TELEDUC
Educación a Distancia

DATA FRAMES Y STRINGS

>>> Motivación y contextualización

Recordemos

Aprendimos herramientas básicas para trabajar con Data Frames

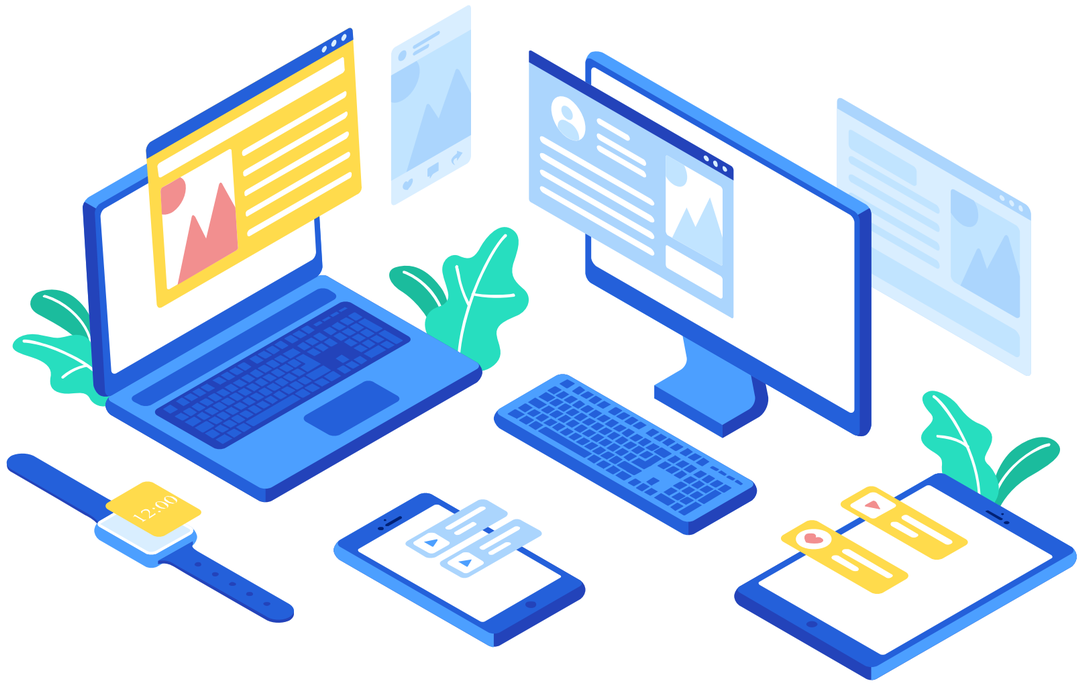
Ver, crear, editar, filtrar y eliminar columnas

Manejo de datos faltantes

Motivación

1. Digamos que tenemos una columna en un Data Frame que contiene los RUT de muchas personas ingresadas manualmente.

2. Es posible que tenga varios errores. Algunos usaron puntos y otros no. Otros usaron guiones y otros no.



¿Cómo podríamos limpiar esta información?

Necesitamos herramientas para trabajar con strings

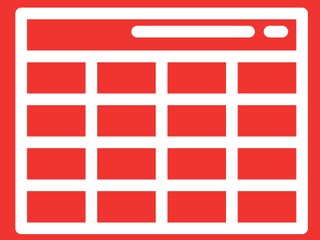
Estas herramientas son muy importantes ya que la mayoría de los datos son strings

¿Cómo podríamos limpiar esta información?

Una forma es recorrer todos estos valores mediante un `for`, y luego ir aplicando diversas funciones de `strings` sobre cada valor. No obstante, esta no es la forma más eficiente de hacerlo.

Existen varias funciones de `strings` que se pueden aplicar directamente sobre la columna.

Estas últimas además requieren de menor tiempo de procesamiento, lo que para grandes volúmenes de datos es muy relevante.



Es posible aplicar diversas funciones de `strings` sobre los valores de cada una de las filas

Iteración sobre Data Frames

Recuerda revisar el archivo **m3-ej1.py** →

Si tenemos un Data Frame de nombre `df`, podemos iterar sobre sus filas de la siguiente manera:

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

for index,row in df.iterrows():
        print(index)
        print(row["Nombre"])
```

RESULTADO

```
0
Javier Andrés López Castro
1
Pedro Andrés Vergara Campos
2
Paula Blanca Marín Campos
3
Isabel Ignacia González Muñoz
4
Bernardo Vicente Campos Rodríguez
```

La lista que recorremos en el `for` se obtiene al ejecutar la función `iterrows()`.

En la declaración del `for` tenemos las variables `index` y `row`. **Se deben declarar estas dos variables obligatoriamente.** En caso contrario, el `for` no funcionará correctamente.

Iteración sobre Data Frames

Recuerda revisar el archivo **m3-ej2.py** →

Si queremos extraer el primer nombre de cada fila, lo podríamos hacer de la siguiente manera:

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

for index,row in df.iterrows():
        primer_nombre = row["Nombre"].split(" ")[0]
        print(primer_nombre)
```

RESULTADO

```
Javier
Pedro
Paula
Isabel
Bernardo
```

Es posible aplicar funciones de `strings` sobre ellas. En este caso, hicimos un `split` con el carácter espacio, luego extraemos el primer elemento que corresponde al primer nombre de cada persona.

A pesar de que podemos aplicar cualquier función de `strings` sobre el valor extraído, al recorrer la columna "**Nombre**", no es recomendado hacer este tipo de operaciones de esta manera para editar datos en un Data Frame.

>>> Funciones de strings sobre columnas de un Data Frame

Funciones de strings sobre columnas de un Data Frame

Si tenemos un Data Frame de nombre `df` y una columna de tipo `object` en *Pandas*, podemos aplicar las funciones de `strings` que ya conoces.

El comando general se ve de la siguiente manera:

```
df["Nombre columna de tipo object"].str.(función de string)
```

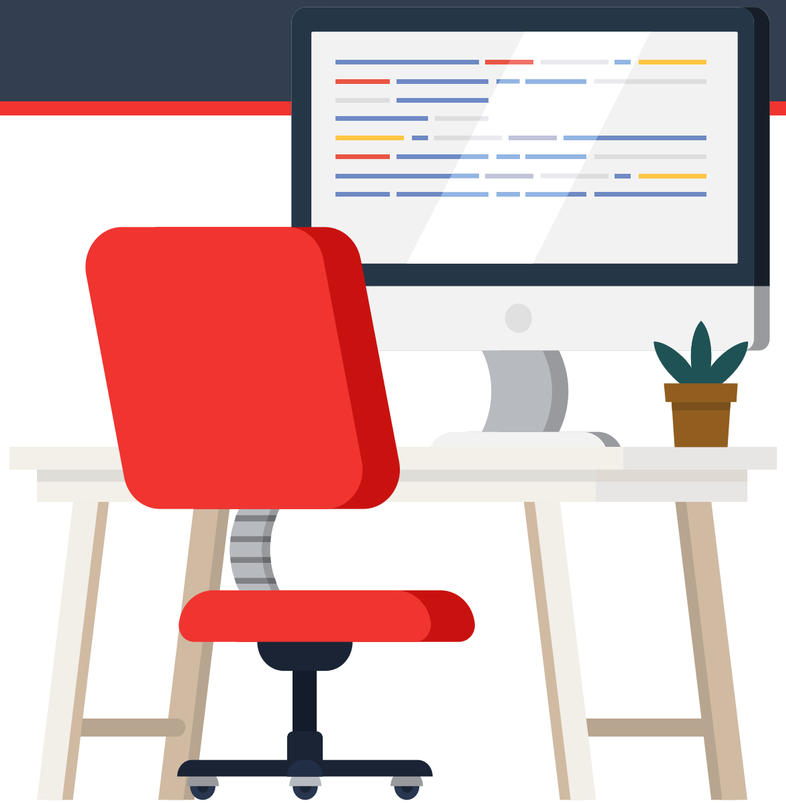
Veamos algunos ejemplos, todos se basan el archivo de ejemplo “**ejemplo.csv**”, para que así puedan ir verificando los outputs.

