clase3

December 1, 2023

1 Clase 3 - Introducción a Pandas

Pandas funciona partiendo de las funcionalidades que trae la librería *numpy*. ## Pero antes... ¿Qué es Numpy? Cuando queremos hacer trabajos analíticos relacionados al procesamiento de información, lo más probable es que nos encontremos trabajando con información estructurada como tablas. Ahora, ¿cómo está compuesta una tabla? **Para ver una guía mas completa con ejemplos pueden entrar en la página oficial:** Numpy quickstart ### Las tablas son matrices con índices: Debemos pensar a las tablas como una forma de organizar datos, pueden haber otras formas (ya les hable de anidar un diccionario adentro de otro y así sucesivamente). Pero en el caso de las tablas (piensen en el excel); matemáticamente, no son otra cosa que matrices de dos dimensiones a las que se le agrega la propiedad de tener un índice para las filas y un índice para las columnas. ¿A qué me refiero con índices? Bueno, en realidad ya estamos familiarizados con eso; al índice de las columnas es al que llamamos encabezado de la tabla y el índice de las filas simplemente es el número de fila. Como ven, esto no está lejos de nuestra experiencia con planillas de cálculo excel. ### Las matrices se componen de vectores: Para la gente que sabe R, ya estamos llegando a un terreno conocido. Los vectores son una nueva forma de estrucutrar un conjunto de datos ordenados en serie que, si bien son parecidos a una lista, internamente funcionan muy diferente y nos van a permitir hacer un monton de operaciones mucho más potentes cuando estemos trabajando con muchísimos más datos. De hecho, hacer un *for loop* sobre una lista es una operación sensiblemente más lenta que iterar y operar sobre un vector. Entonces una matriz de numpy (y una tabla de pandas también), simplemente esta compuesta de muchos vectores, uno al lado del otro. Simplemente imaginen las columnas de una tabla: cada una sun muchos datos en serie, si las ponemos todas juntas tenemos una matriz. Pasa lo mismo si quieren pensarlas en términos de filas, las ponen todas juntas, unas arriba de las otras y listo, tienen una matriz.

¡Igual tranquilos! No vamos a estar trabajando directamente sobre la idea de matrices y con **numpy**, vamos a usar **pandas** y practicar con las viejas y conocidas tablas. Pero es bueno que sepan un poquito qué es lo que sucede detrás, porque quizás nos encontremos que una función o método de **numpy** sea útil para algún proceso.

1.1 Veamos algunas propiedades de los vectores distintas a las listas

NOTA: vamos a usar vectores y arrays (arreglos) como sinónimos. Si un lic. en ciencias de la computación ve esto, me pega.

[ ]:
```
import numpy as np

arr = np.array([1,2,3,4]) # Así de define un array de numpy
print(type(arr))
```

```
<class 'numpy.ndarray'>
```

1.1.1 ¿Cómo operamos sobre un arreglo?

A diferencia de las listas podemos hacer una operación entre un array y un escalar (escalar, en criollo, es un numero solo en vez de otro vector). El resultado de hacer esto es otro array que reprodujo la operación para cada miembro. ¡Sin for loops! Y mucho más rapido

[ ]:
```
print(arr + 1)
```

```
[2 3 4 5]
```

¿Y que pasa si hacemos una operación entre arrays del mismo tamaño?

[ ]:
```
arr * np.array([2,2,3,3])
```

[ ]:
```
array([ 2,  4,  9, 12])
```

Mientras tengan la misma foma todas las operaciones que usamos antes van a funcionar operando entre elementos que ocupen la misma posición en sus arrays. Si no tienen la misma cantidad de elementos tendremos un error, o tendremos que hacer operaciones de álgebra lineal que exceden al curso.

Para los curiosos, una matriz de dos dimensiones se definiría de la siguiente forma (esto es lo que va a funcionar siempre por detrás de nuestras tablas):

[ ]:
```
matriz2d = np.array([[1,2], [3,4]])
matriz2d
```

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

De la misma manera que nuestras tablas, las matrices tienen una propiedad que permite conocer su forma, es la propiedad **.shape**. Esta matriz que definimos es una matriz que tiene 2 elementos de alto y dos de ancho.

