Previamente… Módulo 3 - Tarea 2

Explorando los datos

Una vez los archivos de datos se ha subido, es necesario analizar su contenido para identificar y corregir problemas y errores contenidos.

Visualización de los tipos de datos de las columnas

Es importante mirar a los tipos de datos para identificar si son consistentes con los datos que leemos.

En este caso se ve que los datos son numéricos pero el tipo de dato no identifica los identifica correctamente. Esto implica que valores no numéricos en esta columna tendrán que ser eliminados pues es una columna de datos numéricos.

Input [10]:

```
data.info()
```

Output [10]:

```
<class 'pandas.core.frame.DataFrame'>
Index: 537 entries, European Union (EU6-1958, EU9-1973, EU10-19
81, EU12-1986, EU15-1995, EU25-2004, EU27-2007, EU28-2013) to B
osnia and Herzegovina
Data columns (total 58 columns):
1991                      537 non-null object
Flags and footnotes       0 non-null float64
1992                      537 non-null object
Flags and footnotes.1     0 non-null float64
1993                      537 non-null object
Flags and footnotes.2     0 non-null float64
1994                      537 non-null object
Flags and footnotes.3     0 non-null float64
1995                      537 non-null object
Flags and footnotes.4     0 non-null float64
1996                      537 non-null object
Flags and footnotes.5     0 non-null float64
1997                      537 non-null object
Flags and footnotes.6     0 non-null float64
1998                      537 non-null object
Flags and footnotes.7     0 non-null float64
1999                      537 non-null object
Flags and footnotes.8     0 non-null float64
2000                      537 non-null object
Flags and footnotes.9     0 non-null float64
2001                      537 non-null object
Flags and footnotes.10    0 non-null float64
2002                      537 non-null object
Flags and footnotes.11    0 non-null float64
```

```
2003                        537 non-null object
Flags and footnotes.12      0 non-null float64
2004                        537 non-null object
Flags and footnotes.13      0 non-null float64
2005                        537 non-null object
Flags and footnotes.14      0 non-null float64
2006                        537 non-null object
Flags and footnotes.15      0 non-null float64
2007                        537 non-null object
Flags and footnotes.16      0 non-null float64
2008                        537 non-null object
Flags and footnotes.17      1 non-null object
2009                        537 non-null object
Flags and footnotes.18      26 non-null object
2010                        537 non-null object
Flags and footnotes.19      6 non-null object
2011                        537 non-null object
Flags and footnotes.20      6 non-null object
2012                        537 non-null object
Flags and footnotes.21      14 non-null object
2013                        537 non-null object
Flags and footnotes.22      33 non-null object
2014                        537 non-null object
Flags and footnotes.23      44 non-null object
2015                        537 non-null object
Flags and footnotes.24      12 non-null object
2016                        537 non-null object
Flags and footnotes.25      16 non-null object
2017                        537 non-null object
Flags and footnotes.26      16 non-null object
2018                        537 non-null object
Flags and footnotes.27      41 non-null object
2019                        537 non-null object
Flags and footnotes.28      0 non-null float64
dtypes: float64(18), object(40)
memory usage: 247.5+ KB
```

Estadísticas de las columnas

Otra manera básica de revisar los datos para identificar problemas es a través de un sumario estadístico que nos de una idea de lo que la tabla contiene.

Si el parámetro “include” es “all”, nos muestra todas las columnas.

Algunos valores serán “NaN” porque no tienen sentido para el tipo de dato de la columna.

Para datos numéricos include el número de elementos (count), la media (mean), la desviación estándar (str), el mínimo y máximo (min, max) y los percentiles 25, 50 y 75.

Para el resto de datos (texto y fechas), además del número de elementos (count), nos muestra el valor más común (top) y su frecuencia (freq).

Input [11]:

```
data.describe(include = "all")
```

Output [11]:

	1991	Flags and footnotes	1992	Flags and footnotes.1	1993	Flags and footnotes.2	1994	Flags and footnotes.3	1995	Flags and footnotes.4	...	2015	Flags and footnotes.24	2016	Flags and footnotes.25	2017	Flags a footnotes.
count	537	0.0	537	0.0	537	0.0	537	0.0	537	0.0	...	537	12	537	16	537	
unique	189	NaN	217	NaN	188	NaN	214	NaN	288	NaN	...	317	2	315	2	326	
top	:	NaN	:	NaN	:	NaN	:	NaN	:	NaN	...	:	p	:	p	:	
freq	305	NaN	279	NaN	305	NaN	279	NaN	176	NaN	...	182	11	181	15	176	
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	Na
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	Na
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	Na
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	Na
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	Na
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	Na
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	Na

11 rows × 58 columns

Las dos operaciones previas muestran una primera serie de problemas.

