# ESTRUCTURA DE LA PRESENTACIÓN WEB SCRAPING

1. Web scraping
   1. ¿Qué es web scraping?
   2. Uso e historia de Web Scraping
      1. ¿Dónde y cómo se utiliza?
      2. ¿Cómo se llegó al Web Scraping?
   3. ¿Qué herramientas se emplean?
      1. Python (Beautiful Soup y Scrapy)
      2. Selenium
      3. ParseHub
      4. Octoparse
   4. Ejemplos prácticos de Web Scraping
2. Datasets
   1. ¿Qué es dataset?
   2. Uso e historia de Dataset
      1. ¿Dónde y cómo se utiliza?
      2. ¿Cómo se llegó al Dataset?
   3. ¿Qué herramientas se emplean?
      1. Pandas
      2. SKLearn
      3. Numpy
   4. Ejemplos prácticos de elaboración de Datasets

Web scraping

Web scraping es una técnica de extracción de datos de páginas web de forma automática. Esta técnica emplea programas o scripts que navegan por páginas web, acceden a su contenido y recogen información que luego se almacena en bases de datos o datasets para ser analizados posteriormente.

El web scraping ha evolucionado conjuntamente con la tecnología debido a la necesidad de recopilar la mayor cantidad de información posible. La evolución del web scraping puede dividirse en varias fases:

Origen: Antes, todos los datos se recopilaban de forma manual, navegando por los sitios web y almacenándolos en hojas de cálculos o bases de datos. Este proceso, además de ser lento, se fue complicando a medida que aparecían sitios web dinámicos y crecía la cantidad de datos en línea.

En 1993, Matthew Gray escribió World Wide Web Wanderer, un programa que recorría la web y recopilaba información de páginas web de forma automatizada.

En 1993, surgió el primer motor de búsqueda basado en un robot web. JumpStation dio el paso a que nuevos web crawlers apareciesen durante esos años, como Infoseek, Altavista y Excite; que basaron sus motores en JumpStation, utilizándolo como núcleo.

Aparición de la Web 2.0: En la década de los 2000, empezaron a alzarse los sitios web de contenido dinámico y personalizado. Este hecho impulsó al desarrollo de técnicas más elaboradas.

En 2004, Beautiful Soup proporcionaba ayuda para analizar el contenido web, permitiendo la extracción de datos directamente de los contenedores de las páginas HTML.

Aparecen navegadores automatizados como Selenium, que permite la ejecución de pruebas de software, permitiendo editar acciones o crearlas desde cero, simulando la experiencia de un usuario real.

Democratización y estándares: Con el paso de los años, el web scraping se ha ido regulando, de tal forma que actualmente sigue ciertos estándares a cumplir, aunque a día de hoy siguen surgiendo ciertos problemas legales con los términos de servicio o la privacidad, y problemas con la carga del servidor. Las herramientas se han mejorado, dando lugar a:

Scrapy, un framework de código abierto escrito en Python que permite la construcción de spiders complejos.

Servicios en la nube como ParseHub y Octoparse que hacen más accesible el scraping debido a que se elimina la necesidad de programar.

Usos comunes: b

Análisis de mercado: Se utiliza para recopilar precios y características de productos que están disponibles en el mercado.

Investigación: Permite obtener datos de sitios académicos y gubernamentales.

Contenido de medios: Permite recopilar información acerca de noticias o entradas a blogs.

IA: Se pueden entrenar modelos a partir de los datos recuperados.

Tal y como se ha mencionado anteriormente, existen muchas herramientas que permiten realizar web scraping tanto si eres programador, ingeniero o si lo haces como disfrute personal. Estas herramientas son:

Python y Selenium: Desde Python, mediante librerías como Beautiful Soup o Scrapy, se puede realizar web scraping.

ParseHub y Octoparse: Existen otras formas de hacer web scraping sin necesidad de hacer código.

## EJEMPLO PRÁCTICO: WEB SCRAPING CON BEAUTIFULSOUP

En este ejemplo, se verá una sencilla extracción de datos de una página web. A la hora de realizar cualquier acción de web scraping, siempre se lleva a cabo los mismos procedimientos y pasos:

1. Estudio de la estructura de la página a “scrapear”. La estructura HTML, para poder obtener los datos que almacena cada etiqueta HTML deseada, se debe analizar.
2. Creación de la petición URL.
3. Obtención de los datos de la página HTML.
4. Tratamiento de datos. En este apartado, se debe personalizar respecto al trato que se quiere dar. En este caso, se muestra por pantalla.
5. Por último, se almacenan los datos en alguna variable. En este caso, se guardan en una lista de diccionarios, en el que cada diccionario se trata cada página a “scrapear”. Normalmente, se realiza el almacenamiento de los datos en un archivo csv. Nosotros llevaremos este caso en el caso de los datasets con Pandas

