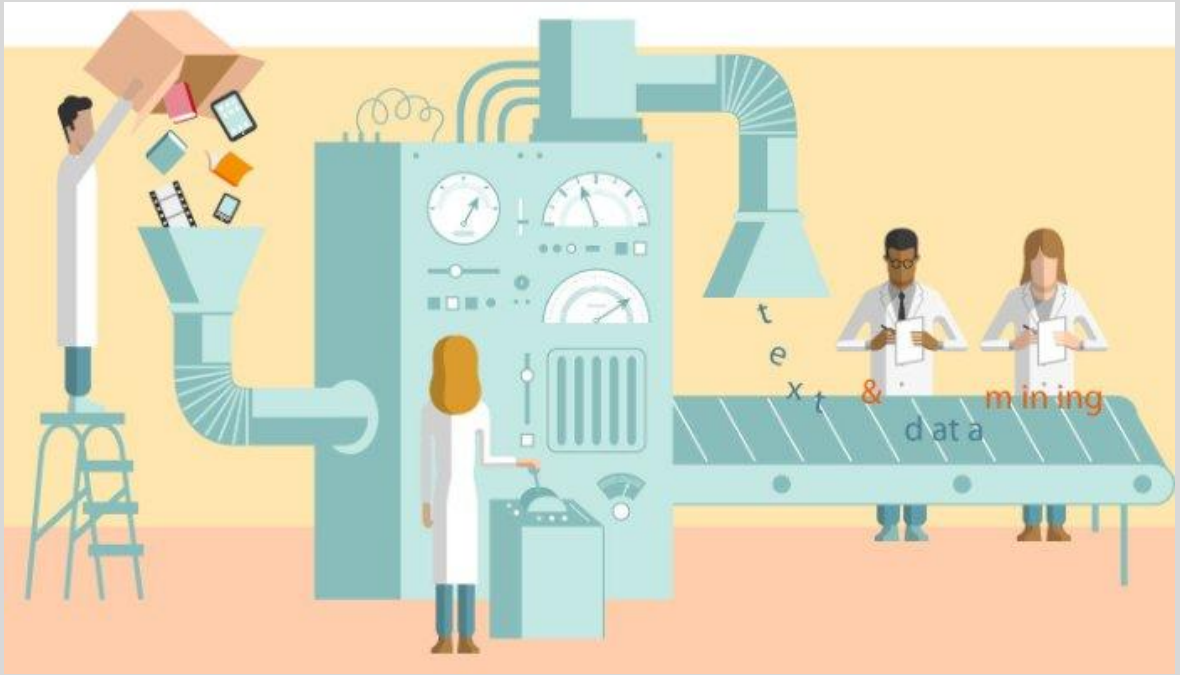

DigitalHouse > Coding School

DATA SCIENCE

UNIDAD 1
MÓDULO 2

Limpieza de datos con Pandas

Agosto 2017

Limpieza de datos con Pandas

1. Conocer las operaciones que abarca la limpieza de datos
2. Presentar el problema de los valores faltantes
3. Conocer e implementar las técnicas del “tidy data”
4. Herramientas para la limpieza de datos: “apply()”, “value_counts()” y expresiones lambda

Limpieza de datos

La limpieza es un paso necesario en todo proyecto de datos.
Podemos resumir el proceso de limpieza de datos refiriéndonos a las siguientes seis tareas:

1. **Resolver problemas de formato**: Por ejemplo cuando al pasar de CSV a Pandas una fecha no se importa correctamente. Ej: 20090609231247 en lugar de 2009-06-09 23:12:47

2. **Corregir valores erróneos:** Por ejemplo un valor numérico o inválido para describir el género. O una edad representada por un número negativo o mucho mayor que 100.

3. Estandarizar categorías: Cuando los datos se recolectaron con un sistema que no tiene los valores tipificados, valores que representan las mismas categorías pueden estar expresados de forma distinta, por ejemplo Arg, AR y Argentina.

