Cargando Datos de un CSV en Python con Pandas

SOLUTION DESIGN CENTROAMERICA
www.solutiondesign.tech

Guatemala, 30 de abril de 2021
Pablo Sao Alonzo[1]
1. Departamento de diseño y desarrollo de Software, Solution Design of Centroamerica, Guatemala.

¿Qué es Pandas?

Es un paquete que se ha tornado esencial para el análisis de información en Python, ganando popularidad desde el 2010. Dicho paquete fue elaborado por Wes Mckinney en el 2008, donde su nombre se deriva de "*panel data*", término utilizado para el conjunto de datos estructurados y multidimencionales (Harrison y Petrou, 2020; Stepanek, 2020).

Este paquete facilita la manipulación, preparación y la limpieza de datos estructurados, como archivos CSV, hojas de cálculo (como Excel) o bases de datos estructuradas. Esta librería cuenta con dos tipos de estructuras; la estructura de *Serie*, conformada por una secuencia de valores similares al tipo de NumPy. Y la otra estructura disponible en este paquete es el *DataFrame*, el cual representa una tabla de datos que contiene una colección de columnas que pueden ser de diferentes tipos (numérico, alfanumérico, booleano, etc.), teniendo está estructura índices de columnas y filas, en la que la información puede ser almacenada en bloques de una o más dimensiones, haciendo más fácil su selección, combinación y transformación de información (Harrison y Petrou, 2020; Walker, 2020, McKinney, 2018).

Cargando Datos de un CSV

La función para poder cargar a memoria los datos de un archivo CSV con pandas se llama **read_csv**, la cual cuenta con diversos parámetros para poder manipular la lectura del archivo y obtener los datos. Viendo los principales parámetros de manipulación que nos pueden llegar a servir en la práctica.

Esta función de Pandas es una de las más completas, en términos de opciones para normalizar los datos, pudiendo incluso manejar dos motores de análisis; uno de estos es el motor de C, el cual es más eficaz que el motor de Python, ya que cuenta con mayor precisión en valores flotantes, que puede ser especificado utilizando el parámetro **float_precision** dentro de la función read_csv. Sin embargo, se debe indicar mediante el parámetro **engine** con una **'c'** dentro de dicha función, de lo contrario utilizará por defecto el motor de Python.

Para poder iniciar la ejemplificación del uso de esta función, supondremos que el lector está familiarizado con la sintaxis de Python y con Jupyter Notebook, donde el archivo de datos que estaremos utilizando; así como, el archivo de Jupyter, puede ser accedido y descargado desde el siguiente repositorio en GitHub: https://github.com/sdesignca/blog-cargando-csv-pandas

Para iniciar, importamos la librería de pandas en nuestro archivo, donde le colocaremos el alías pd.

```
import pandas as pd
```

Luego ubicaremos el path o la ruta donde se encuentra ubicado nuestro archivo "***Datos.csv***". En nuestro caso el archivo se encuentra en la misma ubicación donde está nuestro 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_csv** y la información cargada la asignara a la variable **datos**.

```
datos = pd.read_csv("Datos.csv")
```

En unos casos, podemos llegar a tener otro tipo de delimitador en nuestros archivos CSV, por ejemplo un punto y coma (;), por lo que podemos utilizar el parámetro ***delimiter*** y especificar cual será el carácter que está delimitando la información entre si. En el archivo CSV de ejemplo el delimitador es una coma (,), este es el parámetro por defecto que toma la función ***read_csv***, sin embargo para ejemplificar su uso dentro de la función incluiremos la coma (,) con el parámetro ***delimiter***.

```
datos = pd.read_csv("Datos.csv",delimiter=',')
```

Al ejecutar esta instrucción y asignarla a la variable datos. Mostraremos su contenido utilizando la función head() de Python:

```
datos.head()
```

Al ser ejecutado, mostrará la información cargada en la variable desde el archivo; sin embargo, los caracteres especiales no son desplegados:

	C�digo	Apellidos	Nombres	G�nero	Edad	Fecha de Ingreso	Puesto	Ubicaci�n	Sueldo Base	Bonificaci�n	Total
0	1	Mazariegos	Rene	M	19	1/01/2007	Cajero	105	2,400.00	1,000.00	3,400.00
1	2	Ramirez	Gladys	NaN	35	1/01/2008	Personal de servicio	107	2,300.00	1,000.00	3,300.00
2	3	Ortiz	Silvia	F	42	1/06/2008	Gerente T	302	3,500.00	1,000.00	4,500.00
3	4	Alvarado	Maria	F	22	7/07/2008	Personal de servicio	107	2,300.00	1,000.00	3,300.00
4	5	Perez	Isabel	F	38	9/12/2008	Supervisor	205	2,900.00	1,000.00	3,900.00

Para poder corregir esto, vamos a pasar el parámetro ***encoding*** en la función read_csv, con el valor latin1 (si se desean conocer que otras opciones están disponibles se puede visitar el link https://docs.python.org/3/library/codecs.html#standard-encodings), quedando la instrucción:

```
datos = pd.read_csv("Datos.csv",delimiter=',',encoding='latin1')
```