`string`

¿Para qué sirve la función len()?

FUNCIÓN
`len()`

Para saber la **cantidad de caracteres** de los elementos de una columna de tipo `object`.



Función len()

Recuerda revisar el archivo **m3-ej3.py** →

Para un Data Frame de nombre `df`:

```
df["nombre columna"].str.len()
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

print(df["Nombre"].str.len())
```

RESULTADO

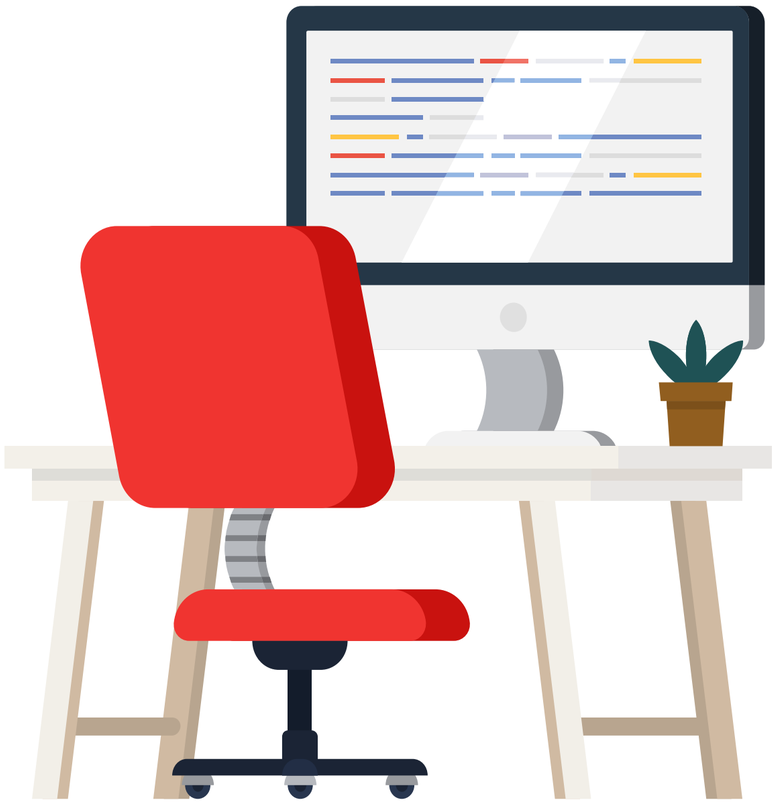
```
0    26
1    27
2    25
3    29
4    33
5    27
```

Aquí tenemos una columna “**Nombre**” con el nombre de distintas personas, y queremos saber la cantidad de caracteres de cada nombre.

¿Cómo extraer carácter o caracteres de Data Frame?

Podemos **extraer un carácter de los valores de una columna** de un Data Frame de tipo object, de igual manera a cómo lo haríamos en un **string normal**.

También se puede extraer una secuencia de caracteres de una columna de un Data Frame de tipo object, de igual manera a cómo lo haríamos en un **string normal**.



Recuerda revisar el archivo **m3-ej4.py** →

Extraer carácter

Es posible extraer un carácter, para un Data Frame de nombre df:

```
df["nombre columna"].str[posición carácter]
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

print(df["RUT"].str[0])
```

RESULTADO

```
0    1
1    1
2    1
3    8
4    1
```

Aquí se puede ver, cómo extraer el primer carácter de la columna “**RUT**” (con el RUT de distintas personas) del Data Frame df. Se podría extraer cualquier carácter ingresando una posición determinada.

Extraer caracteres mediante slice

Recuerda revisar el archivo **m3-ej5.py** →

Es posible extraer un conjunto de carácteres, para un Data Frame de nombre `df`:

```
df["nombre columna"].str[posición carácter inicial:posición carácter final]
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

print(df["RUT"].str[0:4])
```

RESULTADO

```
0    13.6
1    17.1
2    15.3
3    8.29
4    15.7
```

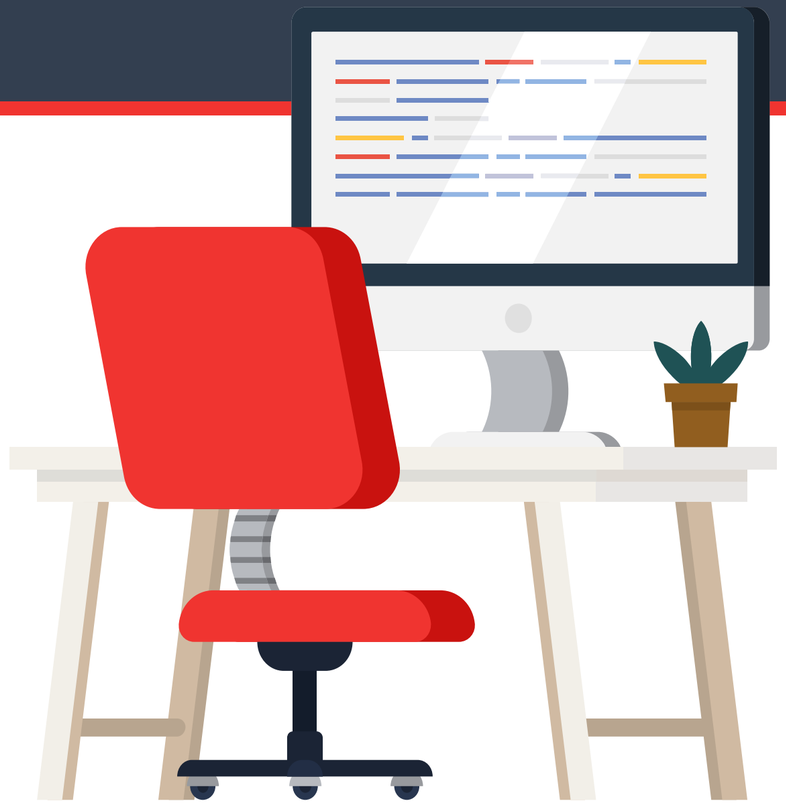
Sabemos que df tiene una columna "**RUT**" con el RUT de distintas personas, y queremos extraer los cuatro primeros dígitos.

