

Inicialmente se usa un entorno de Google Colab con un código que instala las bibliotecas necesarias para realizar web scraping con Selenium. Las bibliotecas que se instalan son:

pip: El administrador de paquetes de Python

xml: Una biblioteca para procesar documentos XML

scrapy: Un marco de trabajo para web scraping

requests-html: Una biblioteca para realizar solicitudes HTTP y procesar el HTML

```
▼ Install required libraries

[1] !pip install lxml
    !pip install scrapy
    !pip3 install requests-html
    !pip3 install selenium

Mostrar salida oculta

%%shell
# Install chromedriver
# Credits: https://medium.com/@MinatoNamikaze02/running-selenium-on-google-colab-a118d10ca5f8
sudo apt -y update
sudo apt install -y wget curl unzip
wget http://archive.ubuntu.com/ubuntu/pool/main/libu/libu2f-host/libu2f-udev_1.1.4-1_all.deb
dpkg -i libu2f-udev_1.1.4-1_all.deb
wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
dpkg -i google-chrome-stable_current_amd64.deb

wget -N https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.62/linux64/chromedriver-linux64.zip
unzip -o /tmp/chromedriver-linux64.zip -d /tmp/
chmod +x /tmp/chromedriver-linux64/chromedriver
mv /tmp/chromedriver-linux64/chromedriver /usr/local/bin/chromedriver

pip install selenium chromedriver_autoinstaller

Mostrar salida oculta
```

La siguiente sección de código implementa un proceso de web scraping para obtener datos de ventas de autos usados del sitio web Tu Carro. El código utiliza la biblioteca Selenium para automatizar las interacciones con el navegador web y la biblioteca BeautifulSoup para extraer los datos de las páginas web.

```
▼ Web Scraping Used Car Sales Data

This section explains the web scraping process implemented to obtain the data from the used car sales web site Tu Carro.

!pip installundetected_chromedriver

Mostrar salida oculta

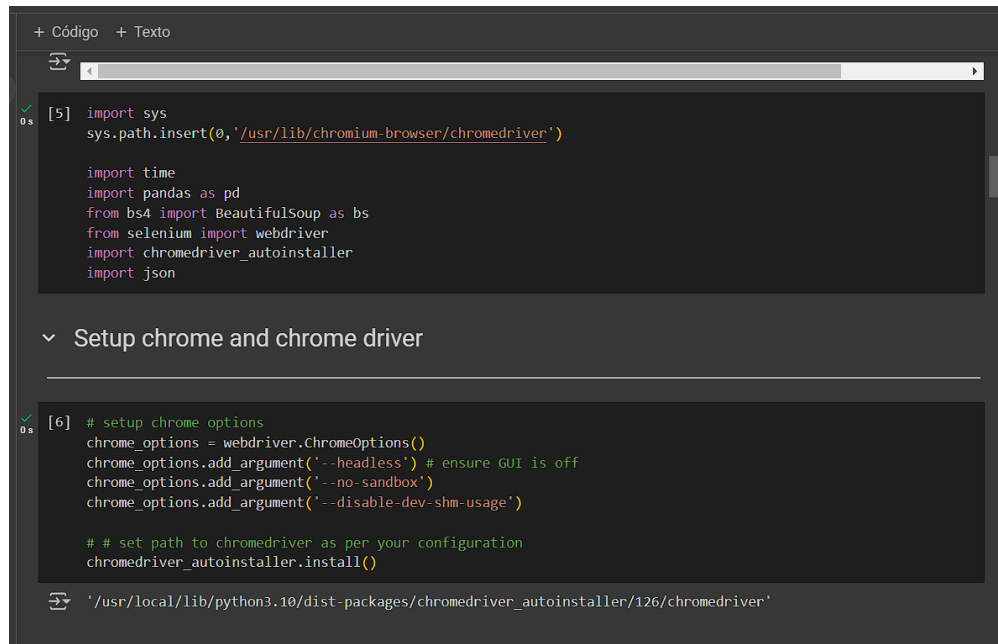
▼ Import required libraries

[4] '''
    credits:
    https://github.com/googlecolab/colabtools/issues/3347
    https://stackoverflow.com/questions/51046454/how-can-we-use-selenium-webdriver-in-colab-research-google-com
    Sept 19, 2023
    '''

    #
    !pip3 install chromedriver-autoinstaller

Requirement already satisfied: chromedriver-autoinstaller in /usr/local/lib/python3.10/dist-packages (0.6.4)
Requirement already satisfied: packaging>=23.1 in /usr/local/lib/python3.10/dist-packages (from chromedriver-autoinstaller)
```