Por una parte, el tipo de datos de las columnas no es numérico. Esto implica que hay valores inválidos, tal y como se ve con la operación “describe”.

Además, hay una serie de “Flags” (banderas) que no contienen ninguna información. Estas son columnas descriptivas que pueden ser eliminadas.

The two previous operations show a first series of problems.

Eliminar columnas innecesarias

Para eliminar columnas innecesarias se necesita saber sus nombres (o índices).

Input [12]:

```
print(data.columns)
```

Output [12]:

```
Index(['1991', 'Flags and footnotes', '1992', 'Flags and footno
tes.1', '1993',
       'Flags and footnotes.2', '1994', 'Flags and footnotes.3'
, '1995',
       'Flags and footnotes.4', '1996', 'Flags and footnotes.5'
, '1997',
       'Flags and footnotes.6', '1998', 'Flags and footnotes.7'
```


```
, '1999',
       'Flags and footnotes.8', '2000', 'Flags and footnotes.9'
, '2001',
       'Flags and footnotes.10', '2002', 'Flags and footnotes.1
1', '2003',
       'Flags and footnotes.12', '2004', 'Flags and footnotes.1
3', '2005',
       'Flags and footnotes.14', '2006', 'Flags and footnotes.1
5', '2007',
       'Flags and footnotes.16', '2008', 'Flags and footnotes.1
7', '2009',
       'Flags and footnotes.18', '2010', 'Flags and footnotes.1
9', '2011',
       'Flags and footnotes.20', '2012', 'Flags and footnotes.2
1', '2013',
       'Flags and footnotes.22', '2014', 'Flags and footnotes.2
3', '2015',
       'Flags and footnotes.24', '2016', 'Flags and footnotes.2
5', '2017',
       'Flags and footnotes.26', '2018', 'Flags and footnotes.2
7', '2019',
       'Flags and footnotes.28'],
      dtype='object')
```

Tienes que eliminar todas las columnas que empiezan con “Flags”.

Una primera manera de eliminarlas es una a una indicando su nombre.

Input [13]:

```
data = data.drop(columns=['Flags and footnotes'])
data.head(1)
```

Output [13]:

GEO/TIME	1991	1992	Flags and footnotes.1	1993	Flags and footnotes.2	1994	Flags and footnotes.3	1995	Flags and footnotes.4	1996	...	2015	Flags and footnotes.24	2016	Flags and footnotes.25	2017	Flags a footnotes
European Union (EU6-1958, EU9-1973, EU10-1981, EU12-1986, EU15-1995, EU25-2004, EU27-2007, EU28-2013)	:	:	NaN	:	NaN	:	NaN	:	NaN	:	...	:	NaN	:	NaN	:	N

1 rows × 57 columns

Pero también se pueden eliminar todas de golpe creando un vector con los nombres de las columnas que se van a borrar.

Esto se hace creando un vector con los nombres de las columnas que contienen “Flags” y después borrando todas las columnas de ese vector.

Input [14]:

```
indices = [i for i, s in enumerate(data.columns) if 'Flags' in s]
colToDelete=data.columns[np.array(indices)]
print(colToDelete)
```

Output [14]:

```
Index(['Flags and footnotes.1', 'Flags and footnotes.2',
       'Flags and footnotes.3', 'Flags and footnotes.4',
       'Flags and footnotes.5', 'Flags and footnotes.6',
       'Flags and footnotes.7', 'Flags and footnotes.8',
       'Flags and footnotes.9', 'Flags and footnotes.10',
       'Flags and footnotes.11', 'Flags and footnotes.12',
       'Flags and footnotes.13', 'Flags and footnotes.14',
       'Flags and footnotes.15', 'Flags and footnotes.16',
       'Flags and footnotes.17', 'Flags and footnotes.18',
       'Flags and footnotes.19', 'Flags and footnotes.20',
       'Flags and footnotes.21', 'Flags and footnotes.22',
       'Flags and footnotes.23', 'Flags and footnotes.24',
       'Flags and footnotes.25', 'Flags and footnotes.26',
       'Flags and footnotes.27', 'Flags and footnotes.28'],
      dtype='object')
```

Input [15]:

```
data = data.drop(columns=colToDelete)
data.head(1)
```

Output [15]:

GEO/TIME	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
European Union (EU6-1958, EU9-1973, EU10-1981, EU12-1986, EU15-1995, EU25-2004, EU27-2007, EU28-2013)	:	:	:	:	:	:	:	:	:	:	...	:	86785.2	:	:	:	:	:	:	:	:

1 rows × 29 columns

Ajustar el nombre de las columnas o filas en la tabla

Podría ser necesario cambiar los nombres de los encabezados para hacerlos más relevantes.

