Python para Análisis de datos: Introducción

Sesión 5

Miguel Expósito Martín (exposito_m@cantabria.es)

9 Abril 2019

Python Data Analysis Library

numpy

introducción a pandas

series

dataframes

pandas desde excel

pyjstat

ejercicios

Python Data Analysis Library

numpy

hoja de ruta

- numpy
- pandas

numpy

es la biblioteca principal para computación científica en Python

- su principal estructura de datos es el array multidimensional
- recuerda a Matlab
- existe desde el año 2000
- es rápido, eficiente y tiene baja huella en memoria

numpy

- almacena datos en bloques contiguos de memoria
- contiene funciones matemáticas sobre arrays sin necesidad de bucles
- implementa funcionalidades de álgebra lineal
- ofrece una API en `C` para conectar `numpy` con `C` o `Fortran`

prueba de rendimiento

```
import numpy as np
my_array = np.arange(1000000)
my_list = list(range(1000000))

%time for _ in range(10): my_array_2 = my_arr * 2
%time for _ in range(10): my_list_2 = [x * 2 for x in my_list]
```

tipos de datos precisos

tipo	descripción
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)

tipos de datos precisos

tipo	descripción
float_	Shorthand for float64.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa

valores especiales

`nan`, `inf`

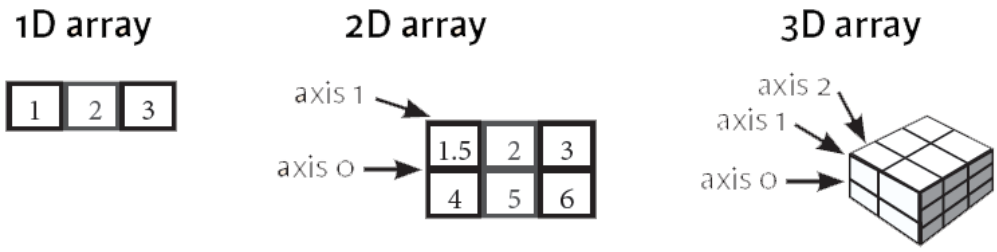
¡ojo! no usar comparadores de igualdad

usar `np.isnan()` o `np.isinf()`

ndarrays

- contenedor flexible y rápido de datasets numéricos
- permite realizar operaciones matemáticas sobre bloques
- `rank`: número de dimensiones
- `shape`: tupla de enteros con el tamaño de cada dimensión
- datos numéricos y homogéneos (**mismo tipo**)

ndarrays



ndarrays

```
data = np.random.randn(2, 3)
data
data * 10
data + data
```

crear ndarrays

- forma más fácil: función `array()`
- acepta cualquier objeto de tipo secuencia

```
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
arr1
```

crear ndarrays

una lista de listas de la misma longitud se convierte a array multidimensional

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
arr2
arr2.ndim # dimension
arr2.shape # forma
arr2.dtype # tipo de datos
```

crear ndarrays

- otras funciones: `zeros`, `ones`, `empty`, `full`, `arange`
- para varias dimensiones, pasar una tupla como argumento
- aceptan `dtype` como argumento

```
np.zeros(10)
np.zeros((3, 6))
np.ones(7)
np.ones(7, dtype=bool)
np.empty((2,3,2))
np.full((2,2), np.inf)
np.arange(15)
```

crear ndarrays

valores aleatorios: `np.random.randn()`

```
arr = np.random.randn(4)
arr

matrix = np.random.randn(4,4)
matrix
```

tipos en ndarrays

`dtype` es un objeto especial que contiene metadatos **importante**: `astype` para convertir tipos

```
arr1 = np.array([1, 2, 3], dtype=np.float64)
arr2 = np.array([1, 2, 3], dtype=np.int32)

arr1.dtype
arr2.dtype

float_arr = arr2.astype(np.float64)
float_arr.dtype
```

¿qué pasa al convertir de `float` a `int`?

ejercicios

- crear un array 1D de números del 0 al 9
- crear un array booleano de 3x3 con todos los valores a True

operaciones vectoriales

- sin bucles for
- entre arrays de la misma longitud, elemento a elemento

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

operaciones matriciales

