El futuro digital es de todos

MinTIC

Pandas – Introducción

OPERADO POR:

Universidad Tecnológica de Pereira

Misión TIC 2022

RUTA DE APRENDIZAJE 1

Pandas

¿Qué es el Data Science?

El **Data Science** es un campo interdisciplinario que involucra métodos científicos, procesos y sistemas para extraer conocimiento o un mejor entendimiento de datos en sus diferentes formas, ya sea estructurados o no estructurados, lo cual es una continuación de algunos campos de análisis de datos como la estadística, la minería de datos, el aprendizaje automático y la analítica predictiva.

El **Data Science** combina software, estadística, matemática, programación y visualización. Y su objetivo es extraer datos factibles de interpretarse e incluso crear nueva información. Las conclusiones que se obtienen permiten desarrollar productos demandados en el mercado o generar oportunidades de negocio de una empresa.

¿Qué es pandas?

El paquete **pandas** es la herramienta más importante a disposición de los científicos y analistas de datos que trabajan en Python en la actualidad. Pandas es la columna vertebral de la mayoría de los proyectos de datos desarrollados con Python.

En Computación y Ciencia de datos, **pandas** es una biblioteca de software escrita como extensión de **NumPy** para manipulación y análisis de datos para el lenguaje de programación Python.

Esta librería ofrece dos de las estructuras más usadas en **Data Science**: la estructura Series y el DataFrame. En estas clases veremos cómo crearlas, las herramientas básicas de uso y algunas de las funciones y métodos que nos permitirán extraer todo el potencial de ellas.

Las características de la biblioteca son:

•El tipo de datos DataFrame para manipulación de datos con indexación integrada. Tiene herramientas para leer y escribir datos entre estructuras de datos en memoria y formatos de archivos variados.

•Permite la alineación de datos y manejo integrado de datos faltantes, la reestructuración y segmentación de conjuntos de datos, la segmentación vertical basada en etiquetas, indexación elegante, y segmentación horizontal de grandes conjuntos de datos, la inserción y eliminación de columnas en estructuras de datos.

•Puedes realizar cadenas de operaciones, dividir, aplicar y combinar sobre conjuntos de datos, la mezcla y unión de datos.

•Permite realizar indexación jerárquica de ejes para trabajar con datos de altas dimensiones en estructuras de datos de menor dimensión, la funcionalidad de series de tiempo: generación de rangos de fechas y conversión de frecuencias, desplazamiento de ventanas estadísticas y de regresiones lineales, desplazamiento de fechas y retrasos.

Primeros pasos de Pandas

Instalar e importar

Pandas es un paquete fácil de instalar. Solo debemos abrir la línea de comandos (para usuarios de PC) e instálelo usando cualquiera de los siguientes comandos:

conda install pandas **o** **pip install pandas**

Si esta trabajando sobre un cuaderno de **Jupyter**, puede ejecutar esta celda:

```
In [13]: ! pip install pandas
```

El ! al principio ejecuta las celdas como si estuvieran en una terminal.

Para importar pandas, generalmente lo importamos con un nombre más corto ya que facilita mucho el proceso de programación:

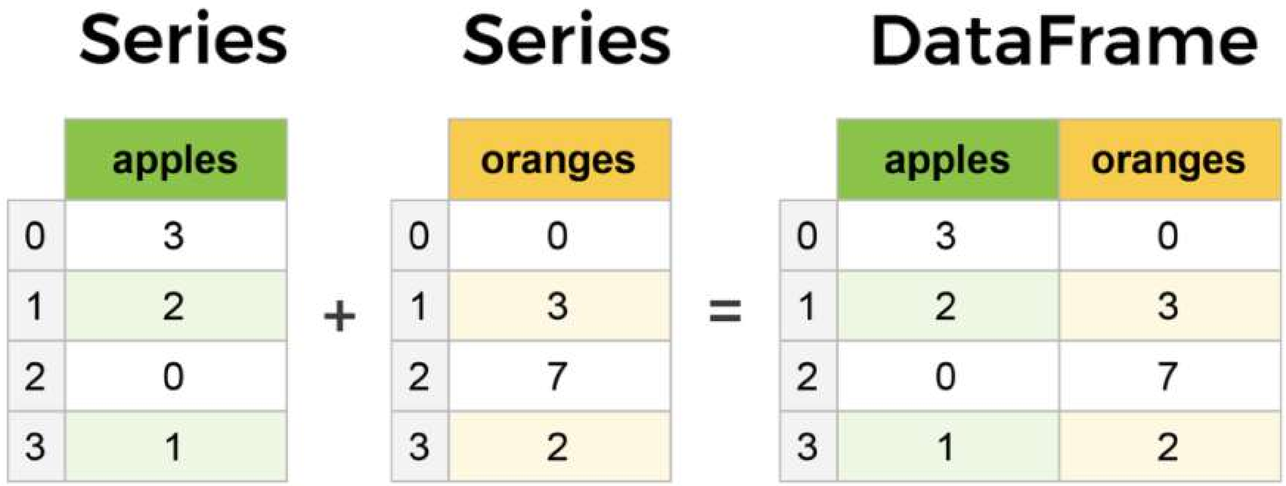
```
In [14]: import pandas as pd
```

Componentes principales de pandas: Series y DataFrames

Los dos componentes principales de los pandas son Series y DataFrame.

Una Serie es esencialmente una columna .

Un DataFrame es una tabla multidimensional formada por una colección de Series.



Series

Las series son estructuras unidimensionales conteniendo un array de datos (de cualquier tipo soportado por **NumPy**) y un array de etiquetas que van asociadas a los datos, llamado índice (**index** en la literatura en inglés):

```
In [15]: ventas = pd.Series([15,12,21], index = ["Ene", "Feb", "Mar"])
         ventas

Out[15]: Ene    15
         Feb    12
         Mar    21
         dtype: int64
```

Los elementos de la serie pueden extraerse con el nombre de la serie y, entre corchetes, el índice (posición) del elemento:

```
In [16]: ventas[0]

Out[16]: 15
```

o con su etiqueta, si la tiene:

```
In [17]: ventas["Ene"]

Out[17]: 15
```

Las etiquetas que forman el índice no necesitan ser diferentes. Pueden ser de cualquier tipo (numérico, textos, tuplas...) siempre que sea posible aplicar la función hash sobre ellas.

Es de destacar que el lazo entre una etiqueta y un valor se mantendrá salvo que lo modifiquemos explícitamente. Esto quiere decir que filtrar una serie o eliminar un elemento de la serie, por ejemplo, no va a modificar las etiquetas asignadas a cada valor.
Otro comentario importante es al respecto de la inmutabilidad del índice de etiquetas: aun cuando es posible asignar a una serie un nuevo conjunto de etiquetas a través del atributo index, intentar modificar un único valor del index va a devolver un error.

Al igual que ocurre con el array **NumPy**, una serie pandas solo puede contener datos de un mismo tipo. En la imagen anterior puede apreciarse el índice a la izquierda ("Ene", "Feb" y "Mar") y los datos a la derecha (15, 12 y 21). El tipo de la serie, accesible a través del atributo **dtype** (Se muestra en la parte inferior: int64), coincide con el tipo de los datos que contiene:

```
In [18]: ventas.dtype
Out[18]: dtype('int64')
```

Podemos acceder a los objetos que contienen los índices y los valores a través de los atributos **index** y **values** de la serie, respectivamente:

```
In [19]: ventas.index
Out[19]: Index(['Ene', 'Feb', 'Mar'], dtype='object')

In [20]: ventas.values
Out[20]: array([15, 12, 21], dtype=int64)
```

Puede apreciarse en el ejemplo que el índice es de tipo "objeto".

La serie tiene, además, un atributo name, atributo que también encontramos en el índice. Una vez los hemos fijado, se muestran junto con la estructura al imprimir la serie:

```
In [21]: ventas.name = "Ventas 2020"
         ventas.name

Out[21]: 'Ventas 2020'

In [22]: ventas

Out[22]: Ene    15
         Feb    12
         Mar    21
         Name: Ventas 2020, dtype: int64

In [23]: ventas.index.name = "Meses"

In [24]: ventas

Out[24]: Meses
         Ene    15
         Feb    12
         Mar    21
         Name: Ventas 2020, dtype: int64
```

Obsérvese cómo, en este ultimo ejemplo, en la salida, tanto la serie como el índice se muestran con su nombre ("Ventas 2020" y "Meses", respectivamente).

