

Módulo 2 – Obtención y Preparación de Datos

# Obtención y Preparación de Datos

Ciencia de Datos

# Objetivos

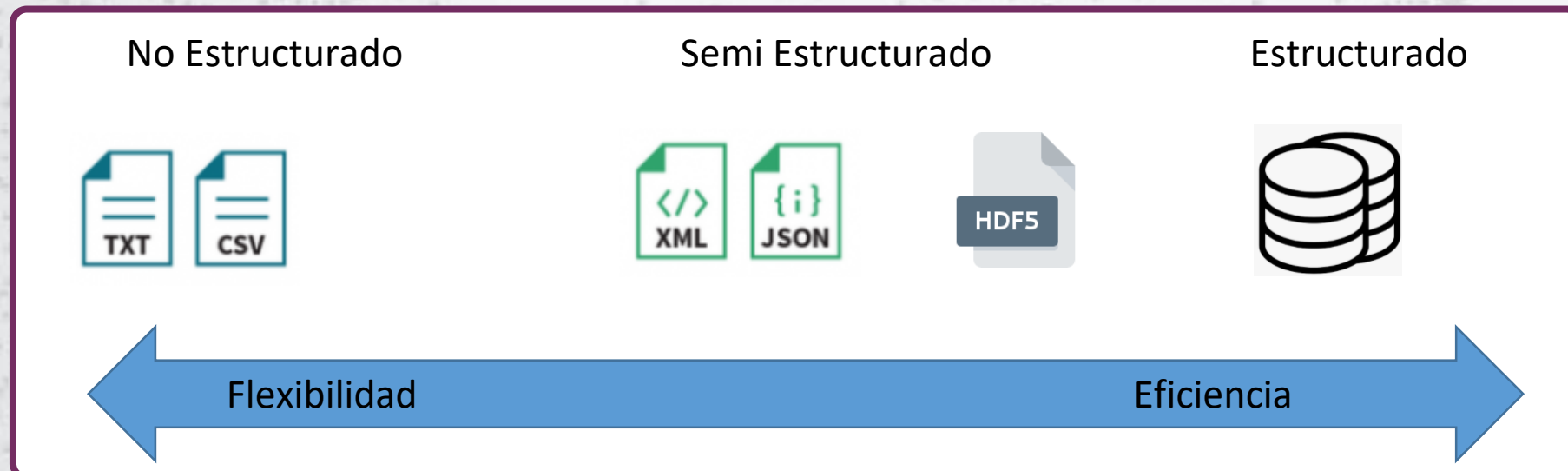


- Fuentes de datos e intercambio.
- Pandas I/O.
- Lectura y escritura de archivos de Texto Plano.
- Lectura y escritura de planillas Excel.
- Lectura de páginas web.
- Lectura de Bases de Datos.
- Lectura de una API.
- Web Scraping.

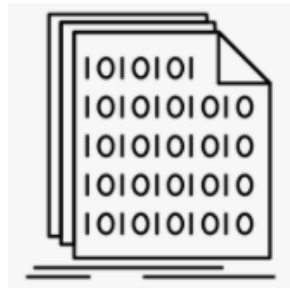
# Fuentes de datos e intercambio

La información y los datos se encuentran diseminados de múltiples formas. Algunas de carácter estructurado (por ejemplo, bases de datos, archivos binarios, servicios de datos o APIs) y otras de carácter no estructurado (por ejemplo, archivos de texto).

Por otra parte, los mecanismos para el intercambio de datos son múltiples, por ejemplo, intercambio de archivos de texto plano, bases de datos compartidas, sistemas de mensajería asíncrona, servicios de datos (RPC, servicios web, APIs REST).



# Archivos de Texto Plano y Binarios



Todos los archivos de computación se componen de pequeñas piezas de información llamada **bits**. En un archivo ASCII, cada byte se correlaciona directamente con un carácter específico, según lo definido con la tabla ASCII estándar. Es por esto que este tipo de archivo es legible fácilmente por una persona.

Los archivos de texto plano están compuestos únicamente por texto sin formato, sólo caracteres, los cuales se pueden codificar de distintos modos dependiendo de la lengua usada. Este tipo de archivo puede ser abierto por un editor de texto cualquiera.

Por otro lado, los archivos llamados binarios, su contenido no representa necesariamente una codificación ASCII, sino que puede almacenar bytes que puede representar la información de la forma en que un determinado programa elija.

# Formatos de Archivo de Texto

Los archivos de texto son un mecanismo utilizado frecuentemente para el traspaso de información. El formato que puede tener un archivo puede ser diverso. A continuación, algunos ejemplos:

- Archivos de un registro por línea.
- Archivos con valores separados por coma (csv).
- Archivos con valores separados por tabulación.
- Archivos con valores de largo fijo.

Archivos con estructura:

- Archivos XML
- Archivos JSON

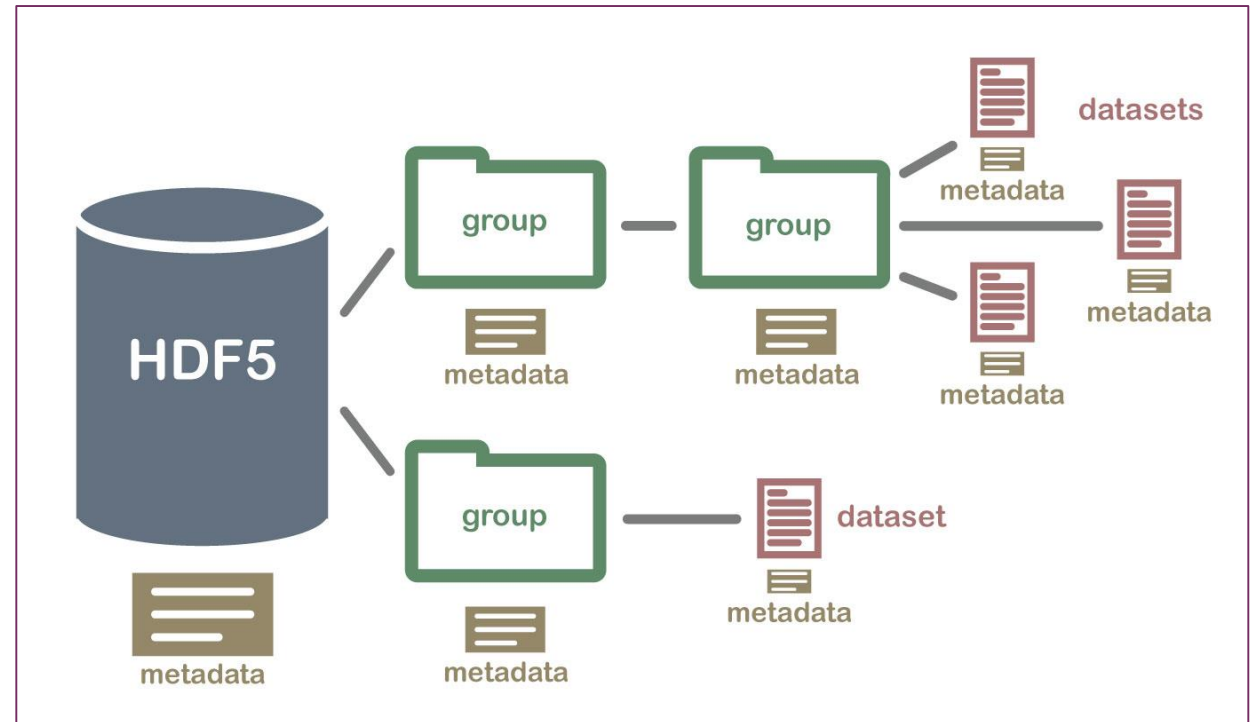
# Formato JSON

El formato JSON almacenan estructuras de dato estilo JavaScript Object Notation y son utilizados comúnmente en los servicios REST API para el intercambio de información.

```
1 {
2   "users": [
3     {
4       "userId": 1,
5       "firstName": "Chris",
6       "lastName": "Lee",
7       "phoneNumber": "555-555-5555",
8       "emailAddress": "clee@fileinfo.com"
9     },
10    {
11      "userId": 2,
12      "firstName": "Action",
13      "lastName": "Jackson",
14      "phoneNumber": "555-555-5556",
15      "emailAddress": "ajackson@fileinfo.com"
16    },
17    {
18      "userId": 3,
19      "firstName": "Ross",
20      "lastName": "Bing",
21      "phoneNumber": "555-555-5557",
22      "emailAddress": "rbing@fileinfo.com"
23    },
24    {
25      "userId": 4,
26      "firstName": "David",
27      "lastName": "Reeves",
28      "phoneNumber": "555-555-5558",
29      "emailAddress": "dreeves@fileinfo.com"
30    },
31    {
32      "userId": 5,
33      "firstName": "Josie",
34      "lastName": "Mac",
35      "phoneNumber": "555-555-5559",
36      "emailAddress": "jmac@fileinfo.com"
37    }
38  ]
39 }
```

# Archivos HDF5

**Hierarchical Data Format v5** es un formato de archivo binario que permite el almacenamiento de datos grandes, complejos y heterogéneos con una estructura jerárquica de tipo directorio, y con meta data descriptiva, todo en un solo archivo.