4. Completar datos faltantes: Los datasets del mundo real suelen venir con datos faltantes que responden a información que se perdió o nunca se recolectó. Existen varias técnicas para completar datos faltantes. Al proceso de completar datos faltantes se lo llama "imputación".

5. Asignar los tipos correctos de datos: El formato en que se encuentran los datos va a afectar nuestro análisis por varias razones. Por ejemplo, las operaciones que se pueden realizar dependen del tipo de datos. Además algunos tipos ocupan menos espacio en memoria que otros.

6. Organizar correctamente el dataset: Es importante estructurar las filas y columnas de la forma más conveniente. Para hacerlo se pueden aplicar las reglas del "tidy data".

Valores faltantes

- En general todo conjunto de datos suele tener datos faltantes (ya sea porque esos datos no fueron recolectado o nunca existieron)
    - Debemos poder detectar, rellenar y eliminar datos faltantes
    - Hay que utilizar conocimiento del dominio para definir cuáles datos faltantes se completarán y cómo.
    - Pandas ofrece varias formas de hacer esto...

Representación de datos faltantes

- Los datasets del mundo real siempre tienen **datos faltantes**
- Cada lenguaje/framework tiene su forma de lidiar con estos
- Pandas utiliza los valores **None, NaN o NaT** debido a que se basa en Numpy
    - **None:** objeto de Python que representa ausencia de dato
    - **NaN:** definición de valor faltante de floats
    - **NaT:** se utiliza para valores faltantes del tipo Timestamp

None VS np.nan

```
In [1]: import numpy as np
        import pandas as pd

In [2]: vals1 = np.array([1, None, 3, 4])
        vals1

Out[2]: array([1, None, 3, 4], dtype=object)
```

```
In [5]: vals2 = np.array([1, np.nan, 3, 4])
        vals2.dtype

Out[5]: dtype('float64')
```

Cuando tenemos un objeto None incluído en una serie, el "upcasting" de Numpy se resuelve a "object". Cuando tenemos un np.nan, conservamos una columna de tipo float y podemos seguir operando de manera eficiente.

```
In [3]: for dtype in ['object', 'int']:
            print("dtype =", dtype)
            %timeit np.arange(1E6, dtype=dtype).sum()
            print()

dtype = object
10 loops, best of 3: 78.2 ms per loop

dtype = int
100 loops, best of 3: 3.06 ms per loop
```

Procesos de generación de datos perdidos

- Missing Completely at Random (MCAR)
  - La probabilidad de que registro tenga un valor perdido en la variable Y no está relacionada ni con los valores de Y , ni con otros valores de la matriz de datos (X)
  - Los valores perdidos son una submuestra al azar de los valores totales
  - Este supuesto se viola si
    - algún grupo o subgrupo tiene mayor probabilidad de presentar datos perdidos en la variable Y y/o
    - si alguno de los valores de Y tiene mayor probabilidad de presentar datos perdidos.

Procesos de generación de datos perdidos

- Missing at Random (MAR)
    - La probabilidad de no respuesta en Y es independiente de los valores de Y , luego de condicionar sobre otras variables.
- Missing Not at Random (MNAR)
    - La probabilidad de no respuesta depende tanto de variables X externas, como de los valores de la variable con datos perdidos (Y)

Procesos de generación de datos perdidos

MCAR

X	Y
0	366,44
0	181,67
0	NR
1	682,61
1	542,28
2	NR
2	577,22
2	219,45
3	209,86
3	NR
3	372,89
3	330,52
4	NR
4	534,75
4	691,69
4	NR
4	629,45

MAR

X	Y
0	28,00
0	13,69
0	181,67
1	219,45
1	209,86
2	366,44
2	372,89
2	330,52
3	NR
3	534,75
3	387,01
3	NR
4	629,45
4	757,86
4	NR
4	691,69
4	NR

MNAR

X	Y
0	13,69
0	28,00
0	181,67
1	209,86
1	219,45
2	330,52
2	366,44
2	372,89
3	387,01
3	416,11
3	534,75
3	542,28
4	NR
4	629,45
4	NR
4	NR
4	NR

Métodos de imputación

Se puede completar los valores faltantes reemplazandolos **por la media** de la serie o **por la media condicionada** a determinada categoría. Por ejemplo, dado un valor de estatura faltante para una mujer, reemplazarlo por la media de las mujeres.