Por ejemplo, puedes renombrar la primera columna, ya que contiene los nombres de las divisiones NUTS.

Input [17]:

```
data.index.names = ['NUTS']
```

También podemos ajustar los nombres de las filas si consideramos que pueden simplificar la lectura de la tabla.

Por ejemplo, podemos cambiar el nombre de la primera fila a “European Union”.

Input [18]:

```
data = data.rename(index={data.index.ravel()[0]: 'European Union'})
data.head()
```

Output [18]:

NUTS	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2010	2011	2012	2013	2014	2015	20
European Union	:	:	:	:	:	:	:	:	:	:	...	:	86785.2	:	:	:	:	
Belgium	3105.5	3099.6	3084.2	3161.1	3158.7	3070.8	2978.4	2984.4	2970.4	3041.6	...	2592.63	2560.32	2484.27	2432.53	2477.24	2503.26	2501.
Région de Bruxelles-Capitale / Brussels Hoofdstedelijk Gewest	0.3	0.5	0.5	0.5	0.4	0.4	0.4	0.3	0.4	0.4	...	0.24	0.25	0.56	0.59	0.79	0.87	
Région de Bruxelles-Capitale / Brussels Hoofdstedelijk Gewest	0.3	0.5	0.5	0.5	0.4	0.4	0.4	0.3	0.4	0.4	...	0.24	0.25	0.56	0.59	0.79	0.87	
Vlaams Gewest	1655.2	1661.4	1637.2	1685.5	1678.5	1613.1	1556.8	1554.4	1536.2	1558.1	...	1303.87	1302.25	1269.41	1255.4	1299.98	1321.01	1326.

5 rows × 29 columns

Limpieza de datos.

El análisis general podria darte una idea de que algo está mal, pero un análisis mas detallado de las columnas es necesario para identificar problemas en los valores.

Una buena estrategia de análisis es visualizar y/o contar el número de valores diferentes, ver el número de valores no numéricos o incluso ver si hay filas o columnas repetidas.

Contado de valores

Contar valores diferentes en las columnas podría mostrar un problema en los datos.

- *dataframe[column].count() muestra el número de valores en la columna.*
- *dataframe[column].unique() muestral el número de valores diferentes en la columna*
- *dataframe[column].value_counts() muestra el número de ocurrencia de cada valor*

Podemos analizar la columna de 2019 mirando su contenido.

Input [19]:

```
print("Number of values", data['2019'].count())
print("Unique values", data['2019'].unique())
```

Output [19]:

```
Number of values 537
Unique values [':' 2373.1 0.72 1275.98 345.21 129.6 311.89 96.0
7 393.21 1096.4 45.33
 279.47 241.17 311.55 218.87 1500 28 72 594 453 354]
```

Input [20]:

```
serie = data['2019'].value_counts()
serie
```

Output [20]:

```
:          517
28           1
279.47       1
129.6        1
2373.1       1
1096.4       1
393.21       1
72           1
594          1
311.55       1
345.21       1
1275.98      1
218.87       1
1500         1
96.07        1
354          1
453          1
45.33        1
241.17       1
311.89       1
0.72         1
Name: 2019, dtype: int64
```

Hay 537 valores en 2019 pero la mayoría de ellos son no-numéricos. Hay que cambiarlos a valores nulos.

Reemplazo de valores no numéricos.

Cambiabos todos los valores no numéricos a NaN (not a number) excepto la primera columna.

Se define una función para identificar si un valor es “float”.

Selecciona las columnas a filtrar y entonces añade la columna con los nombres.

Input [21]:

```
def isfloat(value):
 try:
   float(value)
   return True
 except ValueError:
   return False
data= data.where(data.applymap(lambda x: isfloat(x)))
data.head(5)
```

Output [21]:

NUTS	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2010	2011	2012	2013	2014	2015	20
European Union	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	86785.2	NaN	NaN	NaN	NaN	N
Belgium	3105.5	3099.6	3084.2	3161.1	3158.7	3070.8	2978.4	2984.4	2970.4	3041.6	...	2592.63	2560.32	2484.27	2432.53	2477.24	2503.26	2501.
Région de Bruxelles-Capitale / Brussels Hoofdstedelijk Gewest	0.3	0.5	0.5	0.5	0.4	0.4	0.4	0.3	0.4	0.4	...	0.24	0.25	0.56	0.59	0.79	0.87	
Région de Bruxelles-Capitale / Brussels Hoofdstedelijk Gewest	0.3	0.5	0.5	0.5	0.4	0.4	0.4	0.3	0.4	0.4	...	0.24	0.25	0.56	0.59	0.79	0.87	
Vlaams Gewest	1655.2	1661.4	1637.2	1685.5	1678.5	1613.1	1556.8	1554.4	1536.2	1558.1	...	1303.87	1302.25	1269.41	1255.4	1299.98	1321.01	1326.

