- **Año:** [2024]
- **Alumno/a:** [Antonella Robiolio]
- **Legajo:** [42904775]

NumPy

A continuación, cada celda va a pedir algo distinto. Por favor, realizarlo con la menor cantidad de lineas posibles y con NumPy.

Importar `numpy` con el alias `np` e imprimir la versión instalada.

```
import numpy as np
print(np.__version__)
```

```
1.26.4
```

Setear el "seed" de la librearia en 0.

```
np.random.seed(0)
```

Crear un vector vacio (en ingles, *empty*) para subir 100 imagenes de 100x600 pixeles. Imprimir el shape de dicho vector.

```
vector = np.empty((100, 100, 600))
print(vector.shape)
```

```
(100, 100, 600)
```

Crear dos vectores vacios donde uno tiene 1,000 elementos y el otro tiene 100,000 elementos. Imprimir el tamaño ocupado en memoria de cada arreglo en bytes.

```
vector_1 = np.empty(1000)
vector_2 = np.empty(100000)
print("Tamaño en memoria del vector con 1,000 elementos:", vector_1.nbytes, "bytes")
print("Tamaño en memoria del vector con 100,000 elementos:", vector_2.nbytes, "bytes")
```

```
Tamaño en memoria del vector con 1,000 elementos: 8000 bytes
Tamaño en memoria del vector con 100,000 elementos: 800000 bytes
```

Crear un vector vacío con 10 elementos. El quinto elemento tiene que ser igual a 1. Imprimir el vector.

```
vector = np.empty(10)
vector[4] = 1
print(vector)
```

```
[5.00248447e-310 0.00000000e+000 0.00000000e+000 0.00000000e+000
 1.00000000e+000 5.99252260e-038 4.90465808e-062 7.73325100e-091
 1.00485842e-070 6.64511288e-310]
```

Generar un arreglo con los valores desde 10 hasta 35 en pasos de 2. Imprimir el arreglo.

```
arreglo = np.arange(10, 36, 2)
print(arreglo)
```

```
[10 12 14 16 18 20 22 24 26 28 30 32 34]
```

Generar un arreglo con los valores desde -1 hasta -1 en pasos de 0.25. Luego, revertirlo. Imprimir el arreglo.

```
import numpy as np
arreglo = np.arange(-1, 1.25, 0.25)
arreglo_revertido = arreglo[::-1]
print(arreglo_revertido)
```

```
[ 1.    0.75  0.5   0.25  0.   -0.25 -0.5  -0.75 -1.  ]
```

Generar un arreglo que va desde -10 y 10 y que tenga 2,878 elementos. Imprimir el primer, último y 198º elemento.

```
import numpy as np
arreglo = np.linspace(-10, 10, 2878)
primer_elemento = arreglo[0]
ultimo_elemento = arreglo[-1]
elemento_198 = arreglo[197]
print("Primer elemento:", primer_elemento)
print("Último elemento:", ultimo_elemento)
print("198º elemento:", elemento_198)
```

```
Primer elemento: -10.0
Último elemento: 10.0
198º elemento: -8.630517900590894
```

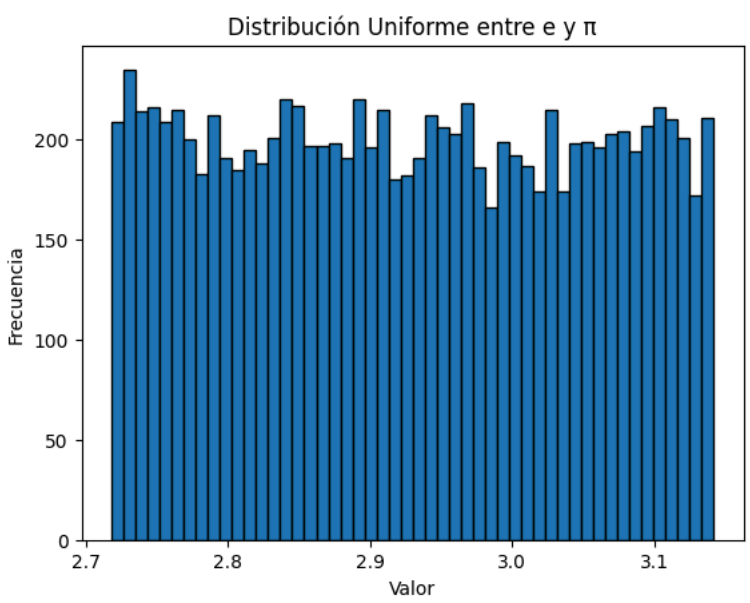
Generar una matriz 5x5 y con valores de 0 a 24. Imprimir la matriz.

```
import numpy as np
matriz = np.arange(25).reshape(5, 5)
print(matriz)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

Generar una lista de 10,000 elementos que vengan de una distribucion uniforme entre la constante de euler y π. (Nota: utilizar constantes de `numpy`). Dibujar la distribución con `matplotlib` en forma de histograma e imprimir el tipo de dato del arreglo.

