MEMORIA del Proyecto de HOSTING Web Scraping

Contenido

[1. Introducción 2](#_Toc157938653)

[2. Importación de Librerías 2](#_Toc157938654)

[3. Configuración de Opciones de Chrome y Creación del WebDriver 4](#_Toc157938655)

[4. Adquisición de HTML con Espera Explícita 5](#_Toc157938656)

[5. Extracción de Elementos y Navegación por Paginación y Código Fuente 6](#_Toc157938657)

[6. Crear df y guardar en Csv. Formato texto. 17](#_Toc157938658)

[8. Información adicional 17](#_Toc157938659)

[9. Conclusiones 17](#_Toc157938660)

## 1. Introducción

Este documento detalla los pasos y fragmentos de código utilizados en un proyecto de web scraping. El objetivo del proyecto es automatizar la extracción de datos de páginas web utilizando Selenium y BeautifulSoup, demostrando el proceso desde la configuración del entorno hasta la navegación por las páginas web y la extracción de la información deseada.

## 2. Importación de Librerías

El proyecto utiliza varias librerías de Python para el scraping web y la automatización. Estas incluyen selenium para la automatización de la interacción con el navegador web, BeautifulSoup para el análisis de HTML, y pandas para la manipulación de datos. A continuación, se detalla la instalación y el propósito de cada librería.

* selenium: Automatiza la interacción con el navegador web.
* BeautifulSoup: Analiza documentos HTML y XML.
* pandas: Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.

Se proporcionan comandos para instalar estas librerías en el código fuente.

!pip install selenium

!pip install fake\_useragent

!pip install requests-html

!pip install webdriver-manager

!pip install --upgrade webdriver-manager

Estas líneas de código están utilizando el comando pip para instalar varias bibliotecas necesarias para este proyecto. Cada comando pip install instala una biblioteca específica. Aquí está una descripción de cada biblioteca:

* selenium: Se utiliza para automatizar la interacción con el navegador web y realizar tareas de web scraping.
* fake\_useragent: Se utiliza para generar un agente de usuario (User-Agent) aleatorio, lo que puede ayudar a evitar la detección como un robot.
* requests-html: Se utiliza para realizar solicitudes HTTP y analizar el contenido HTML de las páginas web.
* webdriver-manager: Ayuda a gestionar y descargar automáticamente el controlador del navegador Chrome.
* --upgrade webdriver-manager: Actualiza la biblioteca webdriver-manager a la última versión.

import time

from bs4 import BeautifulSoup

from selenium import webdriver

from selenium.webdriver.chrome.options import Options

from fake\_useragent import UserAgent

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.chrome.service import Service

from webdriver\_manager.chrome import ChromeDriverManager

from multiprocessing import Pool

import random

import json

import os

import pandas as pd

from datetime import datetime

Estas líneas de código importan diversas bibliotecas y módulos que serán utilizados en el proyecto. Aquí está una descripción de cada uno de ellos:

* time: Proporciona funciones relacionadas con la gestión del tiempo, como sleep para introducir pausas.
* BeautifulSoup: Utilizado para analizar el contenido HTML de las páginas web.
* selenium: La biblioteca principal que permite la automatización del navegador web.
* Options y webdriver.chrome.options: Se utilizan para configurar opciones de Chrome como modo sin cabeza, agente de usuario, etc.
* UserAgent: Genera un agente de usuario aleatorio para ayudar a evitar la detección como un robot.
* By: Proporciona estrategias para buscar elementos en una página web utilizando selecciones basadas en atributos.
* WebDriverWait y expected\_conditions: Se utilizan para esperar hasta que ciertas condiciones se cumplan antes de continuar con la ejecución del código.
* Service y ChromeDriverManager: Ayudan a gestionar y descargar el controlador del navegador Chrome de forma automática.
* multiprocessing.Pool: Se utiliza para realizar procesamiento en paralelo, lo que puede acelerar el proceso de web scraping.
* random: Permite generar números aleatorios.
* json: Se utiliza para trabajar con datos en formato JSON.
* os: Proporciona funciones para interactuar con el sistema operativo.
* pandas: Utilizado para trabajar con datos estructurados, como la creación de DataFrames.
* datetime: Proporciona funciones para trabajar con fechas y horas.

Estas importaciones son esenciales para el proyecto, ya que proporcionan las funciones y clases necesarias para automatizar la navegación web y realizar web scraping de manera efectiva.

## 3. Configuración de Opciones de Chrome y Creación del WebDriver

Configurar las opciones de Chrome y crear un objeto WebDriver son cruciales para la automatización del navegador. Esta sección explica cómo personalizar los ajustes de Chrome para tareas de scraping web, incluyendo la operación en modo sin cabeza y la suplantación del agente de usuario.

1. **Definir la URL: (ejemplo Madrid)**

airbnb\_url\_madrid = f'https://www.airbnb.es/s/Madrid--España/homes?'

