allows to join several DataFrames in a single one, both by rows and by columns

```
pd.concat((dataframe, dataframe), axis = 0)
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



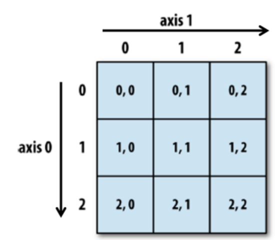
Concatenating a DataFrame

- The **reset_index**() function rebuilds the index ...

```
pd.concat((dataframe, dataframe), axis = 0)\
    .reset_index(drop = True)
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583
6	Afghanistan	1999	745	19987071
7	Afghanistan	2000	2666	20595360
8	Brazil	1999	37737	172006362
9	Brazil	2000	80488	174504898
10	China	1999	212258	1272915272
11	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583



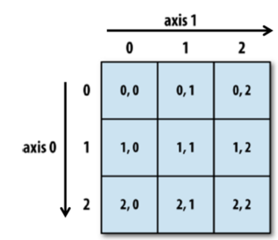
Concatenating a DataFrame

- A shortcut for the **concat()** function when two dataframes are joined is the **append**() function

```
dataframe.append(dataframe)
```

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

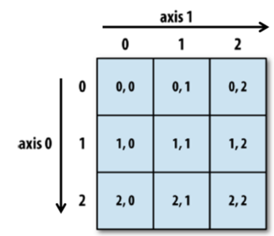


Concatenating a DataFrame

```
rank = dataframe.rank().rename(columns = lambda column: "Rank " + column)
rank
```

	Rank Country	Rank Year	Rank Cases	Rank Population
0	1.5	2.0	1.0	1.0
1	1.5	5.0	2.0	2.0
2	3.5	2.0	3.0	3.0
3	3.5	5.0	4.0	4.0
4	5.5	2.0	5.0	5.0
5	5.5	5.0	6.0	6.0

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583




```
pd.concat((dataframe, rank), axis = 1)
```

	Country	Year	Cases	Population	Rank Country	Rank Year	Rank Cases	Rank Population
0	Afghanistan	1999	745	19987071	1.5	2.0	1.0	1.0
1	Afghanistan	2000	2666	20595360	1.5	5.0	2.0	2.0
2	Brazil	1999	37737	172006362	3.5	2.0	3.0	3.0
3	Brazil	2000	80488	174504898	3.5	5.0	4.0	4.0
4	China	1999	212258	1272815272	5.5	2.0	5.0	5.0
5	China	2000	213766	1280428583	5.5	5.0	6.0	6.0

Join

- The **merge**() function allows to make a join between two DataFrames

```
df_capitals = pd.DataFrame(
    {"Country" : ["Afghanistan", "Brazil", "Spain"],
     "Capital" : ["Kabul", "Brasilia", "Madrid"]},
     columns = ["Country", "Capital"]
)
df_capitals
```

	Country	Capital
0	Afghanistan	Kabul
1	Brazil	Brasilia
2	Spain	Madrid

Join

- The **merge**() function allows to make a join between two DataFrames

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
pd.merge(dataframe, df_capitals, on = "Country")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999	745	19987071	Kabul
1	Afghanistan	2000	2666	20595360	Kabul
2	Brazil	1999	37737	172006362	Brasilia
3	Brazil	2000	80488	174504898	Brasilia

	Country	Capital
0	Afghanistan	Kabul
1	Brazil	Brasilia
2	Spain	Madrid

Join

- The parameters **left_on** and **right_on** allow you to specify different names in both dataframes

```
pd.merge(dataframe, df_capitals,
         left_on = "Country",
         right_on =  "Country")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999	745	19987071	Kabul
1	Afghanistan	2000	2666	20595360	Kabul
2	Brazil	1999	37737	172006362	Brasilia
3	Brazil	2000	80488	174504898	Brasilia

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	Country	Capital
0	Afghanistan	Kabul
1	Brazil	Brasilia
2	Spain	Madrid

Join

- The parameter **how** allow you to specify different join methods: ‘left’, ‘right’, ‘outer’, ‘inner’

