

**DigitalHouse** >  
Coding School

# DATA SCIENCE

## MÓDULO 2

Limpieza de datos con  
Pandas

1

**Aprender sobre las operaciones que abarca la limpieza de datos y métodos asociados**

2

**Aprender a usar herramientas para la limpieza de datos `.map()`, `.apply()`, `applymap()` y expresiones lambda**

3

**Conocer e implementar las técnicas del “tidy data”**

4

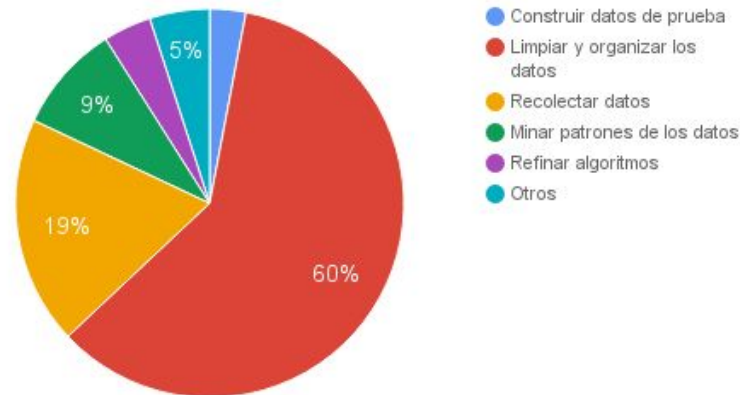
**Introducción a expresiones regulares**

# Limpieza y preparación de datos



- En 2009 Mike Driscoll (data scientist y CEO de Metamarkets) popularizó el término **“data munging”** para referirse al **arduo proceso de limpiar, preparar y validar los datos**

Como invierte su tiempo un data scientist?



Fuente:

<http://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/>

- **En 2013, Josh Wills** (ex director de Data Science de Cloudera y posterior Director of Data Engineering en Slack ) comenta: **"I'm a data janitor.** That's the sexiest job of the 21st century. It's very flattering, but it's also a little baffling."



## Traducción:

*En Data Science, se invierte un 80% del tiempo en preparar los datos y el 20% restante en quejarse de la necesidad de preparar los datos*

# Limpieza de datos con Pandas



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

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

**1. Resolver problemas de formato y asignar los tipos correctos de datos:** 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.

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.



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

**2. 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.

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

**3. 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.

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

**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".

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

**5. 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”.

- 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

```
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 incluido 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.

- 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.

## Análisis con datos completos.

Una forma de proceder consiste en eliminar los registros que presentan algún dato faltante:

- Ventaja: fácil implementación
- Desventaja: pérdida importante de información, posibles sesgos en los estimadores de los parámetros.

Este método asume que la falta de respuesta se generó de forma aleatoria, lo que por lo general no sucede.

Si los datos que faltan son pocos y al azar, entonces este método es el ideal.



## Imputación Simple:

Consiste en asignar un valor por cada dato faltante a partir de la propia variable o las demás variables para lograr una base de datos completa.

## Imputación Múltiple:

A cada valor faltante se le asigna un grupo de  $m$  valores, generando  $m$  bases de datos completas. En cada una de las  $m$  bases de datos se estiman los parámetros de interés y luego se combinan los resultados obtenidos.

## Imputación por Media

Se puede completar los valores faltantes reemplazándolos **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.

## Imputación Hot Deck

Consisten en completar con un dato existente de la muestra, siguiendo algún criterio. Pueden ser aleatorios (se elige un elemento al azar), secuencial (se completa con el valor inmediatamente anterior o posterior) o vecino más cercano.

## Imputación por Regresión

Se emplean modelos de regresión para estimar el dato que falta a partir de las demás variables del data set.

## Introducción a imputación en Pandas

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

- Completar los datos con valores definidos por fuera del data set: ***(method = None)***
- Completar los datos faltantes con el valor anterior o el siguiente (ideal para series de tiempo) ***(method = ffill)*** o ***(method = bfill)***

***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())***

- Decimos que un dataset está ordenado cuando:
  - Cada variable es una columna
  - Cada observación es una fila
  - Cada tipo de unidad observacional forma una tabla
- 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.

## 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



- 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
```



- El método **`apply()`** permite aplicar cualquier función a los elementos de un Dataframe. Se puede utilizar:
  - o Por columna: **`df.apply(mi_funcion)`** (iterando por filas)
  - o Por fila **`df.apply(mi_funcion, axis = 1)`** (iterando por cols)
  - o 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.

# Expresiones regulares



## ¿Qué son las expresiones regulares y para qué sirven?

Una expresión regular es una **secuencia de caracteres** que define un **patrón de búsqueda** de texto. Las regex constituyen lenguaje muy flexible que sirve para identificar y extraer información de un cuerpo de caracteres **no estructurado**.

Ejemplos:

- identificar direcciones de correo electrónico
- identificar DNI, CUIT, CUIL, etc.

Los **metacaracteres especiales** son aquellos que no se encuentran a sí mismos, sino que indican que se debe *matchear* algo de forma no literal, o que afectan a otras partes de la regex repitiendo caracteres o cambiando su significado.

### Algunos metacaracteres frecuentes:

**\d** Cualquier dígito del 0 al 9

**\w** Cualquier caracter alfanumérico (A-Z, a-z, 0-9 y \_)

**\s** Cualquier espacio en blanco (espacio, tabulado, nueva línea, etc.)

**.** Cualquier caracter con excepción de nueva línea (\n)

### Y sus complementos:

**\D** Todo menos cualquier dígito

**\W** Todo menos cualquier caracter alfanumérico

**\S** Todo menos cualquier espacio en blanco

## Cuantificadores:

\* Cero o más del elemento anterior

+ Uno o más del elemento anterior

**{min,max}** Definen mínimos y máximos de repetición

? Opcional, o cero o uno del elemento anterior

## Ejemplos:

**\d+** Encuentra un número o más

**.\*** Encuentra cero o más de cualquier caracter

**\w{2,6}** Encuentra un conjunto de caracteres alfanuméricos con una longitud que va de 2 a 6 caracteres

## Otros metacaracteres importantes:

**[ ]** Definen un conjunto de caracteres

**[A-Za-z0-9\_]** Definen rangos

**( )** Definen un grupo de captura

**|** Operador "o"

**\** Escapa los caracteres especiales

## Metacaracteres de posición:

**^** Comienzo de un *string* (o negación dentro de un conjunto, **[^ ]**)

**\$** Final de un *string*

**\b** Límite de palabra