# Pandas I/O

# Pandas I/O



La librería Pandas provee de soporte para la lectura y escritura de información desde diversas fuentes, de forma simple y flexible. Dentro de las cuales se encuentran las siguientes:

- CSV
- Excel
- HTML
- SQL
- HDF5
- Otras



# Entrada y Salida de Información

## Archivos de Texto

Format Type	Data Description	Reader	Writer
text	CSV	<code>read_csv</code>	<code>to_csv</code>
text	Fixed-Width Text File	<code>read_fwf</code>	
text	JSON	<code>read_json</code>	<code>to_json</code>
text	HTML	<code>read_html</code>	<code>to_html</code>
text	Local clipboard	<code>read_clipboard</code>	<code>to_clipboard</code>
	MS Excel	<code>read_excel</code>	<code>to_excel</code>

# Entrada y Salida de Información

## Archivos de Texto

binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle

# Entrada y Salida de Información

## Formato SQL

SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

# Archivos de Texto Plano

# Leyendo un archivo CSV

Header  
(nombre de las columnas)

- El siguiente, es un archivo de texto plano con formato CSV (Comma Separated Value). En este formato, una línea equivale a un registro de información. Habitualmente, la primera fila contiene el nombre de las columnas. Más información en el siguiente enlace:

[https://en.wikipedia.org/wiki/Comma-separated\\_values](https://en.wikipedia.org/wiki/Comma-separated_values)

```
1  colegio,sexo,edad,ausencias,N1,N2,N3
2  GP,F,18,6,1.8,2.1,2.1
3  GP,F,17,4,1.8,1.8,2.1
4  GP,F,15,10,2.4,2.8,3.5
5  GP,F,15,2,5.2,4.9,5.2
6  GP,F,16,4,2.1,3.5,3.5
7  GP,M,16,10,5.2,5.2,5.2
8  GP,M,16,0,4.2,4.2,3.9
9  GP,F,17,6,2.1,1.8,2.1
10 GP,M,15,0,5.6,6.3,6.6
11 GP,M,15,0,4.9,5.2,5.2
12 GP,F,15,0,3.5,2.8,3.2
```

# Leyendo un Archivo CSV

La función `read_csv()` realiza la lectura y retorna un objeto de tipo DataFrame. En este caso, el archivo se encuentra en el mismo directorio de trabajo que el notebook, pero en caso de no ser así, puede proporcionarse la ruta absoluta al archivo.

```
df = pd.read_csv('notas-alumnos.csv')  
df
```

	colegio	sexo	edad	ausencias	N1	N2	N3
0	GP	F	18	6	1.8	2.1	2.1
1	GP	F	17	4	1.8	1.8	2.1
2	GP	F	15	10	2.4	2.8	3.5
3	GP	F	15	2	5.2	4.9	5.2
4	GP	F	16	4	2.1	3.5	3.5
...	...	...	...	...	...	...	...
390	MS	M	20	11	3.2	3.2	3.2
391	MS	M	17	3	4.9	5.6	5.6
392	MS	M	21	3	3.5	2.8	2.4
393	MS	M	18	0	3.9	4.2	3.5
394	MS	M	19	5	2.8	3.2	3.2

El retorno es un DataFrame  
Pandas

395 rows × 7 columns

# Especificando el Separador de Campos

En oportunidades, los archivos pueden venir con otro separador distinto a la comma (,) En el siguiente ejemplo, el caracter de separación corresponde a la tabulación, por lo tanto, se puede especificar el parámetro `sep='\t'`

```
1  "Gender"    "Height"    "Weight"
2  "Male"      73.847017017515  241.893563180437
3  "Male"      68.7819040458903  162.310472521300
4  "Male"      74.1101053917849  212.74085555565
5  "Male"      71.7309784033377  220.042470303077
6  "Male"      69.8817958611153  206.349800623871
7  "Male"      67.2530156878065  152.212155757083
8  "Male"      68.7850812516616  183.927888604031
9  "Male"      68.3485155115879  167.971110489509
10 "Male"      67.018949662883  175.92944039571
11 "Male"      63.4564939783664  156.399676387112
12 "Male"      71.1953822829745  186.604925560358
13 "Male"      71.6408051192206  213.741169489411
14 "Male"      64.7663291334055  167.127461073476
15 "Male"      69.2830700967204  189.446181386738
16 "Male"      69.2437322298112  186.434168021239
17 "Male"      67.6456197004212  172.186930058117
18 "Male"      72.4183166259878  196.028506330482
19 "Male"      63.974325721061  172.883470208780
```



```
df = pd.read_csv('weight-height.txt', sep='\t')
df
```

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801
...	...	...	...
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461
9999	Female	61.944246	113.649103

# Archivos sin Header

Cuando el archivo de datos no trae en la primera fila los nombres de las columnas, es necesario especificar el parámetro **header** para así no asignar los datos de la primera fila como nombres de columnas. Asimismo, el parámetro **columns** permite especificar los nombres de cada columna.

```
1 GP,F,18,6,1.8,2.1,2.1
2 GP,F,17,4,1.8,1.8,2.1
3 GP,F,15,10,2.4,2.8,3.5
4 GP,F,15,2,5.2,4.9,5.2
5 GP,F,16,4,2.1,3.5,3.5
6 GP,M,16,10,5.2,5.2,5.2
7 GP,M,16,0,4.2,4.2,3.9
8 GP,F,17,6,2.1,1.8,2.1
9 GP,M,15,0,5.6,6.3,6.6
```

```
df = pd.read_csv('notas-alumnos-sin-header.csv', header=None,
               columns=['colegio','sexo','edad','ausencias','N1','N2','N3'])
df
```

	colegio	sexo	edad	ausencias	N1	N2	N3
0	GP	F	18	6	1.8	2.1	2.1
1	GP	F	17	4	1.8	1.8	2.1
2	GP	F	15	10	2.4	2.8	3.5
3	GP	F	15	2	5.2	4.9	5.2
4	GP	F	16	4	2.1	3.5	3.5

El listado de nombres debe ser consistente

# Más Información

## CSV & text files

The workhorse function for reading text files (a.k.a. flat files) is `read_csv()`. See the [cookbook](#) for some advanced strategies.

## Parsing options

`read_csv()` accepts the following common arguments:

### Basic ¶

**filepath\_or\_buffer** : *various*

Either a path to a file (a `str`, `pathlib.Path`, or `py._path.local.LocalPath`), URL (including http, ftp, and S3 locations), or any object with a `read()` method (such as an open file or `StringIO`).

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

### ☰ On this page

#### CSV & text files

##### Parsing options

- Basic
- Column and index locations and names
- General parsing configuration
- NA and missing data handling
- Datetime handling
- Iteration
- Quoting, compression, and file format
- Error handling
- Specifying column data types
- Specifying categorical dtype
- Naming and using columns
- Duplicate names parsing
- Comments and empty lines
- Dealing with Unicode data

# Leyendo un Archivo CSV

Para escribir un Dataframe en un archivo csv, sólo debemos utilizar el método `to_csv()`.

```
In [6]: df = pd.DataFrame({'col1':[1,2,3,4],  
                           'col2':[444,555,666,444],  
                           'col3':['abc','def','ghi','xyz']})
```

```
In [7]: df
```

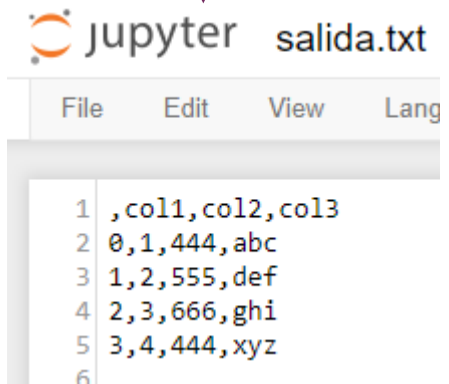
```
Out[7]:
```

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

```
In [8]: df.to_csv('salida.txt')
```

Y se genera el siguiente archivo en el directorio de trabajo.