[ ]:
```
matriz2d.shape
```

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

1.2 Pandas

Ahora sí, ya explicada toda esta parte teórica aburrida, vamos a aprender pandas con ejemplos prácticos. Además pueden ver esta guía para iniciar en la página oficial de pandas: 10 minutes to pandas

1.2.1 Dataframe

Es la estructura que tienen nuestros datos en pandas, nuestras tablas.

Podemos definir un dataframe nuevo de la siguiente forma:

[ ]:
```
import pandas as pd

df = pd.DataFrame({"columna1":[1,2,3,4], "columna2":['a','b','c','d']})

df
```

[ ]:
```
   columna1 columna2
0         1        a
1         2        b
2         3        c
3         4        d
```

Vamos a ver que la mayoría de las operaciónes que hagamos con los dataframe las haremos usando los **métodos** que el propio dataframe de pandas posee. ¿Qué es un **método**? Un **método** es una función interna que tienen algunos tipos de datos complejos como un *dataframe*. Básicamente un dataframe viene con un montón de funciones muy optimizadas para hacer operaciones con él. Vamos a ver muchos ejemplos.

1.2.2 Métodos de exploración

Cuando son muchos datos tenemos que tener algunos medios para entender el esquema de esta tabla o algunos indicadores estadísticos sin leer la tabla entera. Les muestro algunas formas:

Conocer el esquema:

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   columna1  4 non-null      int64 
 1   columna2  4 non-null      object
dtypes: int64(1), object(1)
memory usage: 192.0+ bytes
```

Tener un resumen estadístico (para las columnas numéricas):

[ ]:
```
df.describe()
```

[ ]:
```
        columna1
count  4.000000
mean   2.500000
std    1.290994
min    1.000000
25%    1.750000
50%    2.500000
75%    3.250000
```

```
max    4.000000
```

Conocer solo la forma (recuerden lo que vimos de numpy):

[ ]:
```
# Esto no es un método sino un atributo: simplemente información extra que ya␣
↪tienen todos los dataframes
df.shape
```

[ ]:
```
(4, 2)
```

1.2.3 Con pandas podemos leer varias fuentes de datos como dataframes

[ ]:
```
indices_expo = pd.read_excel("https://www.indec.gob.ar/ftp/ica_digital/
↪ica_d_11_23EB819483A2/data/cuadros/indices_expo.xlsx")

indices_expo.head() # Este método devuelve solo las primeras n filas de la␣
↪tabla. Por defecto son 5
```

[ ]:
```
    Año  Mes  índice de precios de las exportaciones  \
0  2004    1                                97.863194
1  2004    2                                98.637791
2  2004    3                                98.699014
3  2004    4                               101.221201
4  2004    5                               104.373216

   índice de cantidades de las exportaciones  \
0                                  82.338631
1                                  84.271048
2                                  93.490321
3                                 104.331206
4                                 113.006773

   índice de precios de los productos primarios  \
0                                    104.339287
1                                    105.216406
2                                    103.044307
3                                    102.252704
4                                    105.934017

   índice de cantidades de los productos primarios  \
0                                     72.854995
1                                     69.067081
2                                     86.977924
3                                    153.128650
4                                    154.726093

   índice de precios de las manufacturas de origen agropecuario  \
```

```
0                                          105.859154
1                                          107.362138
2                                          107.778709
3                                          109.399487
4                                          111.595279

   índice de cantidades de las manufacturas de origen agropecuario  \
0                                           81.592177
1                                           88.145490
2                                           82.562333
3                                           93.501914
4                                          108.263039

   índice de precios de las manufacturas de origen industrial  \
0                                           91.881963
1                                           93.064163
2                                           93.753874
3                                           96.019831
4                                           97.300169