### HERRAMIENTAS NECESARIAS A UTILIZAR

Para realizar de forma correcta y sin errores esta práctica, es necesario configurar el entorno y las librerías correspondientes. Dependiendo del sistema operativo que se tenga, se utiliza el correspondiente gestor de paquetes de dichos SO. En el caso de Windows, se utiliza el comando **“pip install nom\_libreria”.** Si se tiene Node.js instalado, también se puede utilizar el gestor de paquetes “npm”.

En esta práctica, las herramientas utilizadas son las siguientes:

1. Request: Librería a utilizar para realizar las peticiones a las URL en el que se hace el web scraping. Para instalarlo, se utiliza el comando “pip install request” en Windows.
2. Bs4: Librería correspondiente a BeatifulSoup, utilizada para el web scraping para poder obtener el HTML de la respuesta de la request y obtener los elementos que deseamos, extrayendo de forma sencilla los datos de dicha página. Para instalarlo, se utiliza el comando “pip install bs4”.
3. CSV: Librería para poder extraer, leer y crear archivos .csv. Para instalarlo, se utiliza el comando “pip install csv”.

## CREACIÓN DE LA PETICIÓN Y BEAUTIFULSOUP

En todo código relacionado con el web scraping, lo primero de todo es establecer la petición a la URLs que vamos a querer “scrapear”.

# ----- DEFINICIÓN DE LA PETICIÓN -----  
 # Se inicializa el diccionario donde se almacenan las variables a extraer y sus valores  
 d={}  
 # Se definen las cabeceras para la petición. Desde el modo inspeccionar del navegador se puede obtener el user-agent que utiliza  
 headers = {"user-agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36"}  
 # Se realiza la petición con la que se extraen los datos  
 # Para ello, se muestra un mensaje por terminal indicando sobre qué url se va a realizar la petición  
 print(f'{azul}Realizando la petición: {amarillo}{url}{blanco}')  
 # Se crea la petición con la url y las cabeceras definidas previamente. Además, se añade un timeout por si tardase demasiado tiempo en ejecutar la petición  
 req = requests.get(url, headers=headers, timeout=10)  
 # Se imprime el código de respuesta obtenido de la petición  
 print(f'{azul}Código de respuesta...: {amarillo}{req.status\_code} {req.reason}{blanco}')  
 # En caso de que la petición no sea exitosa  
 **if** req.status\_code != 200:  
 # Salta una excepción con el mensaje de error y el código de estado correspondientes  
 **return** {"error" : f"{req.reason}", "status\_code" : f"{req.status\_code}"}  
 # Si todo va bien, se procede a extraer los datos mediante Web Scraping

Hay que poner foco en la línea subrayada en amarillo, ya que es la función que realiza exactamente la petición y de donde obtendremos el HTML, y en lo subrayado en verde, en el que se configura la cabecera de nuestra petición, sobre el navegador a utilizar (user-agent).

En esta línea, se llama a la función get() de la librería “request”, la cual realiza la petición HTTP con el método GET sobre la URL que se ha guardado en la variable “url”, estableciendo una cabecera, que se ha configurado en la variable “headers” y por último, se establece un timeout de 10s, en el que si no se consigue una respuesta en ese tiempo, se concluye que no se ha podido establecer conexión con dicha URL.

Una vez realizado la petición, se maneja el caso en el que no se pueda obtener una respuesta positiva (lo subrayado en rosa).

## OBTENCIÓN DE LOS DATOS

Una vez que se conoce que se ha obtenido una respuesta positiva por parte de la petición realizada, se guarda toda la petición de respuesta en la variable “req”.

Una vez que ya se ha obtenido la estructura de la página HTML, se necesita “estudiar” aquellas etiquetas que se asocian a los datos que se van a extraer. La manera de proceder es muy sencilla: en el navegador, se accede a la página que se quiere “scrapear”, haciendo click derecho sobre la página y “Inspeccionar elemento”. De esta forma, se abre una pestaña de desarrolladores, en la que se puede observar la estructura de nuestra página a “scrapear”. Se puede observar en la siguiente imagen:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Una vez se tiene dicha pestaña, se procede a obtener de manera sencilla aquellas etiquetas que guardan los datos a obtener. Para ello, se da sobre el primer botón de la esquina superior izquierda de la pestaña de desarrollo que se acaba de abrir. De dicha manera, al pasar el cursor sobre cualquier elemento, se puede obtener la etiqueta HTML que la contiene.

