Práctica 1: Web Scraping

Dataset:

Coches de segunda mano en venta en España

Pablo Campillo Sánchez

Pedro Uceda Martínez

Tipología y Ciclo de Vida de los Datos

Universitat Oberta de Catalunya

1. Contexto

A la hora de abordar esta actividad, nuestro objetivo era realizar un *dataset* cuyo posterior análisis tuviera utilidad en nuestro día a día. Así pues, se ha optado por recoger datos sobre **coches en venta** en nuestro país, de modo que pudiéramos extraer conocimiento posteriormente realizando un análisis sobre el conjunto de datos que extrajésemos. El conjunto de datos tendría infinidad de **utilidades**:

- **Segmentar** los vehículos muy parecidos y dada su distribución de precios determinar si un coche es barato, caro o sobre la media.
- **Implementación de un tasador de coches**, que dadas sus características prediga un precio.
- **Un reconocedor** de modelos de **coche** a partir de la **imagen**.

En nuestro país existen principalmente 2 sitios web que permiten buscar coches en venta: www.coches.net y www.milanuncios.com. Además, en principio, ambos sitios **nos permiten** scrapear los anuncios publicados en los mismos según lo indicado en sus ficheros ***robots.txt***.

En un primer momento, intentamos extraer los datos de **coches.net**, pero encontramos muchísimas **dificultades técnicas**, puesto que, incluso utilizando Selenium, el sitio web identificaba que éramos un robot y nos "expulsaba".

Así pues, optamos por obtener los datos del sitio web **milanuncios**, ya que, aún teniendo un volumen de datos algo menor, para cada anuncio posee una información muy similar y nos permite realizar **scraping** utilizando **Selenium**. El uso del mismo es necesario ya que debido a la manera en la que la web carga los listados, nos vemos obligados a hacer *scroll* sobre las páginas. En caso contrario obtendremos, sólo dos valores por página. Además, **milanuncios es una de las web más visitadas de España**[1].

2. Título

Dado que el conjunto de datos obtenido contiene información sobre anuncios de venta de coches de segunda mano en el territorio español,el título de dicho *dataset* es "**spanish_used_car_market**".

3. Descripción

Como hemos comentado, a través de esta práctica generamos un conjunto de datos o *dataset* conformado por **coches de segunda mano** que se encuentran **en venta** a lo largo del **territorio español**. Entre los datos relevantes se encuentran la marca y modelo de los vehículos, el precio de venta, la provincia donde se venden o una url a la imagen.

4. Representación gráfica

En el centro de la Figura 1 podemos observar la imagen de un coche en venta y alrededor de la misma se muestran las características que ofrece nuestro *dataset* para cada anuncio de coche.

