Unidad: Scraping

Web Scraping: Fundamentos y Aplicaciones

Nicolás Sidicaro
Abril 2025

¿Qué es Web Scraping?

- **Web Scraping** es una técnica que permite **extraer información de sitios web de manera automatizada**
- Funciona como un asistente virtual que puede:
  - Visitar miles de páginas web
  - Extraer precisamente la información requerida
  - Organizar los datos en un formato estructurado
  - Realizar todo a una velocidad imposible de lograr manualmente

¿Por qué nos importa?

Hoy los datos son el PODER

Importancia del Web Scraping

El Web Scraping es **fundamental** porque:

- Permite **obtener datos** que de otra forma serían inaccesibles
- Automatiza tareas **repetitivas y tediosas**
- Facilita el **monitoreo continuo** de información cambiante

Recuperar lo nuestro (?)

Web Scraping en Ciencia de Datos

El Web Scraping es una fase crucial que:

1. Permite la **obtención de datos** no disponibles en APIs o bases estructuradas
2. Requiere **limpieza y transformación** específicas
3. **Alimenta los análisis** posteriores (EDA, modelado)
4. Puede ser parte de un **proceso continuo** de actualización de datos
5. **Complementa** otras fuentes de información

Aplicaciones del Web Scraping

Usos comunes del web scraping:

- **Monitoreo de precios**: Seguimiento de productos en diferentes tiendas
- **Investigación de mercado**: Análisis de productos, opiniones y tendencias
- **Generación de leads**: Extracción de información de contacto
- **Análisis de contenido**: Recopilación de noticias, artículos o publicaciones
- **Seguimiento de redes sociales**: Monitoreo de menciones y comentarios
- **Investigación académica**: Recopilación de datos para estudios

Ejemplos de aplicación en Argentina

El web scraping tiene múltiples aplicaciones en el contexto rioplatense:

- Mercado inmobiliario:
  - Recopilación de precios de propiedades en ZonaProp, Argenprop, Mercado Libre
  - Análisis de tendencias por barrio o zona
  - Detección de oportunidades de inversión
- Monitoreo de precios e inflación:
  - Seguimiento de precios en supermercados online
  - Comparación de precios entre distintos retailers
  - Creación de índices de precios alternativos

Ejemplos de aplicación en Argentina

- **Análisis de opinión pública**:
  - Extracción de comentarios de noticias de medios importantes
  - Análisis de tendencias en redes sociales sobre temas de actualidad
  - Monitoreo de menciones de figuras políticas
- **Seguimiento de ofertas y descuentos**:
  - Monitoreo de promociones bancarias
  - Seguimiento de precios en Hot Sale, Black Friday
  - Comparación de promociones entre diferentes cadenas

Tipos de Web Scraping

Según el nivel de automatización:

- **Scraping manual asistido**: Uso de extensiones de navegador
- **Scraping semi-automatizado**: Scripts con supervisión ocasional
- **Scraping totalmente automatizado**: Sistemas autónomos

Según la complejidad técnica:

- **Scraping estático**: Para sitios con toda la información en el HTML inicial
- **Scraping dinámico**: Para sitios que cargan contenido mediante JavaScript
- **Scraping de APIs**: Uso de interfaces programáticas disponibles

El proceso de Web Scraping

Un proyecto típico de web scraping sigue estos pasos:

1. **Planificación y reconocimiento**
    - Análisis de la estructura del sitio
    - Identificación de datos objetivo
    - Evaluación de dificultad y aspectos legales
2. **Selección de herramientas**
    - Extensiones de navegador
    - Bibliotecas y frameworks de programación (rvest, RSelenium)
    - Servicios en la nube

El proceso de Web Scraping

1. **Desarrollo del scraper**
    - Identificación de selectores (CSS, XPath)
    - Programación de la lógica de extracción
    - Manejo de paginación y errores
2. **Ejecución y almacenamiento**
    - Puesta en marcha con monitoreo
    - Limpieza y procesamiento de datos crudos
    - Almacenamiento en archivos CSV, bases de datos, etc.
3. **Mantenimiento**
    - Monitoreo regular
    - Adaptación a cambios en la estructura del sitio
    - Optimización de rendimiento

Alerta!

Un gran poder conlleva una gran responsabilidad

- **Términos de servicio**: Muchos sitios prohíben explícitamente el scraping
- **Robots.txt**: Indica qué partes de un sitio pueden ser accedidas por bots
- **Propiedad intelectual**: Los datos pueden estar protegidos por derechos de autor
- **Leyes de privacidad**: GDPR y otras regulaciones limitan la recopilación de datos personales

Alerta!