El atributo **axes** nos da acceso a una lista con los ejes de la serie (solo contiene un elemento al tratarse de una estructura unidimensional):

```
In [25]: ventas.axes
Out[25]: [Index(['Ene', 'Feb', 'Mar'], dtype='object', name='Meses')]
```

El atributo **shape** nos devuelve el tamaño de la serie:

```
In [26]: ventas.shape

Out[26]: (3,)
```

DataFrames:

Crear DataFrames directamente en Python y es bastante útil cuando se prueban nuevos métodos y funciones que se encuentran en los documentos de pandas.

Hay muchas formas de crear un DataFrame desde cero, pero una gran opción es usar un simple **dict**.
Digamos que tenemos un puesto de frutas que vende manzanas y naranjas. Queremos tener una columna para cada fruta y una fila para cada compra del cliente.

Para organizar esto como un diccionario para pandas, podríamos hacer algo como:

```
In [27]: datos = { 'manzanas' : [ 3 , 2 , 0 , 1 ], 'naranjas' : [ 0 , 3 , 7 , 2 ] }
```

Y luego páselo al constructor de Pandas DataFrame:

```
In [28]: compras = pd.DataFrame( datos )
         compras
```

Out[28]:

	manzanas	naranjas
0	3	0
1	2	3
2	0	7
3	1	2

¿Cómo funciona esto?

Cada elemento (clave, valor) en "datos" corresponde a una columna en el DataFrame resultante.

El índice de este DataFrame se nos dio en la creación como los números 0-3, pero también podríamos crear el nuestro cuando inicializamos el DataFrame.

Tengamos nombres de clientes como nuestro índice:

```
In [29]: compras = pd.DataFrame( datos , index = [ 'Juno' , 'Robert' , 'Lily' , 'David' ])
         compras
```

Out[29]:

	manzanas	naranjas
Juno	3	0
Robert	2	3
Lily	0	7
David	1	2

Las etiquetas de filas y de columnas -los índices- son accesibles a través de los atributos index y columns, respectivamente:

```
In [30]: compras.index
Out[30]: Index(['Juno', 'Robert', 'Lily', 'David'], dtype='object')

In [31]: compras.columns
Out[31]: Index(['manzanas', 'naranjas'], dtype='object')
```

La nomenclatura usada por pandas puede resultar un tanto confusa en lo que se refiere a los índices: tanto la estructura que contiene las etiquetas de filas como la que contiene las etiquetas de columnas son objetos de tipo **Index** ("índice", en español), pero, como se ha comentado, el índice de filas se denomina también **index** (aunque en minúsculas), y el de columna, **columns**.

Además, el nombre de "indice" se aplica normalmente a la referencia de un dato en una estructura según su posición. Por ejemplo, en la lista m = ["a", "b"], el índice del primer elemento es el número o valor que, añadido entre corchetes tras el nombre de la lista, nos permite acceder al elemento.

Así, el índice del elemento "a" en la lista mencionada es 0, y el índice del elemento "b" es 1, lo que no es del todo coherente con el concepto de "índice" de una estructura pandas cuando lo especificamos explícitamente.

Para evitar esta confusión, hablaremos normalmente de "índices" (en plural) para referirnos a estas dos estructuras (de filas y columnas), de "índice" (en singular) para referirnos al índice de etiquetas del eje vertical, y de "índice de columnas" y de "índice de filas" siempre que sea necesario remarcar a cuál estamos refiriéndonos.

```
In [32]: compras.index
Out[32]: Index(['Juno', 'Robert', 'Lily', 'David'], dtype='object')

In [33]: compras.columns
Out[33]: Index(['manzanas', 'naranjas'], dtype='object')
```

El eje 0 es el correspondiente al índice de filas (eje vertical) y el eje 1 al índice de columnas (eje horizontal). Como puede verse en el ejemplo anterior ejemplo, ambos índices son de tipo "objeto" (ya se ha comentado que, concretamente, son objetos de tipo Index).

El atributo axes devuelve una lista con los ejes de la estructura (dos, al tratarse de una estructura bidimensional):

```
In [34]: compras.axes

Out[34]: [Index(['Juno', 'Robert', 'Lily', 'David'], dtype='object'),
          Index(['manzanas', 'naranjas'], dtype='object')]
```

Al igual que ocurría con las series, los índices de filas y columnas son inmutables. Esto significa que, aunque podemos asignar un nuevo conjunto de datos (etiquetas) a ambas estructuras (index o columns), intentar modificar un único valor devolverá un error.