```
A = np.array( [[1,1], [0,1]] )
B = np.array( [[2,0], [3,4]] )

A @ B
A.dot(B)
```

más álgebra lineal

```
from numpy.linalg import inv, qr

X = np.random.randn(5, 5)
mat = X.T.dot(X)
inv(mat)
mat.dot(inv(mat))

q, r = qr(mat)
```

linear algebra

indexado y rebanado básicos

```
arr = np.arange(10)
arr[5]
arr[5:8]
arr[5:8] = 12
1 / arr
```

las rebanadas son vistas del array original; si se modifica una vista, se refleja en el original

¡¡no se copian!!

indexado y rebanado básicos

con dos dimensiones

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
arr2d[2]
arr2d[0][2]

arr2d[:2]
arr2d[:2, 1:]
```

funciones universales

```
arr = np.arange(10)

np.sqrt(arr) # raíz cuadrada
np.exp(arr) # e elevado a

x = np.random.randn(8)
y = np.random.randn(8)

np.maximum(x, y) # máximo elemento a elemento
```

ufuncs

funciones estadísticas

```
arr = np.random.randn(5, 4)

arr.mean()
arr.mean(axis=1)
arr.sum()
```

statistics

lógica condicional

filtrar por condición

```
arr = np.arange(9)
arr[arr > 2] # elementos mayores que dos
arr[arr % 2 == 0] # elementos pares
```

lógica condicional

ej: obtener x si cond es True; si no, obtener y

```
x = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
y = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
cond = np.array([True, False, True, True, False])

result = np.where(cond, xarr, yarr)
result
```

lógica condicional

ejemplo:

Generar una matriz de 4x4 con valores aleatorios. Reemplazar todos los valores positivos con 2 y los valores negativos con -2.

arrays booleanos

`any()`, `all()`

```
bools = np.array([False, False, True, False])
bools.any()
bools.all()
```

¿cómo contar True en un array booleano?

```
arr = np.random.randn(100)
(arr > 0).sum()
```

ejercicio

- extraer todos los números pares de un array
- reemplazar números impares en array por -1

```
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

introducción a pandas

qué es

paquete Python con estructuras de datos flexibles, expresivas y eficientes para trabajar con datos relacionados o etiquetados

- datos tabulares con columnas heterogéneas (XLS, SQL)
- datos ordenados o no ordenados de series temporales
- datos matriciales arbitrarios con etiquetas de fila y columna
- cualquier otra forma de conjuntos de datos estadísticos observacionales

características

- gestión de datos nulos
- tamaño mutable (borrado e inserción de columnas)
- funcionalidades de agregación (`group by`)
- rebanado inteligente basado en etiquetas
- fusión intuitiva de conjuntos de datos
- robustas herramientas de entrada/salida
- funcionalidad para series temporales
- open source desde 2010

series

qué son

objetos de una dimensión similares a un array que contienen una secuencia de valores y un array asociado de etiquetas (índice)

```
import pandas as pd
obj = pd.Series([4, 7, -5, 3])
obj
obj.values
obj.index
```

si no se especifica índice, se crea uno con enteros de 0 a N-1

índices

a veces es deseable crear una serie con un índice etiquetado

```
import pandas as pd
obj2 = pd.Series([4, 7, -5, 3], index=['Enero', 'Febrero', 'Marzo', 'Abril'])
obj2
```

índices

en contraposición a los arrays, pueden usarse las etiquetas para la selección de valores

```
import pandas as pd
obj2['Marzo']
obj2['Abril'] = 10
obj2[['Enero', 'Marzo', 'Abril']]
```

filtrado simple

la relación valor-índice se preserva

```
obj2[obj2 > 0]
```

desde diccionario

la relación diccionario - series es directa

```
my_dict = {'visitantes': 2000, 'visitas': 50000, 'hits': 200000}
obj3 = pd.Series(my_dict)
obj3
```

desde diccionario

se puede cambiar el orden de las claves

```
metrics = ['visitas',  'hits', 'visitantes', 'tpv']
obj4 = pd.Series(my_dict, index=metrics)
```

ejercicio

crear una serie desde lista, array y dict