```
pd.merge(dataframe, df_capitals, on = "Country", how = "inner")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999	745	19987071	Kabul
1	Afghanistan	2000	2666	20595360	Kabul
2	Brazil	1999	37737	172006362	Brasilia
3	Brazil	2000	80488	174504898	Brasilia

```
pd.merge(dataframe, df_capitals, on = "Country", how = "outer")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999.0	745.0	1.998707e+07	Kabul
1	Afghanistan	2000.0	2666.0	2.059536e+07	Kabul
2	Brazil	1999.0	37737.0	1.720064e+08	Brasilia
3	Brazil	2000.0	80488.0	1.745049e+08	Brasilia
4	China	1999.0	212258.0	1.272815e+09	NaN
5	China	2000.0	213766.0	1.280429e+09	NaN
6	Spain	NaN	NaN	NaN	Madrid

```
pd.merge(dataframe, df_capitals, on = "Country", how = "left")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999	745	19987071	Kabul
1	Afghanistan	2000	2666	20595360	Kabul
2	Brazil	1999	37737	172006362	Brasilia
3	Brazil	2000	80488	174504898	Brasilia
4	China	1999	212258	1272815272	NaN
5	China	2000	213766	1280428583	NaN

```
pd.merge(dataframe, df_capitals, on = "Country", how = "right")
```

	Country	Year	Cases	Population	Capital
0	Afghanistan	1999.0	745.0	19987071.0	Kabul
1	Afghanistan	2000.0	2666.0	20595360.0	Kabul
2	Brazil	1999.0	37737.0	172006362.0	Brasilia
3	Brazil	2000.0	80488.0	174504898.0	Brasilia
4	Spain	NaN	NaN	NaN	Madrid

Elimination of duplicate rows

- The **drop_duplicate()** function allows you to remove duplicate records from a DataFrame

```
dataframe[["Country"]]
```

	Country
0	Afghanistan
1	Afghanistan
2	Brazil
3	Brazil
4	China
5	China

```
dataframe[["Country"]].drop_duplicates()
```

	Country
0	Afghanistan
2	Brazil
4	China

Exercise 39

- Concatenate the DataFrame “liga" so that it duplicates its content (by rows) and assigns it to the variable “liga2”
- Display the number of rows and columns
- Eliminate duplicate rows and display the shape again

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrota:
10	10	Alavés	28.0	10.0	10.0	8.(
10	10	Alavés	28.0	10.0	10.0	8.(
7	7	Ath. Bilbao	28.0	13.0	5.0	10.(
7	7	Ath. Bilbao	28.0	13.0	5.0	10.(
4	4	Atlético Madrid	28.0	16.0	7.0	5.(
4	4	Atlético Madrid	28.0	16.0	7.0	5.(

```
(40, 15)
(20, 15)
```

Exercise 39 - Solution

```
# Concatena el DataFrame "liga" de forma que dulique su contenido
# Asigna el resultado a una variable llamada "liga2"
liga2 = pd.concat([liga, liga], axis = 0)
display(liga2.sort_values("Equipo"))

# Muestra el número de filas y columnas del DataFrame
display(liga2.shape)

# Elimina las filas duplicadas en la variable liga2
liga2 = liga2.drop_duplicates()
# Vuelve a mostrar el número de filas y columnas del DataFrame
display(liga2.shape)
```

Exercise 40

- Create a DataFrame called “equipos" with the content of the Excel file “Equipos.xlsx“

	Equipo	Provincia	Comunidad Autónoma
0	Real Madrid	Madrid	Madrid
1	Barcelona	Barcelona	Cataluña
2	Sevilla	Sevilla	Andalucia
3	Atlético Madrid	Madrid	Madrid
4	Villarreal	Castellón	Comunidad Valenciana

- In the DataFrame “liga“, assign new columns with the province of the team and another one with its autonomous community (materialize the columns)