En el ejemplo práctico, se obtienen los siguientes elementos:

1. Id del objeto.
2. Nombre del producto (almacenado en un h2)
3. La url de la imagen
4. Las plataformas para las que sirve.
5. La valoración
6. Precio original
7. Precio actual

Se procederá de dos formas:

* Sin BeatifulSoup: En el caso del id, se puede obtener directamente de la URL. La cual, se especifica en el último apartado.
* Con BeatifulSoup: Se establece una variable que se crea a partir de la librería Beatiful Soup, de la siguiente manera:

soup = BeautifulSoup(req.text, "html.parser")

En esta línea de código, se define el objeto BeautifulSoup a partir del código que se encuentra en req.text y se convierte a HTML con "html.parser"

### EXTRACCIÓN DEL ID

Existen variables que se pueden extraer directamente de la url desde la que se realiza la petición, como es el caso de la url del producto o el id del producto

Para extraer la url del producto, simplemente se extrae la url de la petición.

d["url"] = req.url  
 # El id del producto se localiza al final de la url hasta el caracter '/'. Para extraerlo, se puede obtener la url, partirla por '/' y extraer el último dato   
 d["id"] = d["url"].split("/")[-1]

### EXTRACCIÓN DE DATOS CON BEATIFULSOUP

La extracción de datos se debe hacer de forma minuciosa, inspeccionando la página y averiguando dónde están localizadas. La extracción de los datos siempre se van a seguir de la siguiente manera:

- find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML

- text: Convierte el resultado a texto

- strip(): Extrae el dato del texto

Por último, el dato se almacena en el diccionario.

En los casos en las que se obtengan un UNICO elemento, como puede ser el nombre del producto, URL de la imagen, etc …

d["nombre\_producto"] = soup.find("h2", class\_="product-title").text.strip()

En el caso de se vayan a obtener una lista de elementos, como puede ser las plataformas a las que va dirigidas el producto, se procederá de la siguiente manera.

- find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML

- find\_all("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza todos los valores que coincidan con la etiqueta, ya que puede estar disponible para varias plataformas.

objs\_plat = soup.find("dd").find\_all("a")  
 d["plataformas"] = []  
 **for** item **in** objs\_plat:  
 # Por cada elemento que se haya localizado, se extrae y se convierte a texto. Por último, se almacena en la colección  
 d["plataformas"].append(item.text.strip())

En “objs\_plat” se guarda la lista que se obtienen de todos los elementos que se localizan en el HTML. Posteriormente, se itera y por cada elemento de lista, se va almacenando en el diccionario.

Para extraer la **valoración**, se realiza el siguiente procedimiento.

- find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML

- attrs: Obtiene todos los parámetros que se encuentran almacenados en el objeto localizado

- get("donde se encuentra el dato"): Extrae el dato concreto de todos los parámetros extraídos con la instrucción anterior

La valoración, como puede estar vacía, se debe tratar con una estructura try/except. Además, en la página desde la que se realiza el ejemplo, la valoración se

expresa en estrellas. La puntuación queda almacenada en el objeto HTML que está definido div class = "review-points-4".

**try**:  
 # Para extraer la valoración se hacen los siguientes pasos:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - attrs: Obtiene todos los parámetros que se encuentran almacenados en el objeto localizado  
 # - get("donde se encuentra el dato"): Extrae el dato concreto de todos los parámetros extraídos con la instrucción anterior  
 c\_point = soup.find("a", class\_="reviews-points-m").attrs.get("class")[-1]  
 # Se extrae la valoración del último lugar de la cadena y se convierte a entero  
 puntos = int(c\_point[-1])  
 # Por último, el dato se almacena en el diccionario como un entero  
 d["valoracion"] = int(puntos)  
 # Si no hay una valoración, se almacena un objeto vacío  
 **except**:  
 d["valoracion"] = None

Los **precios**, como pueden estar vacíos, se debe tratar con una estructura try/except. Además, en la página desde la que se realiza el ejemplo, los precios se encuentran en un mismo objeto, por lo que deben separarse primero y tratarse después. Para ello, se extrae el objeto precio con find().