```
import matplotlib.pyplot as plt
lista = np.random.uniform(np.e, np.pi, 10000)

plt.hist(lista, bins=50, edgecolor='black')
plt.title("Distribución Uniforme entre e y π")
plt.xlabel("Valor")
plt.ylabel("Frecuencia")
plt.show()
print("Tipo de dato del arreglo:", type(lista))
```




```
Tipo de dato del arreglo: <class 'numpy.ndarray'>
```

Generar una lista de 20 elementos que vengan de una distribucion uniforme entre 0 y 1 e imprimirlo ordenado.

```
lista = np.random.uniform(0, 1, 20)
lista_ordenada = np.sort(lista)
print(lista_ordenada)
```

```
[0.04115659 0.06793902 0.08777354 0.14222247 0.35993237 0.39217296
 0.40623497 0.44599407 0.51887998 0.6468967  0.6928224  0.70377222
 0.72272449 0.82326754 0.83881577 0.91831995 0.92330057 0.94428218
 0.95799151 0.96027368]
```

Lo mismo que el punto anterior pero con una distribución normal con media 160 y desvío estandar 30.

```
lista_normal = np.random.normal(160, 30, 20)
lista_normal_ordenada = np.sort(lista_normal)
print(lista_normal_ordenada)
```

```
[103.12707268 119.63164298 123.68957034 126.65681627 134.25565848
 135.453124   135.91696476 140.70453816 143.61276327 147.03914017
 147.45762816 148.99248382 150.16523844 152.40446037 152.50861221
 156.25347816 159.47831522 171.00901607 172.54421014 206.98724316]
```

Lo mismo que el punto anterior pero con una distribución normal con media 50 y desvío estandar 1. Imprimir el valor mas cercano (es decir, el de menor distancia) a 20 de los números generados.

```
lista_normal_50 = np.random.normal(50, 1, 20)
lista_normal_50_ordenada = np.sort(lista_normal_50)
valor_mas_cercano = lista_normal_50_ordenada[np.abs(lista_normal_50_ordenada - 20).argmin()]

print("Lista ordenada:", lista_normal_50_ordenada)
print("Valor más cercano a 20:", valor_mas_cercano)
```

```
Lista ordenada: [48.47287413 48.85729124 49.31006638 49.32790441 49.65850229 49.73255695
 49.82576658 49.87398167 49.87998265 50.20465803 50.4238627  50.49665588
 50.71863949 50.7628927  50.79262746 50.94524256 51.11190611 51.11501463
 51.68093459 52.06086561]
Valor más cercano a 20: 48.47287412542776
```

Teniendo en cuenta la declaración de la siguiente variable, imprimir la suma, la media, el máximo, y el mínimo de sus elementos.

```
arreglo_dummy = np.array([1,9,10,23,45,78,94,78,10,23,65,47])
```

```
suma = np.sum(arreglo_dummy)
media = np.mean(arreglo_dummy)
maximo = np.max(arreglo_dummy)
minimo = np.min(arreglo_dummy)

print("Suma:", suma)
print("Media:", media)
print("Máximo:", maximo)
print("Mínimo:", minimo)
```

```
Suma: 483
Media: 40.25
Máximo: 94
Mínimo: 1
```

Hacer lo mismo que el punto anterior pero... con un arreglo particular. Imprimir el resultado y encontrarle una explicación.