↓



The screenshot shows a Jupyter Notebook window with a menu bar (File, Edit, View, Lang) and a text area containing the following content:


```
1 ,col1,col2,col3  
2 0,1,444,abc  
3 1,2,555,def  
4 2,3,666,ghi  
5 3,4,444,xyz  
6
```

# Escribiendo un archivo CSV

Si queremos que no se guarde en el archivo csv, el índice del dataframe, entonces debemos usar el parámetro `index=False`.

```
In [8]: df.to_csv('salida-sin-indice.txt', index=False)
```



 jupyter salida-sin-indice.txt

File Edit View Language

```
1 col1,col2,col3
2 1,444,abc
3 2,555,def
4 3,666,ghi
5 4,444,xyz
6
```

# Leyendo un Archivo con Largo Fijo

En un archivo de largo fijo, cada fila corresponde a un registro y debemos conocer las posiciones en que vienen los campos. A continuación, un ejemplo:

```
1 id8141 360.242940 149.910199 11950.7
2 id1594 444.953632 166.985655 11788.4
3 id1849 364.136849 183.628767 11806.2
4 id1230 413.836124 184.375703 11916.8
5 id1948 502.953953 173.237159 12468.3
6
```

Especificamos tuplas con índice inicio y fin de cada campo

```
# Leer archivo de largo fijo
specs = [(0, 6), (8, 20), (21, 33), (34, 43)]
names = ['ID', 'Ingresos', 'Egresos', 'Inversion']
df = pd.read_fwf('report-fwf.dat.txt', colspecs=specs, header=None, names=names)
df
```

	ID	Ingresos	Egresos	Inversion
0	id8141	360.242940	149.910199	11950.7
1	id1594	444.953632	166.985655	11788.4
2	id1849	364.136849	183.628767	11806.2
3	id1230	413.836124	184.375703	11916.8
4	id1948	502.953953	173.237159	12468.3

# Leyendo un Archivo Excel

# Archivos Excel

La librería Pandas puede leer y escribir archivos en formato Excel.

✓ Para archivos **.xlsx** modernos se recomienda instalar la librería **openpyxl**.

```
pip install openpyxl
```

✓ Para escribir archivos también se puede utilizar **xlsxwriter**.

```
pip install xlsxwriter
```

	A	B	C	D	E	F
1	_id	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	LUGAR DE SU FUNCION	SUELDO LIQUIDO
2	1	Cecilia Del Carmen Ayala	Enfermera	Encargada Del Cecosf	Cecosf Padre Hugo Cornelissen	1.803.344
3	2	Jesús Ignacio Contreras Viv	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacar	Farmacia/Coordinación	236.489
4	3	Carolina Andrea Estay Pang	T. Enfermería	Preparación De Pacientes/Coordinación	Farmacia/Coordinación	664.647
5	4	Jorge Eduardo García Lagos	Odontólogo	Encargado De Reas	Unidad Dental	1.279.353
6	5	Carolina Lissett Gómez Mo	Conductor	Estafeta Y Conductor	Cecosf Padre Hugo Cornelissen	255.036
7	6	Eugenio Leonardo Hidalgo	Chefer	Estafeta Y Conductor	Cecosf Padre Hugo Cornelissen	215.282

```
df = pd.read_excel('sueldos.xlsx')  
df.head(2)
```

	_id	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	LUGAR DE SU FUNCION	SUELDO LIQUIDO
0	1	Cecilia Del Carmen Ayala Cabrera	Enfermera	Encargada Del Cecosf	Cecosf Padre Hugo Cornelissen	1.803.344
1	2	Jesús Ignacio Contreras Vivar	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacam	Farmacia/Coordinación	236.489

# Archivos Excel

Con el parámetro **sheet\_name** se puede especificar la hoja a leer.

```
df = pd.read_excel('sueldos.xlsx', sheet_name='Hoja Principal')  
df.head(2)
```

	_id	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	LUGAR DE SU FUNCION	SUELDO LIQUIDO
0	1	Cecilia Del Carmen Ayala Cabrera	Enfermera	Encargada Del Cecosf	Cecosf Padre Hugo Cornelissen	1.803.344
1	2	Jesús Ignacio Contreras Vivar	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacam	Farmacia/Coordinación	236.489


	A	B	C	D	
1	_id	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	
2	1	Cecilia Del Carmen Ayala	Enfermera	Encargada Del Cecosf	C
3	2	Jesús Ignacio Contreras Viv	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacam E	
Hoja Principal					
Listo					

# Archivos Excel

Para escribir un **dataframe** en un archivo Excel, utilizamos la función **to\_excel()**. Requiere que se instale la librería mediante el comando **conda install xlwt**.

```
# escribir una planilla  
df[['NOMBRE', 'SUELDO LIQUIDO']].to_excel('resumen_sueldos.xlsx')
```

Por defecto se  
almacena el índice  
del dataframe



	A	B	C	D
1		<b>NOMBRE</b>	<b>SUELDO LIQUIDO</b>	
2	0	Cecilia Del Carmen Ayala Cabrera	1.803.344	
3	1	Jesús Ignacio Contreras Vivar	236.489	
4	2	Carolina Andrea Estay Pangué	664.647	
5	3	Jorge Eduardo García Lagos	1.279.353	
6	4	Carolina Lissett Gómez Morales	255.036	
7	5	Eugenio Leonardo Hidalgo Araya	315.293	

Sheet1

# Archivos Excel

En este ejemplo se especifica que el índice no será almacenado, asimismo, se asigna un nombre a la hoja.

```
# escribir una planilla sin indice y con nombre de hoja  
df[['NOMBRE', 'SUELDO LIQUIDO']].to_excel('resumen_sueldos.xlsx', index=False,  
                                           sheet_name='Hoja Resumen')
```



	A	B	C
1	<b>NOMBRE</b>	<b>SUELDO LIQUIDO</b>	
2	Cecilia Del Carmen Ayala Cabrera	1.803.344	
3	Jesús Ignacio Contreras Vivar	236.489	
4	Carolina Andrea Estay Pangue	664.647	
5	Jorge Eduardo García Lagos	1.279.353	
6	Carolina Lissett Gómez Morales	255.036	
7	Eugenio Leonardo Hidalgo Araya	315.293	

Hoja Resumen

# Leyendo una Tabla HTML

# Tablas HTML

Extraeremos un **dataset** a partir de una tabla en la siguiente página web: [https://es.wikipedia.org/wiki/Regiones\\_de\\_Chile](https://es.wikipedia.org/wiki/Regiones_de_Chile)

es.wikipedia.org/wiki/Regiones\_de\_Chile

WhatsApp generatedata.com

## Regiones de Odeplan y DL 575 [\[ editar \]](#)

La **ODEPLAN** en los años 1960 comenzó a elaborar un programa de desarrollo regional que obedeció a un criterio de regionalización del territorio. Una circular del **presidente de la República** de 2 de enero de 1966, estableciendo una división geoeconómica del país, agrupando las **provincias** permitieran convertirse en polos de desarrollo económico y social. Cada región contaba con un centro urbano que sería el eje de su desarrollo.<sup>3</sup> n.º 1104 de 1969, del Ministerio del Interior.<sup>3</sup>