	Puesto	Equipo	PartidosJugados	Victorias	Empates	Derrotas	GolesFavor	GolesContra	Puntos	DiferenciaGoles	PorcentajeGoles	PorcentajeVictorias	PuntosAcum	Zona	DiferenciaPuntos	Provincia	Comunidad Autónoma
0	1	Real Madrid	27.0	20.0	5.0	2.0	71.0	28.0	65.0	43.0	0.087871	0.740741	65.0	Champions	2.0	Madrid	Madrid
1	2	Barcelona	28.0	19.0	6.0	3.0	81.0	25.0	63.0	56.0	0.100248	0.678571	128.0	Champions	6.0	Barcelona	Cataluña
2	3	Sevilla	28.0	17.0	6.0	5.0	52.0	34.0	57.0	18.0	0.064356	0.607143	185.0	Champions	2.0	Sevilla	Andalucia
3	4	Atlético Madrid	28.0	16.0	7.0	5.0	52.0	23.0	55.0	29.0	0.064356	0.571429	240.0	Champions	7.0	Madrid	Madrid
4	5	Villarreal	28.0	13.0	9.0	6.0	39.0	20.0	48.0	19.0	0.048267	0.464286	288.0	Normal	0.0	Castellón	Comunidad Valenciana

Exercise 40 - Solution

```python
# Carga el fichero "Equipos.xlsx" en una variable llamada "equipos"
equipos = pd.read_excel("Equipos.xlsx")
display(equipos.head())

# En el dataframe "liga", crea una nuevas columnas con la provincia del equipo
# y su comunidad autonoma (materializa las columnas)
liga = liga.merge(equipos, on = "Equipo")
liga
```

Pivot Tables

- The function **pivot_table** () allows us to create a table where we apply a series of grouping functions in a set of values and categories at the same time.

	Account	Name	Rep	Manager	Product	Quantity	Price	Status
0	714466	Trantow-Barrows	Craig Booker	Debra Henley	CPU	1	30000	presented
1	714466	Trantow-Barrows	Craig Booker	Debra Henley	Software	1	10000	presented
2	714466	Trantow-Barrows	Craig Booker	Debra Henley	Maintenance	2	5000	pending
3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won

```
pd.pivot_table(df,
    index=[“Manager”,”Status”],
    columns=[“Product”],
    aggfunc=[np.sum],
    values=[“Price”],
    fill_value=0,
    margins=True,
    dropna=True)
```

Can also use a dictionary:
aggfunc={"Quantity":len,
"Price":[np.sum,np.mean]}

		sum				
		Price				
	Product	CPU	Maintenance	Monitor	Software	All
Manager	Status					
Debra Henley	declined	70000	0	0	0	70000
	pending	40000	10000	0	0	50000
	presented	30000	0	0	20000	50000
	won	65000	0	0	0	65000
Fred Anderson	declined	65000	0	0	0	65000
	pending	0	5000	0	0	5000
	presented	30000	0	5000	10000	45000
	won	165000	7000	0	0	172000
All		465000	22000	5000	30000	522000

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot_table.html

Pivot Tables

- The simplest pivot table must have a DataFrame and an **index**

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

```
dataframe.pivot_table(
    index = "Country"
)
```

	Cases	Population	Year
Country			
Afghanistan	1705.5	2.029122e+07	1999.5
Brazil	59112.5	1.732556e+08	1999.5
China	213012.0	1.276622e+09	1999.5

```
dataframe.pivot_table(
    index = ["Year", "Country"]
)
```

		Cases	Population
Year	**Country**		
1999	**Afghanistan**	745	19987071
	Brazil	37737	172006362
	China	212258	1272815272
2000	**Afghanistan**	2666	20595360
	Brazil	80488	174504898
	China	213766	1280428583

Pivot Tables

- **pivot_table** shows all the numerical values, but it can be filtered with **values** parameter

```
dataframe.pivot_table(
    index = "Country"
)
```

Country	Cases	Population	Year
Afghanistan	1705.5	2.029122e+07	1999.5
Brazil	59112.5	1.732556e+08	1999.5
China	213012.0	1.276622e+09	1999.5

```
dataframe.pivot_table(
    index = "Country",
    values = "Cases"
)
```

Country	Cases
Afghanistan	1705.5
Brazil	59112.5
China	213012.0

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

IMMUNE CODING INSTITUTE

Pivot Tables

- By default, **pivot_table** uses the **mean** as aggregate function. It can be changed with the **aggfunc** parameter

```
dataframe.pivot_table(
    index = ["Country"],
    values = ["Cases"]
)
```