obj\_precio = soup.find("div", class\_="buy--price")  
 **try**:  
 # Para extraer el precio original se hacen los siguientes pasos:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - text: Convierte el objeto a texto  
 # - replace("dato que se quiere reemplazar", "dato por el que se reemplaza"): Reemplaza los elementos que no son necesarios para convertirlos a float  
 # Por último, se convierte el dato a float y se almacena  
 d["precio\_original"] = float(obj\_precio.find("small").text.replace("€","").replace(",","."))  
 # Si no hay un precio original (porque no está en stock), se almacena un objeto vacío  
 **except**:  
 d["precio\_original"] = None  
 **try**:  
 # Para extraer el precio actual hay que extraer la parte entera y la parte decimal porque ambas están separadas.   
 # Para la parte entera se hacen los siguientes pasos:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - text: Convierte el objeto a texto  
 # - strip(): Extrae el dato del texto   
 # - split(): Se separa el dato la parte  
 p\_int = obj\_precio.find("span").text.strip().split("\n")[0].strip()  
 # Para la parte decimal se hacen los siguientes pasos:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - text: Convierte el objeto a texto  
 # - replace("dato que se quiere reemplazar", "dato por el que se reemplaza"): Reemplaza los elementos que no son necesarios para convertirlos a float  
 p\_dec = obj\_precio.find("span", class\_="decimal").text.replace("'",".")  
 # Por último, se juntan ambos datos, el resultado se convierte a float y se almacena  
 d["precio\_actual"] = float(p\_int + p\_dec)  
 # Si no hay un precio actual (porque no hay un descuento), se almacena un objeto vacío  
 **except**:  
 d["precio\_actual"] = None

Por último se devuelve el diccionario en el que se ha ido guardando los datos de la página, en variable “d”.

