Clase 030 — Pandas: operaciones vectorizadas sobre strings

Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.11 Vectorized String Operations. · Duración estimada: 60 min.

Clase 030 — Pandas: operaciones vectorizadas sobre strings

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.11 Vectorized String Operations. · Duración estimada: 60 min.*

Objetivo

Que el alumno limpie y transforme columnas de texto sin caer en apply(lambda x: ...), usando el accessor .str de pandas — vectorizado, NaN-aware, con métodos análogos a los de Python (lower, strip, replace, split, contains, regex).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Usar .str para aplicar operaciones de string vectorizadamente a una Series.
2. Manejar NaN automáticamente (los métodos .str propagan NaN sin error).
3. Aplicar regex con .str.contains(patron), .str.extract(...), .str.replace(...).
4. Dividir y unir con .str.split(sep, expand=True) que produce un DataFrame.
5. Trabajar con categorical cuando el cardinalidad es baja (memoria y speedup).

Temas

#	Tema	Por qué importa
1	Accessor .str	Métodos vectorizados que respetan NaN.
2	Casos típicos: lower, strip, replace, cont	El 80% del trabajo.
3	Regex con .str.extract y grupos nombrados	Extracción estructurada.
4	.str.split(expand=True) → DataFrame	Desnormalizar columnas combinadas.
5	dtype='string' (nullable) vs object	El moderno y NA-aware.
6	Categorical para baja cardinalidad	Menos memoria, groupby más rápido.

Complemento previo: Regex con el módulo re

Antes de meternos con .str.extract / .str.contains, conviene tener una intro mínima a expresiones regulares. Pandas usa regex por debajo en casi todos los métodos de texto, y en scraping (BeautifulSoup, parsing de HTML/logs) son prerrequisito invisible. Sin entender regex, los patrones se vuelven magia negra que se copypastea de Stack Overflow.

Metacaracteres esenciales

Símbolo	Significado	Ejemplo		
.	Cualquier carácter (except	a.c matchea abc, a c, a3c		
*	0 o más repeticiones del ar	ab* matchea a, ab, abbb		
+	1 o más repeticiones	ab+ matchea ab, abbb (no		
?	0 o 1 (opcional)	colou?r matchea color y co		

\d	Dígito [0-9]	\d\d\d matchea 123		
\w	Word char [a-zA-Z0-9_]	\w+ matchea hola_123		
\s	Whitespace (espacio, tab,	\s+ matchea uno o más es		
[]	Set de caracteres	[aeiou] matchea una vocal		
()	Grupo de captura	(\d+)-(\d+) captura ambos		
^	Inicio de string	^Hola matchea solo si arra		
$	Fin de string	\.com$ matchea solo si ter		
`\	`	OR	`gato\	perro` matchea cualquiera
{n,m}	Entre n y m repeticiones	\d{2,4} matchea 2 a 4 dígit		

Funciones clave de re

Función	Para qué sirve
re.search(pat, s)	Busca el primer match en cualquier parte d
re.match(pat, s)	Igual que search pero solo desde el inicio
re.findall(pat, s)	Devuelve lista con todos los matches.
re.sub(pat, repl, s)	Reemplaza todos los matches por repl. Equi
re.compile(pat)	Pre-compila el patrón. Útil si lo vas a us
re.IGNORECASE (flag)	Match case-insensitive. Se pasa como flags

Mini-ejemplo — extraer dominio de email con grupos nombrados:

```
import re

email = "vladimir.acuna@gmail.com"
pat = re.compile(r"(?P<usuario>[\w.]+)@(?P<dominio>[\w.]+)")
m = pat.search(email)
print(m.group("usuario"))  # vladimir.acuna
print(m.group("dominio"))  # gmail.com
```

Raw strings (r"...")

Siempre se usan raw strings con regex. Sin la r, Python interpreta \d, \w, \s como secuencias de escape y muchas veces te las come o te tira DeprecationWarning. Con r"\d+" le decís a Python "esto es literal, no lo toques, pasáselo crudo a re". Regla: toda regex va en raw string, sin excepciones.