Al igual que en los `strings`, el `slice` toma desde la posición inicial a la posición final-1.

¿Para qué sirve la función lower()?

FUNCIÓN
`lower()`

Sirve para transformar los caracteres a **minúsculas** en los valores de una columna de un Data Frame de tipo `object`.



Función lower()

Recuerda revisar el archivo **m3-ej6.py** →

De forma general, para un Data Frame de nombre df:

```
df["nombre columna"].str.lower()
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

print(df["Nombre"].str.lower())
```

RESULTADO

```
0                 javier andrés lópez castro
1               pedro andrés vergara campos
2                 paula blanca marín campos
3             isabel ignacia gonzález muñoz
4         bernardo vicente campos rodríguez
```

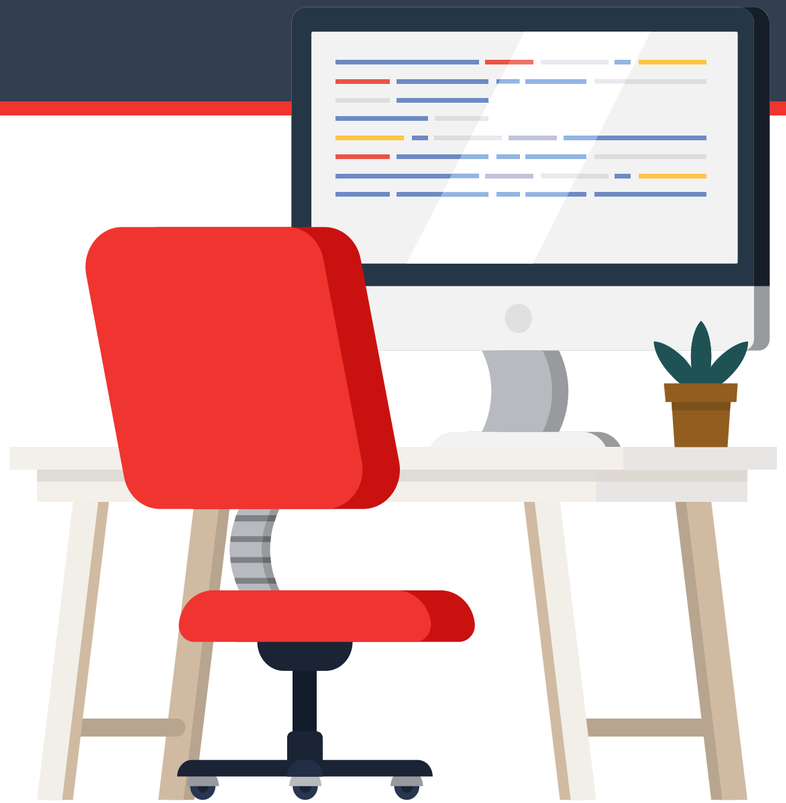
Tenemos un Data Frame de nombre df, que tiene una columna "**Nombre**" con el nombre de distintas personas, y queremos que el nombre quede solo en minúsculas.

Si se desea "**guardar**" este cambio debe volver a asignarlo a la columna original. Esto es igual para las funciones que siguen.

¿Para qué sirve la función upper()?

FUNCIÓN
`upper()`

Sirve para transformar los caracteres a **mayúsculas** de los valores de una columna de un Data Frame de tipo `object`.



Recuerda revisar el archivo **m3-ej7.py** →

Función upper()

De forma general, para un Data Frame de nombre df:

```
df["nombre columna"].str.upper()
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

print(df["Nombre"].str.upper())
```

RESULTADO

```
0                  JAVIER ANDRÉS LÓPEZ CASTRO
1                PEDRO ANDRÉS VERGARA CAMPOS
2                  PAULA BLANCA MARÍN CAMPOS
3              ISABEL IGNACIA GONZÁLEZ MUÑOZ
4          BERNARDO VICENTE CAMPOS RODRÍGUEZ
```

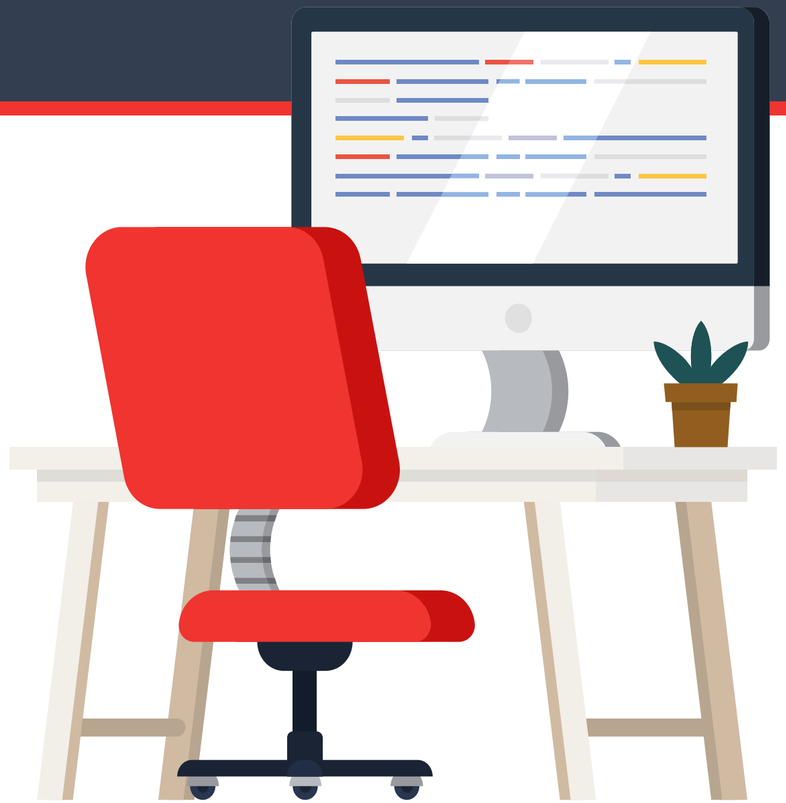
Tenemos un Data Frame de nombre df, que tiene una columna "**Nombre**" con el nombre de distintas personas, y queremos que el nombre quede solo en mayúsculas.

Si uno desea "**guardar**" este cambio debe volver a asignarlo a la columna original.

¿Para qué sirve la función replace(x,y)?

FUNCIÓN
`replace(x,y)`

Se usa para **reemplazar** los `strings x` por `y` en los **valores** de una columna de un Data Frame de tipo `object` mediante la función `replace(x,y)`.