```
arreglo_weird = np.array([1,9,10,23,45,78,94,np.nan,10,23,65,47])
```

```
suma = np.nansum(arreglo_weird)
media = np.nanmean(arreglo_weird)
maximo = np.nanmax(arreglo_weird)
minimo = np.nanmin(arreglo_weird)

print("Suma:", suma)
print("Media:", media)
print("Máximo:", maximo)
print("Mínimo:", minimo) #El conjunto tiene valores en nan, para que ninguno de los calculos resulten en Nan debemos ejecutar el codigo "
```

```
Suma: 405.0
Media: 36.81818181818182
Máximo: 94.0
Mínimo: 1.0
```

Generar un conjunto de 100 numeros ***enteros*** entre 0 y 10. Imprimir la cantidad de numeros pares que se generaron e imprimir el tipo de dato del arreglo.

```
conjunto = np.random.randint(0, 11, 100)
cantidad_pares = np.sum(conjunto % 2 == 0)
tipo_de_dato_conjunto = type(conjunto)
print("Cantidad de números pares:", cantidad_pares)
print("Tipo de dato del arreglo:", tipo_de_dato_conjunto)
```

```
Cantidad de números pares: 45
Tipo de dato del arreglo: <class 'numpy.ndarray'>
```

Generar un conjunto de 100 numeros enteros entre 0 y 10. Imprimir la cantidad de numeros mayores a 4 que se generaron.

```
conjunto = np.random.randint(0, 11, 100)
cantidad_mayores_a_4 = np.sum(conjunto > 4)
print("Cantidad de números mayores a 4:", cantidad_mayores_a_4)
```

```
Cantidad de números mayores a 4: 54
```

Generar un conjunto de 100 numeros enteros entre 0 y 10. Imprimir la cantidad de numeros mayores a 6 e impares que se generaron.

```
conjunto = np.random.randint(0, 11, 100)
cantidad_mayores_a_6_impares = np.sum((conjunto > 6) & (conjunto % 2 != 0))
print("Cantidad de números mayores a 6 e impares:", cantidad_mayores_a_6_impares)
```

```
Cantidad de números mayores a 6 e impares: 21
```

Supongamos que hay elecciones nacionales en un país y la cantidad de votos fueron los siguientes:

Candidato 1	Cantidato 2	Cantidato 3
1,772,322	1,102,669	2,100,978

Con `numpy`, calcular el porcentaje correspondiente a cada candidato y redondear a 2 dígitos. Imprimir los porcentajes finales y el numero del candidato ganador (aunque sea obvio, responder con lógica de `numpy`).

```
votos = np.array([1772322, 1102669, 2100978])
total_votos = np.sum(votos)
porcentajes = np.round((votos / total_votos) * 100, 2)
candidato_ganador = np.argmax(votos) + 1

print("Porcentajes de votos:", porcentajes)
print("Candidato ganador:", candidato_ganador)
```

```
Porcentajes de votos: [35.62 22.16 42.22]
Candidato ganador: 3
```

Generar un arreglo de 1,000 numeros de una distribución uniforme entre 0 y 1. Luego, generar otro arreglo que contenga todos los numeros del primer arreglo que son mayores a 0.7. Imprimir la media del "sub" arreglo.