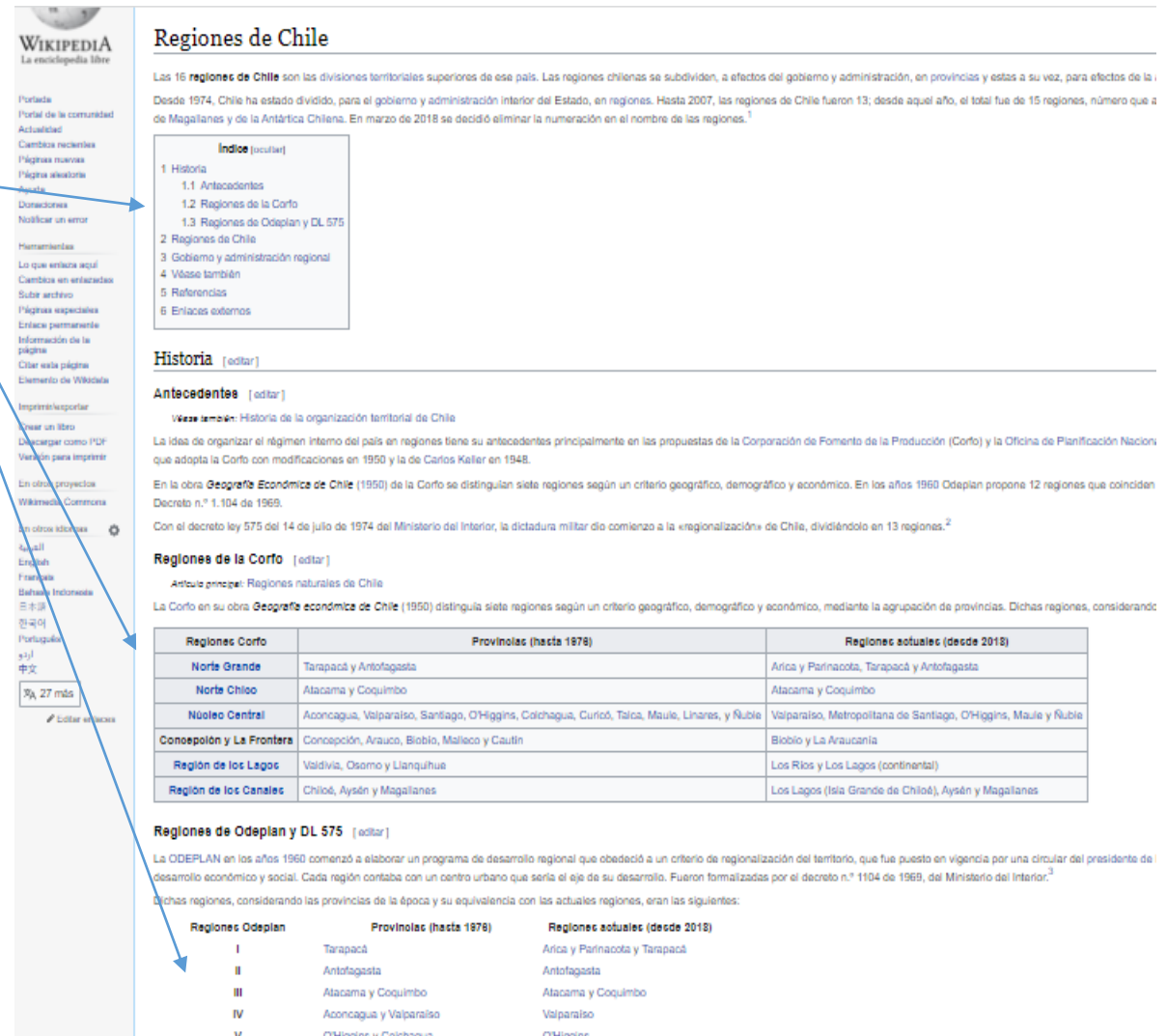
Dichas regiones, considerando las provincias de la época y su equivalencia con las actuales regiones, eran las siguientes:

Regiones Odeplan	Provincias (hasta 1976)	Regiones actuales (desde 2018)
I	Tarapacá	Arica y Parinacota y Tarapacá
II	Antofagasta	Antofagasta
III	Atacama y Coquimbo	Atacama y Coquimbo
IV	Aconcagua y Valparaíso	Valparaíso
V	O'Higgins y Colchagua	O'Higgins
VI	Curicó, Talca, Maule y Linares	Maule
VII	Ñuble, Concepción, Arauco, Biobío y Malleco	Ñuble, Biobío y La Araucanía (norte)
VIII	Cautín	La Araucanía (sur)
IX	Valdivia y Osorno	Los Ríos y Los Lagos (norte)
X	Llanquihue, Chiloé y Aysén	Los Lagos (sur) y Aysén
XI	Magallanes	Magallanes
<b>Zona Metropolitana de Santiago</b>	<b>Santiago</b>	Metropolitana de Santiago

## Regiones de Chile [\[ editar \]](#)

# Tablas HTML

Una página web podría contener más de una tabla de datos.



**Regiones de Chile**

Las 15 **regiones de Chile** son las divisiones territoriales superiores de ese país. Las regiones chilenas se subdividen, a efectos del gobierno y administración, en provincias y estas a su vez, para efectos de la i

Desde 1974, Chile ha estado dividido, para el gobierno y administración interior del Estado, en regiones. Hasta 2007, las regiones de Chile fueron 13; desde aquel año, el total fue de 15 regiones, número que a de Magallanes y de la Antártica Chilena. En marzo de 2018 se decidió eliminar la numeración en el nombre de las regiones.<sup>1</sup>

Índice [ocultar]
1 Historia
1.1 Antecedentes
1.2 Regiones de la Corfo
1.3 Regiones de Odeplan y DL 575
2 Regiones de Chile
3 Gobierno y administración regional
4 Véase también
5 Referencias
6 Enlaces externos

**Historia** [editar]

**Antecedentes** [editar]

**Véase también:** Historia de la organización territorial de Chile

La idea de organizar el régimen interno del país en regiones tiene su antecedentes principalmente en las propuestas de la Corporación de Fomento de la Producción (Corfo) y la Oficina de Planificación Nacion que adopta la Corfo con modificaciones en 1950 y la de Carlos Kellier en 1948.

En la obra **Geografía Económica de Chile** (1950) de la Corfo se distingulan siete regiones según un criterio geográfico, demográfico y económico. En los años 1960 Odeplan propone 12 regiones que coinciden Decreto n.º 1.104 de 1969.

Con el decreto ley 575 del 14 de julio de 1974 del Ministerio del Interior, la dictadura militar dio comienzo a la «regionalización» de Chile, dividiéndolo en 13 regiones.<sup>2</sup>

**Regiones de la Corfo** [editar]

**Artículo principal:** Regiones naturales de Chile

La Corfo en su obra **Geografía económica de Chile** (1950) distingue siete regiones según un criterio geográfico, demográfico y económico, mediante la agrupación de provincias. Dichas regiones, considerando

Regiones Corfo	Provincias (hasta 1978)	Regiones actuales (desde 2018)
<b>Norte Grande</b>	Tarapacá y Antofagasta	Arica y Parícuta, Tarapacá y Antofagasta
<b>Norte Chico</b>	Atacama y Coquimbo	Atacama y Coquimbo
<b>Núcleo Central</b>	Aconcagua, Valparaíso, Santiago, O'Higgins, Colchagua, Curicó, Talca, Maule, Linares, y Ñuble	Valparaíso, Metropolitana de Santiago, O'Higgins, Maule y Ñuble
<b>Concepción y La Frontera</b>	Concepción, Arauco, Biobío, Malleco y Cautín	Biobío y La Araucanía
<b>Región de los Lagos</b>	Valdivia, Osorno y Llanquihue	Los Ríos y Los Lagos (continental)
<b>Región de los Canales</b>	Chiloé, Aysén y Magallanes	Los Lagos (Isla Grande de Chiloé), Aysén y Magallanes

**Regiones de Odeplan y DL 575** [editar]

La ODEPLAN en los años 1960 comenzó a elaborar un programa de desarrollo regional que obedeció a un criterio de regionalización del territorio, que fue puesto en vigencia por una circular del presidente del desarrollo económico y social. Cada región contaba con un centro urbano que sería el eje de su desarrollo. Fueron formalizadas por el decreto n.º 1104 de 1969, del Ministerio del Interior.<sup>3</sup>

Dichas regiones, considerando las provincias de la época y su equivalencia con las actuales regiones, eran las siguientes:

Regiones Odeplan	Provincias (hasta 1978)	Regiones actuales (desde 2018)
I	Tarapacá	Arica y Parícuta y Tarapacá
II	Antofagasta	Antofagasta
III	Atacama y Coquimbo	Atacama y Coquimbo
IV	Aconcagua y Valparaíso	Valparaíso
V	O'Higgins y Colchagua	O'Higgins

# Tablas HTML

En este caso, el elemento índice 1 del listado es el que contiene el dataframe con información.

```
url = 'https://es.wikipedia.org/wiki/Regiones_de_Chile'
tables = pd.read_html(url)
```

```
tables[1]
```

	Regiones Odeplan	Provincias (hasta 1976)	Regiones actuales (desde 2018)
0	I	Tarapacá	Arica y Parinacota y Tarapacá
1	II	Antofagasta	Antofagasta
2	III	Atacama y Coquimbo	Atacama y Coquimbo
3	IV	Aconcagua y Valparaíso	Valparaíso
4	V	O'Higgins y Colchagua	O'Higgins
5	VI	Curicó, Talca, Maule y Linares	Maule
6	VII	Ñuble, Concepción, Arauco, Biobío y Malleco	Ñuble, Biobío y La Araucanía (norte)
7	VIII	Cautín	La Araucanía (sur)
8	IX	Valdivia y Osorno	Los Ríos y Los Lagos (norte)
9	X	Llanquihue, Chiloé y Aysén	Los Lagos (sur) y Aysén
10	XI	Magallanes	Magallanes

# Leyendo una Base de Datos SQL

# SQL

La librería Pandas, también puede trabajar con tablas en una base de datos. Podemos utilizar conexiones a distintos motores de datos, por ejemplo, **Postgres**. En este ejemplo, nos conectaremos a una base de datos en memoria que se llama **SQLite**.

```
from sqlalchemy import create_engine
```

```
# creamos una conexion  
conn = create_engine('sqlite:///memory:')
```

Importamos la librería específica para el motor de datos.