**return** d

Por lo tanto, el código de la obtención de datos con web scraping quedaría de la siguiente forma:  
Por lo tanto, el código de la obtención de datos con web scraping quedaría de la siguiente forma:  
**def** **datos\_Scraping**(url):  
 # ----- DEFINICIÓN DE LA PETICIÓN -----  
 # Se inicializa el diccionario donde se almacenan las variables a extraer y sus valores  
 d={}  
 # Se definen las cabeceras para la petición. Desde el modo inspeccionar del navegador se puede obtener el user-agent que utiliza  
 headers = {"user-agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36"}  
 # Se realiza la petición con la que se extraen los datos  
 # Para ello, se muestra un mensaje por terminal indicando sobre qué url se va a realizar la petición  
 print(f'{azul}Realizando la petición: {amarillo}{url}{blanco}')  
 # Se crea la petición con la url y las cabeceras definidas previamente. Además, se añade un timeout por si tardase demasiado tiempo en ejecutar la petición  
 req = requests.get(url, headers=headers, timeout=10)  
 # Se imprime el código de respuesta obtenido de la petición  
 print(f'{azul}Código de respuesta...: {amarillo}{req.status\_code} {req.reason}{blanco}')  
 # En caso de que la petición no sea exitosa  
 **if** req.status\_code != 200:  
 # Salta una excepción con el mensaje de error y el código de estado correspondientes  
 **return** {"error" : f"{req.reason}", "status\_code" : f"{req.status\_code}"}  
 # Si todo va bien, se procede a extraer los datos mediante Web Scraping  
 # ----- EXTRACCIÓN DE DATOS SIN BEAUTIFULSOUP -----  
 # Existen variables que se pueden extraer directamente de la url desde la que se realiza la petición, como es el caso de la url del producto o el id del producto  
 # Para extraer la url del producto, simplemente se extrae la url de la petición  
 d["url"] = req.url  
 # El id del producto se localiza al final de la url hasta el caracter '/'. Para extraerlo, se puede obtener la url, partirla por '/' y extraer el último dato   
 d["id"] = d["url"].split("/")[-1]  
 # ----- EXTRACCIÓN DE DATOS CON BEAUTIFULSOUP -----  
 # Para extraer las siguientes variables se requiere de Web Scraping y, por tanto, de la librería BeautifulSoup.   
 # Para ello, se define el objeto BeautifulSoup a partir del código que se encuentra en req.text y se convierte a HTML con "html.parser"   
 soup = BeautifulSoup(req.text, "html.parser")  
 # La extracción de datos se debe hacer de forma minuciosa, inspeccionando la página y averiguando dónde están localizadas  
 # El nombre del producto se obtiene de la siguiente manera:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - text: Convierte el resultado a texto  
 # - strip(): Extrae el dato del texto   
 # Por último, el dato se almacena en el diccionario  
 d["nombre\_producto"] = soup.find("h2", class\_="product-title").text.strip()  
 # La url de la imagen del producto se obtiene de la siguiente manera:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - attrs: Obtiene todos los parámetros que se encuentran almacenados en el objeto localizado  
 # - get("donde se encuentra el dato"): Extrae el dato concreto de todos los parámetros extraídos con la instrucción anterior  
 # Por último, el dato se almacena en el diccionario  
 d["url\_imagen"] = soup.find("img", id="product-cover").attrs.get("src")  
 # Las plataformas del producto se obtienen de la siguiente manera:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - find\_all("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza todos los valores que coincidan con la etiqueta, ya que   
 # puede estar disponible para varias plataformas  
 objs\_plat = soup.find("dd").find\_all("a")  
 d["plataformas"] = []  
 **for** item **in** objs\_plat:  
 # Por cada elemento que se haya localizado, se extrae y se convierte a texto. Por último, se almacena en la colección  
 d["plataformas"].append(item.text.strip())  
 # La valoración, como puede estar vacía, se debe tratar con una estructura try/except. Además, en la página desde la que se realiza el ejemplo, la valoración se  
 # expresa en estrellas. La puntuación queda almacenada en el objeto HTML que está definido div class = "review-points-4".  
 **try**:  
 # Para extraer la valoración se hacen los siguientes pasos:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - attrs: Obtiene todos los parámetros que se encuentran almacenados en el objeto localizado  
 # - get("donde se encuentra el dato"): Extrae el dato concreto de todos los parámetros extraídos con la instrucción anterior  
 c\_point = soup.find("a", class\_="reviews-points-m").attrs.get("class")[-1]  
 # Se extrae la valoración del último lugar de la cadena y se convierte a entero  
 puntos = int(c\_point[-1])  
 # Por último, el dato se almacena en el diccionario como un entero  
 d["valoracion"] = int(puntos)  
 # Si no hay una valoración, se almacena un objeto vacío  
 **except**:  
 d["valoracion"] = None  
 # Los precios, como pueden estar vacíos, se debe tratar con una estructura try/except. Además, en la página desde la que se realiza el ejemplo, los precios se  
 # encuentran en un mismo objeto, por lo que deben separarse primero y tartarse después.  
 # Para ello, se extrae el objeto precio con find()   
 obj\_precio = soup.find("div", class\_="buy--price")  
 **try**:  
 # Para extraer el precio original se hacen los siguientes pasos:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - text: Convierte el objeto a texto  
 # - replace("dato que se quiere reemplazar", "dato por el que se reemplaza"): Reemplaza los elementos que no son necesarios para convertirlos a float  
 # Por último, se convierte el dato a float y se almacena  
 d["precio\_original"] = float(obj\_precio.find("small").text.replace("€","").replace(",","."))  
 # Si no hay un precio original (porque no está en stock), se almacena un objeto vacío  
 **except**:  
 d["precio\_original"] = None  
 **try**:  
 # Para extraer el precio actual hay que extraer la parte entera y la parte decimal porque ambas están separadas.   
 # Para la parte entera se hacen los siguientes pasos:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - text: Convierte el objeto a texto  
 # - strip(): Extrae el dato del texto   
 # - split(): Se separa el dato la parte  
 p\_int = obj\_precio.find("span").text.strip().split("\n")[0].strip()  
 # Para la parte decimal se hacen los siguientes pasos:  
 # - find("tipo de elemento", "nombre de la clase, id u objeto donde se almacene"): Localiza la primera coincidencia en el código HTML  
 # - text: Convierte el objeto a texto  
 # - replace("dato que se quiere reemplazar", "dato por el que se reemplaza"): Reemplaza los elementos que no son necesarios para convertirlos a float  
 p\_dec = obj\_precio.find("span", class\_="decimal").text.replace("'",".")  
 # Por último, se juntan ambos datos, el resultado se convierte a float y se almacena  
 d["precio\_actual"] = float(p\_int + p\_dec)  
 # Si no hay un precio actual (porque no hay un descuento), se almacena un objeto vacío  
 **except**:  
 d["precio\_actual"] = None  
 # Devuelve el diccionario creado  
 **return** d

## TRATAMIENTO DE LOS DATOS

En el caso del tratamiento de los datos, se ha creado una nueva función llamada “tratamiento\_de\_datos()”, en la cual se mostrará por pantalla los datos del diccionario que se obtiene con la función “datos\_Scraping()”.

**def** **tratamiento\_Scraping**(datos):  
 # Del diccionario resultante se extraen los datos obtenidos y se realiza el tratamiento   
 **for** clave, valor **in** datos.items():  
 # Para cada clave (que se muestra en mayúsculas), se muestra el valor correspondiente  
 print(f'{azul}{clave.upper()}: {amarillo}{valor}{blanco}')

## ALMACENAMIENTO DE LOS DATOS

En el caso del almacenamiento de los datos, se realizará utilizando dataframes, un objeto que nos proporciona la librería de Pandas, utilizada para el Análisis de datos a nivel mundial. Esta herramienta se verá mas adelante, en el apartado de datasets.