Recuerda revisar el archivo **m3-ej8.py** →

Función replace(x,y)

Para un Data Frame de nombre `df`, donde se desea reemplazar los `strings x` por los `y`.

```
df["nombre columna"].str.replace(x,y)
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

print(df["Fecha_Nac"].str.replace("/","-"))
```

RESULTADO

```
0    1963-2-17
1    1993-10-24
2    1962-5-30
3    1972-3-20
4    1987-7-11
```

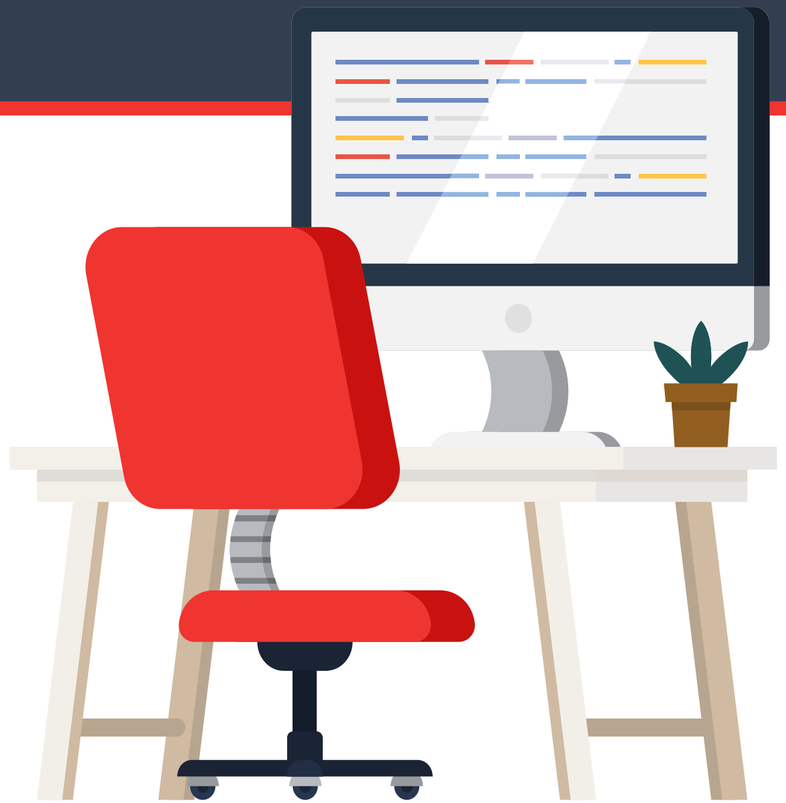
El Data Frame de nombre `df`, tiene una columna "**Fecha_Nac**" que corresponde a la fecha de nacimiento de distintas personas, en este caso se cambia el separador "`/`" por "`-`".

¿Para qué sirve la función contains(x)?

FUNCIÓN

`contains(x)`

Permite saber si los **valores** de una columna de un Data Frame de tipo `object` tienen un `string` específico.



Recuerda revisar el archivo **m3-ej9.py** →

Función contains(x)

De forma general para un Data Frame de nombre `df`, donde se desea buscar un `string x`.

```
df["nombre columna"].str.contains(x)
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

print(df["RUT"].str.contains("-"))
```

RESULTADO

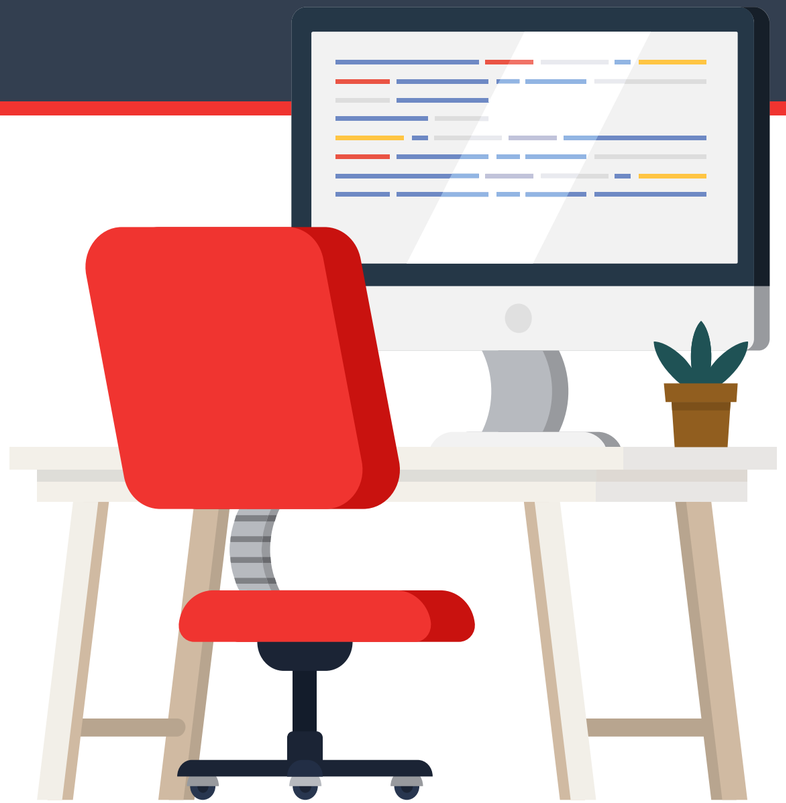
```
0    True
1    True
2    True
3    True
4    True
```

En este caso df tiene una columna “**RUT**” con el RUT de distintas personas, y queremos saber si cada uno de los RUT tiene un guion (“-”).

¿Para qué sirve la función find(x)?

FUNCIÓN
`find(x)`

Podemos saber la **posición** de la primera aparición de un `string x` en una columna de un Data Frame.



Recuerda revisar el archivo **m3-ej10.py** →

Función find(x)

De forma general para un Data Frame de nombre df:

```
df["nombre columna"].str.find(x)
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

print(df["Nombre"].str.find(" "))
```

RESULTADO

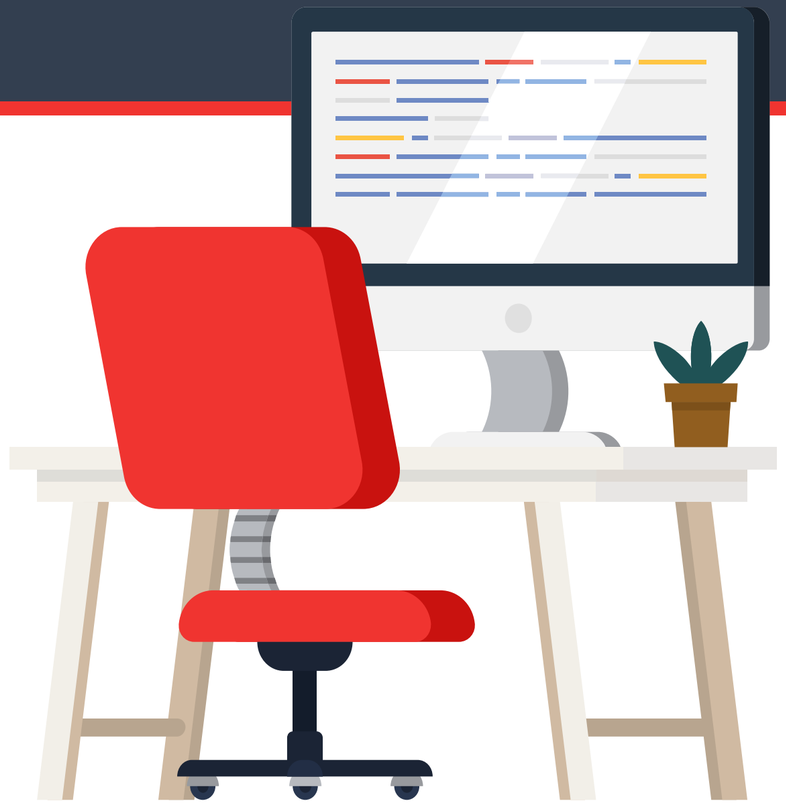
```
0    6
1    5
2    5
3    6
4    8
```

Aquí `df` tiene una columna “**Nombre**” con el nombre completo de distintas personas, y queremos saber en qué posición está el `string` “ ” (espacio).