Creamos una conexión a la base de datos.

Dependiendo del motor de datos puede ser necesario instalar librerías adicionales.

# SQL

Con esta instrucción llevamos un Dataframe a una tabla en la base de datos.

```
In [31]: df.to_sql('mi_tabla',engine)
```

Y de esta forma podemos leer una tabla en la base de datos.

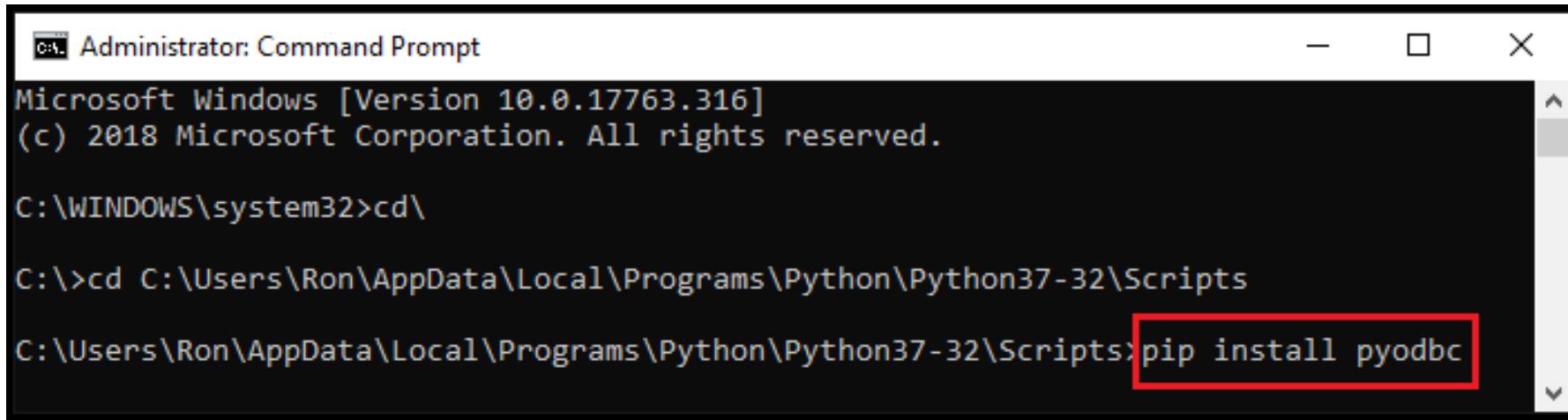
```
In [32]: pd.read_sql('mi_tabla',con=engine)
```

Out[32]:

	index	col1	col2	col3
0	0	1	444	abc
1	1	2	555	def
2	2	3	666	ghi
3	3	4	444	xyz

# SQL

Para establecer una conexión a una BD mediante ODBC, es necesario instalar la librería **pyodbc**. Esto puede realizarse con **pip**, o bien, con **conda**.

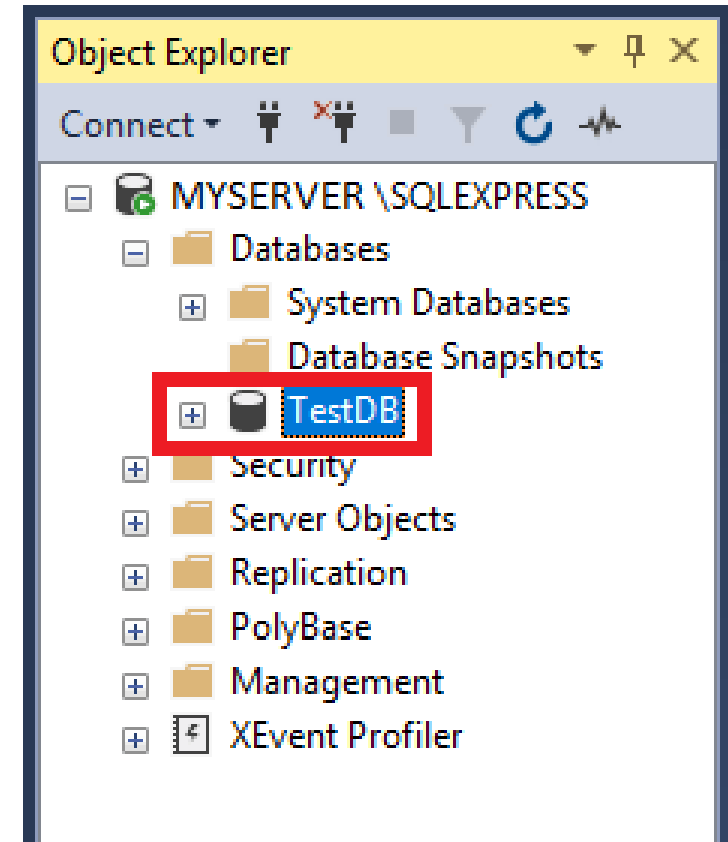
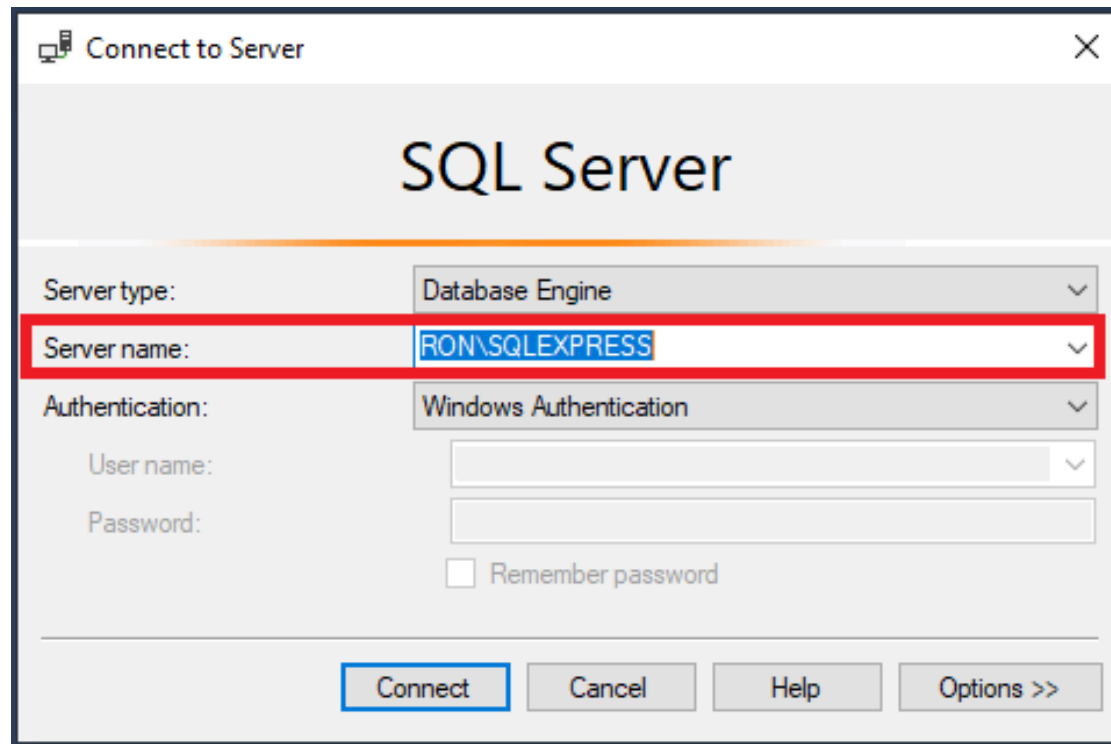


```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.316]
(c) 2018 Microsoft Corporation. All rights reserved.

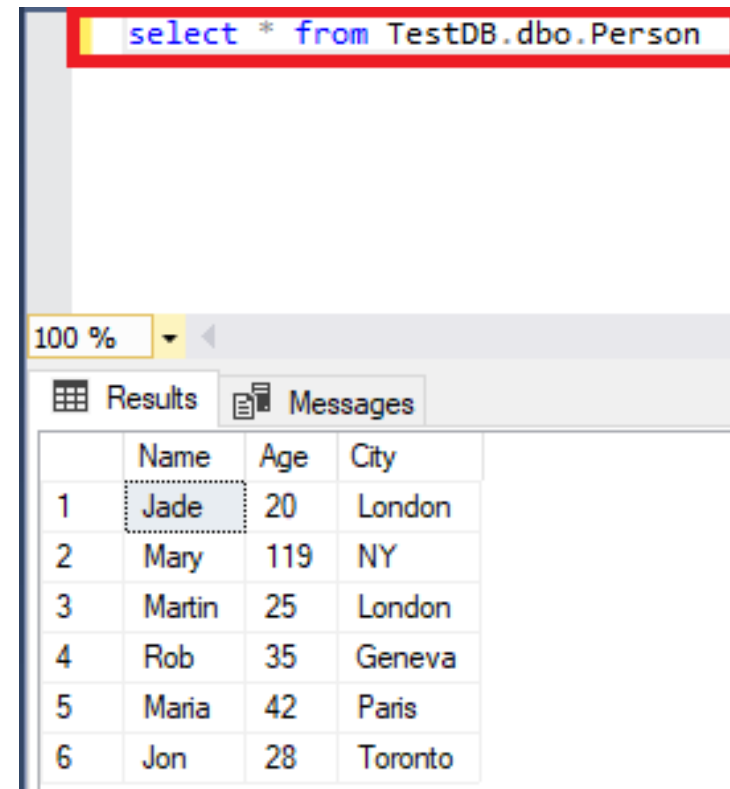
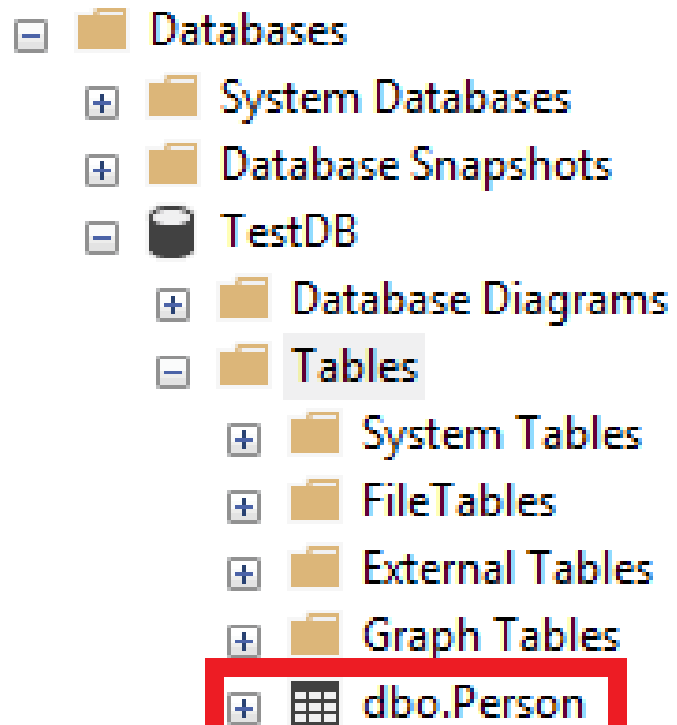
C:\WINDOWS\system32>cd\

C:\>cd C:\Users\Ron\AppData\Local\Programs\Python\Python37-32\Scripts
C:\Users\Ron\AppData\Local\Programs\Python\Python37-32\Scripts>pip install pyodbc
```