Datasets

Un dataset es un conjunto de datos organizados, normalmente, en tablas que se utiliza para el análisis de datos, el entrenamiento de IA y la toma de decisiones. En un dataset, cada fila representa un registro, y cada columna contiene una variable o atributo del registro descrito. Un dataset se compone de variables, registros y metadatos. Un dataset puede ser estructurado, no estructurado o semiestructurado.

Los datasets han ido evolucionando junto a las TIC y a la ciencia de datos, diferenciándose varias fases:

Inicios en Bases de Datos: Entre los años 60 y 80 surgieron los primeros datasets almacenados en las bases de datos relacionales. Para tratar con estos conjuntos de datos se empleaban herramientas como SQL, facilitando la creación y la manipulación de datasets.

Era de Internet: Entre los años 90 y los 2000, con el nacimiento de Internet, la cantidad de datos que había en la red aumentó drásticamente, proveniente de usuarios, redes sociales y sistemas. Para manejar esta cantidad de datos, surgieron herramientas como Hadoop (maneja datasets grandes y complejos) y UCI Machine Learning Repository (proporciona datasets para uso académico y aprendizaje automático) que facilitaron el tratamiento de la cantidad masiva de datos, conocidos como Big Data.

IA y Big Data: A partir de 2010, los gobiernos y las organizaciones científicas decidieron promover la publicación de datasets con el fin de fomentar la transparencia. El auge de las IAs también ha fomentado el cambio en los datasets, surgiendo herramientas como ImageNet y GPT.

Usos comunes: Los datasets se emplean en:

Ciencia de datos e IA: Los datasets son útiles para el entrenamiento de modelos de IA y la detección de patrones, facilitando la predicción de eventos en ciertos campos como los mercados o los sistemas de salud.

Negocio y Marketing: los datasets ayudan a analizar los clientes de cierto mercado, sesgando sus gustos y optimizando las estrategias publicitarias.

Investigación: Los datasets permiten realizar investigaciones y publicaciones, debido a toda la cantidad de información que contiene.

Los datasets siguen una serie de pasos para llevar a cabo su tratamiento:

1. Recopilación de datos:
2. Web scraping (Beautiful Soup, Scrapy, Selenium)
3. APIs (Postman, Kaggle)
4. Datos sintéticos (DataSynthesizer o Faker)
5. Procesamiento de datos
6. Pandas
7. OpenRefine
8. Análisis de datos
   1. Numpy
   2. SKLearn
9. Almacenamiento de datos
10. MySQL/PostgreSQL
11. AWS

MOSTRAR EJEMPLO A PARTIR DEL DATASET QUE HAYAMOS CREADO EN LA PRIMERA PARTE:

1. Usamos los datos recopilados con el Web Scraping
2. Los tratamos con Pandas
3. Los almacenamos en un CSV o en MySQL/PostgreSQL

## DATAFRAMES CON PANDAS

Para poder realizar los ejemplos en condiciones, será necesario configurar nuestro entorno con el software necesario. En este caso, se necesaritará los siguientes requisitos:

* Tener instalado en nuestro sistema operativo, el compilador de Python. Es recomendable tenerlo actualizado al último o la versión más estable del momento.
* La librería “Pandas”: Para ello se debe realizar el siguiente comando en vuestra consola: Pip install Pandas

Texto

Descripción generada automáticamente

* La librería “csv”: Esta librería es necesaria, para poder leer, crear y modificar archivos csv. Además, es la librería que utiliza “Pandas” a la hora de realizar las lecturas y escrituras sobre este tipo de archivos.
* La librería “openpyxl”: Esta librería es necesaria, para poder leer, crear y modificar archivos de tipo Hojas de cálculo. Además, es la librería que utiliza “Pandas” a la hora de realizar las lecturas y escrituras sobre este tipo de archivos.

Captura de pantalla de computadora

Descripción generada automáticamente

* La librería scikit-learn: Scikit-Learn ofrece herramientas para limpiar y preparar los datos antes de crear modelos. Nosotros la utilizaremos únicamente para poder normalizar los datos escogidos de nuestro dataset.

Texto

Descripción generada automáticamente con confianza media

### CARGA DE DATOS EN EL DATAFRAME

En el análisis de datos, la obtención de ellos se puede realizar de distíntas maneras.