   índice de cantidades de las manufacturas de origen industrial  \
0                                           79.862665
1                                           84.353979
2                                           99.783356
3                                           87.491765
4                                           97.681463

   índice de precios de combustibles y energía  \
0                                     87.374956
1                                     85.238800
2                                     88.986244
3                                     92.162363
4                                     97.525000

   índice de cantidades de combustibles y energía
0                                      98.333094
1                                      93.719511
2                                     112.147395
3                                      96.546075
4                                      99.416113
```

[ ]:
```
aeropuertos = pd.read_csv(
    "https://datos.transporte.gob.ar/dataset/
  ↪62b3fe5f-ffe6-4d8f-9d59-bfabe75d1ee8/resource/
  ↪eb54e49e-9a5a-4614-91f4-526c650d0105/download/aeropuertos_detalle.csv"
    , sep=";")
aeropuertos.head()
```

```
[ ]:   local  oaci iata       tipo                        denominacion  \
     0   ACB   NaN  NaN  Aeródromo       CORONEL BOGADO/AGROSERVICIOS
     1   ACH   NaN  NaN  Aeródromo                      GENERAL ACHA
     2   ACM   NaN  NaN  Aeródromo           ARRECIFES/LA CURA MALAL
     3   ADO  SAWD  PUD  Aeródromo                     PUERTO DESEADO
     4   ADT   NaN  NaN  Aeródromo  BANDERA/AGROSERVICIOS DOÑA TERESA

                 coordenadas   latitud  longitud   elev uom_elev  … condicion  \
     0  33°16'20"S  60°34'14"W -60.57066 -33.27226   44.0   Metros  …   PRIVADO
     1  37°24' 6"S  64°36'49"W -64.61351 -37.40164  277.0   Metros  …   PUBLICO
     2  34° 4'33"S  60° 8'30"W -60.14170 -34.07574   37.0   Metros  …   PRIVADO
     3  47°44' 6"S  65°54'15"W -65.90410 -47.73511   82.0   Metros  …   PUBLICO
     4  28°51'19"S  62°15'53"W -62.26462 -28.85541   75.0   Metros  …   PRIVADO

        control region   fir      uso   trafico sna concesionado  \
     0  NOCONTROL   RACE  SAEF  AEROAPP  Nacional  NO           NO
     1  NOCONTROL   RACE  SAEF    CIVIL  Nacional  NO           NO
     2  NOCONTROL   RACE  SAEF    CIVIL  Nacional  NO           NO
     3    AERADIO   RASU  SAVF    CIVIL  Nacional  NO           NO
     4  NOCONTROL   RANO  SACF  AEROAPP  Nacional  NO           NO

                 provincia inhab
     0             SANTA FÉ    NO
     1             LA PAMPA    NO
     2         BUENOS AIRES    NO
     3           SANTA CRUZ    NO
     4  SANTIAGO DEL ESTERO    NO

     [5 rows x 23 columns]
```

Si quiero saber cuantas columnas y de que tipo de datos tiene la tabla "aeropuetos", podemos usar el método info. Reparen en que pandas infirió cuales son los tipos de datos que tiene cada columna, eso puede ser práctico como realmente molesto.

```
[ ]: aeropuertos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 693 entries, 0 to 692
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   local          693 non-null    object
 1   oaci           124 non-null    object
 2   iata           90 non-null     object
 3   tipo           693 non-null    object
 4   denominacion   693 non-null    object
 5   coordenadas    693 non-null    object
 6   latitud        693 non-null    float64
```

