## ETL: Tratamiento de Textos (I)

En esta sesión y la siguiente vamos a hacer el transformado y limpieza de los campos de tipo Texto (tal y como los hemos visto en la sesión anterior), pero también todos aquellos campos que contengan strings (es decir también los categóricos con texto). Además, empezaremos con el apasionante mundo de las expresiones regulares aunque solo de forma introductoria.

Empecemos con nuestro dataset de viajes:

[1]:

**import** pandas **as** pd

**import** random

df\_viajes **=** pd.read\_csv("./data/df\_viajes.csv")

### Introducción Recordatorio

Tal como contamos en la introducción a ETL, una vez hemos obtenido nuestros datos en "crudo", debemos limpiarnos (corregir valores, quitar valores innecesarios o inútiles, etc) y transformarlos (obtener nuevas columnas, simplificar las existentes, etc) y a partir de ahí ya seguiremos con nuestro análisis dirigido por hipótesis o por la propia exploración. En las siguientes píldoras vamos a revisar este proceso de limpieza y transformación, pero vamos a ir de lo particular tratando strings y texto, fechas y series temporales y tipos numéricos por separado para luego juntarlo todo.

### Tratamiento de Strings en pandas

En el sprint anterior tratamos los strings a partir de funciones y aplicando los métodos apply y transform (en mucha menor medida), pero era una versión más del "dar cera"/"pulir cera" porque realmente pandas tiene muchos métodos para tratar los valores strings de Series y DataFrames

En concreto hay que añadir str a nuestra Series o columna del DataFrame y a partir de ahí podemos utilizar un montón de métodos propios de los strings pero valor a valor de la serie o columna.

Veámoslo con un ejemplo que te sonará. Vamos a limpiar los campos "Destino" y "consumo\_kg" de nuestro "df\_viajes". Recordando:

df\_viajes

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

df\_viajes["Destino"].value\_counts()

Tabla

Descripción generada automáticamente

df\_viajes.info()

Texto

Descripción generada automáticamente

df\_viajes["consumo\_kg"].value\_counts()

Tabla

Descripción generada automáticamente

En nuestra práctica utilizamos una función que hacía llamadas a los métodos string, pero con pandas es posible hacerlo de una vez:

[7]:

df\_viajes["Destino\_corregido"]**=**df\_viajes["Destino"].str.lower().str.capitalize()

df\_viajes["Destino\_corregido"].value\_counts()

[7]:

Tabla

Descripción generada automáticamente

Y lo mismo para "consumo\_kg", con un truquito:

[10]:

df\_viajes["consumo\_kg\_corregido"]**=**df\_viajes["consumo\_kg"].str.replace(",",".").astype("float64")

df\_viajes["consumo\_kg\_corregido"]

[10]:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

df\_viajes.info()

Tabla

Descripción generada automáticamente

Para que no te pase que tengas que reescribir lo que ya está escrito, te dejo aquí algunos de los métodos que se pueden acceder con str en pandas (y ten en cuenta que el hace ya la comprobación de que sea string el valor):

Tabla

Descripción generada automáticamente

### Expresiones regulares

El mundo de las expresiones regulares nos daría para un bootcamp entero y aún así... Por eso las vamos a ir viendo poco a poco a lo largo de lo que resta de curso, pero hoy las vamos a introducir con ejemplos, tanto fuera como dentro de pandas.

Y ¿qué son las expresiones regulares? En palabras de ChatGPT: Las expresiones regulares son patrones utilizados para encontrar coincidencias en cadenas de texto.

Un ejemplo, revisa rápidamente este texto:

texto **=** "En 2023, la nave espacial CRX-2049 partió de España. En 2049, encontramos vida en Marte. ¡Increíble! Costó $204M, el evento más importante desde el 1969."

¿Cómo harías para encontrar con poco código todas las fechas que muestra, el dinero y el nombre de la nave?

Podrías crear un programa o función que recorriese el texto, fuera probando cada palabra si es un número, si tiene cuatro cifras... etc, etc

Las expresiones regulares vienen a ayudarte en estas tareas:

**import** re *# para necesidades más complejas regex, pero a nosotros por ahora nos llega con re*

print(re.findall(r"[0-9]{4}",texto))

print(re.match(r".\*?([A-Z]+?-[0-9]{4})", texto).group(1))

['2023', '2049', '2049', '1969']

CRX-2049

Casi, ahora lo corregiremos, lo importante:

* Existen varias funciones dentro del paquete re que nos permiten aplicar busqueda por patrones (eso que aparece como primer argumento y que es como si el gato se hubiera puesto a pisar el teclado)
* La sintaxis es la misma en general una funcion que se le pasa un patrón y luego una variable de tipo string en la que aplicar el patrón.