¿Para qué sirve la función split(x)?

FUNCIÓN
`split(x)`

Se utiliza para **separar**, según un `string x`, los **valores** de una columna de un Data Frame de tipo `object`.



Recuerda revisar el archivo **m3-ej11.py** →

Función split(x)

Para un Data Frame de nombre `df`, donde se desea separar mediante el `string x`.

```
df["nombre columna"].str.split(x)
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

print(df["Nombre"].str.split(" "))
```

RESULTADO

```
0                [Javier, Andrés, López, Castro]
1              [Pedro, Andrés, Vergara, Campos]
2                [Paula, Blanca, Marín, Campos]
3            [Isabel, Ignacia, González, Muñoz]
4          [Bernardo, Vicente, Campos, Rodríguez]
```

En este caso df tiene una columna “**Nombre**” con el nombre de distintas personas, y queremos obtener una lista con los nombres y apellidos.

>>> Estructura y operaciones con Data Frames

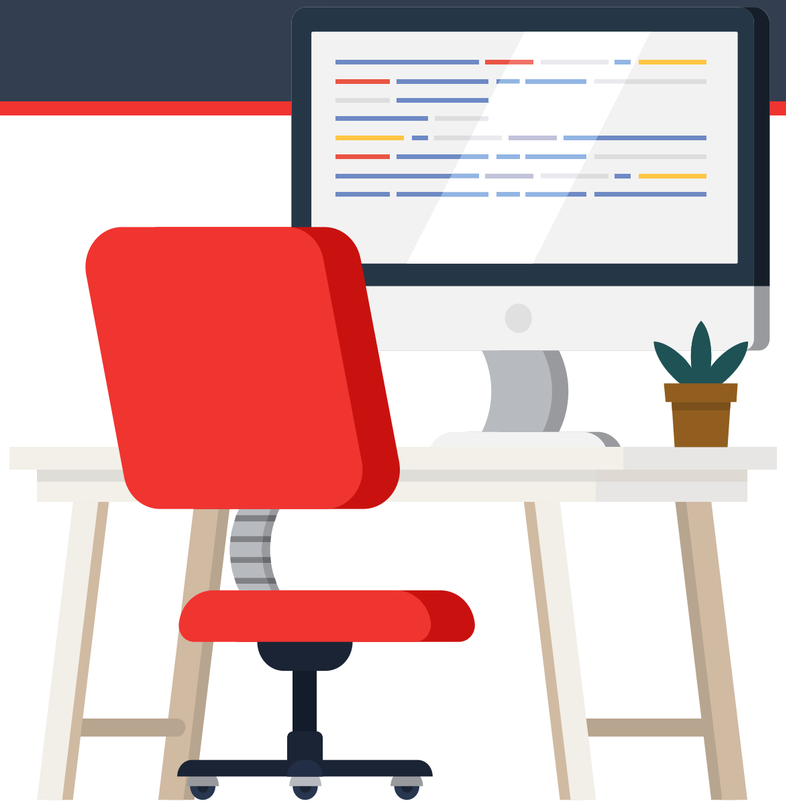
¿Para qué sirve la función split(x,expand=True)?

FUNCIÓN

`split(x,expand=True)`

Podemos crear un Data Frame a partir de un `split` sobre una columna de tipo `object`, añadiendo como parámetro “`expand`” y dándole como valor `True`.

Pensemos en el caso de un Data Frame que tiene una columna “**Nombre**” con el nombre completo de distintas personas. ¿Cómo podríamos separar este nombre completo en primer nombre, segundo nombre, primer apellido y segundo apellido?



Función split(x,expand=True)

Recuerda revisar el archivo **m3-ej12.py** →

Para separar la columna nombre:

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

df2 = df["Nombre"].str.split(" ",expand=True)

print(df2)
```

RESULTADO

	0	1	2	3
0	Javier	Andrés	López	Castro
1	Pedro	Andrés	Vergara	Campos
2	Paula	Blanca	Marín	Campos
3	Isabel	Ignacia	González	Muñoz
4	Bernardo	Vicente	Campos	Rodríguez
5	Isidora	Javiera	Muñoz	López

En este caso, `df2` es un Data Frame de 4 columnas distintas (primer nombre, segundo nombre, primer apellido y segundo apellido). Este se generó al agregar el parámetro `expand=True` al hacer el `split`.

Cambiar nombre a las columnas

Recuerda revisar el archivo **m3-ej13.py** →

El Data Frame creado anteriormente no tiene nombre en sus columnas. De forma general, para un Data Frame de nombre `df`.

```
df.columns = [nombres de las columnas]
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

df2 = df["Nombre"].str.split(" ",expand=True)
df2.columns=["Primer Nombre","Segundo Nombre","Primer Apellido","Segundo Apellido"]
print(df2)
```

RESULTADO

```
  Primer Nombre Segundo Nombre  Primer Apellido Segundo Apellido
0         Javier         Andrés            López           Castro
1          Pedro         Andrés          Vergara           Campos
2          Paula         Blanca            Marín           Campos
3         Isabel        Ignacia         González           Muñoz
4       Bernardo        Vicente           Campos        Rodríguez
```

Usaremos el atributo “columns” del Data Frame. Para acceder a un atributo, simplemente tomamos la variable que almacena el Data Frame, y con un punto accedemos a él.

Cambiar nombre a las columnas

Recuerda revisar el archivo **m3-ej13.py** →

El Data Frame creado anteriormente no tiene nombre en sus columnas. De forma general, para un Data Frame de nombre `df`.

```
df.columns = [nombres de las columnas]
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