5 rows × 29 columns

Conversión del tipo de datos

Para poder hacer cualquier tipo de procesado, debes convertir las columnas a valores numéricos.

La conversion puede poner NaN a todos los valores no numéricos para que el paso anterior no sea necesario.

Input [22]:

```
data = data[data.columns].apply(pd.to_numeric, errors='coerce')
data.info()
```

Output [22]:

```
<class 'pandas.core.frame.DataFrame'>
Index: 537 entries, European Union to Bosnia and Herzegovina
Data columns (total 29 columns):
1991    232 non-null float64
1992    258 non-null float64
1993    232 non-null float64
1994    258 non-null float64
1995    361 non-null float64
1996    382 non-null float64
1997    355 non-null float64
1998    354 non-null float64
1999    384 non-null float64
2000    354 non-null float64
2001    396 non-null float64
2002    364 non-null float64
2003    402 non-null float64
2004    311 non-null float64
2005    151 non-null float64
2006    61 non-null float64
2007    328 non-null float64
2008    352 non-null float64
2009    347 non-null float64
2010    357 non-null float64
2011    352 non-null float64
2012    354 non-null float64
2013    355 non-null float64
2014    347 non-null float64
2015    355 non-null float64
2016    356 non-null float64
2017    361 non-null float64
2018    361 non-null float64
2019    20 non-null float64
dtypes: float64(29)
memory usage: 125.9+ KB
```

Input [23]:

```
data.describe(include = 'all')
```

Output [23]:

	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...
count	232.000000	258.000000	232.000000	258.000000	361.000000	382.000000	355.000000	354.000000	384.000000	354.000000	...
mean	1017.951293	927.348450	956.366810	906.918217	853.977285	831.963874	853.710423	840.977684	798.689583	813.195706	...
std	2198.705273	2031.018765	2104.695487	2035.171217	1813.874991	1750.314308	1788.685585	1769.476449	1701.587877	1731.501156	...
min	-1.000000	-1.000000	-1.000000	-1.000000	0.000000	0.000000	0.100000	0.100000	0.100000	0.100000	...
25%	64.900000	80.100000	63.400000	77.375000	104.500000	125.500000	112.500000	113.250000	117.900000	111.725000	...
50%	413.400000	366.450000	399.300000	375.850000	372.100000	359.300000	379.400000	363.350000	337.450000	340.500000	...
75%	1103.025000	962.925000	944.725000	911.700000	882.100000	794.850000	855.950000	828.250000	763.325000	781.775000	...
max	20804.000000	20264.500000	19906.400000	20507.100000	20836.500000	20540.700000	20334.200000	20055.300000	20196.300000	20324.500000	...

8 rows × 29 columns

Estandarización de los contenidos.

Después de analizar los datos vemos que la columna NUTS tiene los nombres de los NUTS pero no los códigos.

El siguiente paso es estandarizar los contenidos de esa columna reemplazando las etiquetas con los códigos.

Para este propósito, tenemos un archivo que contiene las etiquetas.

Input [24]:

```
file = 'datos/nuts.xlsx'
nuts = pd.read_excel(file, index_col=0)
nuts.head(5)
```

Output [24]:

NUTS	LABEL
EU	European Union (EU6-1958, EU9-1973, EU10-1981,...
BE	Belgium
BE1	Région de Bruxelles-Capitale / Brussels Hoofds...
BE10	Région de Bruxelles-Capitale / Brussels Hoofds...
BE2	Vlaams Gewest

Ya que la información se organiza de manera similar, podemos cambiar un índice por otro.

Input [25]:

```
data.index = nuts.index
data.head(5)
```

Output [25]:

NUTS	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2010	2011	2012	2013	2014	2015	2016
EU	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	86785.24	NaN	NaN	NaN	NaN	NaN
BE	3105.5	3099.6	3084.2	3161.1	3158.7	3070.8	2978.4	2984.4	2970.4	3041.6	...	2592.63	2560.32	2484.27	2432.53	2477.24	2503.26	2501.35
BE1	0.3	0.5	0.5	0.5	0.4	0.4	0.4	0.3	0.4	0.4	...	0.24	0.25	0.56	0.59	0.79	0.87	0.50
BE10	0.3	0.5	0.5	0.5	0.4	0.4	0.4	0.3	0.4	0.4	...	0.24	0.25	0.56	0.59	0.79	0.87	0.50
BE2	1655.2	1661.4	1637.2	1685.5	1678.5	1613.1	1556.8	1554.4	1536.2	1558.1	...	1303.87	1302.25	1269.41	1255.40	1299.98	1321.01	1326.77

5 rows × 29 columns

Eliminando valores inválidos.