En esta línea, se define una variable llamada airbnb\_url\_madrid que contiene la URL de la página de Airbnb que se desea analizar. La URL incluye la ubicación (Madrid, España) y la categoría de alojamiento (en este caso, "homes").

1. **Conexión:**

driver.get(airbnb\_url\_madrid)

time.sleep(10) # Esperar a que se carguen los datos

Aquí, se utiliza Selenium para abrir la URL en un navegador Chrome controlado por el script. driver.get(airbnb\_url\_madrid) abre la URL en el navegador. Luego, time.sleep(10) pausa la ejecución del script durante 10 segundos, permitiendo que la página web se cargue completamente antes de continuar.

1. **Obtener el HTML de la página:**

html = driver.page\_source

Esta línea obtiene el contenido HTML de la página web cargada utilizando Selenium y lo almacena en la variable html.

1. **Parsear el HTML:**

soup = BeautifulSoup(html, 'lxml') (con lxml se supone que va más rápido)

Aquí, se utiliza BeautifulSoup, una biblioteca de Python para analizar documentos HTML y XML, para parsear el contenido de la respuesta HTTP (que es el HTML de la página web). Se crea un objeto BeautifulSoup llamado soup que facilita la extracción de datos de la página web.

1. **Imprimir información sobre la respuesta HTTP:**

print(driver.current\_url) # URL final después de las redirecciones

print(driver.title) # Título de la página

1. Estas líneas imprimen información sobre la respuesta HTTP obtenida:
   * driver.current\_url muestra la URL final después de que se hayan seguido posibles redirecciones.
   * driver.title muestra el título de la página web actual.

En resumen, este código realiza una serie de acciones para abrir la página de Airbnb de Madrid en un navegador controlado por Selenium, obtiene el contenido HTML de la página y luego utiliza BeautifulSoup para facilitar la extracción de datos de esa página. Las líneas finales imprimen información básica sobre la página web actual. Este código sería el punto de partida para realizar operaciones adicionales de extracción de datos o análisis en la página web de Airbnb.

## 4. Adquisición de HTML con Espera Explícita

El documento detalla métodos para cargar eficientemente la página web airbnb y extraer contenido HTML. Se enfatiza el uso de esperas explícitas para asegurar que los elementos web estén completamente cargados antes de la extracción de datos.

1. airbnb\_url\_madrid = f'https://www.airbnb.es/s/Madrid--España/homes?': Se define la URL de la página web de Airbnb que se desea analizar. Esta URL contiene la búsqueda de alojamientos en Madrid, España.
2. driver.get(airbnb\_url\_madrid): Se utiliza Selenium para abrir la URL en un navegador Chrome controlado por el script. Esto carga la página web en el navegador.
3. time.sleep(10): Se añade una pausa de 10 segundos para permitir que la página web se cargue completamente. Esto es útil para asegurarse de que todos los elementos de la página se hayan cargado antes de continuar.
4. html = driver.page\_source: Se utiliza driver.page\_source para obtener el HTML de la página web actualmente cargada en el navegador controlado por Selenium. Esto captura todo el contenido HTML de la página en una variable llamada html.
5. soup = BeautifulSoup(html, 'lxml'): Se utiliza BeautifulSoup para parsear el contenido HTML almacenado en la variable html. La cadena 'lxml' es el analizador que BeautifulSoup utilizará para analizar el HTML. BeautifulSoup crea un objeto soup que facilita la extracción de datos específicos del HTML.
6. print(driver.current\_url): Se imprime la URL final después de posibles redirecciones. Esto es útil para verificar si la URL ha cambiado después de seguir enlaces o redireccionamientos.
7. print(driver.title): Se imprime el título de la página web actualmente cargada en el navegador controlado por Selenium. Esto proporciona información sobre el título de la página, que puede ser útil para identificar la página web.

## 5. Extracción de Elementos y Navegación por Paginación y Código Fuente

Se discuten técnicas para extraer elementos específicos de las páginas web y navegar a través de resultados paginados. Los ejemplos incluyen la extracción de detalles de listados de anuncios por cada ciudad y la navegación a través las páginas para obtener las características individuales de cada anuncio.

Y además, se explica con detalle el código fuente de Python utilizados en el proyecto. Incluye comentarios para mayor claridad y explica la funcionalidad de cada bloque de código.

1. **Acceso a Google Drive**: Guardar automáticamente los csv en el Google *\*Drive\**
2. **En general: En todos los botones se deja esto, porque a veces da fallo y hay que volver a ejecutar el drive.**

# import time # Librería para controlar el tiempo

# from selenium import webdriver # Librería para controlar el navegador

# from selenium.webdriver.chrome.options import Options # Librería para controlar el navegador

# from selenium.webdriver.common.by import By # Librería para controlar el navegador

# from selenium.webdriver.support.ui import WebDriverWait # Librería para controlar el navegador

# from selenium.webdriver.support import expected\_conditions as EC # Librería para controlar el navegador

# from fake\_useragent import UserAgent # Librería para controlar el navegador

# import pandas as pd # Librería para controlar el navegador

# # Configuración inicial del WebDriver y opciones

# ua = UserAgent() # Crear un objeto UserAgent

# chrome\_options = Options() # Crear un objeto Options

# chrome\_options.add\_argument('--headless') # Modo headless

# chrome\_options.add\_argument('--no-sandbox')

# chrome\_options.add\_argument('--disable-dev-shm-usage')

# chrome\_options.add\_argument("--disable-extensions")

# chrome\_options.add\_argument(f'user-agent={ua.random}')

# chrome\_options.headless = True #

# driver = webdriver.Chrome(options=chrome\_options) # Crear un objeto WebDriver

**Botón 1 :**  Botón 1 es la primera página no hace falta botón clic

Explicación detallada de cada paso.