# Conexión Mediante pyodbc



# Conexión Mediante pyodbc



# Conexión Mediante pyodbc

Una vez que contamos con la información, podemos establecer la conexión de la siguiente manera.

```
import pyodbc
conn = pyodbc.connect('Driver={SQL Server};'
                      'Server=RON\SQLEXPRESS;'
                      'Database=TestDB;'
                      'Trusted_Connection=yes;')

cursor = conn.cursor()
cursor.execute('SELECT * FROM TestDB.dbo.Person')

for row in cursor:
    print(row)
```

# Conexión Postgres

En el caso de una conexión a una base de datos **Postgres**, se debe instalar la librería **psycopg2**.

```
import psycopg2
```

```
DB_SCHEMA = 'ipla-db-05'  
cnx = psycopg2.connect(  
    host="10.0.200.110",  
    database="dbacademy",  
    user="dbuser",  
    password="passwd1234",  
    options="-c search_path={}".format(DB_SCHEMA))
```

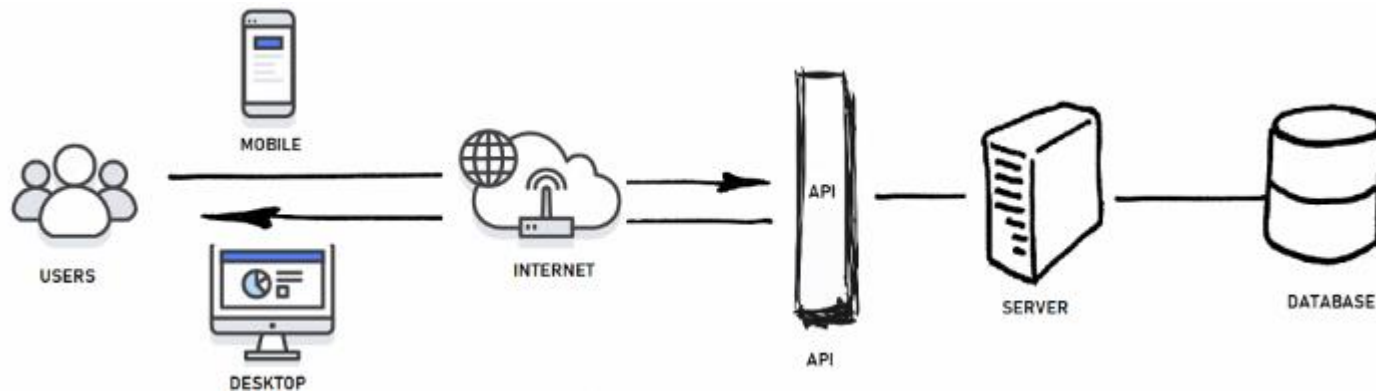
```
query = 'SELECT * FROM escuelas'  
pd.read_sql(query, cnx)
```

	id_escuela	cod_escuela	nom_escuela
0	1	08	ESCUELA DE EDUCACION
1	2	15	ESCUELA DE SALUD
2	3	03	ESCUELA DE TECNOLOGIA
3	4	07	ESCUELA DE DESARROLLO SOCIAL Y SERVICIO PUBLICO
4	5	02	ESCUELA DE ELECTRONICA

# Lectura de una API

# ¿Qué es una API Rest?

El término REST (Representational State Transfer) se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP. Un servicio REST no es una arquitectura software, sino un conjunto de restricciones que tener en cuenta en la arquitectura software que usaremos para crear aplicaciones web respetando HTTP.