df2 = df["Nombre"].str.split(" ",expand=True)
df2.columns=["Primer Nombre","Segundo Nombre","Primer Apellido","Segundo Apellido"]
print(df2)
```

RESULTADO

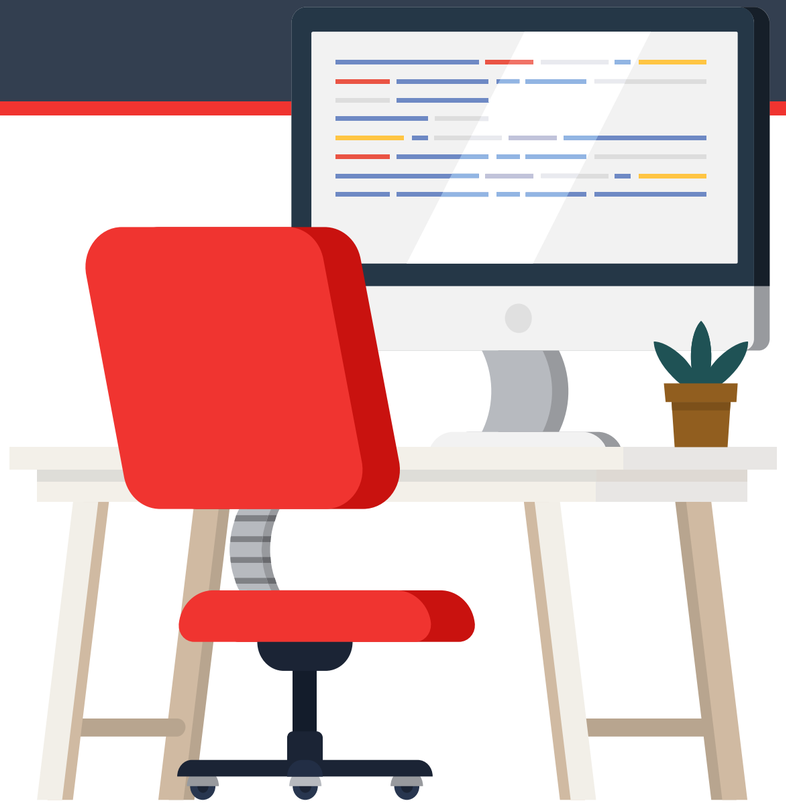
```
  Primer Nombre Segundo Nombre  Primer Apellido Segundo Apellido
0        Javier         Andrés            López           Castro
1         Pedro         Andrés          Vergara           Campos
2         Paula         Blanca            Marín           Campos
3        Isabel        Ignacia         González           Muñoz
4      Bernardo        Vicente           Campos        Rodríguez
```

Luego, asignamos la lista con los nombres de las columnas. Los atributos se definen como características del Data Frame que nos permitirán acceder a mayores funcionalidades con ellos.

¿Para qué sirve la función join()?

FUNCIÓN
`join()`

Esta función **agrega** las columnas de un Data Frame a otro.



Recuerda revisar el archivo **m3-ej14.py →**

Función join()

De forma general, para un Data Frame `df` y otro `df2`.

```
df.join(df2)
```

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

df2 = df["Nombre"].str.split(" ",expand=True)
df2.columns=["Primer Nombre","Segundo Nombre","Primer Apellido","Segundo Apellido"]
df = df.join(df2)
print(df.dtypes)
```

RESULTADO

```
RUT                 object
Nombre              object
Sexo                object
Fecha_Nac           object
Monto                int64
Primer Nombre       object
Segundo Nombre      object
Primer Apellido     object
Segundo Apellido    object
dtype: object
```

¿Qué pasaría si el Data Frame con el que intentamos hacer `join` (en este ejemplo `df2`) tuviera una o más columnas del mismo nombre que el Data Frame original? **En este caso Python arroja un error.**

Función join()

Recuerda revisar el archivo **m3-ej15.py →**

Para evitar el error anterior, se ocupan los **parámetros** `lsuffix` y `rsuffix` que agregan un sufijo a las columnas que tienen el mismo nombre.

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

df2 = df["Nombre"].str.split(" ",expand=True)
df2.columns=["Nombre","Segundo Nombre","Primer Apellido","Segundo Apellido"]
df = df.join(df2, rsuffix = "_df2", lsuffix="_df")
print(df.dtypes)
```

RESULTADO

```
RUT                 object
Nombre_df           object
Sexo                object
Fecha_Nac           object
Monto                int64
Nombre_df2          object
Segundo Nombre      object
Primer Apellido     object
Segundo Apellido    object
dtype: object
```

En el ejemplo, podemos ver qué pasa al ocupar estos parámetros y cómo evitamos el error que apareció anteriormente.

Función join()

Recuerda revisar el archivo **m3-ej16.py →**

¿Qué pasa si queremos hacer `join` sobre dos Data Frames con distinto número de filas?

CÓDIGO

```
import pandas as pd
df = pd.read_csv("ejemplo.csv",encoding="latin-1",sep=";")