1. anuncios\_pisos\_pag1 = []: Se crea una lista vacía llamada anuncios\_pisos\_pag1 que se utilizará para almacenar la información de cada anuncio en la página actual.
2. time.sleep(10): Se hace una pausa de 10 segundos para esperar a que la página cargue. Esto es útil después de hacer clic en el botón de siguiente página para asegurarse de que la página siguiente se haya cargado completamente.
3. try:: Se inicia un bloque de código que maneja posibles excepciones.
4. elementos\_anunciospag1 = WebDriverWait(driver, 10).until(...): Se utiliza WebDriverWait para esperar hasta que todos los elementos de anuncios con la clase 'c4mnd7m' estén presentes en el DOM dentro de un límite de tiempo de 10 segundos. Estos elementos son los anuncios en la página actual.
5. Se realizan operaciones similares de espera para otros elementos, como precios, subtitulos, valoraciones, títulos y precios totales.
6. for index, anunciopag1 in enumerate(elementos\_anunciospag1):: Se inicia un bucle for que recorre todos los elementos de anuncios en la página actual. enumerate se utiliza para obtener tanto el índice (index) como el elemento actual (anunciopag1) en cada iteración.
7. Se extraen datos específicos de cada anuncio, como el título, subtitulo, valoración, precio por noche, precio total, enlace (href) y objetivo (target) del enlace. Si alguno de estos datos no está presente, se establece como 'N/A' para manejar casos en los que falta información.
8. Se imprime la información del anuncio actual para fines de depuración y se utiliza "-" \* 50 para imprimir una línea divisoria.
9. Se agrega un diccionario con la información del anuncio actual a la lista anuncios\_pisos\_pag1 utilizando append.
10. Se manejan posibles excepciones internas en el bucle for para asegurarse de que cualquier error no detenga todo el proceso de extracción de datos.
11. Después de recopilar la información de todos los anuncios en la página actual, se cierra el bloque try.
12. except Exception as e:: Si ocurre alguna excepción durante la extracción de datos de los anuncios, se captura y se imprime un mensaje de error.
13. if anuncios\_pisos\_pag1:: Se verifica si se obtuvieron enlaces de anuncios en la página actual.
14. for anuncio in anuncios\_pisos\_pag1:: Si se obtuvieron enlaces de anuncios, se inicia un bucle for para recorrer estos enlaces y realizar scraping de características específicas de cada anuncio.
15. WebDriverWait(driver, 10).until(...): Se espera a que las características estén presentes en el DOM utilizando WebDriverWait. Estas características se identifican por su clase "lgx66tx.atm\_gi\_idpfg4.atm\_l8\_idpfg4.dir.dir-ltr".
16. Se obtienen las características y se almacenan en caracteristicas\_texto.
17. Se imprime opcionalmente el texto de las características.
18. Se agrega el texto de las características al diccionario del anuncio actual bajo la clave 'caracteristicas'.
19. Se manejan posibles excepciones internas para asegurarse de que cualquier error durante el scraping de características no detenga todo el proceso.
20. except Exception as e:: Si ocurre alguna excepción durante el scraping de características, se captura y se imprime un mensaje de error.
21. Si no se obtuvieron enlaces de anuncios en la página actual, se imprime un mensaje indicando que no se realizará el scraping.