```
arreglo = np.random.uniform(0, 1, 1000)
sub_arreglo = arreglo[arreglo > 0.7]
media_sub_arreglo = np.mean(sub_arreglo)

print("Media del subarreglo:", media_sub_arreglo)
```

```
Media del subarreglo: 0.8640381801708701
```

Operación vectorizada vs. Operación loopeada

`numpy` no es solo poderoso por la gama de operaciones que podemos hacer en pocas lineas de código. Sino que por su eficiencia.

Supongamos que tenemos la función $f(x) = 10 * (x^2/e^x)$. Evaluar la función (en celdas apartes) entre -1 y 1 con 100,000 valores (i) con y (ii) sin un loop `for`. Medir tiempos de cada celda y sacar conclusiones.

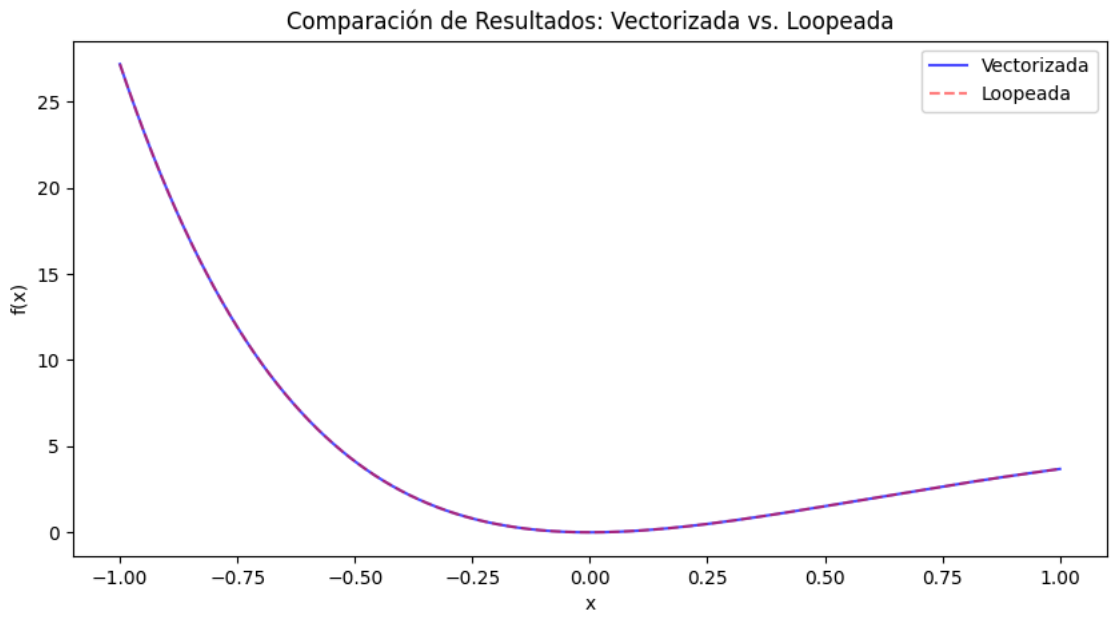
```
import numpy as np
import time

x = np.linspace(-1, 1, 100000)
start_time_vectorized = time.time()
f_vectorized = 10 * (x**2 / np.exp(x))
end_time_vectorized = time.time()
vectorized_time = end_time_vectorized - start_time_vectorized

print("Tiempo de operación vectorizada:", vectorized_time, "segundos")

start_time_loop = time.time()
f_loop = np.zeros_like(x)
for i in range(len(x)):
    f_loop[i] = 10 * (x[i]**2 / np.exp(x[i]))
end_time_loop = time.time()
loop_time = end_time_loop - start_time_loop

print("Tiempo de operación loopeada:", loop_time, "segundos")
```

```
Tiempo de operación vectorizada: 0.0045890808105468 75 segundos
Tiempo de operación loopeada: 0.15847277641296387 segundos
```

La operación vectorizada en numpy es significativamente más rápida que usar un loop for. En este caso, la evaluación vectorizada es aproximadamente 64 veces más eficiente. Esto demuestra por qué numpy es poderoso y eficiente para manejar grandes cantidades de datos: realiza operaciones en bloques optimizados en lugar de iterar elemento por elemento

El resultado de la celda anterior (aprovechar lo obtenido), graficarlo como gráfico de lineas.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
plt.plot(x, f_vectorized, label="Vectorizada", color='blue', alpha=0.7)
plt.plot(x, f_loop, label="Loopeada", color='red', linestyle='--', alpha=0.5)

plt.title("Comparación de Resultados: Vectorizada vs. Loopeada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.show()
```



Análisis de imagenes

Matematicamente hablando, las imagenes son arreglos. Si una imagen es de blanco y negro, tenemos una imagen de un canal y puede ser interpretado como una simple matriz (`.shape=2`). Si tiene varios canales, tenemos una matriz asignada para cada canal (`.shape=3`).