1 Sitios web más visitados en España: https://www.webempresa20.com/blog/webs-mas-visitadas-en-espana.html

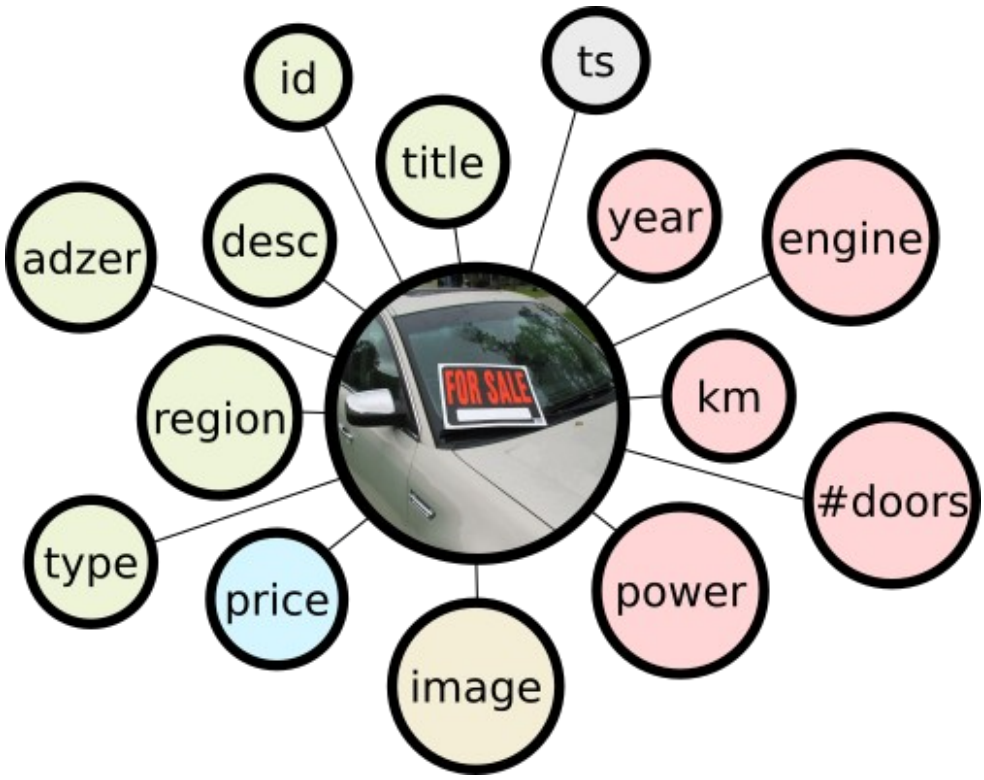

Figura 1: Representación gráfica de las características recopiladas por cada anuncio de coche.

5. Contenido

Para cada vehículo en venta, se recogen las siguientes características:

- **ad_id**: Identificador del anuncio del coche.
- **ad_type**: Tipo de anuncio. En nuestro caso, siempre va a ser Oferta.
- **ad_time**: Tiempo que llevaba publicado el anuncio cuando se recogió la información, en formato X horas o X días. En el caso en que fuese un anuncio destacado, no tenemos esa información.
- **ad_title**: Título del anuncio de venta, con formato {Marca} – {Modelo}.
- **car_desc**: *Preview* de la descripción del anuncio de venta del vehículo.
- **car_km**: Kilómetros que tiene recorridos el coche.
- **car_year**: Año de matriculación del vehículo.
- **car_engine_type**: Tipo de transmisión. Posibles valores: Manual | Automático.
- **car_door_num**: Número de puertas de las que dispone el coche.
- **car_power**: Potencia del vehículo, en formato XXX CV.
- **car_price**: Precio en euros por el que se vende el coche.

- **advertizer_type**: Indica cuál es el tipo de vendedor del vehículo. Valores posibles: Profesional | Particular.
- **image_url**: Foto principal del anuncio de venta del coche.
- **ts**: Hora en la que se recogió la información, con formato YYYY-MM-DD hh:mm:ss.ms.
- **region**: Provincia en la que se está vendiendo el vehículo.

La información relativa a cada vehículo en venta se ha obtenido listando, para cada provincia, los anuncios ordenados de más recientes a más antiguos. Sólo hemos tenido en cuenta los anuncios de ofertas, no de demanda.

Nótese que los datos se han obtenido durante el trascurso de 3 días, en concreto, desde los día 3 al 5 de noviembre de 2020, de manera semiautomática: Se han ido recogiendo los datos provincia por provincia, revisando el proceso por si hubiera algún fallo inesperado.

6. Agradecimientos

Como se ha indicado anteriormente, los datos se han recogido del sitio web www.milanuncios.es, que es el propietario de los mismos y el sitio web de anuncios clasificados más popular en España, creado en el año 2005 .

Dado que en el fichero robots.txt no se prohíbe expresamente el *scrapeo* de los listados de los anuncios, hemos podido extraer los datos implementando técnicas de *Web Scraping* mediante el uso del lenguaje de programación Python, su librería Beautiful Soup y automatizando el proceso con Selenium, motivado principalmente el uso de este último por la necesidad de realizar scroll.

7. Inspiración

Uno de los aspectos que más nos ha motivado al realizar este proyecto es el poder conocer el estado del mercado, es decir, dado un anuncio concreto de un coche, si su precio está en torno a la media por debajo o por encima.