**Botón 2 y 3 :**  Botón 2 y 3 se hace con driver.find\_element\_by\_xpath ya que están visibles

1. boton\_2 = driver.find\_element(By.XPATH, "//a[contains(@class, 'atm\_1y33qqm\_1ggndnn') and contains(text(), '2')]"): Esta línea de código busca un elemento en la página web utilizando una expresión XPath. El elemento que se está buscando es un enlace <a> que contiene la clase "atm\_1y33qqm\_1ggndnn" y el texto "2". En otras palabras, busca el enlace que lleva a la página 2.
2. A continuación, el código intenta hacer clic en el enlace encontrado en el paso anterior utilizando boton\_2.click(). Esto se hace para navegar a la página 2 de resultados en Airbnb.
3. time.sleep(5): Después de hacer clic en el enlace, el código espera 5 segundos. Esto se hace para dar tiempo a que la página 2 se cargue completamente antes de continuar.
4. print("URL actual después de hacer clic en el botón 2:", driver.current\_url): Imprime la URL actual después de hacer clic en el botón 2. Esto es útil para verificar que la navegación a la página 2 haya sido exitosa y para obtener la URL actual.
5. time.sleep(10): El código espera 10 segundos antes de continuar. Esto se hace para asegurarse de que la página 4 (o la página siguiente después de hacer clic en el botón 2) se cargue completamente.
6. Luego, el código inicia una nueva lista llamada anuncios\_pisos\_pag2 para almacenar la información de cada anuncio en la página 2.
7. El código utiliza Selenium y WebDriverWait para esperar hasta que los elementos de los anuncios estén presentes en el DOM de la página 2. Esto se hace para asegurarse de que los anuncios se hayan cargado antes de intentar extraer información de ellos.
8. Se repiten los mismos pasos que en el código anterior para extraer información de los anuncios, como el título, el subtitulo, la valoración, el precio por noche, el precio total, el enlace y el target del anuncio.
9. También se maneja la posibilidad de que no se pueda encontrar alguno de los elementos y se asigna 'N/A' en caso de que falten datos.
10. El código intenta realizar scraping de las características específicas de cada anuncio utilizando un bucle for. Se hace clic en el enlace de cada anuncio, se espera a que las características estén presentes en el DOM y se extraen.
11. Las características se almacenan en una lista llamada caracteristicas\_texto.
12. Se imprime el texto de las características, y las características se añaden al diccionario del anuncio correspondiente.
13. Finalmente, el código intenta guardar los resultados en un DataFrame de Pandas si se han extraído enlaces. Si no se han extraído enlaces, se imprime un mensaje indicando que no se realiza scraping.

***Botón 4, suele dar fallos, aunque sea visible, se decide, por tanto, aplicar el código de navegación.***

**Botón 4 a la 15: Este código se encarga de navegar a una página deseada y extraer datos de anuncios en esa página.**

A continuación, explicaré en detalle cada parte del código:

1. driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);").Realiza un scroll en la página.Este script de JavaScript se ejecuta para desplazar la página hacia abajo y asegurarse de que el enlace a la página siguiente sea visible
2. pagina\_deseada y pagina\_actual: Estas variables se utilizan para definir la página deseada a la que se quiere llegar (en este caso, la página 4) y realizar un seguimiento de la página actual durante la navegación.
3. while pagina\_actual < pagina\_deseada:: Esto inicia un bucle while que continuará hasta que la pagina\_actual sea igual a la pagina\_deseada. El propósito de este bucle es navegar a través de las páginas haciendo clic en el botón "Siguiente" hasta llegar a la página deseada.
4. try:: Se inicia un bloque try para manejar posibles excepciones durante la navegación.
5. boton\_siguiente = WebDriverWait(driver, 10).until(...): Este código utiliza WebDriverWait para esperar hasta que el botón "Siguiente" esté presente en el DOM y sea clickeable. Utiliza una expresión XPath para encontrar el botón que contiene el atributo 'aria-label' con el valor 'Siguiente'.
6. boton\_siguiente = driver.find\_element(By.XPATH, "//a[contains(@class, 'atm\_1y33qqm\_1ggndnn') and contains(text(), 'Siguiente')]"): En esta línea de código, se busca y se selecciona el botón "Siguiente" en la página actual utilizando una expresión XPath. El XPath se utiliza para localizar elementos en el DOM del navegador. En este caso, se busca un elemento <a> (hipervínculo) que contenga ciertas clases y texto específico. Si se encuentra el botón "Siguiente", se almacena en la variable boton\_siguiente.
7. time.sleep(5): Se espera 5 segundos después de hacer clic en el botón "Siguiente" para permitir que la página se cargue completamente antes de continuar.
8. boton\_siguiente.click(): Luego, se hace clic en el botón "Siguiente" para avanzar a la siguiente página de resultados
9. pagina\_actual += 1: Se incrementa la variable pagina\_actual en 1 para realizar un seguimiento de la página actual.
10. print(f"Avanzando a la página {pagina\_actual + 1}"): Se imprime un mensaje que indica a qué página se está avanzando.
11. print("URL actual:", driver.current\_url): Se imprime la URL actual después de cambiar de página.
12. except Exception as e:: En caso de que no se pueda hacer clic en el botón "Siguiente" (por ejemplo, si ya no hay más páginas), se captura una excepción y se imprime un mensaje. Luego, se rompe el bucle con break para salir del bucle while.
13. Después de salir del bucle, se verifica si pagina\_actual es igual a pagina\_deseada. Si son iguales, significa que se ha llegado a la página deseada y se procede a extraer datos de los anuncios en esa página.
14. Se crea una lista vacía llamada anuncios\_pisos\_pag4 para almacenar la información de los anuncios en la página 4.
15. Se utilizan expresiones WebDriverWait para esperar a que los elementos de los anuncios estén presentes en el DOM antes de extraer datos. Estos elementos incluyen títulos, subtitulos, valoraciones, precios por noche, precios totales, entre otros.
16. Se recorren todos los elementos de anuncios en la página deseada utilizando un bucle for, y se extrae información de cada uno de ellos, incluyendo enlaces y atributos relevantes.
17. Se maneja la posibilidad de que no se puedan encontrar algunos elementos y se asigna 'N/A' en caso de que falten datos.
18. Se intenta realizar scraping de características específicas de cada anuncio, similar a lo explicado en bloques de código anteriores.
19. Si se obtienen enlaces y se extrae información de los anuncios, se crea un DataFrame de Pandas (df) con los datos de los anuncios.
20. Si no se obtienen enlaces en la página deseada, se imprime un mensaje indicando que no se realiza scraping.