El código de la imagen se utiliza para configurar Chromium y Chrome Driver para su uso en scripts de automatización Python. La configuración incluye la ejecución de Chromium en modo sin cabeza, la desactivación del uso de la GPU y la instalación de Chrome Driver si es necesario.



```
+ Código + Texto

[5] import sys
    sys.path.insert(0, '/usr/lib/chromium-browser/chromedriver')

    import time
    import pandas as pd
    from bs4 import BeautifulSoup as bs
    from selenium import webdriver
    import chromedriver_autoinstaller
    import json

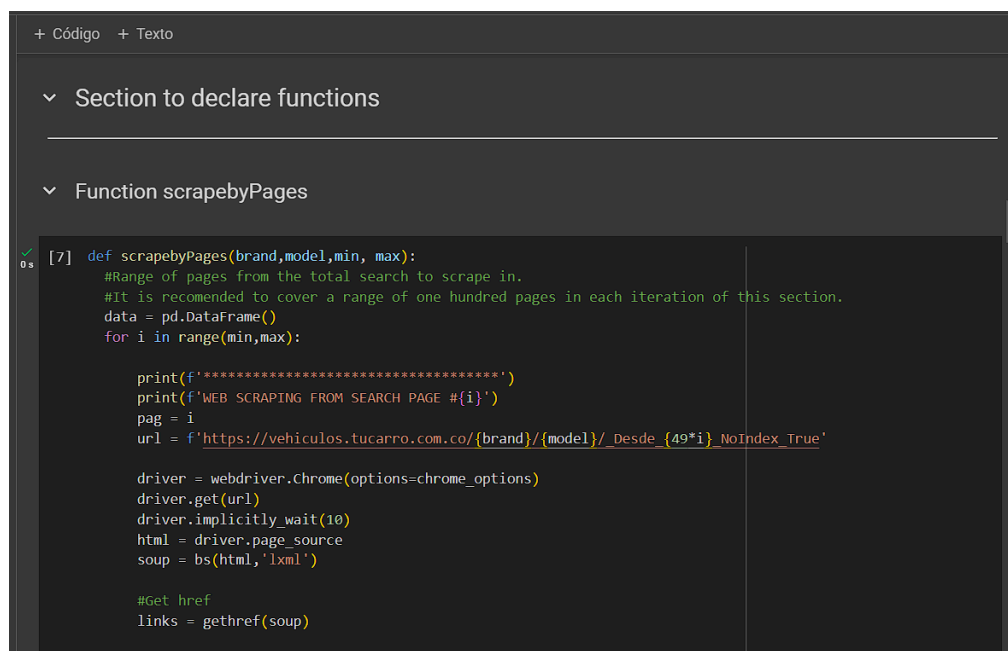
  Setup chrome and chrome driver

[6] # setup chrome options
    chrome_options = webdriver.ChromeOptions()
    chrome_options.add_argument('--headless') # ensure GUI is off
    chrome_options.add_argument('--no-sandbox')
    chrome_options.add_argument('--disable-dev-shm-usage')

    # # set path to chromedriver as per your configuration
    chromedriver_autoinstaller.install()

    '/usr/local/lib/python3.10/dist-packages/chromedriver_autoinstaller/126/chromedriver'
```

El código de la imagen se utiliza para extraer datos de una página web y almacenarlos en un archivo CSV. El código utiliza las bibliotecas BeautifulSoup y Pandas para extraer y procesar los datos.



```
+ Código + Texto

  Section to declare functions

  Function scrapebyPages

[7] def scrapebyPages(brand,model,min, max):
    #Range of pages from the total search to scrape in.
    #It is recommended to cover a range of one hundred pages in each iteration of this section.
    data = pd.DataFrame()
    for i in range(min,max):

        print(f'*****')
        print(f'WEB SCRAPING FROM SEARCH PAGE #{i}')
        pag = i
        url = f'https://vehiculos.tucarro.com.co/{brand}/{model}/ Desde {49*i} NoIndex True'

        driver = webdriver.Chrome(options=chrome_options)
        driver.get(url)
        driver.implicitly_wait(10)
        html = driver.page_source
        soup = bs(html,'lxml')

        #Get href
        links = gethref(soup)
```