```
 7   longitud       693 non-null    float64
 8   elev           693 non-null    float64
 9   uom_elev       693 non-null    object
 10  ref            683 non-null    object
 11  distancia_ref  688 non-null    float64
 12  direccion_ref  679 non-null    object
 13  condicion      693 non-null    object
 14  control        693 non-null    object
 15  region         693 non-null    object
 16  fir            693 non-null    object
 17  uso            470 non-null    object
 18  trafico        693 non-null    object
 19  sna            693 non-null    object
 20  concesionado   693 non-null    object
 21  provincia      693 non-null    object
 22  inhab          693 non-null    object
dtypes: float64(4), object(19)
memory usage: 124.6+ KB
```

Puedo acceder al encabezado de la tabla

[ ]:
```
aeropuertos.columns
```

[ ]:
```
Index(['local', 'oaci', 'iata', 'tipo', 'denominacion', 'coordenadas',
       'latitud', 'longitud', 'elev', 'uom_elev', 'ref', 'distancia_ref',
       'direccion_ref', 'condicion', 'control', 'region', 'fir', 'uso',
       'trafico', 'sna', 'concesionado', 'provincia', 'inhab'],
      dtype='object')
```

Como a su índice:

[ ]:
```
aeropuertos.index
```

[ ]:
```
RangeIndex(start=0, stop=693, step=1)
```

1.2.4 ¿Cómo seleccionar datos?

A veces queremos quedarnos con algunas columnas, a veces queremos ver algunas filas en específico. #### Leer por índice Los nombres de las columnas son un índice al igual que los números de fila, podemos usarlos para leer datos específicos:

[ ]:
```
aeropuertos["local"]
```

[ ]:
```
0      ACB
1      ACH
2      ACM
3      ADO
4      ADT
      ...
```

```
688    YOS
689    YPY
690    ZAP
691    ZLM
692    ZUL
Name: local, Length: 693, dtype: object
```

Reparen como devuelve los datos de la columna local y además devuelve el índice de filas de la tabla. Esto es una **serie** de pandas, los dataframe están compuestos de ellas.

Nos quedamos con una fila, para eso usamos el método iloc. Fíjense que leer una fila del dataframe da como resultado una serie, que tiene las columnas como índice:

[ ]:
```
aeropuertos.iloc[5] # este método funciona proporcionandole el índice de la fila
```

[ ]:
```
local                              ADU
oaci                               NaN
iata                               NaN
tipo                        Aeródromo
denominacion              BANDERA/DUTTO
coordenadas      28°52' 1"S  62°14'17"W
latitud                       -62.23812
longitud                      -28.86691
elev                               87.0
uom_elev                         Metros
ref                             Bandera
distancia_ref                       3.0
direccion_ref                        NE
condicion                       PRIVADO
control                       NOCONTROL
region                             RANO
fir                                SACF
uso                             AEROAPP
trafico                        Nacional
sna                                  NO
concesionado                         NO
provincia           SANTIAGO DEL ESTERO
inhab                                NO
Name: 5, dtype: object
```

Lógicamente con la coordenada de una celda también podemos recuperar su valor, usando iloc:

[ ]:
```
aeropuertos.iloc[0,21]
```

[ ]:
```
'SANTA FÉ'
```

Con estos métodos podemos elegir que columnas usar del dataframe

[ ]:
```
indices_expo.columns
```

[ ]:
```
Index(['Año', 'Mes', 'índice de precios de las exportaciones',
       'índice de cantidades de las exportaciones',
       'índice de precios de los productos primarios',
       'índice de cantidades de los productos primarios',
       'índice de precios de las manufacturas de origen agropecuario',
       'índice de cantidades de las manufacturas de origen agropecuario',
       'índice de precios de las manufacturas de origen industrial',
       'índice de cantidades de las manufacturas de origen industrial',
       'índice de precios de combustibles y energía',
       'índice de cantidades de combustibles y energía'],
      dtype='object')
```

[ ]:
```
indices_expo[['Año', 'Mes','índice de precios de las exportaciones', 'índice de␣
↪cantidades de las exportaciones']]
```

[ ]:
```
      Año  Mes  índice de precios de las exportaciones  \
0    2004    1                              97.863194
1    2004    2                              98.637791
2    2004    3                              98.699014
3    2004    4                             101.221201
4    2004    5                             104.373216
..    …   …                                     …
233  2023    6                             194.953569
234  2023    7                             189.442813
235  2023    8                             187.634697
236  2023    9                             185.265495
237  2023   10                             183.306995