* Archivos (CSV, JSON, hojas de cálculo, etc…): Cualquier archivos que guarde información. Incluso puedes utilizar un documento de texto (.txt).
* Base de datos
* Web Scraping

En este caso, cargaremos los datos obtenidos mediante un Web Scraping de la página de Game, sobre algunos productos que hemos elegido. En esta práctica, se guardarán la siguiente información de cada página (producto):

* URL
* Id del objeto, en la base de datos de Game
* Nombre del producto
* URL de la imagen
* Las plataformas en las que se pueden utilizar (Play 5, Xbox, PC, etc…)
* Valoración
* Precio original
* Precio actual

La obtención y almacenamiento de los datos del Web Scraping se realizará en formato de diccionario, un formato que el constructor de Dataframes de Pandas acepta. Además, es una buena práctica realizarla de dicha forma, ya que nos podemos ahorar el segundo parámetro del constructor, que es para especificar el nombre de las columnas del Dataframe, ya que cogerá los nombres de la “key” del diccionario.

### EXPLICACIÓN DEL CÓDIGO

La función principal que se utilizará en esta práctica, referente a los Dataframes y procesados y guardado de datasets es el siguiente.

**def** **almacenamiento\_dataframe**(datos, columnas=[]):  
 # 1- SE CREAR EL DATAFRAME A PARTIR DEL WEB SCRAPING.  
# 2- SE LIMPIA LOS DATOS  
# 3- SE GUARDAN LOS DATOS EN UN ARCHIVO EXCEL Y JSON.  
 ruta\_json = 'datos.json'  
 ruta\_csv = 'datos.csv'  
 ruta\_excel = 'datos.xlsx'  
 #1- Con el constructor por defecto, se pueden crear dataframes de cualquier forma, ya sea a partir de una lista de listas, lista de objetos, diccionarios.  
 **if** columnas:  
 df = pd.DataFrame(datos, columns=columnas)   
 **else**:  
 df = pd.DataFrame(datos)  
   
 #2 - SE LIMPIAN LOS DATOS  
 df = limpiar\_datos(df)  
  
 #3 - SE GUARDAN LOS DATOS EN UN ARCHIVO EXCEL Y JSON, O CSV  
 guardar\_datos(df, ruta\_excel=ruta\_excel, ruta\_json=ruta\_json)   
 df.to\_csv(ruta\_csv)

La carga de los datos se verá de la siguiente forma:

#1- Con el constructor por defecto, se pueden crear dataframes de cualquier forma, ya sea a partir de una lista de listas, lista de objetos, diccionarios.

**if** columnas:  
 df = pd.DataFrame(datos, columns=columnas)   
 **else**:  
 df = pd.DataFrame(datos)

Este bloque condicionar mira si, en nuestra función, se ha querido establecer unos nombres de las columnas diferentes a las establecidas en el diccionario. Sino que se carga directamente en el Dataframe.

#2 - SE LIMPIAN LOS DATOS  
 df = limpiar\_datos(df)

Se “limpian los datos” en el caso de que hubiese datos mal procesados. En estos ejemplos, que son muy controlados, no sería muy necesario, pero siempre es recomendable, ya que al hacer Web Scraping exhaustivo existirán productos u objetos que se encuentre repetidos o mal formados.

**def** **limpiar\_datos**(df):  
 """  
 Limpia datos nulos y duplicados en el DataFrame.  
  
 :param df: DataFrame a limpiar.  
 :return: DataFrame limpio.  
 """  
 #SE COMPRUEBA QUE TODAS LAS COLUMNAS SON HASHEABLES  
 **for** col **in** df.columns:  
 **if** df[col].apply(**lambda** x: isinstance(x, (list, dict))).any():  
 print(f"La columna '{col}' tiene valores no hashables.")  
 print("Concivertiendo la columna en valores hasheables")  
 convertir\_columnas\_hasheables(df,col)  
  
 # Eliminar duplicados  
 df = df.drop\_duplicates()  
   
 # Rellenar valores nulos con métodos adecuados (media, moda, etc.)  
 **for** column **in** df.columns:  
 **if** df[column].isnull().sum() > 0:  
 **if** df[column].dtype == "object":  
 df[column].fillna(df[column].mode()[0], inplace=True) # Rellenar con la moda  
 **else**:  
 df[column].fillna(df[column].mean(), inplace=True) # Rellenar con la media  
 print("Datos nulos y duplicados eliminados.")  
  
 **return** df  
# Función para guardar el DataFrame en distintos formatos

Esta función también realiza un procesamiento de objetos hasheables. Es decir, al intentar eliminar los duplicados, no se pueden realizar sobre objetos que no sean hasheables, como las listas. Es por eso, que también se debe realizar esas comprobaciones antes de realizar las eliminaciones.