El código mostrado en la imagen define dos funciones Python que se pueden utilizar para extraer información de una página web. Las funciones utilizan bibliotecas populares para analizar HTML y navegar por la web.

```

  0s [8] #Function to get 'href' from each article item
      def gethref(soup):

          links = []
          for link in soup.findAll('a'):
              url_car = link.get('href')
              if 'MCO-' in url_car:
                  # print(url_car)           %Print each car url as a validity test
                  links.append(url_car)

          print("Href obtained: ", len(links))

          return links
      # return

  0s [9] #Function to call housing_features routine on each href
      def scrapper(url_car):

          # set up the webdriver
          driver = webdriver.Chrome(options=chrome_options)

          # Scrape

```

El código mostrado en es una herramienta útil para extraer características de automóviles de un archivo CSV. El código es fácil de usar y se puede modificar para adaptarse a diferentes necesidades.

```

  0s [10] # Version 1.0
      def extract_cars_features(soup):

          features_list = []

          # car_name
          try:
              car_name = soup.find('h1',{'class': 'ui-pdp-title'}).text
              features_list.append(car_name)
              # print(f"Car's name is: {car_name}")
          except:
              car_name = ' '
              features_list.append(car_name)

          # price
          try:
              price=soup.find('div',{'class': 'ui-pdp-price__second-line'}).text
              features_list.append(price)
              # print(f"Car's price is: {price}")
          except:
              price = 0
              features_list.append(price)

          # year_car
          try:
              year_car_dataPub = soup.find('div',{'class': 'ui-pdp-header__subtitle'}).text.split(' ')

```

El programa de raspado de datos que se muestra en la imagen parece ser una herramienta efectiva para extraer información sobre automóviles del sitio web tucarro.com de la pagina 1 a la 10 de acuerdo a los parámetros ingresados anteriormente

+ Código

+ Texto

Start scraping

0s

[11] from selenium.webdriver.support.ui import WebDriverWait

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

from selenium.webdriver.common.by import By

The input parameters for the 'scrapebyPages' function are: Brand name, Car model name. Be careful to write the brand and model names exactly as they are in tucarro.com. The third input parameter is the initial results page (always initialize to 1) and the fourth input parameter is the final results page you want to download data from; this parameter depends on the amount of results pages your car returns for the brand and model you want to get data from. So, it is recommended to search the web portal first to find out how many pages of results you can get for the car you want to get data from.

car\_brand = 'chevrolet' # Brand car name. Ej: chevrolet, renault, kia.

car\_model = 'spark' # Model car name. Ej: duster, onix, rio.

data = scrapebyPages(car\_brand,car\_model,1,10)

# scrapebyPages(1,2)

Este es el valor de p[i]: ['Chevrolet Spark 1.2 Gt Ltz', '\$37.900.000', '2019', '80.000', 'Plateado', 'Gasolina']

Scraping 26 / 48 ...

Este es el valor de p[i]: ['Chevrolet Spark 1.2 Gt Lt', '\$41.900.000', '2022', '17.000', 'Gris', 'Gasolina']

Scraping 37 / 48

La imagen muestra una ventana de un navegador web con una lista de archivos. La lista está ordenada por nombre y cada archivo tiene las siguientes columnas:

- **Nombre:** El nombre del archivo.
- **Código Texto:** El tipo de archivo.
- **GamaIA:** Un nombre de usuario o identificador.
- **Moomar salida-coulta:** Un nombre de archivo o directorio.
- **Q:** Una letra que puede ser un indicador de estado o clasificación.

Los archivos de la lista son todos archivos de texto, con la excepción de un archivo que tiene la extensión ".csv". Los archivos parecen estar relacionados con el análisis de datos, ya que algunos de los nombres de archivo incluyen términos como "data.head()" y "used Cars"

Archivos

+ Código + Texto

Mostrar salida oculta

0s

cols = ['car\_model','price','year\_model','kms','color','fueltype']

data.columns = cols

print(data.shape)

data.head()

(384, 6)