     índice de cantidades de las exportaciones
0                                    82.338631
1                                    84.271048
2                                    93.490321
3                                   104.331206
4                                   113.006773
..                                         …
233                                  96.357135
234                                 110.830245
235                                 109.046317
236                                 107.423438
237                                 101.956874

[238 rows x 4 columns]
```

[ ]:
```
indices_expo_filtrada = indices_expo[['Año', 'Mes','índice de precios de las␣
↪exportaciones', 'índice de cantidades de las exportaciones', 'índice de␣
↪cantidades de combustibles y energía']]
indices_expo_filtrada.head()
```

[ ]:
```
   Año  Mes  índice de precios de las exportaciones  \
0  2004    1                                97.863194
1  2004    2                                98.637791
2  2004    3                                98.699014
3  2004    4                               101.221201
4  2004    5                               104.373216

   índice de cantidades de las exportaciones  \
0                                  82.338631
1                                  84.271048
2                                  93.490321
3                                 104.331206
4                                 113.006773

   índice de cantidades de combustibles y energía
0                                       98.333094
1                                       93.719511
2                                      112.147395
3                                       96.546075
4                                       99.416113
```

Tambien puedo eliminar filas o columnas

[ ]:
```
indices_expo_filtrada = indices_expo_filtrada.drop('índice de cantidades de␣
↪combustibles y energía', axis=1) # axis 1 quiere decir que quiero eliminar␣
↪columnas, de otra forma podría eliminar filas tambien
indices_expo_filtrada.head()
```

[ ]:
```
   Año  Mes  índice de precios de las exportaciones  \
0  2004    1                                97.863194
1  2004    2                                98.637791
2  2004    3                                98.699014
3  2004    4                               101.221201
4  2004    5                               104.373216

   índice de cantidades de las exportaciones
0                                  82.338631
1                                  84.271048
2                                  93.490321
3                                 104.331206
4                                 113.006773
```

Puedo filtrarla por las mismas condiciones lógicas que ya vimos:

[ ]:
```
indices_expo_filtrada.Año > 2010 # al usar un operádor lógico en la serie de␣
↪pandas, pandas nos devuleve una serie de booleanos.
```

[ ]:
```
0      False
1      False
2      False
3      False
4      False
       …
233     True
234     True
235     True
236     True
237     True
Name: Año, Length: 238, dtype: bool
```

Ese array de booleanos es el que permite filtrar las tablas con condiciones que le impongamos

[ ]:
```
indices_expo_filtrada[indices_expo_filtrada.Año > 2010]
```

[ ]:
```
      Año  Mes  índice de precios de las exportaciones  \
84   2011    1                              180.058997
85   2011    2                              184.026354
86   2011    3                              190.789107
87   2011    4                              187.680033
88   2011    5                              190.600870
..    …  …                                    …
233  2023    6                              194.953569
234  2023    7                              189.442813
235  2023    8                              187.634697
236  2023    9                              185.265495
237  2023   10                              183.306995

     índice de cantidades de las exportaciones
84                                    99.960269
85                                   101.901966
86                                   111.037765
87                                   130.277500
88                                   144.443923
..                                          …
233                                   96.357135
234                                  110.830245
235                                  109.046317
236                                  107.423438
237                                  101.956874