```
import numpy as np
mylist = list('abcedfghijklmnopqrstuvwxyz')
myarr = np.arange(26)
mydict = dict(zip(mylist, myarr))
```

operaciones desde numpy

se pueden aplicar operaciones sin modificar la estructura

```
np.square(obj3)
```

alineación automática

```
obj3 + obj4
```

ejercicio

obtener los elementos de A que no están en B

usar ~ , `isin`

```
ser1 = pd.Series([1, 2, 3, 4, 5])
ser2 = pd.Series([4, 5, 6, 7, 8])
```

ejercicio

obtener la media y la desviación estándar de una serie

```
s = pd.Series([1,2,3,4,5,6,7,8,9,5,3])
```

dataframes

qué son

objetos que representan una tabla rectangular conteniendo una colección ordenada de columnas, cada una de las cuales puede ser de un tipo de datos distinto

```
data = {'ca': ['Cantabria', 'Cantabria', 'Cantabria', 'Canarias', 'Canarias'],
'year': [2000, 2001, 2002, 2001, 2002],
'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = pd.DataFrame(data) # inicialización con list of dicts
frame.head() # visualizar las 5 primeras filas
```

ejemplo

```
    Trimestre         Sexo  ...                          Variables  value
0        2005  Ambos sexos  ...                          Población    NaN
1        2005  Ambos sexos  ...                            Activos    NaN
2        2005  Ambos sexos  ...                           Ocupados    NaN
3        2005  Ambos sexos  ...                            Parados    NaN
4        2005  Ambos sexos  ...    Parados que buscan primer empleo    NaN
5        2005  Ambos sexos  ...                          Inactivos    NaN
6090  2018 - 2      Mujeres  ...                          Población  253.7
6091  2018 - 2      Mujeres  ...                            Activos  126.7
6092  2018 - 2      Mujeres  ...                           Ocupados  109.5
6093  2018 - 2      Mujeres  ...                            Parados   17.2
6094  2018 - 2      Mujeres  ...    Parados que buscan primer empleo    2.2
6095  2018 - 2      Mujeres  ...                          Inactivos  127.1
```

creación y manejo básico

- se puede especificar el orden de las columnas
- si una columna no existe, se creará con valores nulos
- una columna se puede recuperar como un objeto de tipo series
- también pueden recuperarse filas

```
pd.DataFrame(data, columns=['year', 'state', 'pop'])
frame2 = pd.DataFrame(data, columns=['year', 'ca', 'pop', 'debt'],
                      index=['uno', 'dos', 'tres', 'cuatro', 'cinco'])
frame2
frame2['ca'] # columna
frame2.loc['tres'] # fila
```

lectura desde archivo

`.read_csv()`

acepta:

- URLs
- `sep`: separador (,)
- `delimiter`: delimitador (`None`)
- `header`: cabecera (se infiere)

escritura desde archivo

```
.write_csv()
```

manejo básico

- se puede obtener una lista de columnas
- el atributo `dtypes` indica el tipo de dato de cada columna
- las dimensiones del dataframe se obtienen con `.shape`

```
frame2.columns
frame2.dtypes
frame2.shape
```

manejo básico

la serie devuelta al recuperar una columna es una vista

```
vals = frame['pop']
vals[0] = 0
vals
data
```

usar `copy()` para eliminar el warning

manejo básico

un dataframe se puede transponer

```
frame2.T
```

manejo básico

cálculo de frecuencias

```
frame2.ca.value_counts()
```

manejo básico

cálculo de valores únicos

```
frame2.ca.nunique()
```

manejo básico

funciones estadísticas

```
frame2['pop'].mean()
frame2['pop'].sum()
frame2['pop'].max()
frame2['pop'].min()
frame2['pop'].median()
```

manejo básico

valores nulos: `isnull()`: `NaN`, `None/NaN`, `NaT`

```
array = np.array([[1, np.nan, 3], [4, 5, np.nan]])
pd.isnull(array)
```

inspección

```
frame2.describe() # información estadística
frame2.info() # información sobre la estructura de datos
frame2.tail()
```

selección de columnas

un dataframe puede verse como un conjunto de series que comparten un índice (las cabeceras de las columnas)