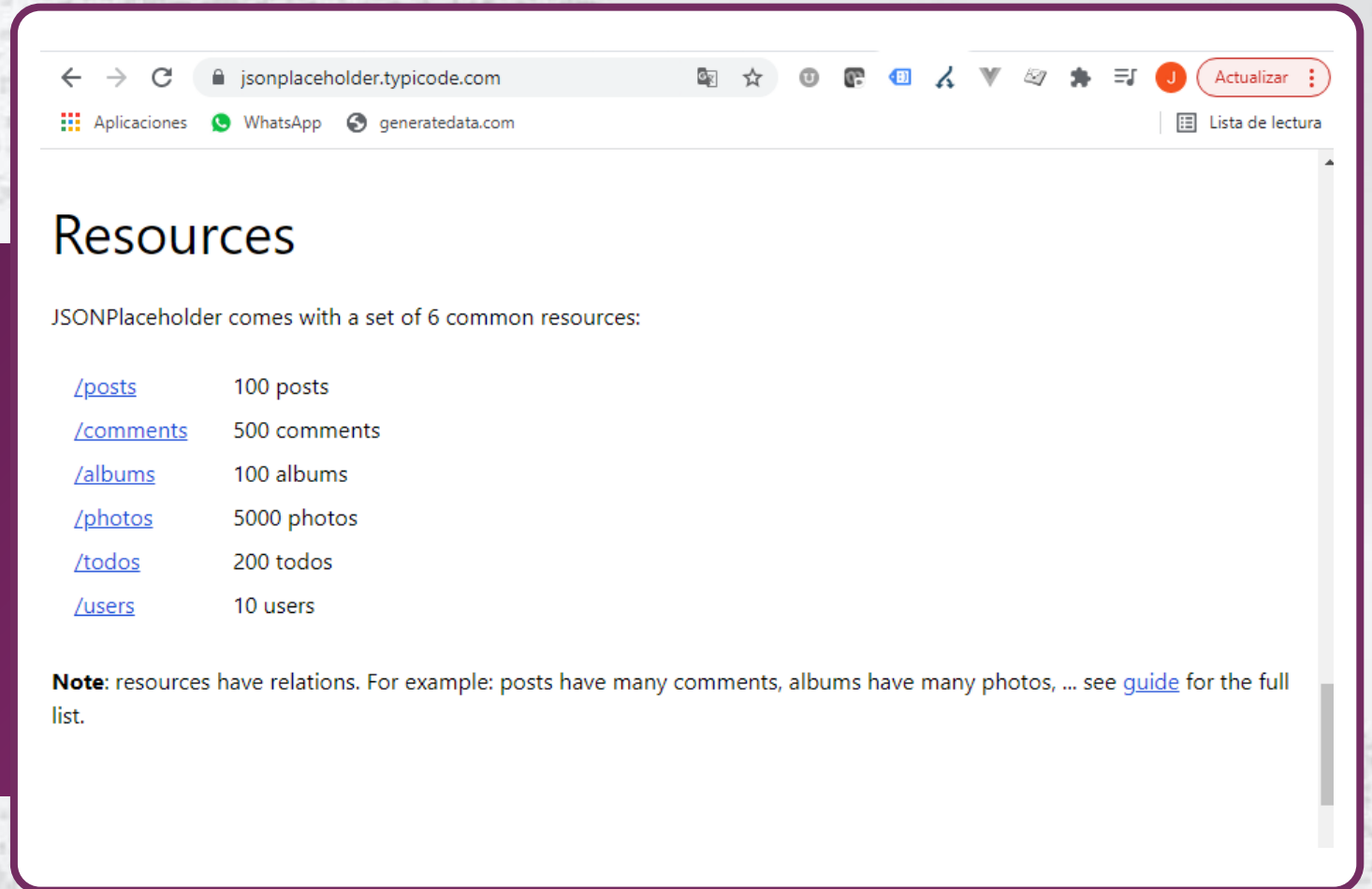
Hoy en día, las API REST son ampliamente utilizadas para el intercambio de información entre sistemas.

# ¿Qué es una API Rest?

Para este ejemplo, utilizaremos una API pública que disponibiliza distintos recursos.

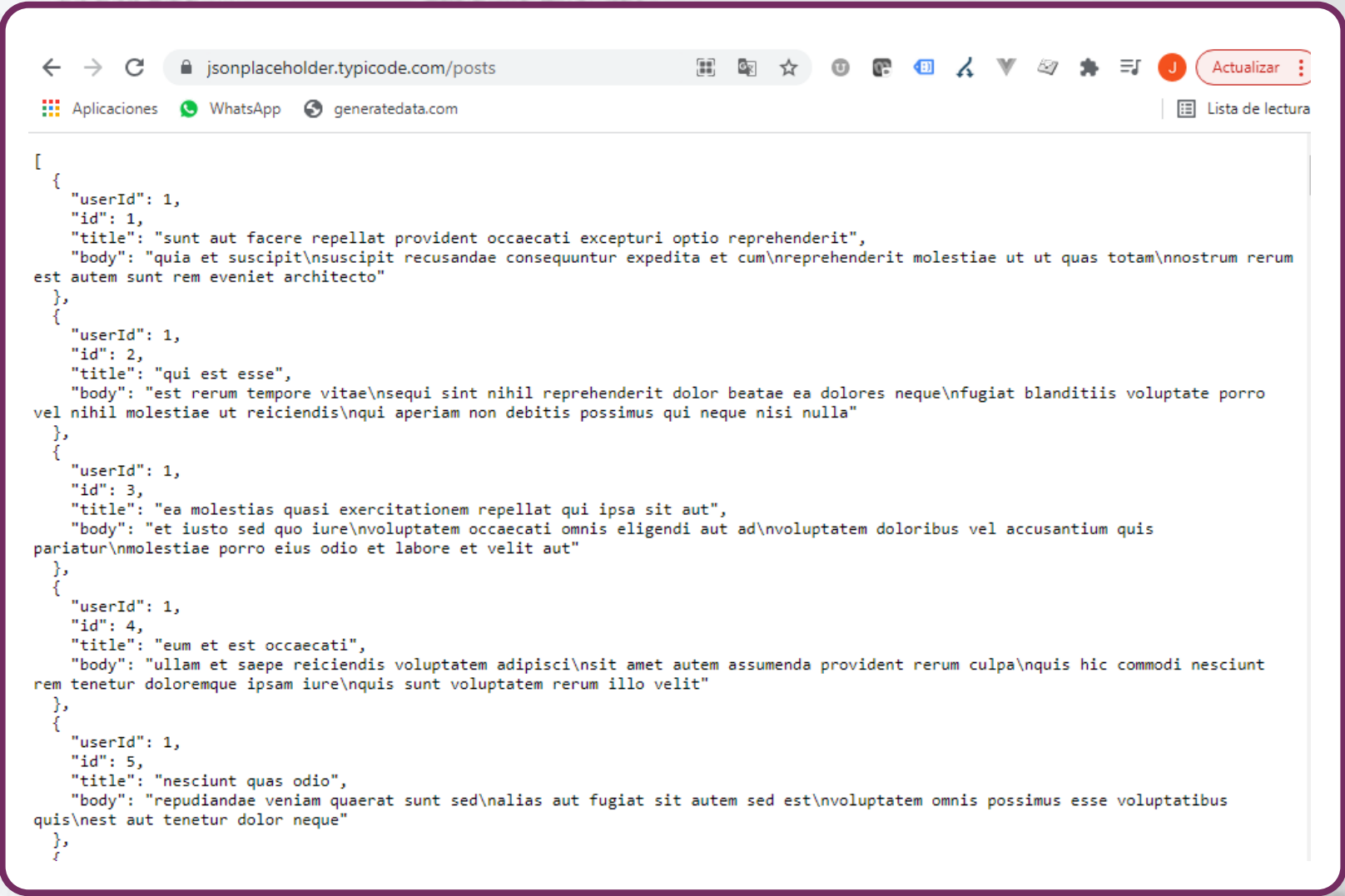
Puede ver más información en la siguiente web:

<https://jsonplaceholder.typicode.com/>



# ¿Qué es una API Rest?

Para este ejemplo, utilizaremos el servicio que pone a disposición un listado de posts de una red social.



The screenshot shows a web browser window with the address bar displaying `jsonplaceholder.typicode.com/posts`. The browser's address bar includes navigation buttons (back, forward, refresh), a lock icon, and a search icon. Below the address bar, there are tabs for 'Aplicaciones', 'WhatsApp', and 'generatedata.com'. On the right side of the browser, there is a red 'Actualizar' button and a 'Lista de lectura' icon. The main content area of the browser displays a JSON array of five post objects. Each object contains 'userId', 'id', 'title', and 'body' fields. The JSON is formatted with syntax highlighting.

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
  },
  {
    "userId": 1,
    "id": 3,
    "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
    "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut"
  },
  {
    "userId": 1,
    "id": 4,
    "title": "eum et est occaecati",
    "body": "ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit"
  },
  {
    "userId": 1,
    "id": 5,
    "title": "nesciunt quas odio",
    "body": "repudiandae veniam quaerat sunt sed\nalias aut fugiat sit autem sed est\nvoluptatem omnis possimus esse voluptatibus quis\nest aut tenetur dolor neque"
  }
]
```

# Consumiendo una API Rest

Para realizar **requests** a una API Rest, debemos importar la librería **requests**, que es parte de la librería estándar de Python.

```
import requests
```

```
url = 'https://jsonplaceholder.typicode.com/posts'
```

```
headers = {"Accept": "*/*", "Content-Type": "application/json"}  
response = requests.get(url, headers=headers)
```

```
if response.status_code != 200:  
    raise IOError(f'Error Code: {response.status_code}. Reason: {response.reason}')
```

# Consumiendo una API Rest

Posteriormente, la respuesta la podemos interpretar como un dataframe con la función `json_normalize()`.

