Cargando Datos de un Excel en Python con Pandas

Guatemala, 21 de mayo de 2021
Pablo Sao Alonzo[1]
1. Departamento de diseño y desarrollo de Software, Solution Design of Centroamerica, Guatemala.

Como vimos en la entrada anterior (Cargando Datos de un CSV en Python con Pandas), pandas es un paquete para Python que nos facilita la manipulación de los datos; sin embargo, dichos datos son cargados en memoria, por lo que la velocidad de respuesta en su uso, dependerá de la cantidad de datos y los recursos del equipo que estemos utilizando

Cargando Datos de un Excel

La función **read_excel** es la función de pandas para cargar datos de un archivo de Excel, donde posee diversos parámetros para manipular los datos al momento de cargarlos en la memoria, donde nos enfocaremos en los principales parámetros que pueden llegar a sernos de utilidad en la práctica (McKinney, 2018; Harrison y Petrou, 2020).

Según la documentación de Pandas, al igual que Harrison y Petrou (2020), internamente esta librería hace uso de los paquetes **xlrd** y **openpyxl** para leer archivos XLS y XLSX, si en dado caso no lo llegáramos a tener instalado, deberá realizarse de forma manual con **pip** o **conda**. Dentro de esta función de lectura de archivos, también podemos especificar con el parámetro **engine**, que paquete utilizar al momento de cargar los datos, teniendo disponibles las siguientes opciones:

- xlrd
- openpyxl
- odf
- pyxlsb

Antes de dar inicio a la explicación del funcionamiento de esta función, es importante recalcar que una hoja de cálculo, no es necesariamente un set de datos y no es requerido que las personas registre la información en un formato determinado, por lo que en algunos casos los tipos de datos no concuerdan con lo que ha registrado la persona o el software que creo la información.

Para poder iniciar la explicación, supondremos que el lector está familiarizado con la sintaxis de Python y con Jupyter Notebook, donde los archivos de datos que estaremos utilizando; así como, el archivo de Jupyter, puede ser accedidos y descargados desde el siguiente repositorio en GitHub: https://github.com/sdesignca/blog-cargando-excel-pandas

Para iniciar, importamos la librería de pandas en nuestro archivo de Jupyter Notebook, donde le colocaremos el alías, o sobre nombre **pd**.

```
import pandas as pd
```

Luego ubicaremos el path o la ruta donde se encuentra nuestro primer archivo de ejemplo "**datos.xlsx**", en mi caso el archivo "***datos.csv***" se encuentra en la misma ubicación que el archivo de Jupyter, por lo que solo necesitamos colocar el nombre del archivo con su extensión como un parámetro de tipo *String* o cadena a la función **read_excel**. Está información se cargará y se asignará a la variable **datos**.

```
datos = pd.read_excel("datos.xlsx")
```

Mostraremos el contenido de la variable datos con la función *head()* del paquete de pandas:

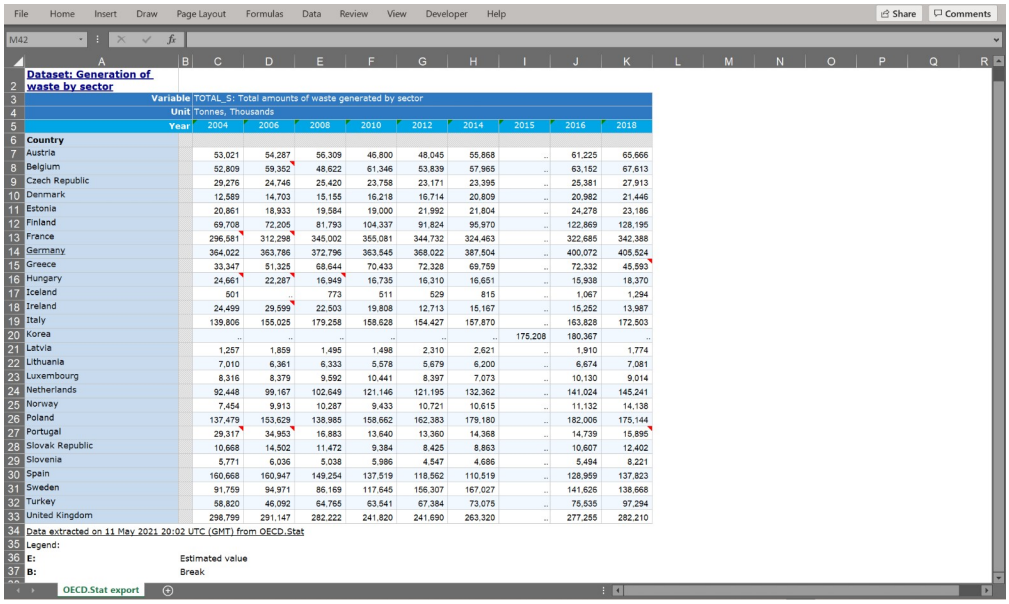
```
datos.head()
```

Como puede observarse, la información es asignada a la variable:

	No.	Apellidos	Nombres	Género	Edad	Fecha de Ingreso	Puesto	Ubicación	Sueldo Base	Bonificación	Total
0	1	Mazariegos	Rene	M	19	2007-01-01	Cajero	105	2400	250	2650
1	2	Ramirez	Gladys	F	35	2008-01-01	Personal de servicio	107	2300	250	2550
2	3	Ortiz	Silvia	F	42	2008-06-01	Gerente T	302	3500	250	3750
3	4	Alvarado	Maria	F	22	2008-07-07	Personal de servicio	107	2300	250	2550
4	5	Perez	Isabel	F	38	2008-12-09	Supervisor	205	2900	250	3150

Sin embargo, como se mencionó anteriormente, una hoja de cálculo no es necesariamente un set de datos, donde podemos tener la necesidad de poder realizar una exploración de información y un análisis a partir de archivos que pueden ser generados por un software o persona, en la cual siempre mantiene su estructura. Para ejemplificar esto utilizaremos el archivo "**WasteBySector.xlsx**", el cual es un reporte generado por "*The Organisation for Economic Co-operation and Development (OECD)*" sobre la generación de desperdicio por país, accesible desde el siguiente enlace: https://stats.oecd.org o disponible en nuestro repositorio en GitHub: https://github.com/sdesignca/blog-cargando-excel-pandas

Al abrir el reporte de Excel podemos observar que posee la siguiente estructura:



Dataset: Generation of waste by sector										
Variable	TOTAL_S: Total amounts of waste generated by sector									
Unit	Tonnes, Thousands									
Year	2004	2006	2008	2010	2012	2014	2015	2016	2018	
Country										
Austria	53,021	54,287	56,309	46,800	48,045	55,868	..	61,225	65,666	
Belgium	52,809	59,352	48,622	61,346	53,839	57,965	..	63,152	67,613	
Czech Republic	29,276	24,746	25,420	23,758	23,171	23,395	..	25,381	27,913	
Denmark	12,589	14,703	15,155	16,218	16,714	20,809	..	20,982	21,446	
Estonia	20,861	18,933	19,584	19,000	21,992	21,804	..	24,278	23,186	
Finland	69,708	72,205	81,793	104,337	91,824	95,970	..	122,869	128,195	
France	296,581	312,298	345,002	355,081	344,732	324,463	..	322,685	342,388	
Germany	364,022	363,786	372,796	363,545	368,022	387,504	..	400,072	405,524	
Greece	33,347	51,325	68,644	70,433	72,328	69,759	..	72,332	45,593	
Hungary	24,661	22,287	16,949	16,735	16,310	16,651	..	15,938	18,370	
Iceland	501	..	773	511	529	815	..	1,067	1,294	
Ireland	24,499	29,599	22,503	19,808	12,713	15,167	..	15,252	13,987	
Italy	139,806	155,025	179,258	158,628	154,427	157,870	..	163,828	172,503	
Korea	..	..	..	..	..	..	175,208	180,367	..	
Latvia	1,257	1,859	1,495	1,498	2,310	2,621	..	1,910	1,774	
Lithuania	7,010	6,361	6,333	5,578	5,679	6,200	..	6,674	7,081	
Luxembourg	8,316	8,379	9,592	10,441	8,397	7,073	..	10,130	9,014	
Netherlands	92,448	99,167	102,649	121,146	121,195	132,362	..	141,024	145,241	
Norway	7,454	9,913	10,287	9,433	10,721	10,615	..	11,132	14,138	
Poland	137,479	153,629	138,985	158,662	162,383	179,180	..	182,006	175,144	
Portugal	29,317	34,953	16,883	13,640	13,360	14,368	..	14,739	15,895	
Slovak Republic	10,668	14,502	11,472	9,384	8,425	8,863	..	10,607	12,402	
Slovenia	5,771	6,036	5,038	5,986	4,547	4,686	..	5,494	8,221	
Spain	160,668	160,947	149,254	137,519	118,562	110,519	..	128,959	137,823	
Sweden	91,759	94,971	86,169	117,645	156,307	167,027	..	141,626	138,668	
Turkey	58,820	46,092	64,765	63,541	67,384	73,075	..	75,535	97,294	
United Kingdom	298,799	291,147	282,222	241,820	241,690	263,320	..	277,255	282,210	