```
frame2[['year', 'pop']] # selección columnas
frame2[frame2['pop'] > 2] # filtrado con condición
frame2[(frame2['pop'] > 2) & (frame2['year'] != 2002)] # filtrado con condición
frame2.iloc[:, 0:2] # todas las filas, columnas 0 a 2 no incluida
frame2.iloc[:, :-1] # todas las filas, todas las columnas excepto la última
```

indexado

```
frame4 = frame2.set_index('year')
frame4
```

selección de filas

```
frame2.iloc[3] # posición
frame2.iloc[[3,4]] # posición
frame4.loc[2001] # etiqueta
```

reseteo de índice

```
frame4.reset_index(inplace=True)
```

resumen de selección

- `loc` para indexado basado en etiquetas
- `iloc` para indexado posicional

aunque hay otras formas...

para modificar el propio dataframe, usar el atributo `inplace=True`

ejercicio inspección

importar datos desde:

```
DATA_URL = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.us
usuarios = pd.read_csv(DATA_URL, sep='|')
```

asignar a una variable llamada `users` y utilizar `user_id` como índice

ejercicio inspección

- ver las 25 primeras filas
- ver las 10 últimas filas
- obtener el número de observaciones en el dataset
- obtener el número de columnas en el dataset
- mostrar los nombres de las columnas
- mostrar el índice del dataset

ejercicio inspección

- mostrar los tipos de datos de cada columna
- mostrar sólo la columna de ocupación
- mostrar cuántas ocupaciones diferentes hay en el dataset
- mostrar la ocupación más frecuente
- resumir el dataframe
- calcular la edad media de los usuarios

joins: merge

inner join

```
left_frame = pd.DataFrame({'key': range(5),
                           'left_value': ['a', 'b', 'c', 'd', 'e']})
right_frame = pd.DataFrame({'key': range(2, 7),
                           'right_value': ['f', 'g', 'h', 'i', 'j']})
left_frame
right_frame

pd.merge(left_frame, right_frame, on='key', how='inner')
```

joins: merge

left, right, full outer join

```
pd.merge(left_frame, right_frame, on='key', how='left')
pd.merge(left_frame, right_frame, on='key', how='right')
pd.merge(left_frame, right_frame, on='key', how='outer')
```

combinación: concat

```
pd.concat([left_frame, right_frame]) # vertical
pd.concat([left_frame, right_frame], axis=1) # horizontal
```

añadir filas al final: append

los dos dataframes deben tener las mismas columnas

```
left_frame.append(left_frame)
```

groupby

```
ANIMALS_URL='https://gist.githubusercontent.com/predicador37/bccac851999eaf10b6
animals = pd.read_csv(ANIMALS_URL)
```

groupby

¿cuál es el peso medio por animal?

```
# Agrupar por cada categoría de animal
animal_groups = animals.groupby("animal")
# Aplicar la media aritmética a la columna de peso
animal_groups['weight'].mean()
```

agg

permite agregar por varias funciones según un eje

```
animal_groups['weight'].agg(['mean', 'median'])
```

ejemplo

análisis de dataset de netflix

```
NETFLIX_URL='https://gist.githubusercontent.com/predicador37/d081821c1538cc6c26
df = pd.read_csv(NETFLIX_URL)
```

ejemplo: limpieza

- dropna(): elimina filas con valores nulos
- drop_duplicates(): elimina filas duplicadas

```
df.dropna(inplace=True)
df.drop_duplicates(inplace=True)
```

ejemplo: inspección

```
df.describe()
```

ejemplo: split

se dividen los datos en grupos, donde cada grupo es el conjunto de películas estrenadas en un año determinado

```
df_by_year = df.groupby('release year')
type(df_by_year)
```

ejemplo: apply

aplicar `.describe()` a cada grupo

```
df_by_year.describe().head()
```

ejemplo: combine

obtener la media o la mediana de la puntuación de usuario por año

```
df_med_by_year = df_by_year.median()
df_med_by_year.head()
df_med_by_year['user rating score']
```

ejercicio: agrupaciones

importar datos desde:

```
DATA_URL = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.us
```

¡inspeccionar separadores!

ejercicio: agrupaciones

- inspeccionar dataset
- mostrar edad media por ocupación
- para cada ocupación, calcular las edades mímina y máxima
- para cada combinación de ocupación y sexo, calcular la edad media

aplicar funciones a filas, columnas y elementos

- `.apply()`: función a arrays 1D a cada fila o columna
- `.applymap()`: elemento a elemento en dataframe
- `.map()`: elemento a elemento en series

funciones anónimas

son funciones definidas sin nombre

```
lambda arguments: expression
```

```
def square_root(x):
    return math.sqrt(x)

square_root = lambda x: math.sqrt(x)
```

se usan mucho para aplicar funciones a dataframes

ejemplo: aplicar operaciones en dataframes

```
import pandas as pd
import numpy as np

data = {'name': ['Pedro', 'Paco', 'Lorena', 'Juan', 'Patxi'],
        'year': [2012, 2012, 2013, 2014, 2014],
        'reports': [4, 24, 31, 2, 3],
        'coverage': [25, 94, 57, 62, 70]}
df = pd.DataFrame(data, index = ['Santander', 'Torrelavega', 'Laredo', 'Camargo
df
```

ejemplo

crear función que pase a mayúsculas usando lambda

```
capitalizer = lambda x: x.upper()
```

ejemplo

aplicar la función a la columna `name`

```
df['name'].apply(capitalizer)
```

ejemplo

equivalente al anterior

```
df['name'].apply(lambda x: x.upper())
```

ejemplo

mapear la función sobre cada elemento de la series `name`

```
df['name'].map(capitalizer)
```

ejemplo

aplicar una raiz cuadrada a todas las celdas del dataframe

```
df = df.drop('name', axis=1) # eliminar la columna de texto
df.applymap(np.sqrt)
```

ejemplo

aplicar una función que multiplica por 100 los números

```
def times100(x):
    if type(x) is str:
        return x
    elif x:
        return 100 * x
    else:
        return
df.applymap(times100)
```

pandas desde excel

cargar fichero de ejemplo

```
import pandas as pd
import numpy as np
DATA_URL = 'https://gist.githubusercontent.com/predicador37/24e8e4ee465956aa923
df = pd.read_csv(DATA_URL)
df.head()
```

añadir columna de totales

```
df["total"] = df["Jan"] + df["Feb"] + df["Mar"]
df.head()
```

análisis básico a nivel de columna

- ¿total, media, mínimo, máximo del mes de enero?

```
df["Jan"].sum()
df["Jan"].mean()
df["Jan"].min()
df["Jan"].max()
```

añadir subtotales por mes y total general

```
sum_row = df[["Jan","Feb","Mar","total"]].sum() # suma para cada columna
df_sum = pd.DataFrame(data=sum_row).T # crear un nuevo dataframe transpuesto
df_sum
df_sum = df_sum.reindex(columns=df.columns) # añadir las columnas que faltan
df_sum
df_final = df.append(df_sum, ignore_index=True)
df_final.tail()
```

añadir subtotales por estado

```
from money import Money

df_sub = df_final[["state","Jan","Feb","Mar","total"]].groupby('state').sum()
formatted_df = df_sub.applymap(lambda x: Money(x, currency='EUR'))
formatted_df

sum_row=df_sub[["Jan","Feb","Mar","total"]].sum() # calcular subtotales como en
sum_row
df_sub_sum=pd.DataFrame(data=sum_row).T
def money(x):
    return Money(x, currency='EUR')
df_sub_sum = df_sub_sum.applymap(money)
df_sub_sum
```

añadir subtotales por estado

```
final_table = formatted_df.append(df_sub_sum)
final_table

final_table = final_table.rename(index={0:"Total"})
final_table
```

otro ejemplo

lectura del fichero

```
import pandas as pd
import numpy as np
DATA_URL = 'https://gist.githubusercontent.com/predicador37/29a4c89cc652d3b201e
df = pd.read_csv(DATA_URL)
df.head()
df.dtypes
```

otro ejemplo

conversión de objeto a fecha