**En general y en cada botón**

El fragmento de código extracción de elementos página principal:

try: # Realizar clic en el botón Siguiente

WebDriverWait(driver, 10).until(EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, 'c4mnd7m'))) # Esperar hasta que se carguen todos los elementos de anuncios en el DOM (10 segundos)

elementos\_anunciospag4 = driver.find\_elements(By.CLASS\_NAME, 'c4mnd7m') # Encontrar todos los elementos de anuncios en la página deseada

# Esperas para elementos adicionales en el DOM

precios\_noche = WebDriverWait(driver, 30).until(EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, '\_1jo4hgw'))) # Esperar hasta que se carguen todos los elementos de anuncios en el DOM (30 segundos)

subtitulos\_anuncio = WebDriverWait(driver, 30).until(EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, 't6mzqp7')))# Esperar hasta que se carguen todos los elementos de anuncios en el DOM (30 segundos)

valoraciones\_anuncio = WebDriverWait(driver, 30).until(EC.presence\_of\_all\_elements\_located((By.CSS\_SELECTOR, ".r1dxllyb.atm\_7l\_18pqv07.atm\_cp\_1ts48j8.dir.dir-ltr")))# Esperar hasta que se carguen todos los elementos de anuncios en el DOM (30 segundos)

titulos\_anuncio = WebDriverWait(driver, 30).until(EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, 't1jojoys')))# Esperar hasta que se carguen todos los elementos de anuncios en el DOM (30 segundos)

precios\_total = WebDriverWait(driver, 30).until(EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, '\_tt122m')))# Esperar hasta que se carguen todos los elementos de anuncios en el DOM (30 segundos)

for index, anunciopag4 in enumerate(elementos\_anunciospag4): # Recorrer todos los elementos de anuncios en la página deseada. Index es el índice de cada elemento . Enumerate es el índice de cada elemento

enlace\_anuncio = anunciopag4.find\_element(By.TAG\_NAME, 'a') # Encontrar el enlace del anuncio

href\_anuncio = enlace\_anuncio.get\_attribute('href') # Extraer la URL del enlace

target\_anuncio = enlace\_anuncio.get\_attribute('target') # Extraer el atributo target

# Extraer información usando el índice correspondiente

titulo = titulos\_anuncio[index].text if index < len(titulos\_anuncio) else 'N/A' # Extraer el texto de la etiqueta si existe

subtitulo = subtitulos\_anuncio[index].text if index < len(subtitulos\_anuncio) else 'N/A' # Extraer el texto de la etiqueta si existe

valoracion = valoraciones\_anuncio[index].text if index < len(valoraciones\_anuncio) else 'N/A' # Extraer el texto de la etiqueta si existe

precio\_noche = precios\_noche[index].text if index < len(precios\_noche) else 'N/A' # Extraer el texto de la etiqueta si existe

precio\_total = precios\_total[index].text if index < len(precios\_total) else 'N/A' # Extraer el texto de la etiqueta si existe

anuncios\_pisos\_pag4.append({ #append es para agregar un elemento a la lista

'texto': anunciopag4.text,

'titulo': titulo,

'subtitulo': subtitulo,

'valoracion': valoracion,

'precio\_noche': precio\_noche,

'precio\_total': precio\_total,

'href': href\_anuncio,

'target': target\_anuncio

})

**Explicación General**:

1. **Inicialización de la Lista**: Antes de comenzar a extraer información de los anuncios en la página actual, se crea una lista vacía llamada anuncios\_pisos\_pag\_actual. Esta lista se utilizará para almacenar la información de cada anuncio en la página actual.
2. **Espera de Elementos en el DOM**: El código utiliza la función WebDriverWait de Selenium para esperar hasta que ciertos elementos estén presentes en el DOM del navegador. Estos elementos incluyen los anuncios principales y otros elementos relevantes como precios, subtítulos, valoraciones, títulos y precios totales. La espera se establece en un tiempo límite (por ejemplo, 10 o 30 segundos) para asegurarse de que la página haya cargado completamente y todos los elementos necesarios estén disponibles para su procesamiento.

elementos\_anunciospag\_actual = WebDriverWait(driver, 10).until(

EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, 'c4mnd7m'))

)