Data extracted on 11 May 2021 20:02 UTC (GMT) from OECD.Stat

Legend:

E: Estimated value

B: Break

En este caso, vemos que el archivo tiene una estructura distinta al del primer ejemplo, donde la información que necesitamos extraer son los países y la cantidad de desperdicio generado por países en los diferentes años, en la cual utilizaremos varios parámetros de la función **read_excel**. Para ello pasaremos como *String* o cadena el nombre del archivo con su extensión, el cual es "**WasteBySector.xlsx**".

En algunos casos la hoja de calculo puede tener varias hojas, por lo que con el parámetro **sheet_name** podemos seleccionar la hoja de la cual deseamos cargar la información, donde podemos pasar el número de la hoja, el nombre de la hoja o una lista de las hojas de las cuales deseamos cargar la información, en nuestra hoja de ejemplo solo existe una por lo que no es necesario colocar el nombre, pero con el fin de ejemplificarlo pasaremos como parámetro el nombre que tiene nuestra hoja: "**OECD.Stat export**".

Al analizar la estructura del Excel, hay secciones que no son de nuestro interés para un análisis. Si empezamos a analizar la parte superior del Excel debemos omitir las primeras cuatro filas, es importante ver que la primera fila esta oculta en nuestro archivo, sin embargo esto no significa que no sea tomada en cuanta por la función al momento de cargar información. Para poder omitir las filas usamos el parámetro **skiprows**, con el valor 4.

	A	B	C	D	E	F
2	**Dataset: Generation of waste by sector**					
3	**Variable**		TOTAL_S: Total amounts of waste generated by			
4	**Unit**		Tonnes, Thousands			
5	**Year**		2004	2006	2008	2010
6	**Country**					
7	Austria		53,021	54,287	56,309	46,800
8	Belgium		52,809	59,352	48,622	61,346
9	Czech Republic		29,276	24,746	25,420	23,758
10	Denmark		12,589	14,703	15,155	16,218

Ahora si vemos la parte inferior de los datos de nuestro Excel, también contiene información que no necesitamos para nuestro análisis, por lo cual utilizaremos el parámetro **skipfooter**, donde enviaremos el valor 4, ya que debemos de omitir las ultimas cuatro filas al momento de cargar nuestra información.

26	Poland	137,479	153,629	138,985	158,662	162,383	179,180	..	182,006	175,144
27	Portugal	29,317	34,953	16,883	13,640	13,360	14,368	..	14,739	15,895
28	Slovak Republic	10,668	14,502	11,472	9,384	8,425	8,863	..	10,607	12,402
29	Slovenia	5,771	6,036	5,038	5,986	4,547	4,686	..	5,494	8,221
30	Spain	160,668	160,947	149,254	137,519	118,562	110,519	..	128,959	137,823
31	Sweden	91,759	94,971	86,169	117,645	156,307	167,027	..	141,626	138,668
32	Turkey	58,820	46,092	64,765	63,541	67,384	73,075	..	75,535	97,294
33	United Kingdom	298,799	291,147	282,222	241,820	241,690	263,320	..	277,255	282,210
34	Data extracted on 11 May 2021 20:02 UTC (GMT) from OECD.Stat									
35	Legend:									
36	**E:**	Estimated value								
37	**B:**	Break								
38										
39										
40										

Por lo que tendríamos a este momento la siguiente instrucción:

```
datos = pd.read_excel("WasteBySector.xlsx",
                      sheet_name='OECD.Stat export',
                      skiprows=4,
                      skipfooter=4,
                      usecols="A,C:K")
```

Ahora con la función *head(30)* mostraremos los datos que se han cargado a nuestra variable datos, es importante mencionar que el valor 30 nos mostrará los primeros 30 registros que contenga nuestra variable, donde hemos utilizado este valor para mostrar que no han sido cargados los datos de las primeras cuatro filas y las ultimas cuatro filas.

```
datos.head(30)
```

Podemos observar en el resultado que obtendremos, aún hay información que debemos ajustar, como cambiar el nombre de la columna "*Year*" (año) por "*Country*" (País), eliminar la primera fila de los datos y la segunda columna ("*Unnamed: 1*").