df2 = pd.DataFrame([["1",2],["3",4]])
df = df.join(df2)
print(df)
```

Se observa que al unir ambos Data Frames, la mayoría de las filas quedó con datos “NaN”, dado que tenían distintos números de filas.

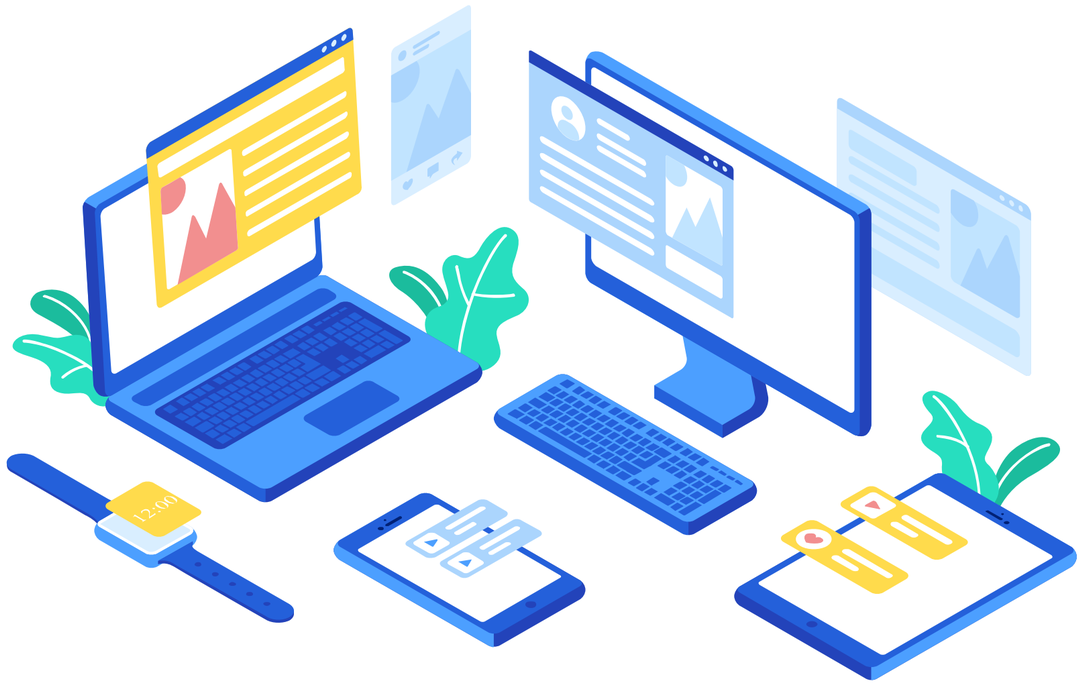
RESULTADO

	RUT	Nombre	Sexo	Fecha_Nac	Monto	0	1
0	13.626.365-6	Javier Andrés López Castro	MASCULINO	1963/2/17	9889868	1	2.0
1	17.135.958-1	Pedro Andrés Vergara Campos	MASCULINO	1993/10/24	4071184	3	4.0
2	15.391.058-0	Paula Blanca Marín Campos	FEMENINO	1962/5/30	9004634	NaN	NaN
3	8.296.689-7	Isabel Ignacia González Muñoz	FEMENINO	1972/3/20	1260523	NaN	NaN
4	15.755.894-8	Bernardo Vicente Campos Rodríguez	MASCULINO	1987/7/11	6515702	NaN	NaN

Unión de dos Data Frames

1

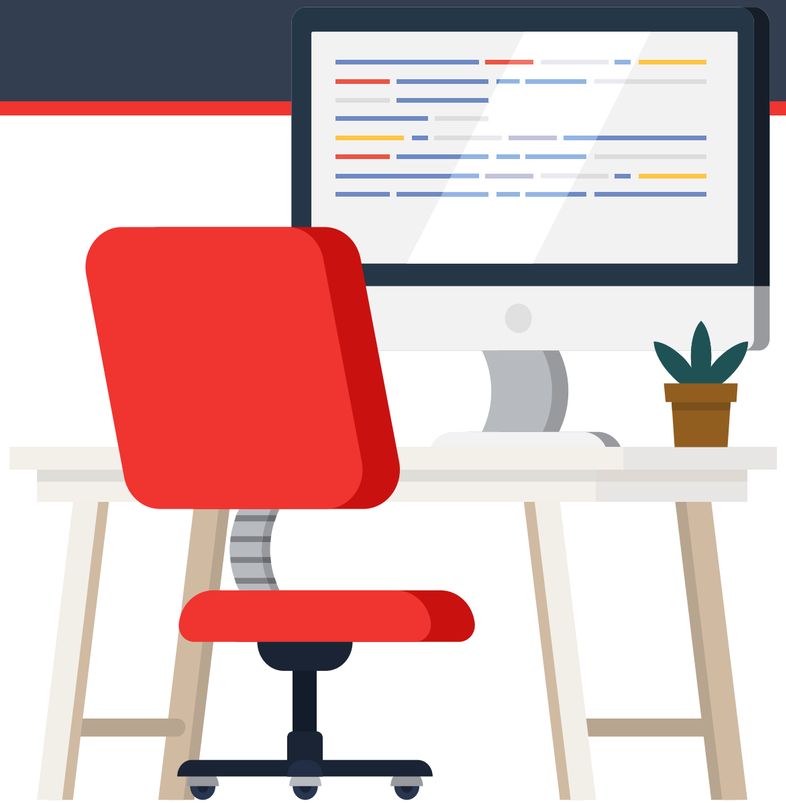
Hacer un `join` solo para unir las columnas de dos Data Frames distintos, es una visión muy acotada de esta funcionalidad. Pero **¿si quisiéramos unir dos Data Frame en base a valores en común entre ambos?**



¿Para qué sirve la función merge()?

FUNCIÓN
`merge()`

La función `merge()` nos permite **unir** la información de dos Data Frames distintos basados en los valores de una columna, cuando haya un valor en común.



Función merge()

Digamos que tenemos la información de los clientes de dos bancos distintos. Pueden notar que las personas con RUT “**13.626.365-6**” y “**17.135.958-1**” tienen cuenta en ambos bancos.

Banco A	
RUT	**Monto**
13.626.365-6	9889868
17.135.958-1	4071184
15.391.058-0	9004634
8.296.689-7	1260523

Banco B	
RUT	**Monto**
13.626.365-6	7565403
17.135.958-1	1237464
15.755.894-8	6515702
17.399.932-8	5507746

¿Cómo podríamos crear un Data Frame que solo tuviese la información de las personas que tienen cuenta en ambos bancos?

Función merge()

Según lo planteado anteriormente, nos gustaría crear el siguiente Data Frame:

RUT	Monto Banco A	Monto Banco B
13.626.365-6	9889868	7565403
17.135.958-1	4071184	1237464

Recuerda revisar el archivo **m3-ej17.py →**

Función merge()

Para unir dos Data Frames `df1` y `df2` distintos en base a una columna que comparta valores en común:

```
df1.merge(df2,on=columna en común)
```

Para el ejemplo del banco presentado anteriormente:

CÓDIGO

```
import pandas as pd
df_banco_A = pd.read_csv("bancoA.csv",encoding="latin-1",sep=";")
df_banco_B = pd.read_csv("bancoB.csv",encoding="latin-1",sep=";")

df_banco_A = df_banco_A.merge(df_banco_B,on="RUT",suffixes=("_bancoA","_bancoB"))

print(df_banco_A)
```

RESULTADO

```
            RUT  Monto_bancoA  Monto_bancoB
0  13.626.365-6       9889868       7565403
1  17.135.958-1       4071184       1237464
```

Función merge()

¿Qué pasaría si queremos hacer la misma operación, pero los dos Data Frames tienen columnas de RUT con distinto nombre? Podemos hacerlo sin necesidad de renombrar las columnas.

Digamos que los Data Frames de los bancos A y B respectivamente son los siguientes:

Banco A	
ID	**Monto**
13.626.365-6	9889868
17.135.958-1	4071184
15.391.058-0	9004634
8.296.689-7	1260523

Banco B	
RUT	**Monto**
13.626.365-6	7565403
17.135.958-1	1237464
15.755.894-8	6515702
17.399.932-8	5507746

Recuerda revisar el archivo **m3-ej18.py** →

Función merge()

Podemos manejar este tipo de casos utilizando la función `merge` de la siguiente manera:

CÓDIGO

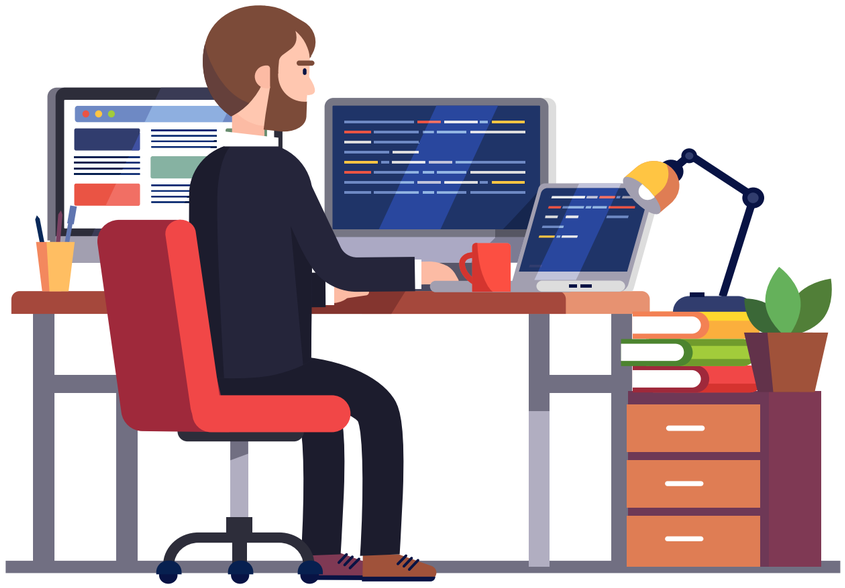
```
import pandas as pd
df_banco_A = pd.read_csv("bancoA.csv",encoding="latin-1",sep=";")
df_banco_B = pd.read_csv("bancoB.csv",encoding="latin-1",sep=";")