Buenas prácticas éticas

Para usar el web scraping de manera responsable:

1. **Identificación adecuada**: Usar un User-Agent que identifique el bot
2. **Respeto de límites**: Evitar bombardear el servidor; implementar retrasos
3. **Minimización del impacto**: Extraer solo lo necesario en momentos adecuados
4. **Consideración de usuarios**: No afectar la experiencia de otros visitantes
5. **Evaluación de alternativas**: Si existe una API oficial, utilizarla primero

¿Qué necesitamos saber?

Fundamentos web:

- HTML: Estructura y contenido
  - Etiquetas, atributos, jerarquía de elementos
  - Componentes como tablas, listas, formularios
- CSS: Estilos y selectores
  - Selectores por tipo, clase, ID, atributo
  - Combinadores y pseudo-clases
- JavaScript (básico):
  - Comprensión de cómo afecta al contenido dinámico
  - Cómo identificar sitios que dependen de JS

Herramientas para Web Scraping

- **Extensiones de navegador**:
  - Web Scraper, Data Miner
  - Ideal para tareas simples y aprendizaje
- **Bibliotecas en R**:
  - `rvest`: Para sitios estáticos
  - `RSelenium`: Para sitios dinámicos con JavaScript
  - `httr`: Para solicitudes HTTP y APIs
- **DevTools del navegador**:
  - Inspector de elementos
  - Pestaña Network
  - Consola para pruebas

Sitios estáticos vs. dinámicos

Sitios estáticos:

- Contenido completamente renderizado desde el servidor
- Todo el contenido está en el HTML inicial
- El código fuente contiene todos los datos visibles
- Se pueden scrapear con herramientas simples como `rvest`

Sitios dinámicos:

- El HTML inicial es solo un "esqueleto"
- Datos cargados posteriormente mediante JavaScript/AJAX
- El código fuente no contiene todos los datos visibles
- Requieren herramientas como `RSelenium`

Ejemplos en R: Sitios estáticos

```r
# Cargar bibliotecas
library(rvest)
library(dplyr)

# Obtener la página
url ← "https://ejemplo.com/productos"
pagina ← read_html(url)

# Extraer datos usando selectores CSS
titulos ← pagina %>%
  html_nodes(".producto h2") %>%
  html_text()

precios ← pagina %>%
  html_nodes(".producto .precio") %>%
  html_text()

# Crear dataframe
productos ← data.frame(
  titulo = titulos,
  precio = precios
)
```

Ejemplos en R: Sitios dinámicos

```r
# Cargar bibliotecas
library(RSelenium)

# Iniciar navegador controlado
driver ← rsDriver(browser = "firefox", port = 4444L)
remote_driver ← driver$client

# Navegar a la página
remote_driver$navigate("https://ejemplo.com/productos")

# Esperar a que cargue el contenido dinámico
Sys.sleep(3)

# Extraer elementos
elementos_producto ← remote_driver$findElements(
  using = "css selector", ".producto")

# Extraer datos de cada elemento
productos ← lapply(elementos_producto, function(elem) {
  titulo ← elem$findChildElement(
    using = "css selector", "h2")$getElementText()[[1]]
  precio ← elem$findChildElement(
    using = "css selector", ".precio")$getElementText()[[1]]

  c(titulo = titulo, precio = precio)
})
```

XPath: Una herramienta poderosa

XPath es un lenguaje para seleccionar nodos en documentos XML/HTML que ofrece:

- **Mayor expresividad** que los selectores CSS
- **Navegación bidireccional** en el árbol DOM
- **Selección por contenido textual**

Ejemplos:

```
# Seleccionar por texto
nodos ← html_nodes(pagina,
                   xpath = "//p[contains(text(), 'oferta')]")

# Seleccionar elementos con múltiples condiciones
nodos ← html_nodes(pagina,
                   xpath = "//div[@class='producto' and .//span[text()='En stock']]")

# Navegar hacia arriba (imposible con CSS)
nodos ← html_nodes(pagina,
                   xpath = "//span[text()='Agotado']/ancestor::div[@class='producto']'
```

Desafíos comunes en Web Scraping

Bloqueos y detección de bots:

- CAPTCHAs y verificaciones
- Limitación de frecuencia (rate limiting)
- Bloqueo de IP
- Fingerprinting del navegador