	Year	Unnamed: 1	2004	2006	2008	2010	2012	2014	2015	2016	2018
0	Country	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Austria	NaN	53020.95	54286.6	56308.77	46799.58	48045.09	55868.3	..	61225.040	65666.13
2	Belgium	NaN	52809.34	59351.72	48621.91	61345.8	53839.47	57965.39	..	63152.380	67613.23
3	Czech Republic	NaN	29275.74	24745.75	25419.7	23757.57	23171.36	23394.96	..	25381.430	27913.45
4	Denmark	NaN	12588.95	14703.14	15155.21	16217.74	16713.82	20808.84	..	20981.930	21445.9
5	Estonia	NaN	20860.68	18932.9	19583.86	19000.2	21992.34	21804.04	..	24277.880	23185.58
6	Finland	NaN	69708.48	72205.48	81792.85	104336.9	91824.2	95969.89	..	122869.200	128195.3
7	France	NaN	296580.9	312297.8	345002.2	355081.3	344731.9	324463	..	322685.300	342387.9
8	Germany	NaN	364021.9	363786.1	372796.3	363545	368022.2	387504.3	..	400071.700	405523.6
9	Greece	NaN	33346.96	51324.66	68643.96	70432.7	72328.28	69758.87	..	72332.350	45592.6
10	Hungary	NaN	24660.92	22287.48	16949.2	16735.42	16310.15	16650.64	..	15938.080	18369.59
11	Iceland	NaN	501.426	..	772.584	510.941	529.351	815.148	..	1067.319	1293.511
12	Ireland	NaN	24499.14	29599.18	22502.82	19807.59	12713.02	15166.83	..	15251.690	13986.76
13	Italy	NaN	139806.1	155025	179257.5	158627.6	154427	157870.3	..	163827.800	172502.8
14	Korea	NaN	..	..	..	..	..	..	175208	180367.000	..
15	Latvia	NaN	1257.225	1858.551	1495.084	1498.2	2309.581	2621.495	..	1909.631	1773.726
16	Lithuania	NaN	7010.178	6361.109	6333.352	5578.134	5678.751	6200.45	..	6674.238	7080.538
17	Luxembourg	NaN	8315.766	8378.911	9592.144	10441.47	8397.228	7072.758	..	10130.080	9014.397
18	Netherlands	NaN	92448.12	99166.56	102648.6	121145.5	121194.5	132362.3	..	141024.000	145241
19	Norway	NaN	7453.565	9913.286	10286.64	9432.997	10720.87	10614.91	..	11131.590	14137.72
20	Poland	NaN	137478.5	153628.9	138984.6	158662	162383	179179.9	..	182005.700	175143.5
21	Portugal	NaN	29317.29	34952.77	16882.92	13640.08	13359.52	14368	..	14739.130	15894.87
22	Slovak Republic	NaN	10668.41	14501.5	11472.01	9384.112	8425.384	8862.778	..	10606.970	12401.87
23	Slovenia	NaN	5770.505	6035.829	5038.401	5986.106	4546.506	4686.417	..	5494.362	8220.679
24	Spain	NaN	160668.1	160946.6	149254.2	137518.9	118561.7	110518.5	..	128958.500	137822.9
25	Sweden	NaN	91759.47	94971.3	86168.59	117645.2	156306.5	167026.9	..	141625.700	138667.6
26	Turkey	NaN	58820.31	46091.63	64764.5	63540.63	67383.77	73075.12	..	75534.650	97294.07
27	United Kingdom	NaN	298798.8	291147.4	282222.1	241820	241690.4	263319.5	..	277255.000	282209.8

Primero usaremos el parámetro **usecols** dentro de la función **reas_excel** para cargar unicamente las columnas don los datos que son de nuestro interés, donde podemos pasar una cadena o *String* con el rango de las columnas o una lista de las columnas que deseamos cargar. En nuestro caso utilizaremos la

opción de *String*, donde indicaremos que cargue unicamente la información letra de la columna **A** y la información de la columna **C** hasta la **K**.

```
datos = pd.read_excel("WasteBySector.xlsx",
                      sheet_name='OECD.Stat export',
                      skiprows=4,
                      skipfooter=4,
                      usecols="A,C:K")
```