df_banco_A = df_banco_A.merge(df_banco_B,left_on="ID",right_on="RUT",suffixes=("_bancoA","_bancoB"))

print(df_banco_A)
```

RESULTADO

```
              ID  Monto_bancoA           RUT  Monto_bancoB
0  13.626.365-6       9889868  13.626.365-6       7565403
1  17.135.958-1       4071184  17.135.958-1       1237464
```

Basta con que agreguemos los parámetros `left_on` y `right_on` para identificar en qué columnas vamos a buscar valores en común.

Función merge()



¿Cuáles otros modos hay?
Vamos a ver dos más: left y outer

En realidad el “`merge`” que estamos haciendo tiene un modo, que se llama “`inner`”.

Para aquellos que están familiarizados con SQL, es el equivalente a un *Inner Join* entre dos tablas.

Aquí, el Data Frame resultante estará compuesto solo por aquellos valores que se encuentren en ambos.

Función merge()

Tenemos los siguientes Data Frames representando cuentas de dos bancos:

Banco A	
ID	**Monto**
13.626.365-6	9889868
17.135.958-1	4071184
15.391.058-0	9004634
8.296.689-7	1260523

Banco B	
RUT	**Monto**
15.755.894-8	6515702
13.626.365-6	7565403
17.135.958-1	1237464
15.755.894-8	6515702

Función merge() modo “left”

Recuerda revisar el archivo **m3-ej19.py** →

CÓDIGO

```
import pandas as pd
df_banco_A = pd.read_csv("bancoA.csv",encoding="latin-1",sep=";")
df_banco_B = pd.read_csv("bancoB_2.csv",encoding="latin-1",sep=";")

df_banco_A = df_banco_A.merge(df_banco_B,left_on="ID",right_on="RUT",suffixes=("_bancoA","_bancoB“), how="left")

print(df_banco_A)
```

RESULTADO

```
             ID  Monto_bancoA           RUT  Monto_bancoB
0  13.626.365-6       9889868  13.626.365-6     7565403.0
1  17.135.958-1       4071184  17.135.958-1     1237464.0
2  15.391.058-0       9004634           NaN           NaN
3   8.296.689-7       1260523           NaN           NaN
```

Importante

En el `merge` modo “`left`”, se mantienen todas las filas del Data Frame donde estamos haciendo el `merge` y solo se agregan las columnas del otro Data Frame que tengan valores en común según la columna que estemos analizando.

Función merge() modo “left”

Recuerda revisar el archivo **m3-ej19.py** →

CÓDIGO

```
import pandas as pd
df_banco_A = pd.read_csv("bancoA.csv",encoding="latin-1",sep=";")
df_banco_B = pd.read_csv("bancoB_2.csv",encoding="latin-1",sep=";")

df_banco_A = df_banco_A.merge(df_banco_B,left_on="ID",right_on="RUT",suffixes=("_bancoA","_bancoB"), how="left")

print(df_banco_A)
```

RESULTADO

```
             ID  Monto_bancoA           RUT  Monto_bancoB
0  13.626.365-6       9889868  13.626.365-6     7565403.0
1  17.135.958-1       4071184  17.135.958-1     1237464.0
2  15.391.058-0       9004634           NaN           NaN
3   8.296.689-7       1260523           NaN           NaN
```

En el Data Frame resultante, todas las personas del Banco A quedaron. Además, *Pandas* buscó cuáles de las personas del Banco A estaban en el B y también rescató esos valores.

Recuerda revisar el archivo **m3-ej20.py** →

Función merge() modo “outer”

Si tenemos los mismos Data Frames anteriores representando cuentas de dos bancos, veamos qué ocurre al aplicar esta función.

CÓDIGO

```
import pandas as pd
df_banco_A = pd.read_csv("bancoA.csv",encoding="latin-1",sep=";")
df_banco_B = pd.read_csv("bancoB_2.csv",encoding="latin-1",sep=";")

df_banco_A = df_banco_A.merge(df_banco_B,left_on="ID",right_on="RUT",suffixes=("_bancoA","_bancoB“), how="outer")

print(df_banco_A)
```

RESULTADO

```
              ID  Monto_bancoA           RUT  Monto_bancoB
0  13.626.365-6     9889868.0  13.626.365-6     7565403.0
1  17.135.958-1     4071184.0  17.135.958-1     1237464.0
2  15.391.058-0     9004634.0           NaN           NaN
3   8.296.689-7     1260523.0           NaN           NaN
4           NaN           NaN  15.755.894-8     6515702.0
```

Importante

En el `merge` modo “`outer`”, se mantienen todas las filas del Data Frame donde estamos haciendo el `merge` y se agregan todas las filas y columnas del otro Data Frame que estemos agregando. En aquellas donde hay un valor en común en la columnas que se está analizando, se juntan las columnas. En aquellas que no, se llena con valores nulo para el Data Frame donde se está haciendo el `merge` e igualmente para el Data Frame que se está agregando.

Función merge() modo “outer”

Recuerda revisar el archivo **m3-ej20.py** →

Si tenemos los mismos Data Frames anteriores representando cuentas de dos bancos, veamos qué ocurre al aplicar esta función.

CÓDIGO

```
import pandas as pd
df_banco_A = pd.read_csv("bancoA.csv",encoding="latin-1",sep=";")
df_banco_B = pd.read_csv("bancoB_2.csv",encoding="latin-1",sep=";")

df_banco_A = df_banco_A.merge(df_banco_B,left_on="ID",right_on="RUT",suffixes=("_bancoA","_bancoB“), how="outer")

print(df_banco_A)
```

RESULTADO

```
              ID  Monto_bancoA           RUT  Monto_bancoB
0  13.626.365-6     9889868.0  13.626.365-6     7565403.0
1  17.135.958-1     4071184.0  17.135.958-1     1237464.0
2  15.391.058-0     9004634.0           NaN           NaN
3   8.296.689-7     1260523.0           NaN           NaN
4           NaN           NaN  15.755.894-8     6515702.0
```

En el Data Frame resultante, todas las personas del Banco A y del Banco B quedaron. Es decir, se buscan aquellas filas que tengan valores en común en la columna que se está analizando y se juntan (como podemos ver en las dos primeras filas).

Función merge()

Importante

Al hacer `merge`, las columnas donde se buscarán valores en común deben ser del mismo tipo.

Esto se puede revisar con el comando `dtypes`.

Podemos transformar el tipo de una columna con la función `astype(tipo_de_dato)`. Específicamente, el parámetro que se ingresa es el tipo de dato al que se quiere convertir la columna.

`merge()`

Conclusiones

- En este módulo aprendimos varias funciones de `strings` que pueden ser aplicadas sobre un Data Frame. Estas son sumamente importantes al procesar datos con la librería *Pandas*, especialmente en lo respectivo a la limpieza de información.
- Además, vimos cómo operar con más de un Data Frame, ya que en la vida real se trabaja con múltiples fuentes de información, y manejar las herramientas necesarias para integrarlas es muy útil.

>>> Cierre

Has finalizado la revisión de los contenidos de esta clase.

A continuación, te invitamos a realizar las actividades y a revisar los recursos del módulo que encontrarás en plataforma.