FORMATO CSV

In [ ]:
```
import pandas as pd # Importamos pandas con el alias, que es "pd"

# Ahora, lo que hacemos es crear un dataframe (una clase de pandas)
# pd.read_csv: lee un archivo.
# con "delimiter" puedo cambiar el delimitador. delimiter = ";"
# tiene el atributo del encoding
    # Aomprende el español: vocales con acento y 'ñ'
dat_csv = pd.read_csv('empleados.csv', encoding = "ISO-8859-1")

# Visualizamos el contenido, en forma de tabla.
# la tabla es una representacion grafica del dataframe
# CSV = "Comma Separated Values" = "Valores separados por comas"
dat_csv
```

> Si queremos que en lugar de todos los datos, se restrinja la salida a las primeras cinco filas, podemos utilizar el método head().

In [ ]:
```
type(dat_csv) # me dice que estamos trabajando con un
              # dataframe de Pandas
```

In [ ]:
```
dat_csv.head() # vemos los primeros 5 elementos (1, 2, 3, ...)
```

> En los dos casos anteriores, aparecen los elementos de la primera fila como los nombres de las columnas de datos, si no quisiéramos este comportamiento podemos agregar el atributo header = None, de la siguiente manera:

In [ ]:
```
dat_csv = pd.read_csv('empleados.csv', encoding = "ISO-8859-1", header = None)
dat_csv
# en este caso es incorrecto, pero eventualmente
# queremos que esa primera linea no sean headers
```

> Si quisiéramos que tomar una determinada línea como cabecera, debemos asignarle al parámetro header el número de fila correspondiente, en el siguiente ejemplo se toma la línea 3 como cabecera:

In [ ]:
```
dat_csv = pd.read_csv('empleados.csv', encoding = "ISO-8859-1", header = 3)
dat_csv
```

> Por defecto, read_csv asigna un índice numérico predeterminado que comienza con cero al leer los datos. Sin embargo, es posible alterar este comportamiento pasando el nombre de la columna que utilizaremos como índice. A continuación se dejara la columna id como índice de la tabla:

In [ ]:
```
dat_csv = pd.read_csv('empleados.csv', encoding = "ISO-8859-1", index_col='id')
dat_csv
# antes teníamos que id era un atributo, pero ahora el index
```

> En el caso de que queramos restringir la tabla a algunas columnas específicas, podemos realizarlo con el parámetro **usecols** indicando en una lista las columnas seleccionadas.

In [ ]:
```
# A veces no nos interesan todos los datos. Usando el usecols, doy una lista
# de strings que se corresponden con los datos que necesito
dat_csv  = pd.read_csv('empleados.csv', encoding = "ISO-8859-1", usecols = ['Nombr
dat_csv.head()
```

> Si queremos eliminar una fila en particular, podemos utilizar el parámetro skiprows, en el siguiente caso se eliminan las filas 1 y 4:

In [ ]:
```
# Lo mismo podemos hacer con las filas
dat_csv  = pd.read_csv('empleados.csv', encoding = "ISO-8859-1", skiprows = [1,4])
dat_csv
```

> Para presentar un número dado de filas desde el inicio, podemos utilizar el parámetro nrows, en el siguiente ejemplo se presentan las primeras tres filas de datos.

In [ ]:
```
dat_csv  = pd.read_csv('empleados.csv', encoding = "ISO-8859-1",  nrows = 3)
dat_csv
```

> Para seleccionar un rango específico de filas podemos utilizar el método query especificando los límites aplicables sobre una determinada columna:

In [ ]:
```
# Puedo hacerle consultas al dataframe con el método query
dat_csv  = pd.read_csv('empleados.csv', encoding = "ISO-8859-1")
dat_csv.query('2 < id < 6') # -> recordar que id era un atributo de los datos
```

> Para tratar con los datos de una columna, podemos recuperarlos dentro de una lista mediante el uso de un bucle for, y luego si es necesario convertir la lista en una array como se muestra a continuación:

In [ ]:
```
import pandas as pd
import numpy as np
dat_csv  = pd.read_csv('datos.csv', encoding = "ISO-8859-1")

datos_x = dat_csv.nombre # estoy copiando las columnas
datos_y = dat_csv.sexo

# Lo imprimo para mirar--
# ----------------------
print(datos_x)
print('\n')
```

```
print(datos_y)
print('\n')
# ----------------------
# ----------------------

x = []
y = []

# append() agrega el elemento completo al final de la lista

for i in dat_csv.nombre:
    x.append(i)

for j in dat_csv.sexo:
    y.append(j)

print(x)
print(y)
print('\n')

# Las transformo en array de NumPy
# Lo que nos permite hacer es el pasaje a Numpy, que es nuestra interfaz
# numérica que tenemos para manejar con los modelos
# => es necesario hacer el pasaje
x_array = np.array(x)
print(type(x_array))
print(x_array)
```

FORMATO EXCEL

> En el caso de trabajar directamente con un archivo de Excel, podemos utilizar el método **read_excel()**

In [ ]:
```
dat_exc  = pd.read_excel('empleados.xlsx') # No entiende el encoding!!!
dat_exc
```

FORMATO JSON

> Json es un formato nativo de Javascript que se popularizó y ahora se utiliza como interfaz en un monton de otros lenguajes. La estructura de los datos es un poquito más compleja

{ "nombre": "Ana", "edad": 33, "estado_civil": true, "esposo": "Pablo", "hijos": ["Cecilia","Luis"], "autos": [{ "modelo": "Ford", "color": "rojo" }, { "modelo": "Chevrolet", "color": "azul" }] }

In [ ]:
```
import pandas as pd
movies_json = pd.read_json('json1.json')
movies_json.head()
```

FORMATO HTML

In [ ]:
```
import pandas as pd
pd.read_html('index.html')
```

BASE DE DATOS

> Una base de datos es una recopilacion de elementos (de datos) con relaciones predefinidas entre ellos. Entonces: los datos están organizados en tablas, y estas tablas tienen relaciones entre ellas.

In [ ]:
```
import pandas as pd
import sqlite3 # Hace una llamada a sql
conn = sqlite3.connect("mibase.sqlite") # -> puedo conectar con mi BD
df = pd.read_sql_query("SELECT * FROM producto;", conn)
df.head()
```

FORMATO XML

> También es un formato de etiquetas. De hecho, el formato HTML es un caso especial del XML

In [ ]:
```
import xml.etree.cElementTree as et # Elemnt tree del módulo XML
import pandas as pd

# Definimos una función
def obtenerValorDeNodo(node):
    return node.text if node is not None else None

# Definimos la función main
def main():
    parsed_xml = et.parse("datos.xml") # Parseamos el archivo en cuestion
    dfcols = ['nombre', 'email', 'telefono', 'calle'] # Nosotros nos quedemos queda
    df_xml = pd.DataFrame(columns = dfcols) # creamos un dataframe de Pandas

    for node in parsed_xml.getroot():
        nombre = node.attrib.get('nombre') # estaba como atributo de la etiqueta "
        email = node.find('email')
        telefono = node.find('telefono')
        calle = node.find('direccion/calle')

        df_xml = df_xml.append(  pd.Series([nombre, obtenerValorDeNodo(email), obte
    print(df_xml)

main()

# The frame.append method is deprecated and will be removed from pandas in a future
```

In [ ]:
```
pip install nbconvert
```

In [1]:
```
pip install nbconvert[webpdf] --allow-chromium-download
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
Usage:
  C:\Users\vguar\anaconda3\python.exe -m pip install [options] <requirement specif
ier> [package-index-options] ...
  C:\Users\vguar\anaconda3\python.exe -m pip install [options] -r <requirements fi
le> [package-index-options] ...
  C:\Users\vguar\anaconda3\python.exe -m pip install [options] [-e] <vcs project u
rl> ...
  C:\Users\vguar\anaconda3\python.exe -m pip install [options] [-e] <local project
path> ...
  C:\Users\vguar\anaconda3\python.exe -m pip install [options] <archive url/path>
...

no such option: --allow-chromium-download
```

In [6]:

```
--to webpdf notebook.ipynb
```

```
  File "C:\Users\vguar\AppData\Local\Temp\ipykernel_11344\1195906325.py", line 1
    --to webpdf notebook.ipynb
          ^
SyntaxError: invalid syntax
```