Si mostramos los datos asignado a nuestra variable con el comando *head()*

```
datos.head()
```

Se observa que la columna "*Unamed: 1*" ya no fue tomada en cuenta al cargar la información:

	Year	2004	2006	2008	2010	2012	2014	2015	2016	2018
0	Country	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Austria	53,020.95	54,286.6	56,308.77	46,799.58	48,045.09	55,868.3	..	61,225.04	65,666.13
2	Belgium	52,809.34	59,351.72	48,621.91	61,345.8	53,839.47	57,965.39	..	63,152.38	67,613.23
3	Czech Republic	29,275.74	24,745.75	25,419.7	23,757.57	23,171.36	23,394.96	..	25,381.43	27,913.45
4	Denmark	12,588.95	14,703.14	15,155.21	16,217.74	16,713.82	20,808.84	..	20,981.93	21,445.9

Luego cambiaremos el nombre de la columna "*Year*" (año) a "*Country*" (país) con la siguiente instrucción. Donde el parámetro **inplace** con el valor **True**, indica que deseamos que sea cambiado de forma permanente en la información contenida en nuestra variable **datos**, de lo contrario este cambio no será permanente para la siguiente instrucción que realicemos con nuestro set de datos :

```
datos.rename(columns={'Year':'Country'},inplace=True)
```

Al mostrar nuevamente la información en nuestra variable, veremos que se ha renombrado el nombre de la columna.

	Country	2004	2006	2008	2010	2012	2014	2015	2016	2018
0	Country	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Austria	53,020.95	54,286.6	56,308.77	46,799.58	48,045.09	55,868.3	..	61,225.04	65,666.13
2	Belgium	52,809.34	59,351.72	48,621.91	61,345.8	53,839.47	57,965.39	..	63,152.38	67,613.23
3	Czech Republic	29,275.74	24,745.75	25,419.7	23,757.57	23,171.36	23,394.96	..	25,381.43	27,913.45
4	Denmark	12,588.95	14,703.14	15,155.21	16,217.74	16,713.82	20,808.84	..	20,981.93	21,445.9

A este punto no se ha discutido sobre las estructura de *DataFrame* y *Serie* de Pandas; sin embargo, debemos hacer la mención de que ambas estructuras manejan de forma interna una secuencia de indexado de los registros. Es importante mencionarlo, ya que utilizaremos este indexado para la siguiente instrucción que utilizaremos.

	Country	2004	2006	2008	2010	2012	2014	2015	2016	2018
0	Country	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Austria	53,020.95	54,286.6	56,308.77	46,799.58	48,045.09	55,868.3	..	61,225.04	65,666.13
2	Belgium	52,809.34	59,351.72	48,621.91	61,345.8	53,839.47	57,965.39	..	63,152.38	67,613.23
3	Czech Republic	29,275.74	24,745.75	25,419.7	23,757.57	23,171.36	23,394.96	..	25,381.43	27,913.45
4	Denmark	12,588.95	14,703.14	15,155.21	16,217.74	16,713.82	20,808.84	..	20,981.93	21,445.9

Ahora proseguimos con la eliminación de la primera fila:

	Country	2004	2006	2008	2010	2012	2014	2015	2016	2018
0	Country	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Austria	53,020.95	54,286.6	56,308.77	46,799.58	48,045.09	55,868.3	..	61,225.04	65,666.13
2	Belgium	52,809.34	59,351.72	48,621.91	61,345.8	53,839.47	57,965.39	..	63,152.38	67,613.23
3	Czech Republic	29,275.74	24,745.75	25,419.7	23,757.57	23,171.36	23,394.96	..	25,381.43	27,913.45
4	Denmark	12,588.95	14,703.14	15,155.21	16,217.74	16,713.82	20,808.84	..	20,981.93	21,445.9

Para hacer esto, utilizamos la función **drop**, donde utilizaremos el parámetro **labels** con el valor 0, para indicar que son los registros de la primera fila, aquí también puede ser enviada una lista de índices o columnas. Luego indicamos que eliminaremos la filas pasando el valor "***index***" en el parámetro **axis**, si quisiéramos eliminar la columna, deberíamos enviar el parámetro "**columns**" y por último enviaremos en el parámetro **inplace** con el valor **True**, para que el cambio se preserve en la información de nuestra variable **datos**:

```
datos.drop(labels=0,axis="index",inplace=True)
```