Sitios dinámicos:

- JavaScript y AJAX
- Scroll infinito
- Single-Page Applications (SPAs)

Desafíos comunes en Web Scraping

Estructuras inconsistentes:

- Diferentes diseños para el mismo tipo de contenido
- Contenido faltante
- Cambios de formato

Mantenimiento:

- Cambios frecuentes en el diseño de los sitios
- Pequeñas modificaciones en clases o estructura
- Nuevas medidas anti-scraping

Análisis preliminar de sitios web

Un enfoque paso a paso para analizar un sitio antes de comenzar a programar:

1. **Exploración inicial**
   - Navegar manualmente como usuario normal
   - Identificar secciones principales
2. **Análisis técnico preliminar**
   - Verificar robots.txt y términos de servicio
   - Determinar si el sitio es estático o dinámico
3. **Análisis estructural**
   - Examinar patrones en URLs
   - Analizar estructura HTML para cada tipo de página

Automatización vs. Exploración Manual

Ventajas y limitaciones de cada enfoque:

- **Automatización**:
  - ✓ Eficiente para grandes volúmenes de datos
  - ✓ Consistente y reproducible
  - ✓ Ideal para recopilación periódica
  - ✗ Más susceptible a bloqueos
  - ✗ Requiere mantenimiento ante cambios del sitio
- **Exploración manual**:
  - ✓ Más flexible para sitios complejos
  - ✓ Menos susceptible a detección
  - ✓ Mejor para análisis profundo de pocos elementos
  - ✗ No escalable

Documentación del Scraping

La documentación es crucial:

- ¿Por qué documentar?
  - Facilita el mantenimiento futuro
  - Permite reproducir y mejorar el proceso
  - Ayuda a entender la estructura de los datos
- ¿Qué documentar?
  - Estructura del sitio y selectores
  - Decisiones de diseño
  - Tratamiento de casos especiales
  - Proceso de limpieza post-scraping
  - Limitaciones conocidas

Errores Comunes en Web Scraping

Errores a evitar:

- **No respetar los términos de servicio** o robots.txt
- **Bombardear el servidor** con demasiadas solicitudes
- **Confiar en selectores frágiles** que cambian frecuentemente
- **No manejar errores** y casos excepcionales
- **No considerar el mantenimiento** a largo plazo
- **Omitir la documentación** del proceso y decisiones
- **Ignorar la calidad de los datos** extraídos

Pasos Clave para un Scraping Efectivo

1. **Definir claramente tus objetivos**
    - ¿Qué datos específicos necesitas?
    - ¿Con qué frecuencia necesitas actualizarlos?
2. **Analizar el sitio objetivo**
    - Estructura, tecnologías, protecciones
    - Aspectos legales y éticos
3. **Elegir las herramientas adecuadas**
    - Según la complejidad del sitio
    - Según tus conocimientos técnicos
4. **Desarrollar de manera incremental**
    - Comienzar con un subconjunto de datos
    - Probar y refinar antes de escalar

Pasos Clave para un Scraping Efectivo

1. **Implementar buenas prácticas**
    - Retrasos entre solicitudes
    - Identificación apropiada
    - Respeta límites de uso
2. **Limpiar y validar los datos**
    - Verifica consistencia y completitud
    - Implementa transformaciones necesarias
3. **Documentar y mantener**
    - Registra decisiones y estructura
    - Monitorea y adapta a cambios

Web Scraping y Análisis de Datos

El web scraping es el primer paso de un proceso más amplio:

- **Obtención de datos → Limpieza → Exploración → Análisis → Comunicación**

Para un análisis efectivo:

- Estructura los datos extraídos en formatos adecuados (CSV, JSON, bases de datos)
- Documenta metadatos importantes (fecha de extracción, URL fuente)
- Implementa procesos de validación para detectar anomalías
- Considera la automatización de la extracción periódica
- Diseña pipelines que conecten la extracción con el análisis

Recursos adicionales

- **Documentación**:
  - W3Schools: Para aprender HTML, CSS
  - Documentación de rvest
  - Documentación de RSelenium
- **Bibliotecas en R**:
  - `rvest`: Para sitios estáticos
  - `RSelenium`: Para sitios con JavaScript
  - `httr`: Para APIs y solicitudes HTTP
  - `xml2`: Para procesamiento de HTML/XML
- **Herramientas**:
  - DevTools de Chrome/Firefox
  - SelectorGadget para identificar selectores CSS
  - XPath Helper para probar expresiones XPath