En general, este *dataset* es útil para desarrollar una herramienta que nos ayude a encontrar posibles oportunidades de coches en venta y a la vez nos permite aprender en el proceso de cómo realizar *web scraping*. Además, es un dominio conocido: Si nosotros mismos no nos hemos visto en la tesitura de comprar un coche, seguro que nuestros padres o algún conocido sí.

En el apartado en el que se situaba en contexto el conjunto de datos se presentaban posibles aplicaciones del mismo. Relacionado con aquellas, podríamos hacernos las siguientes preguntas en torno al *dataset*:

- Partiendo de un presupuesto determinado y una marca de coche deseada, ¿qué rango de km o años de antigüedad debería esperar, de media, que tenga el vehículo que puedo adquirir?
- Voy a vender mi coche porque me voy a comprar otro nuevo. Si tiene 5 años y 120.000 km, ¿Cuánto dinero podría pedir por él?.
- ¿Cuál es el modelo de este vehículo que aparece en la foto?

8. Licencia

Se selecciona la licencia **CC BY-NC-SA 4.0 License**[2] por los siguientes motivos:

- **Autoría**: Se pueden utilizar, producir, o reproducir los datos siempre y cuando se reconozca a los creadores de la obra y, en el caso en que se realizasen cambios, se ha de describir cuáles son aquellos que se han realizado. Así, obtendríamos el reconocimiento por hacer realizado este trabajo de recopilación.
- **No se permite el uso comercial.** Dado que no tenemos el consentimiento expreso del propietario de los datos originales para realizar la extracción de los mismos, preferimos que no se pueda utilizar el material, **de manera total ni parcial**, **para obtener un beneficio monetario o una ventaja competitiva** por parte de posibles competidores de www.milanuncios.com.
- **Licencia de trabajos derivados:** Al compartir el material de manera total o parcial, o utilizarlo, el trabajo derivado debe tener la misma licencia, o bien una compatible de la siguiente lista: https://creativecommons.org/compatiblelicenses. Así, el trabajo se distribuirá con los mismas condiciones que las que hemos decidido.

9. Código

El código se compone principalmente de dos ficheros:

- **bin/scrap**: Comando para lanzar la herramienta desde consola. Se basa en la librería para desarrollar comandos **click**.
- **brand_car_scraper/main.py**: Contiene toda la lógica del parserador repartido en 3 clases:
  - **MilanunciosScraper**: gestiona todo el flujo de ir parseando páginas y guardándolas en el fichero de salida tipo csv.
  - **SeleniumBrowser**: Clase que gestiona la obtención de todo el contenido de una página aceptando las cookies y haciendo scroll para que se cargue todo el contenido.
  - **MilAnunciosPageParser**: Dato el html de una página, obtiene todas las características de todos los anuncios de la página en una lista de diccionarios.

Las **librerías** que se han utilizado principalmente son **selenium** y **beautifulsoup4**.

El código fuente se puede encontrar en el repositorio:
https://github.com/pablo-campillo/brand_car_scraper y en Anexo: Código.

10. Zenodo

Se ha subido el fichero .CSV obtenido como fruto del desarrollo de esta práctica a https://zenodo.org/record/4252636#.X6W8IHX0lhF.

2 Puede consultarse en https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode

El **DOI obtenido** es **10.5281/zenodo.4252636**.

11. Contribuciones

A continuación se presentan las contribuciones de los integrantes por cada bloque en el que podemos en el que podemos dividir la práctica:

Contribuciones	Firma
Investigación previa: idea	P.C.S., P.U.M.
Estudio viabilidad	P.C.S., P.U.M.
Desarrollo del código	P.C.S., P.U.M.
Generación del dataset	P.C.S., P.U.M.
Redacción del documento	P.C.S., P.U.M.
Subida a Zenodo	P.C.S., P.U.M.

