# Cómo trabajar con variables tipo texto en Pandas

6 de Mayo de 2020

Pandas cuenta con una gran funcionalidad a la hora de interactuar con texto, es super versatil si estas interesado en crear modelos de análisis de lenguaje natural.

Comencemos cargando nuestra librería y creando un diccionario con nombres de personas.

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

data = {'names':['Sara Moreno 34',

'jUAn GOMez 23',

'CArlos mArtinez 89',

'Alfredo VelaZques 3',

'luis Mora 56',

'@freddier #platzi 10',pd.NA]}

Usemos los datos del diccionario para crear nuestro DataFrame. Nuestro DataFrame contiene una columna tipo texto, con variedades de caracteres especiales, números, mayúsculas e inclusive variables nulas.

df = pd.DataFrame(data)

df

|  |  |
| --- | --- |
|  | names |
| 0 | Sara Moreno 34 |
| 1 | jUAn GOMez 23 |
| 2 | CArlos mArtinez 89 |
| 3 | Alfredo VelaZques 3 |
| 4 | luis Mora 56 |
| 5 | @freddier #platzi 10 |
| 6 |  |

Para usar las funciones asociadas a texto usamos str en nuestro DataFrame, por ejemplo, si se quiere colocar el texto en minúscula, basta con escribir:

df['names'].str.lower()

|  |  |
| --- | --- |
|  | names |
| 0 | sara moreno 34 |
| 1 | juan gomez 23 |
| 2 | carlos martinez 89 |
| 3 | alfredo velazques 3 |
| 4 | luis mora 56 |
| 5 | @freddier #platzi 10 |
| 6 |  |

Para mayúsculas igualmente:

df['names'].str.upper()

|  |  |
| --- | --- |
|  | names |
| 0 | SARA MORENO 34 |
| 1 | JUAN GOMEZ 23 |
| 2 | CARLOS MARTINEZ 89 |
| 3 | ALFREDO VELAZQUES 3 |
| 4 | LUIS MORA 56 |
| 5 | @FREDDIER #PLATZI 10 |
| 6 |  |

O si queremos solo la primera letra en mayúscula:

df['names'].str.capitalize()

|  |  |
| --- | --- |
|  | names |
| 0 | Sara moreno 34 |
| 1 | Juan gomez 23 |
| 2 | Carlos martinez 89 |
| 3 | Alfredo velazques 3 |
| 4 | Luis mora 56 |
| 5 | @freddier #platzi 10 |
| 6 |  |

Para contar la longitud de nuestro texto usamos:

df['names'].str.len()

|  |  |
| --- | --- |
|  |  |
| 0 | 14 |
| 1 | 13 |
| 2 | 18 |
| 3 | 19 |
| 4 | 12 |
| 5 | 20 |
| 6 |  |

Para dividir el texto por espacios usamos split y definimos el carácter por  
el que queremos dividir, en este caso, un espacio vacío ' ' o '#':

df['names'].str.split(' ')

|  |  |
| --- | --- |
|  | names |
| 0 | [‘Sara’, ‘Moreno’, ‘34’] |
| 1 | [‘jUAn’, ‘GOMez’, ‘23’] |
| 2 | [‘CArlos’, ‘mArtinez’, ‘89’] |
| 3 | [‘Alfredo’, ‘VelaZques’, ‘3’] |
| 4 | [‘luis’, ‘Mora’, ‘56’] |
| 5 | [’@freddier’, ‘#platzi’, ‘10’] |
| 6 |  |

df['names'].str.split('#')

|  |  |
| --- | --- |
|  | names |
| 0 | [‘Sara Moreno 34’] |
| 1 | [‘jUAn GOMez 23’] |
| 2 | [‘CArlos mArtinez 89’] |
| 3 | [‘Alfredo VelaZques 3’] |
| 4 | [‘luis Mora 56’] |
| 5 | [’@freddier ', ‘platzi 10’] |
| 6 |  |

Para seleccionar los primeros o últimos 5 caracteres usamos:

df['names'].str[:5]

|  |  |
| --- | --- |
|  | names |
| 0 | Sara |
| 1 | jUAn |
| 2 | CArlo |
| 3 | Alfre |
| 4 | luis |
| 5 | @fred |
| 6 |  |

df['names'].str[-5:]

|  |  |
| --- | --- |
|  | names |
| 0 | no 34 |
| 1 | ez 23 |
| 2 | ez 89 |
| 3 | ues 3 |
| 4 | ra 56 |
| 5 | zi 10 |
| 6 |  |

Podemos reemplazar una secuencia de caracteres por otra mediante:

df['names'].str.replace('Alfredo','Antonio')

|  |  |
| --- | --- |
|  | names |
| 0 | Sara Moreno 34 |
| 1 | jUAn GOMez 23 |
| 2 | CArlos mArtinez 89 |
| 3 | Antonio VelaZques 3 |
| 4 | luis Mora 56 |
| 5 | @freddier #platzi 10 |
| 6 |  |

También podemos buscar una secuencia de texto en específico, en este caso,  
'ara':

df['names'].str.findall('ara')

|  |  |
| --- | --- |
|  | names |
| 0 | [‘ara’] |
| 1 | [] |
| 2 | [] |
| 3 | [] |
| 4 | [] |
| 5 | [] |
| 6 |  |

También podemos crear un filtro basándonos en una secuencia de texto en  
específico, en este caso, las filas que tengan 'or':

df['names'].str.contains('or')

|  |  |
| --- | --- |
|  | names |
| 0 | True |
| 1 | False |
| 2 | False |
| 3 | False |
| 4 | True |
| 5 | False |
| 6 |  |

Así mismo, podemos contar el número de ocurrencias de un caracter en específico,  
por ejemplo, cuántas veces aparece la letra 'a':

df['names'].str.lower().str.count('a')

|  |  |
| --- | --- |
|  | names |
| 0 | 2 |
| 1 | 0 |
| 2 | 0 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 |  |

Existen comandos más avanzados usando Regex, por ejemplo, si quiero extraer los caracteres numéricos:

df['names'].str.extract('([0-9]+)', expand=**False**)

|  |  |
| --- | --- |
|  | names |
| 0 | 34 |
| 1 | 23 |
| 2 | 89 |
| 3 | 3 |
| 4 | 56 |
| 5 | 10 |
| 6 | nan |

O, por ejemplo, si quiero extraer las menciones '@xxxx' del texto:

df['names'].str.replace('@[^\s]+','')

|  |  |
| --- | --- |
|  | names |
| 0 | Sara Moreno 34 |
| 1 | jUAn GOMez 23 |
| 2 | CArlos mArtinez 89 |
| 3 | Alfredo VelaZques 3 |
| 4 | luis Mora 56 |
| 5 | #platzi 10 |
| 6 |  |

Si te gustó todo lo que puedes hacer con texto, te invito a que busques nuestro  
[Curso de Expresiones Regulares](https://platzi.com/clases/1301-expresiones-regulares/11873-uso-de-regex-para-descomponer-querys-get/) y así mismo nuestro [Curso de Fundamentos de Procesamiento de Lenguaje Natural con Python y NLTK](https://platzi.com/clases/python-lenguaje-natural/) para mejorar tus habilidades de análisis.