Al mostrar nuevamente los valores con la función *head()*, obtendremos la información cargada en el *DataFrame* con la codificación latina, mostrando así los caracteres especiales contenidos:

	Código	Apellidos	Nombres	Género	Edad	Fecha de Ingreso	Puesto	Ubicación	Sueldo Base	Bonificación	Total
0	1	Mazariegos	Rene	M	19	1/01/2007	Cajero	105	2,400.00	1,000.00	3,400.00
1	2	Ramirez	Gladys	NaN	35	1/01/2008	Personal de servicio	107	2,300.00	1,000.00	3,300.00
2	3	Ortiz	Silvia	F	42	1/06/2008	Gerente T	302	3,500.00	1,000.00	4,500.00
3	4	Alvarado	Maria	F	22	7/07/2008	Personal de servicio	107	2,300.00	1,000.00	3,300.00
4	5	Perez	Isabel	F	38	9/12/2008	Supervisor	205	2,900.00	1,000.00	3,900.00

En este punto, ya tenemos nuestra información cargada en nuestra variable datos, con los caracteres especiales disponibles; sin embargo, tenemos una columna que contiene fechas y podríamos pensar que estos valores son de tipo fecha (o *datetime*), por lo que revisaremos el tipo de dato de cada una de las columnas con la siguiente propiedad del *DataFrame* de Pandas:

```
datos.dtypes
```

Obteniendo el siguiente resultado:

```
Código              int64
Apellidos          object
Nombres            object
Género             object
Edad                int64
Fecha de Ingreso   object
Puesto             object
Ubicación           int64
Sueldo Base       float64
Bonificación      float64
Total             float64
dtype: object
```

Como se puede observar, la columna “Fecha de Ingreso” es de tipo **object**, para poder cambiar esto, podemos convertir el tipo de la columna, luego de cargar la información; sin embargo, en este documento, estamos mostrando el uso de la función **read_csv**, por lo que vamos a convertir esta columna a tipo fecha desde el momento en el que se carga la información utilizando el parámetro **parse_dates**, donde enviamos como parámetro un arreglo con el nombre de las columnas que deseamos convertir a tipo fecha, en nuestro caso el arreglo seria **[‘Fecha de Ingreso’]**:

```
datos = pd.read_csv("Datos.csv"
                    ,delimiter=','
                    ,encoding='latin1'
                    ,parse_dates=['Fecha de Ingreso']
                    )
```

Luego de ejecutar nuevamente esta instrucción y cargar los datos del archivo a la variable datos, verificamos nuevamente el tipo de datos de las columnas:

```
datos.dtypes
```

Al obtener el resultado, podemos observar que la columna “Fecha de Ingreso” ya es de tipo fecha o *datetime*

```
Código                      int64
Apellidos                  object
Nombres                    object
Género                     object
Edad                        int64
Fecha de Ingreso    datetime64[ns]
Puesto                     object
Ubicación                   int64
Sueldo Base               float64
Bonificación              float64
Total                     float64
dtype: object
```

Sin embargo en un análisis no siempre es necesaria toda la información contenida en un archivo, por lo que podemos pasar como parámetro las columnas que deseamos sean cargadas en la memoria. Esto lo hacemos pasando un arreglo con el nombre de las columnas que contiene nuestro archivo CSV en el parámetro **usecols**, en nuestro caso vamos a cargar únicamente las columnas “Código”, “Fecha de Ingreso”, “Sueldo Base” y “Bonificación”:

```
datos = pd.read_csv("Datos.csv"
          ,delimiter=','
          ,encoding='latin1'
          ,parse_dates=['Fecha de Ingreso']
          ,usecols=['Código','Fecha de Ingreso','Sueldo Base','Bonificación']
          )
```

Al ejecutar este comando y hacer uso de la función *head()* sobre la variable que contiene el *DataFrame*, obtenemos unicamente la información de las columnas que hemos especificado.

	Código	Fecha de Ingreso	Sueldo Base	Bonificación
0	1	2007-01-01	2,400.00	1,000.00
1	2	2008-01-01	2,300.00	1,000.00
2	3	2008-01-06	3,500.00	1,000.00
3	4	2008-07-07	2,300.00	1,000.00
4	5	2008-09-12	2,900.00	1,000.00

Estos suelen ser los parámetros de más ayuda al momento de cargar un archivo CSV, sin embargo existen más parámetros dentro de la función **read_csv** que nos pueden llegar a ser de utilidad, por lo que si se desea profundizar en las demás opciones que no fueron cubiertas, puede visitarse el siguiente link de documentación:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

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.: . 124 – 130 pp.

Stepanek, H. (2020). *Thinking in Pandas: How to use the Python Data Analysis Library the Right Way*. Portland, E.E.U.U: Apress. 67 – 69 pp.

Walker, M. (2020). *Python Data Cleaning: cookbook*. Birmingham, U.K: Packt Publishing. 2 – 8 pp.