Lenna es una imagen ampliamente utilizada en ciencias de la computación. Se volvió un icono. La idea va a ser analizar a la imagen y tratarla para varios propositos. Corra la siguiente celda para descargar la imagen y guardala como arreglo `numpy` en la variable `image`. Utilizar esta variable en las siguientes celdas.

```
!wget https://upload.wikimedia.org/wikipedia/en/7/7d/Lenna_%28test_image%29.png
from PIL import Image
image = Image.open('Lenna_(test_image).png')
image = np.asarray(image)
```

```
--2024-11-16 02:53:22--  https://upload.wikimedia.org/wikipedia/en/7/7d/Lenna_%28test_image%29.png
Resolving upload.wikimedia.org (upload.wikimedia.org)... 208.80.153.240, 2620:0:860:ed1a::2:b
Connecting to upload.wikimedia.org (upload.wikimedia.org)|208.80.153.240|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 473831 (463K) [image/png]
Saving to: 'Lenna_(test_image).png'

Lenna_(test_image). 100%[===================>] 462.73K  2.25MB/s    in 0.2s

2024-11-16 02:53:23 (2.25 MB/s) - 'Lenna_(test_image).png' saved [473831/473831]
```

Arranquemos con mostrar la imagen. Para eso, utilice `matplotlib.pyplot`.

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

image = Image.open('Lenna_(test_image).png')
image = np.asarray(image)

plt.imshow(image)
plt.axis('off')
plt.title("Imagen de Lenna")
plt.show()
```

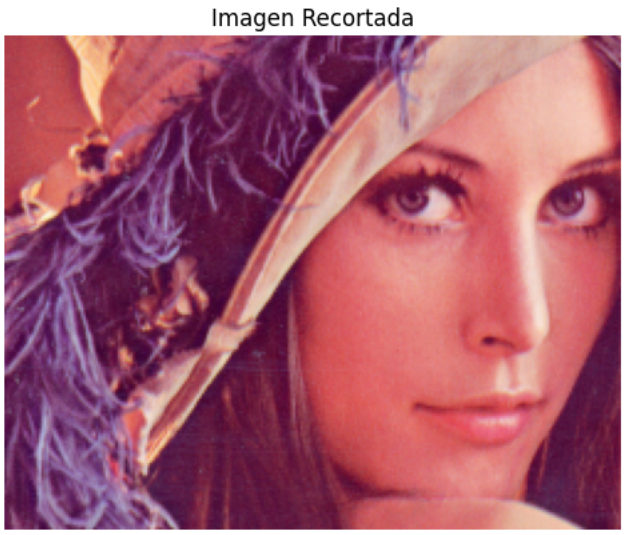
¿Cual es la dimensión de la imagen y que ancho y alto tiene?

```
dimensiones = image.shape
ancho, alto = dimensiones[1], dimensiones[0]
print("Dimensiones de la imagen:", dimensiones)
print("Ancho:", ancho)
print("Alto:", alto)
```