> *Herramienta recomendada para iterar patrones: regex101.com — testa en vivo, explica cada token y soporta flavor Python.*

Definiciones y características

Accessor .str

: Espacio de nombres en Series con métodos string vectorizados. Análogos a los de Python (.lower(), .split(), .replace()) pero aplicados elementwise y NaN-aware (propagan NaN sin error).

.str.extract(pattern)

: Aplica regex con grupos () y devuelve DataFrame con una columna por grupo. Soporta grupos nombrados ((?P<dominio>...)).

.str.split(sep, expand=True)

: Divide cada string y opcionalmente expande a DataFrame de columnas. Útil para denormalizar 'Apellido, Nombre' → 2 cols.

dtype 'string' (nullable)

: Versión moderna del dtype para texto. Diferencias con object: NA-aware (usa pd.NA), futuras optimizaciones. Recomendado en pandas 2+.

Categorical

: Dtype para columnas con cardinalidad baja (pocos valores únicos). Almacena cada valor como entero + diccionario. Ahorra ~10× memoria y acelera groupby/sort.

Dataset / recursos

Sintético: emails, nombres con espacios, fechas como string.

Ejercicios

1. Lower + strip. Lista de emails con mayúsculas y espacios. Normaliza con .str.lower().str.strip().

2. Extract dominio. De una columna de emails, extrae el dominio con regex (@(.+)$).

3. Split nombre completo. Columna 'Ana García' → nombre, apellido en columnas separadas.

4. Filtro por contains. Filas donde la columna descripcion contiene la palabra (case-insensitive) 'urgente'.

5. Categorical. Convierte una columna con 5 valores únicos en 100k filas a Categorical. Compara memoria.

Homework verificable

Notebook con CSV sintético de contactos (nombre, email, teléfono): (a) normalizar email (lower+strip); (b) extraer dominio; (c) separar nombre/apellido; (d) flag de email corporativo (no gmail/yahoo/hotmail); (e) convertir país a Categorical y reportar memoria.

Criterio de aceptación: Operaciones manejan NaN sin error. Categorical reduce memoria al menos 5×.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
'NoneType' has no attribute 'lower' al hac	Hay NaN/None en la Series. Fix: usa .str.l
Regex no captura nada con .str.extract	Falta () para definir grupo, o el pattern
.str.contains('foo') lanza error con NaN	Por default, na=NaN propaga. Fix: s.str.co
Convertí a Categorical y el sort sale alfa	Categorical por default es no-ordenado. Fi
.str.split(',') da listas, no columnas	Sin expand=True. Fix: s.str.split(',', exp
Regex con "\d+" no matchea o tira Deprecat	Olvidaste la r del raw string — Python int

Preguntas frecuentes

¿.str.lower() o .apply(str.lower)?

.str.lower() siempre — vectorizado, maneja NaN, mucho más rápido en N grande. apply es loop Python disfrazado.

¿Cuándo convertir a Categorical?

Cuando la cardinalidad es baja (~<5% de N filas) y vas a hacer groupby/sort. Para 100k filas de 5 países: enorme ganancia. Para 100k filas de 80k strings únicos: no ayuda.

¿'string' o object dtype?

'string' para código nuevo (NA-aware). object sigue siendo default por compat. Conviértelo explícito: df['col'] = df['col'].astype('string').

¿Regex case-insensitive?

s.str.contains('foo', case=False) o flags=re.IGNORECASE. También s.str.lower().str.contains('foo') (más explícito).

¿Cómo elimino acentos?

Pandas no trae nativo. Usa unidecode o s.str.normalize('NFKD').str.encode('ascii', 'ignore').str.decode('ascii').

¿Por qué \d a veces falla?

Porque te olvidaste de la r del raw string. "\d" en Python plano no es un escape válido y según la versión te lo come o te tira DeprecationWarning (y en Python 3.12+ va directo a SyntaxWarning). Con r"\d" el backslash se pasa literal a re, que es quien lo interpreta como "dígito". Regla mecánica: toda regex en raw string, sin pensarlo.

Referencias

- VanderPlas, cap. 3 § 3.11.
- pandas Text data user guide
- pandas Categorical
- Python re HOWTO

Siguiente clase

Clase 031 — Pandas: series de tiempo, resampling, rolling

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import pandas as pd
import numpy as np
```

Archivos complementarios

- notebook.ipynb