[154 rows x 4 columns]
```

Miren como ahora la tabla tiene menos observaciones. Tambien podemos hacer esto con otro método:

[ ]:
```
indices_expo_filtrada = indices_expo_filtrada.query('Año > 2010')
indices_expo_filtrada.head()
```

[ ]:

	Año	Mes	índice de precios de las exportaciones	índice de cantidades de las exportaciones
84	2011	1	180.058997	99.960269
85	2011	2	184.026354	101.901966
86	2011	3	190.789107	111.037765
87	2011	4	187.680033	130.277500
88	2011	5	190.600870	144.443923

Además podemos agregar columnas:

[ ]:
```
indices_expo_filtrada["suma"] = indices_expo_filtrada["índice de cantidades de␣
↪las exportaciones"] + indices_expo_filtrada["índice de precios de las␣
↪exportaciones"]
indices_expo_filtrada
```

[ ]:

	Año	Mes	índice de precios de las exportaciones	índice de cantidades de las exportaciones	suma
84	2011	1	180.058997	99.960269	280.019266
85	2011	2	184.026354	101.901966	285.928320
86	2011	3	190.789107	111.037765	301.826871
87	2011	4	187.680033	130.277500	317.957532
88	2011	5	190.600870	144.443923	335.044793
..	…	…	…	…	…
233	2023	6	194.953569	96.357135	291.310704
234	2023	7	189.442813	110.830245	300.273058
235	2023	8	187.634697	109.046317	296.681014
236	2023	9	185.265495		
237	2023	10	183.306995		

```
236                                  107.423438  292.688933
237                                  101.956874  285.263869

[154 rows x 5 columns]
```

Tambien podemos ordenar y agrupar la tabla por distintas variables

[ ]:
```
indices_expo_agrupada_ordenada = (
    indices_expo_filtrada
    .drop("Mes", axis=1) #elimino la columna mes
    .groupby("Año").sum() #agrupo por año sumando los valores de cada mes del␣
 ↪mismo año
    .sort_values("suma", ascending=False) #ordenamos descentemente por los␣
 ↪valores de la colmna "suma"
)

indices_expo_agrupada_ordenada
```

[ ]:
```
      índice de precios de las exportaciones  \
Año
2022                              2564.296148
2011                              2261.829831
2012                              2317.096610
2021                              2203.350486
2013                              2284.280725
2014                              2227.903170
2019                              1812.682336
2018                              1930.430239
2017                              1825.365888
2015                              1888.938146
2016                              1797.388588
2020                              1761.900388
2023                              1948.797150

      índice de cantidades de las exportaciones         suma
Año
2022                                1434.000000  3998.296148
2011                                1525.800000  3787.629831
2012                                1436.400000  3753.496610
2021                                1467.671991  3671.022476
2013                                1383.600000  3667.880725
2014                                1276.200000  3504.103170
2019                                1495.800000  3308.482336
2018                                1335.000000  3265.430239
2017                                1338.600000  3163.965888
2015                                1255.200000  3144.138146
2016                                1340.400000  3137.788588
```

```
2020                                  1300.500000  3062.400388
2023                                  1009.156874  2957.954025
```

Al usar "group by", el nuevo índice de las filas es la variable por la que agrupamos. Siempre podemos volver al índice numérico que vimos antres

[ ]:
```
indices_expo_agrupada_ordenada.reset_index()
```

[ ]:
```
    Año  índice de precios de las exportaciones  \
0   2022                              2564.296148
1   2011                              2261.829831
2   2012                              2317.096610
3   2021                              2203.350486
4   2013                              2284.280725
5   2014                              2227.903170
6   2019                              1812.682336
7   2018                              1930.430239
8   2017                              1825.365888
9   2015                              1888.938146
10  2016                              1797.388588
11  2020                              1761.900388
12  2023                              1948.797150