Además de los elementos principales, se espera a que otros elementos relevantes, como precios por noche, subtítulos, valoraciones, títulos y precio total, estén presentes en el DOM de la página:

precios\_noche = WebDriverWait(driver, 30).until(

EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, '\_1jo4hgw'))

)

subtitulos\_anuncio = WebDriverWait(driver, 30).until(

EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, 't6mzqp7'))

)

valoraciones\_anuncio = WebDriverWait(driver, 30).until(

EC.presence\_of\_all\_elements\_located((By.CSS\_SELECTOR, ".r1dxllyb.atm\_7l\_18pqv07.atm\_cp\_1ts48j8.dir.dir-ltr"))

)

titulos\_anuncio = WebDriverWait(driver, 30).until(

EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, 't1jojoys'))

)

precio\_total = WebDriverWait(driver, 30).until(

EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, '\_tt122m'))

)

Esto asegura que la página haya cargado completamente y todos los elementos necesarios estén disponibles para su procesamiento.

1. Luego, se inicia un bucle for para recorrer todos los elementos de anuncio en la página actual. Esto se hace con la ayuda de enumerate, que proporciona un índice para cada elemento en la lista elementos\_anunciospag\_actual. El índice se usa para acceder a los elementos correspondientes en las listas de precios, subtítulos, valoraciones, títulos y precio total. Aquí se procesa cada anuncio en detalle:

for index, anunciopag\_actual in enumerate(elementos\_anunciospag\_actual):

try:

# Extraer información de cada elemento del anuncio

```

En este bucle for, se recorren todos los elementos de anuncio en la página actual (no necesariamente la página 2, ya que el código generaliza para cualquier página). A continuación, explicaré cada parte del bucle en detalle:

1. for index, anunciopag\_actual in enumerate(elementos\_anunciospag\_actual):: Este bucle for utiliza la función enumerate para recorrer la lista de elementos\_anunciospag\_actual. La función enumerate asigna un índice (index) a cada elemento de la lista y lo almacena en la variable anunciopag\_actual. Esto permite recorrer la lista y, al mismo tiempo, realizar un seguimiento del índice del elemento actual.
2. try:: Se inicia un bloque try, que se utiliza para manejar cualquier excepción que pueda ocurrir durante el procesamiento de un anuncio.
3. # Extraer información de cada elemento del anuncio: Aquí se comentó una línea de código que vendría a continuación. En esta parte, se accede a cada elemento del anuncio (como el título, subtitulo, valoración, etc.) dentro de anunciopag\_actual y se extrae su contenido.
4. El código dentro del bloque try se encarga de extraer información específica de cada elemento del anuncio en la página actual, como el título, el subtítulo, la valoración, el precio por noche, el precio total, el enlace y el target del anuncio.
5. Cada una de estas piezas de información se extrae utilizando métodos y propiedades de los elementos HTML dentro de anunciopag\_actual. Por ejemplo, titulo\_anuncio se extrae del elemento titulos\_anuncio[index], y lo mismo se aplica a las otras variables.
6. Los valores extraídos se almacenan en las variables correspondientes (titulo\_anuncio, subtitulo\_anuncio, valoración\_anuncio, precio\_noche, precio\_total\_texto, href\_anuncio, target\_anuncio).
7. A continuación, se imprimen estos valores en la consola para que el usuario pueda verlos durante la ejecución del programa. Esto puede ser útil para el proceso de depuración y seguimiento del progreso del scraping.
8. Finalmente, todos estos valores se almacenan en un diccionario y se agregan a la lista anuncios\_pisos\_pag\_actual como un elemento del diccionario. El diccionario almacena todos los detalles del anuncio, y la lista anuncios\_pisos\_pag\_actual se usa para recopilar estos diccionarios para cada anuncio en la página actual.
9. titulo\_anuncio = titulos\_anuncio[index].text if index < len(titulos\_anuncio) else 'N/A'
   * + Sirve para extraer el título de un anuncio de una página web y almacenarlo en la variable titulo\_anuncio. Vamos a desglosarla paso a paso:
     + titulos\_anuncio: Esto es una lista que contiene todos los elementos HTML que tienen la clase "t1jojoys" en la página web. En el contexto de tu código, estos elementos son los títulos de los anuncios que estás buscando.
     + index: Es una variable que está siendo utilizada en un bucle for para recorrer los elementos de titulos\_anuncio. En cada iteración, index tomará el valor del índice actual del bucle.
     + titulos\_anuncio[index]: Esto accede al elemento de la lista titulos\_anuncio en la posición dada por index. En otras palabras, obtiene el título del anuncio en la posición actual del bucle.
     + .text: Esto es una propiedad de un elemento HTML en Selenium que te permite obtener el texto contenido dentro de ese elemento. En este caso, se está extrayendo el texto del título del anuncio.
     + if index < len(titulos\_anuncio) else 'N/A': Esta es una expresión condicional que se asegura de que la extracción del título sea segura. Verifica si el valor de index es menor que la longitud de la lista titulos\_anuncio. Si es cierto, significa que hay un título en la posición actual y se asigna ese título a titulo\_anuncio. Si es falso, lo que significa que no se encontró un título en esa posición, se asigna 'N/A' (que significa "No disponible") a titulo\_anuncio.

En resumen, esta línea de código se utiliza para extraer el título de un anuncio de la página web de Airbnb. Si se encuentra un título válido en la posición actual, se almacena en la variable titulo\_anuncio. Si no se encuentra un título válido, se asigna 'N/A' como un valor por defecto para indicar que no se encontró ningún título en esa posición.

En resumen, este bucle for recorre cada elemento de anuncio en la página actual, extrae información relevante de esos elementos y almacena esos datos en una estructura de datos (anuncios\_pisos\_pag\_actual) para su posterior procesamiento o almacenamiento. La utilización de enumerate permite un acceso ordenado a los elementos mientras se realiza un seguimiento de su posición en la lista.

El fragmento de código extracción p de características adicionales de cada anuncio haciendo click en los enlaces:

# Intentar realizar scraping solo si se obtuvieron enlaces

if anuncios\_pisos\_pag4: # Verificación de que se hayan extraído anuncios

try:

for anuncio in anuncios\_pisos\_pag4: # Recorrer todos los anuncios

driver.get(anuncio['href']) # Ir a la página del anuncio extraído

WebDriverWait(driver, 10).until(EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, "lgx66tx.atm\_gi\_idpfg4.atm\_l8\_idpfg4.dir.dir-ltr"))) # Esperar a que se carguen las características

caracteristicas = driver.find\_elements(By.CLASS\_NAME, "lgx66tx.atm\_gi\_idpfg4.atm\_l8\_idpfg4.dir.dir-ltr") # Encontrar todas las características

caracteristicas\_texto = [c.text for c in caracteristicas] # Extraer el texto de las características

print(f"Características para {anuncio['href']}: {caracteristicas\_texto}") # Imprimir las características

anuncio['caracteristicas'] = caracteristicas\_texto # Agregar las características al anuncio

except Exception as e: # En caso de error

print(f"No se pudo realizar scraping de características para {anuncio['href']}: {e}")

Ahora, desglosemos cómo funciona esta parte del código:

1. **Verificación de Enlaces**: Primero, se verifica si la lista anuncios\_pisos\_pag contiene enlaces válidos. Esto se hace con la condición if anuncios\_pisos\_pag:. Si la lista no está vacía (es decir, si se han extraído anuncios), el código procederá a realizar el scraping de características adicionales.
2. **Recorriendo Anuncios y Accediendo a las Páginas de Anuncios**: Se utiliza un bucle for para recorrer cada anuncio en la lista anuncios\_pisos\_pag. Dentro del bucle, se obtiene el enlace (href) de cada anuncio y se navega a la página de ese anuncio utilizando driver.get(anuncio['href']).
3. **Espera de las Características**: Después de cargar la página del anuncio, el código espera a que se carguen las características adicionales. Esto se hace mediante WebDriverWait y la función EC.presence\_of\_all\_elements\_located. El código espera a que los elementos con la clase lgx66tx.atm\_gi\_idpfg4.atm\_l8\_idpfg4.dir.dir-ltr estén presentes en el DOM antes de continuar.