Anexo: Código

bin/scrap

```
#!/usr/bin/env python3

import click
from brand_car_scraper.main import car_scraper


@click.command()
@click.argument('output_file', type=click.Path(exists=False))
@click.option('--fp', type=click.INT, default=1, help='First page to be parsed. Default value is
1.')
@click.option('--tp', type=click.INT, default=None, help='Last page to be parsed, i.e. until the
last page.')
@click.option('--region', type=click.STRING, default="madrid", help='Region to be parsed e.g.
madrid.')
def scrap(output_file, fp=1, tp=None, region="madrid"):
    """This tool scraps list of second car ads at https://www.coches.net/segunda-mano/?
pg=<page_number> where
    <page_number is an integer greater or equal than 1.

    If no OPTIONS are provided all the pages will be parsed.

    OUTPUT_FILE a path where the csv file will be stored.

    \b
    Examples:
        - For scraping from page 1 to the end, region madrid:
            $ scrap out.csv
        - For scraping from page 1 to 10, region madrid:
            $ scrap --tp 10 out.csv
        - For scraping from page 5 to 10, region madrid:
            $ scrap -fp 5 -tp 10 out.csv
        - For scraping from page 5, region madrid:
            $ scrap -fp 5 out.csv
        - For scraping from page 5 to 10, region murcia:
            $ scrap -fp 5 -tp 10 --region murcia out.csv

    \b
    AUTHORS:
        - Pedro Uceda Martinez\b
        - Pablo Campillo Sánchez
    """
    click.secho(f"car_scraper({output_file}, {fp}, {tp}, {region})", fg='green')
    car_scraper(output_file, fp, tp, region)


if __name__ == '__main__':
    scrap()
```

brand_car_scraper/main.py

```
from bs4 import BeautifulSoup
import time
import pandas as pd
import click

from uuid import uuid4
from datetime import datetime

from selenium import webdriver
from selenium.webdriver.chrome.options import Options
```

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.common.keys import Keys
from selenium.webdriver import ActionChains
from pathlib import Path


def car_scraper(output_file, fp, tp, region):
    milanuncios_scraper = MilanunciosScraper()
    milanuncios_scraper.scrap(output_file, fp, tp, region)


class MilanunciosScraper:
    """ Class to scrap vehicle ads from https://www.milanuncios.com/coches-de-segunda-mano-en-
{region}/?fromSearch={page}&orden=date
    Where:
        - province is a string validated by the function validate_region()
        - page is the number of page result to be requested
    """

    def __init__(self, executable_path="brand_car_scraper/chromedriver",
log_path="chromedriver.log"):
        self._response_delay = 0.5
        self._executable_path = executable_path

    def scrap(self, output_file, fp, tp, region):
        output_file_path = Path(output_file)

        self.current_page = 1 if not fp else fp
        self.max_page = tp

        if not self._validate_region(region):
            click.secho(f"Not valid region name error: {region}", fg='red')
            click.secho(f"Valid region names: {self.regions}", fg='red')
            return

        dataset = pd.DataFrame()

        while True:
            try:
                self.parse_pages(dataset, output_file_path, region)
                break
            except TimeoutException as e:
                click.secho(f"Browser Timeout Exception!", fg="red")
                continue
        dataset.to_csv(output_file)

    def parse_pages(self, dataset, output_file_path, region):
        with SeleniumBrowser() as sb:
            for page_number in self._pages_to_scrap():
                click.echo(page_number)
                page_content = sb.read(self._get_url(region, page_number))

                if page_content:
                    parser = MilAnunciosPageParser(page_content)
                    self._update_max_page(parser)

                    records = parser.get_records()
                    if len(records) > 0:
                        df = pd.DataFrame(records)
                        df['region'] = region
                        df.to_csv(output_file_path.parent / f"{page_number:03d}-
{output_file_path.name}")
                        dataset = pd.concat([dataset, pd.DataFrame(records)])
                        dataset['region'] = region

    def _get_url(self, region, page_number):
```

```
        return f"https://www.milanuncios.com/coches-de-segunda-mano-en-{region}/?
pagina={page_number}&orden=date"

    def _pages_to_scrap(self):
        while True:
            if self.max_page and self.current_page > self.max_page:
                break
            yield self.current_page
            self.current_page += 1

    def _update_max_page(self, page_parser):
        if not self.max_page and page_parser:
            self.max_page = page_parser.get_total_number_of_pages()
            click.secho(f"Number of pages: {self.max_page}", fg="green")

    def _validate_region(self, region):
        return region.lower() in self.regions