#SE COMPRUEBA QUE TODAS LAS COLUMNAS SON HASHEABLES  
 **for** col **in** df.columns:  
 **if** df[col].apply(**lambda** x: isinstance(x, (list, dict))).any():  
 print(f"La columna '{col}' tiene valores no hashables.")  
 print("Concivertiendo la columna en valores hasheables")  
 convertir\_columnas\_hasheables(df,col)  
  
 # Eliminar duplicados  
 df = df.drop\_duplicates()

**def** **convertir\_columnas\_hasheables**(df,nombre\_columna):  
 df[nombre\_columna] = df[nombre\_columna].apply(**lambda** x: str(x) **if** isinstance(x, list) **else** x)

Por último, se guarda dicho dataframe, ya procesado, en algún archivo utilizado en el mundo del análisis de datos, como pueden ser CSV, JSON, hojas de cálculo, etc…

En nuestro caso, hemos decidido pasarlo a:

* JSON
* CSV
* Excel

#### ¿CÓMO SE GUARDA UN DATAFRAME EN UN ARCHIVO?

Muy sencillo. La librería de “Pandas” ya nos ofrece funciones que nos permiten fácilmente pasar nuestro Dataframe a cualquier archivo de almacenamiento de datos. Dichas funciones mantienen siempre la misma terminología.

Df.to\_**tipo\_archivo**(“ruta\_archivo”)

Siendo “df” nuestro dataframe que queremos guardar, “tipo\_archivo” es el formato en que deseamos guardar nuestro dataframe y “ruta\_archivo” la ruta del directorio en que que queremos que se guarde.

El tipo de archivo, según el tipo, será:

* to\_csv(ruta\_csv)
* to\_json(ruta\_json)
* to\_excel(ruta\_excel)

También es muy importante que, el nombre de la ruta y el nombre del archivo, tengan una extensión que el tipo de archivo pueda aceptar. En el caso de Excel, puede aceptar cualquier ruta con un nombre de archivo de Hojas de Cálculo (.xlsx. etc…), pero si el tipo de archivo es JSON, el nombre del archivo tiene que tener una extensión “.json”.

Además, cada tipo de función de guardado de archivos tiene diferentes configuraciones que se le pueden pasar. En nuestro caso, hemos utilizado las siguientes:

* to\_excel: En esta función, hemos deseado que no se establezca como una columna, los índices que tienen en el dataframe, utilizando la propiedad **“index=False”.**

df.to\_excel(ruta\_excel, index=False) #Index = False para que no se guarden como una columna el índice de cada tupla.

* To\_json: En esta función hemos utilizado 2 configuraciones:
  + **Orient**: Indica el formato del JSON. Hay diferentes formatos. Nosotros utilizamos “records” ya que se muestra como una lista de “columna: valor” por cada objeto del dataframe. Si se tiene curiosidad de conocer los diferentes formatos de JSON, puedes visitar la página oficial de [Pandas](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_json.html).
  + **Lines**: Si 'orient' es 'records' escribe en formato json delimitado por líneas. Lanzará ValueError si 'orient' es incorrecto ya que otros no son tipo lista.

La ruta del archivo puede ser tanto absoluta como relativas a la ubicación del archivo que se está ejecutando.

## REFERENCIAS

1. <https://www.youtube.com/watch?v=nBYIbwgTR7E&list=PLheIVUbpfWZ17lCcHnoaa1RD59juFR06C>
2. <https://www.youtube.com/watch?v=bK3EwIMHm94&list=PLcpNLPfo7LVIZj8mU1ka9DLhMIbarxpFK>
3. <https://www.octoparse.es/blog/como-comenzo-y-sucedera-en-futuro>
4. <https://en.wikipedia.org/wiki/World_Wide_Web_Wanderer>
5. <https://en.wikipedia.org/wiki/JumpStation>
6. <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
7. <https://sentrio.io/blog/que-es-selenium/>
8. <https://scrapy.org>
9. <https://es.wikipedia.org/wiki/Scrapy>
10. <https://elmundodelosdatos.com/extraccion-datos-sitios-web-productos-zara/>
11. <https://github.com/scrapy/scrapy>
12. <https://www.parsehub.com>
13. <https://www.octoparse.es>
14. <https://aws.amazon.com/es/what-is/hadoop/>
15. <https://archive.ics.uci.edu>
16. <https://www.postman.com>
17. <https://www.kaggle.com/datasets>
18. <https://openrefine.org>
19. <https://faker.readthedocs.io/en/master/>
20. <https://github.com/DataResponsibly/DataSynthesizer>