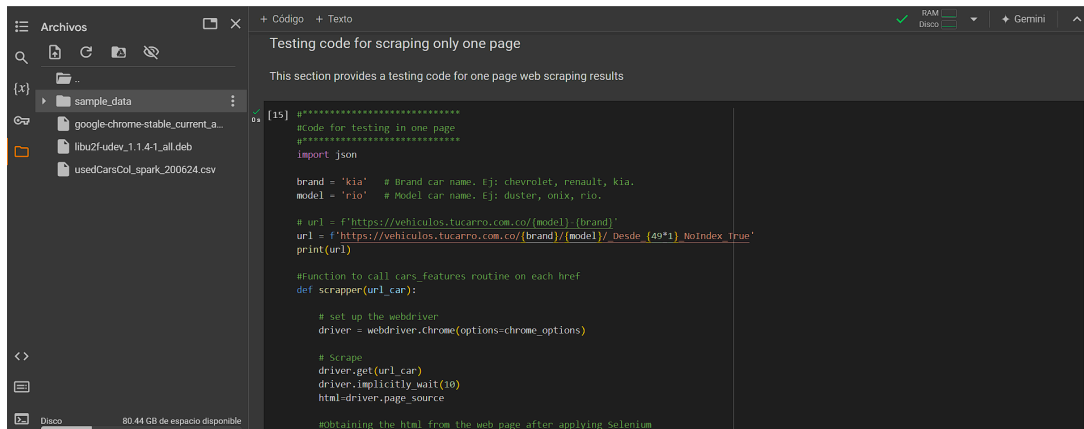
	car_model	price	year_model	kms	color	fueltype
0	Chevrolet Spark Gt Ltz	\$34.500.000	2016	46.100	Gris	Gasolina
1	Chevrolet Spark Gt Ltz 1.2 Mecanico	\$31.000.000	2015	116.000	Blanco	Gasolina
2	Chevrolet Spark GT 1.2 Ltz	\$41.900.000	2021	18.000	Blanco	Gasolina
3	Chevrolet Spark LT LIFE	\$25.500.000	2016	122.807	Rojo	Gasolina
4	Chevrolet Spark 1.2 Gt M300 Ltz Mecánica	\$33.800.000	2017	123.500	Blanco	Gasolina

[14] saved\_name=f'usedCarsCol\_{car\_model}\_200624.csv'

data.to\_csv(saved\_name, encoding='utf-8', index=False)

El código define una función llamada descrappor que toma dos argumentos: url y car. El argumento url es la URL de la página web que se va a raspar y el argumento car es un diccionario que especifica los datos que se deben raspar.

La función `descrappor` comienza obteniendo el código HTML de la página web utilizando la biblioteca `requests`. Luego, analiza el código HTML utilizando la biblioteca `BeautifulSoup` para extraer los datos especificados en el diccionario `car`. Finalmente, la función devuelve un diccionario que contiene los datos extraídos.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like `sample_data`, `google-chrome-stable_current_a...`, `libu2f-udev_1.1.4-1_all.deb`, and `usedCarsCol_spark_200624.csv`. The code editor has a tab titled "Testing code for scraping only one page" and contains the following Python code:

```
[15] *****
#Code for testing in one page
*****
import json

brand = 'kia' # Brand car name. Ej: chevrolet, renault, kia.
model = 'rio' # Model car name. Ej: duster, onix, rio.

# url = f'https://vehiculos.tucarro.com.co/{model}/{brand}'
url = f'https://vehiculos.tucarro.com.co/{brand}/{model}/Desde_{49*1}_NoIndex_True'
print(url)

#function to call cars_features routine on each href
def scrapper(url_car):

    # set up the webdriver
    driver = webdriver.Chrome(options=chrome_options)

    # scrape
    driver.get(url_car)
    driver.implicitly_wait(10)
    html=driver.page_source

    #obtaining the html from the web page after applying Selenium
```

el anterior proceso se realizó en 2 días distintos para realizar un comparativo de la información extraída en días distintos, en el día 4 de julio se descargaron 433 registros en formato csv y de los cuales 2 de registros tuvieron como resultado “,0,0,0,,”, el día 6 de julio se extrajeron 385 registros en formato csv y de los cuales 2 de registros tuvieron como resultado “,0,0,0,,”

Conclusión:.

Es importante tener en cuenta que el raspado de datos puede ser una actividad ilegal o no ética, dependiendo de los términos de servicio del sitio web de destino. Es importante verificar los términos de servicio antes de raspar datos de cualquier sitio web.