Tanto el índice de filas como el de columnas poseen el atributo name. Una vez fijado, se muestra al imprimir la estructura:

```
In [35]: compras.index.name = "Clientes"

In [36]: compras.columns.name = "Frutas"

In [37]: compras

Out[37]:
```

Frutas / Clientes	manzanas	naranjas
Juno	3	0
Robert	2	3
Lily	0	7
David	1	2

De forma semejante a como ocurría con las series, el atributo values de un dataframe nos permite acceder a los valores del **dataframe**, con formato array **NumPy** 2d:

```
In [38]: compras.values

Out[38]: array([[3, 0],
                [2, 3],
                [0, 7],
                [1, 2]], dtype=int64)
```

Este array tendrá un tipo u otro en función de los tipos de las columnas del **dataframe**, acomodándose de forma que englobe a todos ellos.

Y un **dataframe** también tiene un atributo **shape** que nos informa de su dimensionalidad y del número de elementos en cada dimensión. En el siguiente ejemplo Podemos ver que el **dataframe** compras tiene 4 filas y 2 columnas:

```
In [39]: compras.shape

Out[39]: (4, 2)
```

Creación de series:

El constructor para la creación de una serie pandas es pandas.Series. Este constructor acepta tres parámetros principales:

- **data:** estructura de datos tipo array, iterable, diccionario o valor escalar que contendrá los valores a introducir en la serie.
- **index:** estructura tipo array con la misma longitud que los datos. Si este argumento no se añade al crear la serie, se agregará un índice por defecto formado por números enteros desde 0 hasta n-1, siendo n el número de datos.
- **dtype:** tipo de datos para la serie. Si no se especifica, se inferirá a partir de los datos.

Los valores del índice, como ya se ha comentado anteriormente, no tienen que ser necesariamente distintos aunque ciertas operaciones pueden generar un error si no soportan la posibilidad de tener índices duplicados.

```
In [40]: s = pd.Series([7,5,3])
         s

Out[40]: 0    7
         1    5
         2    3
         dtype: int64
```

Al no haberse especificado un índice, se asigna uno automáticamente con los valores 0, 1 y 2.

Si repetimos esta instrucción especificando un índice:

```
In [41]: s = pd.Series([7,5,3], index = ["Ene", "Feb", "Mar"])
         s

Out[41]: Ene    7
         Feb    5
         Mar    3
         dtype: int64
```

Aquí vemos cómo el índice por defecto ha sido sustituido por el indicado. En este caso, la longitud del índice deberá coindicir con el número de elementos de la lista.

Los mismos comentarios podrían hacerse si, en lugar de una lista, hubiésemos partido de un array NumPy para crear la serie.

Utilizando un diccionario:

```
In [42]: d = {"Ene":7, "Feb":5, "Mar":3 }
         s = pd.Series(d)
         s

Out[42]: Ene    7
         Feb    5
         Mar    3
         dtype: int64
```

Aquí vemos cómo el constructor utiliza las claves como etiquetas del índice, y los valores del diccionario como valores de la serie. Si incluimos el índice explícitamente en el constructor, los valores en la serie se tomarán en el orden en el que estén en el índice explícito. Además, si en éste hay valores que no pertenecen al conjunto de claves del diccionario, se añaden a la serie con un valor NaN:

```
In [43]: d = {"Ene":7, "Feb":5, "Mar":3 }
         s = pd.Series(d, index = ["Abr", "Mar", "Feb", "Ene"],dtype=int)
         s

Out[43]: Abr    NaN
         Mar    3.0
         Feb    5.0
         Ene    7.0
         dtype: float64
```

En este ejemplo, hemos creado la serie especificando el índice que hemos formado dando la vuelta a las claves del diccionario ("Mar", "Feb" y "Ene") y hemos añadido a la lista de etiquetas el valor "Abr", que no pertenece al conjunto de claves del diccionario. Se ha añadido a la serie, pero se le ha asignado el valor NaN. Es precisamente la presencia de este valor lo que modifica el tipo de la serie a float.

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022