	Cases
Country	
Afghanistan	1705.5
Brazil	59112.5
China	213012.0

```
dataframe.pivot_table(
    index = "Country",
    values = "Cases",
    aggfunc = "sum"
)
```

	Cases
Country	
Afghanistan	3411
Brazil	118225
China	426024

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Pivot Tables

- You can use a list of aggregate funcions instead a single value

```
dataframe.pivot_table(
    index = "Country",
    values = "Cases",
    aggfunc = [np.sum, "mean"]
)
```

	sum	mean
	Cases	Cases
Country		
Afghanistan	3411	1705.5
Brazil	118225	59112.5
China	426024	213012.0

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Pivot Tables

- The **columns** parameter provide an additional way to segment the actual values you care about.

```
dataframe.pivot_table(
    index = "Country",
    columns = "Year",
    values = "Cases",
    aggfunc = np.sum
)
```

Year	1999	2000
Country		
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

	Country	Year	Cases	Population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Exercise 41

- About the DataFrame “liga",
- Create a pivot table with the average points per autonomous community
- Sort the result from highest to lowest

	Puntos
Comunidad Autónoma	
Cataluña	51.500000
Madrid	48.666667
Pais Vasco	43.250000
Comunidad Valenciana	39.000000
Canarias	35.000000
Andalucia	33.500000
Galicia	32.500000
Asturias	21.000000
Navarra	11.000000

Exercise 41 - Solution

```
# Crea una pivot table con la media de puntos por comunidad autonoma
pivot = liga.pivot_table(index = "Comunidad Autónoma",
                         values = "Puntos",
                         aggfunc="mean")

# Ordena el resultado de mayor a menor
pivot.sort_values("Puntos", ascending=False)
```

Exercise 42

- Create a pivot table with the average of “Puntos” and “DiferenciaGoles” by autonomous community and province

Comunidad Autónoma	Provincia	DiferenciaGoles	Puntos
Andalucia	Granada	-33.000000	19.000000
	Malaga	-12.000000	27.000000
	Sevilla	2.500000	44.000000
Asturias	Asturias	-26.000000	21.000000
Canarias	Las Palmas	-1.000000	35.000000
Cataluña	Barcelona	28.500000	51.500000
Comunidad Valenciana	Castellón	19.000000	48.000000
	Valencia	-13.000000	30.000000
Galicia	A Coruña	-12.000000	27.000000
	Pontevedra	-5.000000	38.000000
Madrid	Madrid	17.666667	48.666667
Navarra	Navarra	-39.000000	11.000000
Pais Vasco	Guipúzcoa	4.000000	44.500000
	Vizcaya	3.000000	44.000000
	Álava	-4.000000	40.000000

Exercise 42 - Solution

```
# Crea una pivot table con la media de puntos y diferencia de goles por comunidad autónoma y provincia
liga.pivot_table(index = ["Comunidad Autónoma", "Provincia"],
                 values = ["Puntos", "DiferenciaGoles"],
                 aggfunc= "mean")
```

IMMUNE CODING INSTITUTE

Exercise 43

- Create a pivot table comparing the “Zona” and the autonomous community, where the average points are shown, using a fill value of 0 (fill_value)

Zona	Champions	Normal	Descenso
Comunidad Autónoma			
Andalucia	57	29.00	19
Asturias	0	21.00	0
Canarias	0	35.00	0
Cataluña	63	40.00	0
Comunidad Valenciana	0	39.00	0
Galicia	0	32.50	0
Madrid	60	26.00	0
Navarra	0	0.00	11
Pais Vasco	0	43.25	0

Exercise 43 - Solution

```
# Crea una pivot table comparando Comunidades autónomas y Zonas
# Donde se muestren la media de puntos
# Utiliza 0 como valor de relleno (fill_value)

liga.pivot_table(index = "Comunidad Autónoma",
                 columns = "Zona",
                 values = "Puntos",
                 aggfunc = "mean",
                 fill_value = 0
)
```

THANKS FOR YOUR ATTENTION

Daniel Villanueva Jiménez
daniel.villanueva@immune.institute
@dvillaj