Si mostramos los datos con la función *head()*, veremos que se ha eliminado del *DataFrame* la primera fila:

	Country	2004	2006	2008	2010	2012	2014	2015	2016	2018
1	Austria	53,020.95	54,286.6	56,308.77	46,799.58	48,045.09	55,868.3	..	61,225.04	65,666.13
2	Belgium	52,809.34	59,351.72	48,621.91	61,345.8	53,839.47	57,965.39	..	63,152.38	67,613.23
3	Czech Republic	29,275.74	24,745.75	25,419.7	23,757.57	23,171.36	23,394.96	..	25,381.43	27,913.45
4	Denmark	12,588.95	14,703.14	15,155.21	16,217.74	16,713.82	20,808.84	..	20,981.93	21,445.9
5	Estonia	20,860.68	18,932.9	19,583.86	19,000.2	21,992.34	21,804.04	..	24,277.88	23,185.58

Ahora reindexaremos los datos del *DataFrame* asignado a nuestra variable **datos** utilizando la función **reset_index**. En esta función utilizaremos el parámetro **drop** con el valor **True**, si no utilizamos esta propiedad de la función, al momento de realizar el reindexado, la secuencia de indices anteriores no sera eliminado y será creado como una nueva columna dentro de nuestro *DataFrame*, por lo que esta propiedad elimina por completo la secuencia de índices que vamos a remplazar. Y por último agregaremos el parámetro **inplace** con el valor True para que se preserven los cambios en la estructura de nuestra variable **datos**:

```
datos.reset_index(drop=True,inplace=True)
```

Utilizaremos la función info() para poder obtener la información de la memoria utilizada, la cantidad de columnas y el detalle sobre las columnas de nuestro *DataFrame*, en la cual se mostrará el índice, el nombre, la cantidad de datos no nulos y el tipo de dato de las columnas.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27 entries, 0 to 26
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Country  27 non-null     object 
 1   2004     27 non-null     object 
 2   2006     27 non-null     object 
 3   2008     27 non-null     object 
 4   2010     27 non-null     object 
 5   2012     27 non-null     object 
 6   2014     27 non-null     object 
 7   2015     27 non-null     object 
 8   2016     27 non-null     float64
 9   2018     27 non-null     object 
dtypes: float64(1), object(9)
memory usage: 2.2+ KB
```

Podemos observar en el resultado que de las nueve columnas con datos decimales (flotantes o *floats*), solo uno fue reconocido como tipo *float64;* donde el resto de columnas, incluyendo la del nombre de

los países es del tipo *object* (objeto). Por lo que ahora nos corresponde realizar la transformación del tipo de dato de las columnas de datos numéricos, la cual es posible con la función **to_numeric**, donde el primer argumento corresponde a los datos que deseamos convertir y el segundo parámetro que utilizaremos es **errors** con el valor **"coerce"**, el cual indicará si hay un error al hacer la conversión, sea colocado un valor *NaN* (nulo)

```
for columna in datos.columns[1:]:
    datos[columna] = pd.to_numeric(datos[columna],
                                   errors='coerce'
                                   )
```

Hay que hacer mención que la propiedad **columns** nos permite acceder al listado de las columnas del *DataFrame*, en este caso para convertir los datos a decimales o *floats*, omitimos la primera columna, por lo que indicamos que sean tomadas desde la segunda columna (con el valor de índice 1), hasta la ultima columna existente, quedando la restricción de la siguiente forma **[1:]**

Al obtener nuevamente la información del *DataFrame* con la función *info()*, observaremos que las columnas de datos numéricos ya es de tipo *float*, y nos da un dato más real de los datos que no son nulos en cada una de las columnas.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27 entries, 1 to 27
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Country  27 non-null     object 
 1   2004     26 non-null     float64
 2   2006     25 non-null     float64
 3   2008     26 non-null     float64
 4   2010     26 non-null     float64
 5   2012     26 non-null     float64
 6   2014     26 non-null     float64
 7   2015     1 non-null      float64
 8   2016     27 non-null     float64
 9   2018     26 non-null     float64
dtypes: float64(9), object(1)
memory usage: 2.3+ KB
```

Luego de todo este proceso de transformación, nuestro set de datos queda de la siguiente forma:

	Country	2004	2006	2008	2010	2012	2014	2015	2016	2018
0	Austria	53,020.95	54,286.60	56,308.77	46,799.58	48,045.09	55,868.30	NaN	61,225.04	65,666.13
1	Belgium	52,809.34	59,351.72	48,621.91	61,345.80	53,839.47	57,965.39	NaN	63,152.38	67,613.23
2	Czech Republic	29,275.74	24,745.75	25,419.70	23,757.57	23,171.36	23,394.96	NaN	25,381.43	27,913.45
3	Denmark	12,588.95	14,703.14	15,155.21	16,217.74	16,713.82	20,808.84	NaN	20,981.93	21,445.90
4	Estonia	20,860.68	18,932.90	19,583.86	19,000.20	21,992.34	21,804.04	NaN	24,277.88	23,185.58
5	Finland	69,708.48	72,205.48	81,792.85	104,336.90	91,824.20	95,969.89	NaN	122,869.20	128,195.30
6	France	296,580.90	312,297.80	345,002.20	355,081.30	344,731.90	324,463.00	NaN	322,685.30	342,387.90
7	Germany	364,021.90	363,786.10	372,796.30	363,545.00	368,022.20	387,504.30	NaN	400,071.70	405,523.60
8	Greece	33,346.96	51,324.66	68,643.96	70,432.70	72,328.28	69,758.87	NaN	72,332.35	45,592.60
9	Hungary	24,660.92	22,287.48	16,949.20	16,735.42	16,310.15	16,650.64	NaN	15,938.08	18,369.59
10	Iceland	501.43	NaN	772.58	510.94	529.35	815.15	NaN	1,067.32	1,293.51
11	Ireland	24,499.14	29,599.18	22,502.82	19,807.59	12,713.02	15,166.83	NaN	15,251.69	13,986.76
12	Italy	139,806.10	155,025.00	179,257.50	158,627.60	154,427.00	157,870.30	NaN	163,827.80	172,502.80
13	Korea	NaN	NaN	NaN	NaN	NaN	NaN	175,208.00	180,367.00	NaN
14	Latvia	1,257.22	1,858.55	1,495.08	1,498.20	2,309.58	2,621.49	NaN	1,909.63	1,773.73
15	Lithuania	7,010.18	6,361.11	6,333.35	5,578.13	5,678.75	6,200.45	NaN	6,674.24	7,080.54
16	Luxembourg	8,315.77	8,378.91	9,592.14	10,441.47	8,397.23	7,072.76	NaN	10,130.08	9,014.40
17	Netherlands	92,448.12	99,166.56	102,648.60	121,145.50	121,194.50	132,362.30	NaN	141,024.00	145,241.00
18	Norway	7,453.56	9,913.29	10,286.64	9,433.00	10,720.87	10,614.91	NaN	11,131.59	14,137.72
19	Poland	137,478.50	153,628.90	138,984.60	158,662.00	162,383.00	179,179.90	NaN	182,005.70	175,143.50
20	Portugal	29,317.29	34,952.77	16,882.92	13,640.08	13,359.52	14,368.00	NaN	14,739.13	15,894.87
21	Slovak Republic	10,668.41	14,501.50	11,472.01	9,384.11	8,425.38	8,862.78	NaN	10,606.97	12,401.87
22	Slovenia	5,770.51	6,035.83	5,038.40	5,986.11	4,546.51	4,686.42	NaN	5,494.36	8,220.68
23	Spain	160,668.10	160,946.60	149,254.20	137,518.90	118,561.70	110,518.50	NaN	128,958.50	137,822.90
24	Sweden	91,759.47	94,971.30	86,168.59	117,645.20	156,306.50	167,026.90	NaN	141,625.70	138,667.60
25	Turkey	58,820.31	46,091.63	64,764.50	63,540.63	67,383.77	73,075.12	NaN	75,534.65	97,294.07
26	United Kingdom	298,798.80	291,147.40	282,222.10	241,820.00	241,690.40	263,319.50	NaN	277,255.00	282,209.80

Referencias

Harrison, M., y Petrou, T. (2020). *Pandas 1.x: coockbook*. 2da. Edicion. Birmingham, U.K: Packt Publishing. 86 – 93 pp.

McKinney, W. (2018). *Python for Data Analysis*. 2 da edicion. California, E.E.U.U.: . 186 – 187 pp.