```
df['date'] = pd.to_datetime(df['date'])
df.head()
df.dtypes
```

otro ejemplo

filtrar por número de cuenta 307599

```
df[df['account number'] == 307599].head()
```

otro ejemplo

filtrar por cantidad mayor a 22

```
df[df['quantity'] > 22].head()
```

otro ejemplo

filtrar por referencia (sku) que empiece por B1

```
df[df['sku'].map(lambda x: x.startswith('B1'))].head()
```

otro ejemplo

combinar los dos filtros anteriores

```
df[(df['quantity'] > 22) & (df['sku'].map(lambda x: x.startswith('B1')))].head(
```

otro ejemplo

encontrar todos los registros que incluyen dos números de cuenta específicos

```
df[df['account number'].isin([714466, 218895])].head()
```

otro ejemplo

query: requiere instalación previa de `numexpr` recuperar lista de clientes por nombre

```
df.query('name == ["Kulas Inc","Barton LLC"]').head()
```

otro ejemplo

trabajar con fechas

```
df = df.sort('date')
df.head()
df[df['date'] >='20140905'].head() # por fecha exacta
df[df['date'] >='2014-03'].head() # por mes
df[df['date'] >= 'Oct-2014'].head() # por mes en otro formato
df[df['date'] >= '10-10-2014'].head() # por fecha en otro formato
```

otro ejemplo

series temporales: utilizar la fecha como índice con `set_index()`

```
df2 = df.set_index(['date'])
df2.head()
df2["20140101":"20140201"].head() # filtrado por rango de fechas
df2["2014"].head() # filtrado por año
```

otro ejemplo

identificar referencias que contienen un determinado valor

```
df[df['sku'].str.contains('B1')].head()
df[(df['sku'].str.contains('B1-531')) & (df['quantity']>40)].sort_values(by=['q
```

otro ejemplo

obtener una lista de valores únicos

```
df['name'].unique()
```

pyjstat

qué es

paquete Python para convertir JSON-stat a pandas dataframe

https://json-stat.org/

se puede visualizar en:

- http://jsonviewer.stack.hu/
- https://json-stat.org/format/browser/
- https://json-stat.org/format/viewer/
- http://json-stat.com/explorer/

ejemplo: lectura de archivo

```
from pyjstat import pyjstat
DATA_URL = 'http://www.icane.es/data/api/active-population-aged-16-more-gender-
dataset = pyjstat.Dataset.read(DATA_URL) # lee dataset de json-stat
df = dataset.write('dataframe') # genera un dataframe
print(df)
```

ejemplo: consulta

```
query = [{'sexo': 'hombres'}, {'trimestre': '2016-1'}, {'grupo-de-edad': 'total
dataset.get_value(query)
```

obtener el total de activos mujeres para todos los grupos de edad en el segundo trimestre de 2017

ejercicios

ejercicio: dataset ICANE

importar y explorar el archivo JSON-stat desde la siguiente URL:

```
DATA_URL = 'http://www.icane.es/data/api/municipal-register-annual-review-munic
```

- crear un índice temporal
- filtrar los últimos 5 años de tu municipio de residencia en una variable `df_muni`
- obtener una serie con la población para ese filtro

ejercicio: dataset ICANE

consultar en el dataset original la variación interanual de la población total de Cantabria en 2017.

¡ojo! inspeccionar dataset en http://jsonviewer.stack.hu/ para identificar nombres de dimensiones y variables

ejercicio: nombres de niños en US

crear un dataframe desde la siguiente URL y asignárselo a la variable `baby_names`

```
DATA_URL = 'https://raw.githubusercontent.com/guipsamora/pandas_exercises/maste
```

ejercicio: nombres de niños en US

- visualizar los primeros diez registros
- deshacerse de las columnas `Unnamed: 0` e `Id` (se puede usar `.drop()`)
- ¿hay más nombres de niñas o de niños?
- agrupar por nombre agregando con suma y asignarlo a la variable `names`
- ¿cuántos nombres diferentes hay en el dataset?
- ¿cuál es el nombre más frecuente?
- obtener un resumen con la media, mínimo, máximo, std y y cuartiles