4. **Extracción de Características**: Una vez que los elementos de las características adicionales están presentes, se utilizan driver.find\_elements para encontrar todos estos elementos. Luego, se extrae el texto de cada elemento y se almacena en la lista caracteristicas\_texto. Estos textos representan las características adicionales del anuncio.
5. **Almacenamiento de Características**: Se imprime en la consola el texto de las características y se agrega a la clave caracteristicas del diccionario del anuncio correspondiente en la lista anuncios\_pisos\_pag. Esto permite almacenar las características adicionales junto con la información básica del anuncio.
6. **Manejo de Errores**: Se incluye un bloque except para manejar cualquier excepción que pueda ocurrir durante el scraping de características. Si hay un error, se imprime un mensaje en la consola indicando que no se pudo realizar el scraping de características para ese anuncio específico.

En resumen, esta parte del código se encarga de extraer características adicionales de cada anuncio en la página actual, siempre que haya enlaces válidos en la lista de anuncios. Cada conjunto de características se almacena junto con la información básica del anuncio en la lista anuncios\_pisos\_pag.

## 6. Crear df y guardar en Csv. Formato texto.

# Lista principal para almacenar la información de todos los anuncios

todos\_los\_anuncios = []

todos\_los\_anuncios.extend(anuncios\_pisos\_pag1)

……………………

# Crear un DataFrame a partir de la lista de todos los anuncios

df\_anuncios = pd.DataFrame(todos\_los\_anuncios)

# Limpiar los saltos de línea y comas en las columnas de texto

df\_anuncios['titulo'] = df\_anuncios['titulo'].str.replace('\n', ' ').str.replace(',', '')

df\_anuncios['subtitulo'] = df\_anuncios['subtitulo'].str.replace('\n', ' ').str.replace(',', '')

# Repetir para cualquier otra columna que pueda tener este problema

# Ruta donde deseas guardar el archivo CSV (nombre único con marca de tiempo)

timestamp = datetime.now().strftime("%Y%m%d%H%M%S")

nombre\_archivo\_unico = f'anuncios\_airbnb\_girona\_{timestamp}.csv'

ruta\_guardado = f'/content/drive/MyDrive/Hosting/Hosting/Analisis Scrapping/csv/{nombre\_archivo\_unico}'  # Ajusta esta ruta según sea necesario

# Guardar el DataFrame en un archivo CSV único basado en la marca de tiempo

df\_anuncios.to\_csv(ruta\_guardado, sep='\t', encoding='utf-16', index=False)

print(f"Archivo CSV único con marca de tiempo guardado en: {ruta\_guardado}")

Este bloque de código tiene el propósito de consolidar información de varios conjuntos de datos, limpiar esa información, y luego almacenarla en un archivo CSV. Aquí hay una explicación de cada línea:

* Se crea una lista vacía llamada todos\_los\_anuncios, que almacenará información de múltiples páginas.
* La lista se extiende con los datos de las variables anuncios\_pisos\_pagX, que aparentemente contienen anuncios de diferentes páginas (del 1 al 15) de una fuente como Airbnb.
* Se crea un DataFrame de pandas llamado df\_anuncios a partir de la lista combinada de todos los anuncios.
* Se limpian los títulos y subtítulos de los anuncios en el DataFrame para eliminar saltos de línea y comas, lo que podría afectar el formato del CSV.
* Se genera un nombre de archivo único para el CSV utilizando la fecha y hora actual con un formato específico.
* Se establece la ruta de guardado del archivo CSV, la cual incluye el nombre del archivo único.
* Se guarda el DataFrame en un archivo CSV en la ruta especificada, utilizando una tabulación como separador y la codificación UTF-16.
* Finalmente, se imprime la ruta de guardado para notificar al usuario dónde se ha guardado el archivo CSV.

## 8. Información adicional

Se ha incluido al análisis, los listings que se han descargado a través de la página<http://insideairbnb.com/get-the-data>, teniendo en cuenta que sólo podremos obtener los del año 2023, ya que es la única información pública que se puede obtener. Quedándose por tanto, fuera del análisis el año 2022.

## 9. Conclusiones

Esta sección resume los resultados del proyecto, los desafíos encontrados y las posibles mejoras para futuras tareas de web scraping.

Desafíos:

1. **Página Dinámica:** Airbnb utiliza una página web dinámica, lo que significa que su contenido cambia cada vez que haces clic en ella o interactúas con elementos como botones de carga o paginación.
2. **Límite de Extracción:** Airbnb limita la cantidad de anuncios que puedes extraer en una sola solicitud a aproximadamente 270. Esto hace que sea necesario ejecutar el código varias veces para recopilar una gran cantidad de datos. Además, la información que se puede extraer corresponde al año actual, 2024. Tal y como se ha comentado en punto 8 de este informe, se ha obtenido más información de los listings del último año en la página <http://insideairbnb.com/get-the-data>, quedando el año 2022 sin información en este análisis.
3. **Evolución del Sitio:** Los CSS pueden cambiar con el tiempo debido a actualizaciones del sitio, lo que puede romper el código de scraping existente.
4. **Restricciones de Request:** Airbnb puede bloquear o limitar el acceso a su sitio web cuando se realizan demasiadas solicitudes en un corto período de tiempo. Esto hace que sea necesario utilizar Selenium (driver) en lugar de la biblioteca requests para simular interacciones más humanas y evitar bloqueos.
5. **Manejo de WebDriver y ChromeDriver:**

* **Errores de Inicialización:** Problemas al inicializar el WebDriver de Selenium, específicamente con ChromeDriverManager, incluyendo errores como AttributeError y TypeError. Esto podría deberse a problemas con la versión específica del ChromeDriver o problemas al intentar instalarlo automáticamente.
* **Configuración del Navegador:** Dificultades para configurar las opciones del navegador Chrome, como el modo headless y otros parámetros de inicialización.

1. **Selección de Elementos y Esperas:**

* **Localización de Elementos**: Problemas al encontrar y seleccionar elementos en la página utilizando Selenium, debido a cambios potenciales en la estructura del DOM o en los selectores de clases utilizados.
* **Esperas Implícitas y Explícitas**: Errores al usar WebDriverWait para esperar a que los elementos estén presentes en el DOM, y problemas con elementos obsoletos o referencias estancadas.

1. **Navegación de las páginas ocultas:**
2. **Uso de Try except para evitar que el código se pare.**
3. **Se han creado varios códigos**

En resumen, el web scraping en Airbnb presenta desafíos debido a su naturaleza dinámica y a las limitaciones de extracción impuestas. Para abordar estos desafíos, es importante utilizar herramientas y estrategias adecuadas y mantenerse atento a las posibles actualizaciones del sitio.