TELEDUC
Educación a Distancia

Herramientas avanzadas de programación en *Python* para procesamiento de datos

Resumen

Módulo 3

Importante: Para todos los códigos que se entregarán en este resumen, se asume que ya se ejecutó la siguiente línea de código:

```
import pandas as pd
```

Además, que existe un Data Frame creado de nombre df

En este módulo se explicarán las funciones para trabajar con *strings* en un Data Frame y operar entre Data Frames.

Para comenzar, es posible iterar sobre los valores de un Data Frame. Esto podría particularmente aplicar cualquier modificación que se requiera hacer sobre cada una de las filas y/o columnas de un Data Frame. No obstante, no se recomienda hacer ya que es poco eficiente, y en general estas modificaciones se pueden de hacer de otras maneras que no afecten la eficiencia.

Para poder iterar en un Data Frame df:

```
for index,row in df.iterrows():
```

    #código que se desea ejecutar.

Mediante la variable *index* del *for* uno accede al *index*, y *row* accede a la fila completa. Es posible acceder al valor de cualquier columna mediante:

```
row[nombre columna]
```

Es importante mencionar que index,row **debe** ir en la declaración del for, en caso contrario *Python* arrojará un error.

Como se mencionó anteriormente, no se recomienda hacer operaciones de *strings* interando sobre los valores de un Data Frame sino que es mejor aplicar funciones de *strings* sobre estos. Es posible aplicar funciones de *strings* sobre un df cualquiera con columnas de tipo *object* de la siguiente manera:

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

En particular, se mostrarán algunas funciones que se pueden aplicar sobre columnas de tipo *object* para un df cualquiera:

- **Funcion len():** Nos permite saber la cantidad de caracteres de los elementos de una columna de tipo *object*. De forma general:

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

En *Pandas* también es posible extraer uno o más caracteres de una columna:

- **Extraer un carácter:** Es posible extraer un carácter mediante su posición. De forma general:

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

- **Extraer varios carácter:** Es posible extraer un carácter mediante un intervalo. De forma general:

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

Importante: Este intervalo es un equivalente al *slice* de los *strings*. Es decir, si uno tiene un intervalo de 0 a 4, entonces extraerá los caracteres 0,1,2, y 3.

Volviendo a las funciones de strings:

- **Función lower():** Sirve para transformar los caracteres a minúsculas de una columna de un Data Frame. Si se desea guardar este cambio debe volver a asignarlo a la columna original. De forma general:

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

- **Función upper():** Sirve para transformar los caracteres a mayúsculas de una columna de un Data Frame. Si se desea guardar este cambio debe volver a asignarlo a la columna original. De forma general:

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

- **Función replace(x,y):** Recibe como parámetros los *strings* x e y. Se usa para reemplazar los *strings* x por el *string* y en los valores de una columna de un Data Frame. Si se desea guardar este cambio debe volver a asignarlo a la columna original. De forma general:

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

- **Función contains(x):** Recibe como parámetro un *string* x. Permite saber si los valores de una columna de un Data Frame tienen este *string* x. De forma general:

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

- **Función find(x):** Recibe como parámetro un *string* x. Permite saber la primera ocurrencia del *string* x en una columna de un Data Frame. De forma general:

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

- **Función split(x):** Recibe como parámetro un *string* x. Permite separar, según un *string* x, los valores de una columna de un Data Frame. De forma general:

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

El retorno de esta función es una lista dentro de una columna. No obstante, es posible crear un Data Frame nuevo a partir de esta función añadiendo un parámetro *expand*, con el valor *True*, de la siguiente manera:

```
df["nombre columna"].str.split(x,expand=True)
```

El retorno de la función con el parámetro anterior es un Data Frame cuyas columnas no tienen nombre. Es posible cambiar el nombre de estas columnas de la siguiente manera:

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

Donde cada valor de la lista que está al lado derecho del signo "igual" corresponderá al nombre de cada una de las columnas de df. Obviamente, se esperaría que esta lista tenga tantos elementos como columnas tenga df.

Con las funciones anteriores, y el *split* particularmente, se hace útil conocer cómo operar con más de un Data Frame. Para eso se inicia aprendiendo la función *join*.

- **Función join():** Esta función agrega las columnas de un Data Frame a otro. Si se tiene otro dataframe df2, de forma general:

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

Es importante mencionar que si hay diferencias en la cantidad de filas entre df y df2, quedarán valores NaN (en el lado del Data Frame con menos filas).

- **Función merge():** Esta función actúa como un *join* de SQL. Tiene varios modos (equivalentes a los distintos *join* de SQL). A continuación, se detallará el modo *inner*:

Modo *Inner*:

Su función es buscar valores en una columna específica de dos Data Frames distintos. Cuando encuentra un valor que está contenido en ambos Data Frames de esta columna en específico, entonces une ambas filas completas. De forma general:

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

Esta columna tiene que estar en df1 y df2.

Por ejemplo, si df1 es equivalente a la siguiente tabla (Banco A del ejemplo de clases):

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

Y df2 es equivalente a la siguiente tabla (Banco B del ejemplo de clases):

RUT	Monto
13.626.365-6	7565403
17.135.958-1	1237464

15.755.894-8	6515702
17.399.932-8	5507746

Entonces al aplicar:

```
df1.merge(df2, on=RUT)
```

Resulta:

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

Ya que **sólo** los RUT 13.626.365-6 y 17.135.958-1 se encontraban en df1 y df2 simultáneamente. Se observa entonces que el Data Frame resultante tiene **solamente** las filas donde había un valor de la columna en específico (RUT en este caso) que se compartía entre ambos Data Frames.

Importante:

Al aplicar

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

Se entiende que df1 es el lado “izquierdo” y df2 el lado “derecho”.

Puede ocurrir que la columna RUT esté en ambos Data Frames, pero con nombres distintos. En este caso, se especifica dos parámetros "*left_on*", y "*right_on*", diciendo cómo se llama la columna donde se buscarán los valores en cada Data Frame respectivamente.

Modo *Left*:

De forma general:

```
df1.merge(df2,on=columna en específico,how="left")
```

En este caso, el Data Frame resultante tiene **todas** las filas de df1. Asimismo, donde encuentre valores en común con df2, agregará sus columnas a las filas de df1. Del ejemplo de clases, el Data Frame resultante sería:

RUT	Monto_BancoA	Monto_BancoB
13.626.365-6	9889868	7565403
17.135.958-1	4071184	1237464
15.391.058-0	9004634	NaN
8.296.689-7	1260523	NaN

Modo *Outer*:

De forma general:

```
df1.merge(df2,on=columna en específico,how="outer")
```

En este caso, el Data Frame resultante tiene **todas** las filas de df1 y df2, independiente si hay valores en común o no. Donde encuentre valores en común con df2, agregará sus columnas a las filas de df1. Del ejemplo de clases, el Data Frame resultante sería:

RUT	Monto_BancoA	Monto_BancoB
13.626.365-6	9889868	7565403
17.135.958-1	4071184	1237464
15.391.058-0	9004634	NaN
8.296.689-7	1260523	NaN
15.755.894-8	NaN	6515702
17.399.932-8	NaN	5507746