```
# respuesta json  
json_resp = response.json()
```

```
# transformamos en dataframe  
df = pd.json_normalize(json_resp)  
df.head()
```

	userId	id	title	body
0	1	1	sunt aut facere repellat provident occaecati e...	quia et suscipit\nsuscipit recusandae consequu...
1	1	2	qui est esse	est rerum tempore vitae\nsequi sint nihil repr...
2	1	3	ea molestias quasi exercitationem repellat qui...	et iusto sed quo iure\nvoluptatem occaecati om...
3	1	4	eum et est occaecati	ullam et saepe reiciendis voluptatem adipisci...
4	1	5	nesciunt quas odio	repudiandae veniam quaerat sunt sed\nalias aut...

# Web Scrapping

# Web Scraping

- Muchas veces necesitamos obtener datos que están publicadas en distintas páginas web, con mucho menos estructuración de la que nos brinda la librería Pandas. Entonces, es necesario el uso de herramientas más sofisticadas para la obtención de datos.
- Un web **scraper** es una herramienta especializada que permite la extracción de textos desde sitios y portales web completos, a partir de etiquetas y selectores, con el propósito de armar una base de datos de información.



# Web Scraping con Python

¿Por qué web scraping con Python?

Fundamentalmente, por ser un lenguaje intuitivo, fácil de usar y, por las librerías y herramientas que han surgido en torno al mismo para llevar a cabo esta técnica.

En función del proyecto y sus características, existen dos alternativas:

- Utilizar las **librerías requests** y **Beautiful Soup** de manera conjunta (es lo que veremos en este tutorial).
- Utilizar el **framework Scrapy**. Este framework, además de hacer scraping (extraer información de una página) permite hacer crawling fácilmente (descubrir los enlaces de una web y navegar a través de ellos).

# Beautiful Soup

**Beautiful Soup** es una librería Python que permite extraer información de contenido en formato HTML o XML. Para usarla, es necesario especificar un parser, que es responsable de transformar un documento HTML o XML en un árbol complejo de objetos Python. Esto permite, por ejemplo, que podamos interactuar con los elementos de una página web como si estuviésemos utilizando las herramientas del desarrollador de un navegador. A la hora de extraer información de una web, uno de los **parsers** más utilizado es el parser HTML de lxml.

A continuación, te muestro cómo instalar tanto la **librería Beautiful Soup** como el **parser lxml** utilizando el gestor de paquetes pip.

Para instalar Beautiful Soup, ejecuta el siguiente comando:

```
1. $> pip install beautifulsoup4
```

Para instalar el parser **lxml**, ejecuta el siguiente comando:

```
1. $> pip install lxml
```

# Pasos para hacer Web Scraping

Identifica los elementos de la página de los que puede extraer la información. Las páginas web son documentos estructurados formados por una jerarquía de elementos. El primer paso para extraer información es identificar correctamente el elemento o elementos que contienen la información deseada. Para ello, lo más fácil es abrir la página en un navegador e inspeccionar el elemento. Esto se consigue haciendo clic con el botón derecho sobre el elemento en cuestión y pulsando sobre la opción Inspeccionar o Inspeccionar elemento (depende del navegador).

Descarga el contenido de la página. Para ello, utiliza la **librería requests**. El contenido de la respuesta, el que contiene la página en HTML, será el que pasemos posteriormente a BeautifulSoup para generar el árbol de elementos y poder hacer consultas al mismo.

```
import requests
from bs4 import BeautifulSoup

r = requests.get('http://unapagina.xyz')
soup = BeautifulSoup(r.text, 'lxml')
```

# Pasos para hacer Web Scrapping

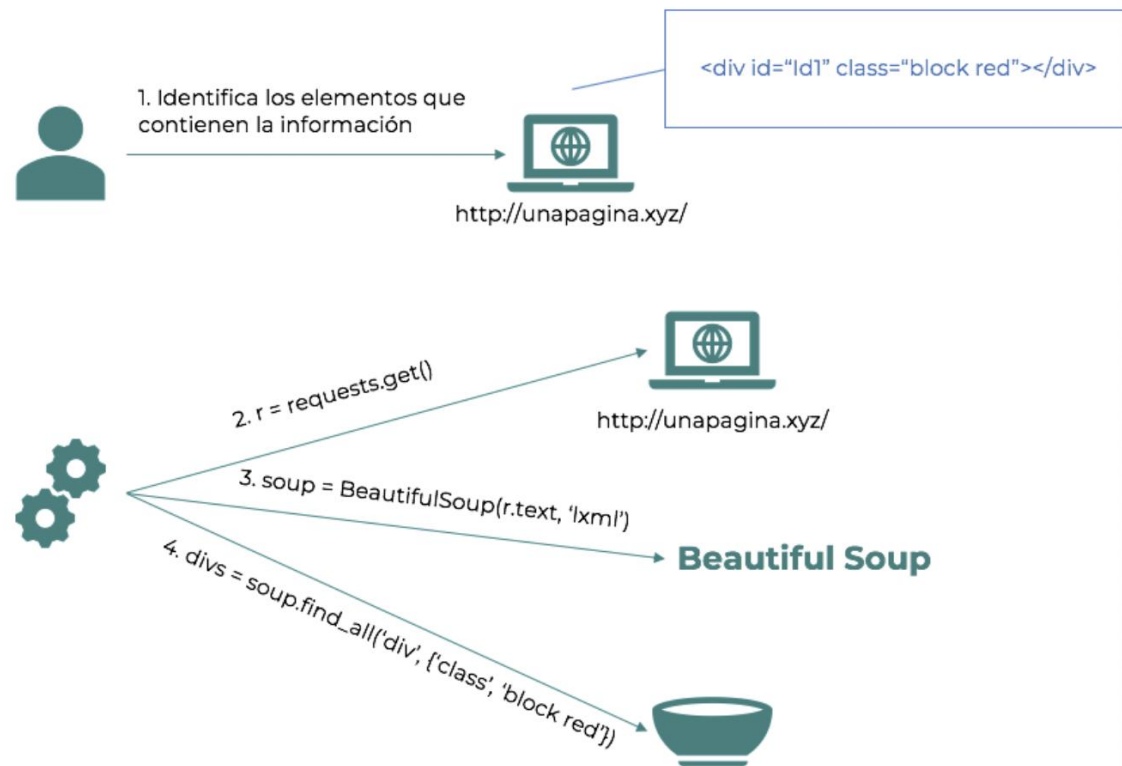
Crear la “sopa”. El contenido de la página obtenido en el paso anterior será el que utilizemos para crear la «sopa», esto es, el árbol de objetos Python que representan al documento HTML. Para ello, hay que crear un objeto de tipo **BeautifulSoup**, al cual se le pasa el texto en formato HTML y el identificador del parser a utilizar:

```
import requests
from bs4 import BeautifulSoup

r = requests.get('http://unapagina.xyz')
soup = BeautifulSoup(r.text, 'lxml')
```

# Consumiendo una API Rest

Busca los elementos en la sopa y obtén la información deseada.



# Tipos de objetos de BeautifulSoup

☞ Imagina que este documento *HTML* se ha obtenido al consultar una página con la librería `requests`.

```
1. <html lang="es">
2. <head>
3.     <meta charset="UTF-8">
4.     <title>Página de prueba</title>
5. </head>
6. <body>
7. <div id="main" class="full-width">
8.     <h1>El título de la página</h1>
9.     <p>Este es el primer párrafo</p>
10.    <p>Este es el segundo párrafo</p>
11.    <div id="innerDiv">
12.        <div class="links">
13.            <a href="https://pagina1.xyz/">Enlace 1</a>
14.            <a href="https://pagina2.xyz/">Enlace 2</a>
15.        </div>
16.        <div class="right">
17.            <div class="links">
18.                <a href="https://pagina3.xyz/">Enlace 3</a>
19.                <a href="https://pagina4.xyz/">Enlace 4</a>
20.            </div>
21.        </div>
22.    </div>
23.    <div id="footer">
24.        <!-- El footer -->
25.        <p>Este párrafo está en el footer</p>
26.        <div class="links footer-links">
27.            <a href="https://pagina5.xyz/">Enlace 5</a>
28.        </div>
29.    </div>
30. </div>
31. </body>
32. </html>
```

```
from bs4 import BeautifulSoup
```

```
contenido = """
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Página de prueba</title>
</head>
<body>
    <div id="main" class="full-width">
        <h1>El título de la página</h1>
        <p>Este es el primer párrafo</p>
        ...
    </div>
</body>
</html>
"""
```

```
soup = BeautifulSoup(contenido, 'lxml')
```

# Dudas y consultas

Fin presentación