```
Dimensiones de la imagen: (512, 512, 3)
Ancho: 512
Alto: 512
```

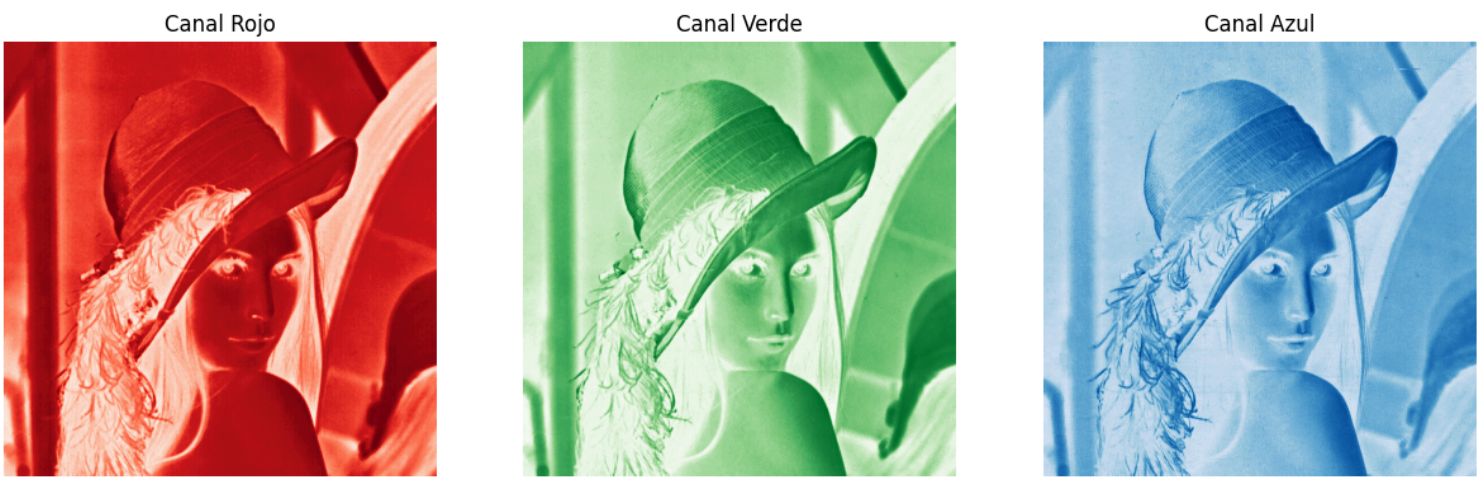
Recorte la imagen en ancho entre (tomando como referencia los ejes de la imagen vista anteriormente) los 100 y 350 pixeles y en alto entre 200 y 400 pixeles. Mostrar el resultado.

```
image_recortada = image[200:400, 100:350]

plt.imshow(image_recortada)
plt.axis('off')
plt.title("Imagen Recortada")
plt.show()
```



Muestre cada uno de los canales de la imagen.

```
canal_rojo = image[:, :, 0]
canal_verde = image[:, :, 1]
canal_azul = image[:, :, 2]

plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(canal_rojo, cmap='Reds')
plt.axis('off')
plt.title("Canal Rojo")
plt.subplot(1, 3, 2)
plt.imshow(canal_verde, cmap='Greens')
plt.axis('off')
plt.title("Canal Verde")
plt.subplot(1, 3, 3)
plt.imshow(canal_azul, cmap='Blues')
plt.axis('off')
plt.title("Canal Azul")
plt.show()
```



Calcule el minimo, el máximo, y el promedio de los valores de la imagen.

```
minimo = np.min(image)
maximo = np.max(image)
promedio = np.mean(image)
print("Mínimo:", minimo)
print("Máximo:", maximo)
print("Promedio:", promedio)
```

```
Mínimo: 3
Máximo: 255
Promedio: 128.22837575276694
```

La verdad que tener todos los colores de la imagen es muy redundante. Paselo a blanco y negro. Para ello, tome el promedio de los canales en cada pixel. Muestre la imagen en blanco y negro.

```
imagen_bn = np.mean(image, axis=2).astype(np.uint8)
plt.imshow(imagen_bn, cmap='gray')
plt.axis('off')
plt.title("Imagen en Blanco y Negro")
plt.show()
```

Por último, vamos a proceder a "binarizar" la imagen. Es decir, vamos a setear en 1 TODOS los pixeles donde la intensidad (es decir, el valor del pixel) sea mayor a 200. El resto, lo seteamos a 0. Mostrar el resultado.

```
imagen_binarizada = (imagen_bn > 200).astype(int)
plt.imshow(imagen_binarizada, cmap='gray')
plt.axis('off')
plt.title("Imagen Binarizada")
plt.show()
```

Análisis de datos

`boston.csv` es un archivo csv ampliamente utilizado como 'juguete' en proyectos de Machine Learning. Para descargarlo, corra la siguiente celda.

