



Module 3 - Task 3

## WRITING DATA FILES



### Previamente... Módulo 3 - Tarea 2

#### Conversión del tipo de datos

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

La conversión 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 x 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]:



Module 3 - Task 3

# WRITING DATA FILES



	LABEL
<b>NUTS</b>	
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]:**

	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2010	2011	2012	2013	2014	2015	2016
<b>NUTS</b>																		
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 x 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()` muestra 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()` muestra los valores nulos del conjunto de datos

Input [28]:

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

Output [28]:





### Module 3 - Task 3

# WRITING DATA FILES



```
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	0.0	
mean	NaN	NaN	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	NaN	NaN
min	NaN	NaN	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	NaN	NaN
50%	NaN	NaN	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	NaN	NaN
max	NaN	NaN	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]:

	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2010	2011	2012	2013	
NUTS																
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	86785.24
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	2411.10
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	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	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	1255.40

5 rows × 29 columns



### Rellenar columnas vacías

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.





Module 3 - Task 3

## WRITING DATA FILES



### Escribiir archivos de datos

#### PARA APRENDER MÁS...

The formats supported in the Pandas library and how to write in them is described in:

[https://pandas.pydata.org/pandas-docs/stable/user\\_guide/io.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html)

Format	Read	Write
csv	pd.read_csv()	pac.to_csv()
json	pd.read_json()	pac.to_json()
excel	pd.read_excel()	pac.to_excel()
hdf	pd.read_hdf()	pac.to_hdf()
sql	pd.read_sql()	pac.to_sql()
...	...	...

Each write operation in a format has different parameters to adjust the settings.

#### Writing to an Excel file

**Pandas** allows us to save the data in Excel format with `dataframe.to_excel()`.

The most relevant parameters are:

- *io* = The location of the file
- *sheet\_name* = The sheet to be written
- *index* = If we want you to put an index column or not in the file

Input [32]:

```
data.to_excel('datos/animalEurostatNuts2_DataSheet_corrected.xlsx', sheet_name='Data')
```

#### Procesado de multiples hojas

El archivo usado como ejemplo contiene múltiples hojas que pueden ser procesadas de la misma manera ya que tienen la misma estructura.

Para automatizar el proceso, puedes leer todas las hojas y aplicar el proceso a cada una de ellas y guardar el resultado en un ahoja diferente del archivo de salida.





### Module 3 – Task 3

## WRITING DATA FILES



Primero, tenemos que definir una función que, dado un marco de datos, aplica el proceso de limpieza de datos entero.

Después, la función llama al proceso para cada hoja.

#### Input [33]:

```
import pandas as pd
import numpy as np

nuts = pd.read_excel('datos/nuts.xlsx', index_col=0)

#function for Data cleaning
def cleanData(data):
    #we eliminate leftover columns
    indices = [i for i, s in enumerate(data.columns) if 'Flags' in s]
    colToDelete=data.columns[np.array(indices)]
    data = data.drop(columns=colToDelete)
    #we change the header and first column
    data.index = nuts.index
    #we change data type and remove no numbers
    data = data[data.columns].apply(pd.to_numeric, errors='coerce')
    #we eliminate duplicate rows
    data = data.loc[~data.index.duplicated(keep='first')]
    #we delete negative values
    data[data < 0] = np.nan
    #we interpolate null values
    data = data.interpolate(axis=1, limit_direction='both')
    #we add zeros to the empty rows
    data = data.fillna(0)
    return data
```

#### Input [34]:

```
reader = pd.ExcelFile('datos/animalEurostatNuts2.xlsx')
writer = pd.ExcelWriter('datos/animalEurostatNuts2_corrected.xlsx')
for sheet in reader.sheet_names():
    data = pd.read_excel(reader, sheet_name=sheet, skiprows=9, skipfooter=13, index_col=0)
    data = cleanData(data)
    data.to_excel(writer, sheet_name=sheet)
writer.save()
print('Done')
```

#### Output [34]:







Done

[Continua... Módulo 3 – Tarea 4](#)