    índice de cantidades de las exportaciones         suma
0                                 1434.000000  3998.296148
1                                 1525.800000  3787.629831
2                                 1436.400000  3753.496610
3                                 1467.671991  3671.022476
4                                 1383.600000  3667.880725
5                                 1276.200000  3504.103170
6                                 1495.800000  3308.482336
7                                 1335.000000  3265.430239
8                                 1338.600000  3163.965888
9                                 1255.200000  3144.138146
10                                1340.400000  3137.788588
11                                1300.500000  3062.400388
12                                1009.156874  2957.954025
```

2 ¡Tengan un machete a mano!

1. Cheatsheet oficial de pandas
2. Cheatsheet en español (no es tan clara)

Recuerden googlear por más información, hay muchos documentos como este en internet, muchos son más completos que este incluso. Por suerte también hay varios articulos muy buenos en español tambien. Google, google y chatGPT .

2.1 Ejercicio en clase

Leemos la tabla de índices de valor, precio y cantidad, de las exportaciones por grandes rubros de comercio exterior y calculamos el promedio ponderado para un rubro cualquiera.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i} = \frac{x_1 w_1 + x_2 w_2 + x_3 w_3 + \ldots + x_n w_n}{w_1 + w_2 + w_3 + \ldots + w_n}$$

[ ]:
```
df_vpq_grubros = pd.read_excel("https://www.indec.gob.ar/ftp/cuadros/economia/
 ↪serie_mensual_indices_expo.xls", skiprows=3, skipfooter=5)
df_vpq_grubros = (
    df_vpq_grubros.dropna(how="all")
    .dropna(how="all",axis=1))

df_vpq_grubros.columns = [
    "año", "mes",
    "nivel_general_v","nivel_general_p","nivel_general_q",
    "pp_v","pp_p","pp_q",
    "moa_v","moa_p","moa_q",
    "moi_v","moi_p","moi_q",
    "cye_v","cye_p","cye_q"]

df_vpq_grubros.año = df_vpq_grubros.año.ffill().str.slice(0,4)


df_vpq_grubros
```

[ ]:
```
      año         mes  nivel_general_v  nivel_general_p  nivel_general_q  \
1    2004       Enero        80.603953        97.863194        82.338631
2    2004     Febrero        83.123100        98.637791        84.271048
3    2004       Marzo        92.234875        98.699014        93.490321
4    2004       Abril       105.498772       101.221201       104.331206
5    2004        Mayo       117.807332       104.373216       113.006773
..    …          …              …                …                …
234  2023       Junio       187.932320       194.953569        96.357135
235  2023       Julio       210.332815       189.442813       110.830245
236  2023      Agosto       205.124199       187.634697       109.046317
237  2023  Septiembre       199.596631       185.265495       107.423438
238  2023     Octubre       186.894082       183.306995       101.956874

           pp_v        pp_p        pp_q       moa_v       moa_p       moa_q  \
1     76.123614  104.339287   72.854995   86.380968  105.859154   81.592177
2     72.702140  105.216406   69.067081   94.632818  107.362138   88.145490
3     89.558461  103.044307   86.977924   88.934241  107.778709   82.562333
4    156.440215  102.252704  153.128650  102.194786  109.399487   93.501914
5    163.674736  105.934017  154.726093  120.702251  111.595279  108.263039
```

```
..          …         …         …         …         …         …
234  223.099790  203.825746  109.152802  220.416743  204.294681  107.822932
235  248.626891  189.738936  130.564084  216.811726  202.650379  106.782106
236  256.413659  178.606070  142.982170  205.935271  197.131707  104.172009
237  250.466241  178.300695  139.853746  181.814207  198.137504   91.445885
238  199.640977  173.268796  115.220387  170.241974  194.166939   87.678147

          moi_v      moi_p      moi_q      cye_v      cye_p      cye_q
1     73.327082   91.881963   79.862665   85.744250   87.374956   98.333094
2     78.440031   93.064163   84.353979   79.751869   85.238800   93.719511
3     93.456245   93.753874   99.783356   99.670501   88.986244  112.147395
4     83.921482   96.019831   87.491765   88.971806   92.162363   96.546075
5     94.934964   97.300169   97.681463   96.958675   97.525000   99.416113
..          …         …         …         …         …         …
234  180.216415  169.141244  106.567241   98.270366  227.957365   43.225433
235  234.267129  169.938767  137.409772  118.145320  226.316698   52.187652
236  212.487738  171.676269  122.961426  135.246981  248.074757   54.407745
237  243.488711  160.967301  150.328802  109.233849  252.355535   43.104604
238  223.142815  160.227662  139.266100  148.501880  260.601599   56.984255

[238 rows x 17 columns]
```