Este enfoque tiene ventajas y desventajas:

- Ventaja: Es muy probable acercarme al verdadero valor del dato faltante
- Desventajas:
  - Reduzco artificialmente la variabilidad y la aleatoriedad de los datos, lo cual me puede llevar a conclusiones equivocadas.
  - Si existía correlación entre esta variable y otras, ese valor puede verse afectado.

Para resolver estos problemas surgen los métodos de **Imputación Múltiple** que tratan de conservar las relaciones observadas entre las variables del dataset sin descuidar que existe aleatoriedad en esas relaciones.

Métodos de imputación

El método ***fillna()*** de Pandas, permite varios tipos de imputación que puede especificarse en el parámetro “method”:

- Completar los datos con un valor escalar ***(method = None)***
- Completar los datos faltantes con el valor anterior o el siguiente (ideal para series de tiempo) ***(method = bfill) o (method = ffill)***

fillna() también permite recibir un dataframe donde los índices de los datos faltantes están asociados a algún valor:

- Completar por la media, moda o la mediana. ***dff.fillna(dff.mean())***

Otra posibilidad para lidiar con los datos faltantes es eliminar los casos que contienen alguno (complete case deletion). Este método es ideal cuando los datos faltantes son pocos y faltan completamente al azar. ***df.dropna()***

Las reglas del “tidy data”

- Decimos que un dataset está ordenado cuando:
  - Cada variable es una columna
  - Cada observación es una fila
  - Cada unidad observacional es una tabla
  - Cada valor pertenece a una fila y una columna
- Algunas definiciones:
  - Variable: Es la medición de un atributo, por ejemplo, peso, altura, etc
  - Valor: Es la medida que toma una variable para una observación
  - Observación: Todas las observaciones toman el mismo tipo de valores para cada variable.

Tidy data: ejemplos

Messy data

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

Tidy data

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

- Cada unidad observacional debería ser un tratamiento sobre una persona
- En los formatos “messy data”, si quisiera agregar tratamientos que no aplican a todas las personas se llenaría el dataset de valores nulos.

Herramientas para la limpieza de datos

Expresiones lambda

- Recordemos brevemente cómo construir una función usando def:
  - Este forma de declarar funciones sirve, entre otras cosas, para reducir código duplicado y para modularizar el código.

```
>>> def square(x): return x**2
>>> square(10)
100
```

- ¿Tendría sentido definir una función de este modo si vamos a invocar dicha función una única vez?
  - Las funciones lambda pueden ser directamente definidas en el código que las va a utilizar y sin necesidad de otorgarles un nombre (son funciones anónimas).
  - Toman una única expresión y no contienen la expresión return.
  - Se definen en una única línea

```
>>> square = lambda x: x**2
>>> square(10)
100
```

La función apply() en Pandas

- El método **apply()** permite aplicar cualquier función a los elementos de un Dataframe. Se puede utilizar:
    - Por columna: **df.apply(mi_funcion)**
    - Por fila **df.apply(mi_funcion, axis = 1)**
    - Elemento por elemento **df.applymap(mi_funcion)**
- **apply()** también se puede utilizar sobre una Serie, elemento por elemento.
- Una buena propiedad de **apply()** es que permite aplicar las operaciones vectorizadas de numpy.

Práctica Guiada Parte I

Práctica Guiada Parte II

Práctica Guiada Parte III

Práctica Independiente