#### Funciones básicas

Hay muchas pero te interesan:

* findall(patron, variable), devuelve una lista con todas las partes de la variable que cumplen el patrón
* match(patron, variable), devuelve un objeto con grupos (ahora veremos) con las ocurrencias que cumplen el patrón buscando desde el principio de la variable
* sub(patron1,patron2, variable), devuelve un string con todas las ocurrencias de patron1, que encuentre en variable, sustituidas por patron2

#### Patrones

Esta es la madre del cordero y aquí es donde entramos no sólo en Matrix sino en MatrixFriki++, te dejo [aquí](https://docs.python.org/3/library/re.html) un enlace a todos los patrones sencillos (que se pueden combinar entre sí) y ahora veamos los principales para empezar:

##### 1. Sencillos

".": Cualquier caracter (si quiero buscar el carácter "." tenemos que poner ".")  
"\d": Un dígito cualquiera (o sea de 0 a 9)  
"\D": Un carácter que no es un número  
"\w": Una letra (no incluye signos de puntuación)  
"\W": Un carácter que no es una letra  
"\s": Espacios,tabuladores ("\t"), retornos de carro/linea\_nueva ("\n")  
"\S": Un carácter que no sea el anterior.

Ejemplos:

*# Todas las palabras de 4 caracteres rodeadas por espacios:*

re.findall(" .... ",texto)

[' nave ', ' vida ']

*# Todos los números de 4 cifras a los que no les siga una letra:*

re.findall("\d\d\d\d\W",texto)

['2023,', '2049 ', '2049,', '1969.']

*# Eliminar todos los espacios y tabuladores:*

re.sub("\s","",texto)

'En2023,lanaveespacialCRX-2049partiódeEspaña.En2049,encontramosvidaenMarte.¡Increíble!Costó$204M,eleventomásimportantedesdeel1969.'

##### Anclas

Con lo anterior ya podemos hacer muchas cosas, pero existen unas cuantas "anclas" o modificadores que le dicen a nuestras funciones de expresiones regulares que se comporten de una manera bastante útil:

* "^", Busca al principio de la variable o cadena.
* "$", Busca al final de la variable o cadena.
* "[]", Es válido para cualquiera de los caracteres o patrones dentro de los corchetes (ojo, si quiero buscar el carácter corchete necesito "[" o "]")
* "[^ ]", Es válido si no aparece ninguno de los caracteres o patrones dentro de los corchetes
* "|", Es válido para lo que haya a la izquierda o derecha del |
* "()", Lo que encuentres que coincida con el patrón de dentro de los paréntesis forma un grupo

*# Encuentra cualquier número de 3 o 4 cifras*

re.findall("\d\d\d\d|\d\d\d",texto)

['2023', '2049', '2049', '204', '1969']

*# Encuentra cualquier carácter que no sea a, b, ni un número, ni un espacio:*

re.findall("[^ab\d ]",texto)

['E',

# Sustituye, las letras e y a por E:

re.sub("[ea]","E",texto)

'En 2023, lE nEvE EspEciEl CRX-2049 pErtió dE EspEñE. En 2049, EncontrEmos vidE En MErtE. ¡IncrEíblE! Costó $204M, El EvEnto más importEntE dEsdE El 1969.'

##### Cuantificadores

Y aquí de verdad llega la potencia (y también la complejidad mayor):

* "<patrón>\*" encuentra 0 o más ocurrencias del patrón <patrón>
* "<patrón>+" encuentra 1 o más ocurrencias del patrón <patrón>
* "<patrón>?" encuentra 0 o 1 ocurrencia del patrón <patrón>
* "<patrón>{(num)}" encuentra exactamente (num) ocurrencias seguidas del patrón <patrón>
* "<patrón>{(numero),(numero2)}" encuentra ocurrencias que se repitan seguidas entre (numero) y (numero2) del <patrón> (si no pones (numero2), es mínimo (numero))
* "[a-z]" encuentra cualquier letra de la "a" a la "z", sólo minúsculas (no acentos, no ñ)
* "[a-zA-Z]" encuentra cualquier letra de la "a" a la "z" o de la "A" a la "Z"
* "[0-9]" encuentra cualquier número del 0 al 9 (equivale a \d)
* "[2-6]" encuentra cualquier número del 2 al 6 (y puedes cambiar el intervalo)

*# Ahora sí, encuentra los números de 4 cifras que estén rodeados por espacios, comas o puntos:*

re.findall(r"[ ,.]\d{4}[ ,.]",texto)

[' 2023,', ' 2049,', ' 1969.']

re.findall(r"[ ,.](\d{4})[ ,.]",texto)

['2023', '2049', '1969']

*# Encuentra las fechas:*

print(re.findall(r"[^-][0-9]{4}[^A-Z]",texto))

[' 2023,', ' 2049,', ' 1969.']

*# Encuentra el nombre de las naves sabiendo que son tres letras mayúsculas seguidas de un guión y 4 números:*

nave**=**re.match(r".\*([A-Z]{3}-\d{4})",texto).group(1)

print(nave)

CRX-2049

*# Encuentra todas las palabras (sin números) de 4 o más letras que empiecen por e, rodeadas por espacios, puntos o comas:*

re.findall(r"[eE][a-zA-Z]{3,}",texto)

['espacial', 'Espa', 'encontramos', 'evento', 'esde']

[ ]:

*# Y sólo la palabra*

[Aqui](https://regex101.com/) tienes un enlace para que experimentes con tus expresiones regulares.