Dependiendo del problema puedes saber que los valores de las celdas deben estar entre un determinado rango. Por tanto, cualquier valor fuera de este rango no es válido y tiene que ser ajustado (min, max o NaN).

En el ejemplo discutido, los valores pueden ser solo positivos. Sin embargo, hay algunos valores negativos en la tabla.

- *dataframe.lt() muestral el número de valores más bajos que los indicados*

Input [26]:

```
data.lt(0).sum()
```

Output [26]:

```
1991    3
1992    3
1993    3
1994    3
1995    0
1996    0
1997    0
1998    0
1999    0
2000    0
2001    0
2002    0
2003    0
2004    0
2005    0
2006    0
2007    0
2008    0
2009    0
2010    0
2011    0
2012    0
2013    0
```

```
2014    0
2015    0
2016    0
2017    0
2018    0
2019    0
dtype: int64
```

Estos valores se pueden poner como 0 o cambiarlos a NaN

Input [27]:

```
data[data < 0] = np.nan
```

Visualización de datos nulos

Podemos ver cuantos de los 537 valores no numéricos tiene cada columna.

Dependiendo del problema a resolver puede ser necesario eliminar filas o columnas con NaN para tener datos completos.

- *dataframe.isna() muestral los valores nulos del conjunto de datos*

Input [28]:

```
data.isna().sum()
```

Output [28]:

```
1991    308
1992    282
1993    308
1994    282
1995    176
1996    155
1997    182
1998    183
1999    153
2000    183
2001    141
2002    173
2003    135
2004    226
2005    386
2006    476
2007    209
2008    185
2009    190
```

```
2010    180
2011    185
2012    183
2013    182
2014    190
2015    182
2016    181
2017    176
2018    176
2019    517
dtype: int64
```

Eliminar datos nulos

En caso de que necesitemos limpiar más datos y eliminar columnas que contengan algunos datos nulos, en este caso tendríamos que eliminar 2019, ya que los datos no están terminados. Así, tenemos que eliminar las columnas que no nos son útiles.

- *dataframe.dropna() te permite eliminar columnas con datos nulos.*

Input [29]:

```
data2 = data.dropna()
data2.describe()
```

Output [29]:

	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
count	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
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

8 rows × 29 columns

Se puede ver que en la colección de datos, hay columnas no completas.

Inteporlación de datos nulos.

Si los análisis a realizar no permiten hacer nulos los datos o eliminarlos ya que muchas filas se eliminarían, una alternativa es interpolar los valores.

Input [30]:


```
data = data.interpolate(axis=1, limit_direction='both')
data.head()
```

Output [30]:

NUTS	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2010	2011	2012	2013	
EU	86785.24	86785.24	86785.24	86785.24	86785.24	86785.24	86785.24	86785.24	86785.24	86785.24	...	86785.24	86785.24	86785.24	86785.24	867
BE	3105.50	3099.60	3084.20	3161.10	3158.70	3070.80	2978.40	2984.40	2970.40	3041.60	...	2592.63	2560.32	2484.27	2432.53	247
BE1	0.30	0.50	0.50	0.50	0.40	0.40	0.40	0.30	0.40	0.40	...	0.24	0.25	0.56	0.59	
BE10	0.30	0.50	0.50	0.50	0.40	0.40	0.40	0.30	0.40	0.40	...	0.24	0.25	0.56	0.59	
BE2	1655.20	1661.40	1637.20	1685.50	1678.50	1613.10	1556.80	1554.40	1536.20	1558.10	...	1303.87	1302.25	1269.41	1255.40	129

5 rows × 29 columns

Rellenar columnas vacias

La interpolación es imprecise con columnas con pocos datos, ya que apenas hay datos e información para deducir los valores que faltan. Un ejemplo es la primera fila de la tabla, donde solo hay 1 dato para la Unión Europea, por lo que la interpolación rellena todas las casillas con ese valor.

Otro problema son las filas completamente vacías. Aquí la interpolación no puede hacer nada. Una solución si necesitan tener valores es poner todas esas filas con valores 0.

- *dataframe.fillna() te permite cambiar un valor NaN a otro valor.*

Continua… Módulo 3 – Tarea 3