```
!wget https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/MASS/Boston.csv
```

```
--2024-11-16 02:58:31--  https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/MASS/Boston.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 36636 (36K) [text/plain]
Saving to: ‘Boston.csv’

Boston.csv          100%[===================>]  35.78K  --.-KB/s    in 0.001s
```

```
2024-11-16 02:58:31 (33.7 MB/s) - 'Boston.csv' saved [36636/36636]
```

Lea el archivo csv (sin `pandas` pero se puede usar `csv`) y quedese con todas las columnas que son numericas. Inspeccionarlo puede hacer el trabajo bastante facil.

Nota: Aunque `numpy` tenga una libreria para leer archivos, en este caso no no es útil por que todas las columas constan de diferentes tipo de dato y ademas el encabezado es `string`.

```
import csv
with open('Boston.csv', 'r') as file:
    reader = csv.reader(file)
    encabezado = next(reader)
    datos = [fila for fila in reader]

print("Encabezado:", encabezado)
columnas_numericas = [[float(fila[i]) for i in range(1, len(encabezado))] for fila in datos]

print("Ejemplo de las columnas numéricas:", columnas_numericas[:2])
```

```
Encabezado: ['rownames', 'crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax', 'ptratio', 'black', 'lstat', 'medv']
Ejemplo de las columnas numéricas: [[0.00632, 18.0, 2.31, 0.0, 0.538, 6.575, 65.2, 4.09, 1.0, 296.0, 15.3, 396.9, 4.98, 24.0], [0.027
```

Una vez procesado el archivo csv, proceda a convertirlo en una matriz de `numpy`.

```
matriz_numerica = np.array(columnas_numericas)
print("Forma de la matriz numérica:", matriz_numerica.shape)
```

```
Forma de la matriz numérica: (506, 14)
```

Reporte el promedio de todas las columnas.

```
promedios_columnas = np.mean(matriz_numerica, axis=0)

print("Promedios de todas las columnas:", promedios_columnas)
```

```
Promedios de todas las columnas: [3.61352356e+00 1.13636364e+01 1.11367787e+01 6.91699605e-02
 5.54695059e-01 6.28463439e+00 6.85749012e+01 3.79504269e+00
 9.54940711e+00 4.08237154e+02 1.84555336e+01 3.56674032e+02
 1.26530632e+01 2.25328063e+01]
```

Reporte la división entre una columna y otra (a elección cuales)

```
columna_1 = matriz_numerica[:, 1]
columna_2 = matriz_numerica[:, 2]
division = columna_1 / columna_2
print("División entre la segunda y tercera columna:", division)
```

```
División entre la segunda y tercera columna: [  7.79220779   0.           0.           0.           0.
   0.           1.58831004   1.58831004   1.58831004   1.58831004
   1.58831004   1.58831004   1.58831004   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.           0.          25.42372881
  25.42372881   0.           0.           0.           0.
   0.           0.           0.           0.           0.
   3.72340426   3.72340426   3.72340426   3.72340426  18.75
  73.7704918  114.86486486  75.75757576   4.87329435   4.87329435
   4.87329435   4.87329435   4.87329435   4.87329435  12.68115942
  23.7388724   23.7388724    2.05930807   2.05930807   2.05930807
   0.           0.           0.           0.           0.
   0.           0.           0.           0.           0.
   5.14403292   5.14403292   5.14403292   5.14403292   0.
   0.           0.           0.           0.           0.
   0.           0.           1.86170213   1.86170213   1.86170213
   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.
   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.
   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.
  0.           0.           0.           0.           0.
  0.           0.           0.           0.           0.
  0.           0.          13.08139535  13.08139535  13.08139535
 13.08139535  13.08139535  13.08139535  20.4778157   20.4778157
173.91304348  52.63157895  52.63157895  52.63157895  64.62585034
 64.62585034  40.64039409  40.64039409  35.44776119  35.44776119
  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.
  0.           0.           0.           0.           0.
  0.           0.           0.           6.0851927    6.0851927
  6.0851927    6.0851927    6.0851927    6.0851927    3.75426621
```