    @property
    def regions(self):
        return [
            'alava', 'albacete', 'alicante', 'almeria', 'andalucia',
            'asturias', 'avila', 'badajoz', 'baleares', 'barcelona', 'burgos',
            'caceres', 'cadiz', 'cantabria', 'canarias', 'castellon',
            'castilla_la_mancha', 'castilla_y_leon', 'catalunya', 'ceuta',
            'ciudad_real', 'cordoba', 'cuenca', 'extremadura', 'galicia',
            'girona', 'granada', 'guadalajara', 'guipuzcoa', 'huelva',
            'huesca', 'jaen', 'la_coruna', 'la_rioja', 'las_palmas', 'leon',
            'lleida', 'lugo', 'madrid', 'malaga', 'melilla', 'murcia', 'navarra',
            'ourense', 'pais_vasco', 'palencia', 'pontevedra', 'salamanca',
            'segovia', 'sevilla', 'soria', 'tarragona', 'tenerife', 'teruel',
            'toledo', 'valencia', 'comunidad_valenciana', 'valladolid', 'vizcaya',
            'zamora', 'zaragoza'
        ]


class MilAnunciosPageParser:
    def __init__(self, page_content):
        self.soup = BeautifulSoup(page_content, 'html.parser')

    def get_records(self):
        return self._extract_all_cars_data(self.soup.find_all('article', {'class': 'ma-AdCard'}))

    def get_total_number_of_pages(self):
        result = None
        div = self.soup.find('div', 'ma-NavigationPagination-pagesContainer')
        if div:
            pages = div.find_all('span', 'ma-ButtonBasic-content')
            if pages:
                result = int(pages[-1].text)
                click.secho(f"Total number of paged found: {result}", fg='green')
        return result

    def _extract_all_cars_data(self, articles: list) -> list:
        result = []
        for article in articles:
            car_record = self._extract_cars_record(article)
            if car_record is None:
                continue
            result.append(car_record)
        return result

    def _extract_cars_record(self, article) -> dict:
        ad_type = self._get_text(article, 'p', 'ma-AdCard-sellType', default=None)

        if ad_type.upper() != "OFERTA":
            return None
```

```
        ad_id = self._get_text(article, 'p', 'ma-AdCard-adId', default=None)
        ad_time = self._get_text(article, 'p', 'ma-AdCard-time', default=None)

        ad_title = self._get_text(article, 'h3', 'ma-AdCard-bodyTitle', default=None)
        car_desc = self._get_text(article, 'p', 'ma-AdCard-text', default=None)

        fields = [p.text for p in article.find('ul', 'ma-AdTagList').find_all('span', 'ma-AdTag-
label')]
        if len(fields) != 5:
            return None
        car_km, car_year, car_engine_type, car_door_num, car_power = fields

        price_section = article.find('div', 'ma-AdCard-metadataActions')
        car_price = self._get_text(price_section, 'span', 'ma-AdCard-price', default=None)
        if car_price:
            car_price = car_price.replace('.', '').replace('€', '')
        advertizer_type = self._get_text(price_section, 'span', 'ma-AdTag-label', default=None)

        image_url = article.find('img', 'ma-AdCard-photo').get('src') if article.find('img', 'ma-
AdCard-photo') else ""

        return {
            'ad_id': ad_id,
            'ad_type': ad_type,
            'ad_time': ad_time,
            'ad_title': ad_title,
            'car_desc': car_desc,
            'car_km': car_km,
            'car_year': car_year,
            'car_engine_type': car_engine_type,
            'car_door_num': car_engine_type,
            'car_power': car_power,
            'car_price': car_price,
            'advertizer_type': advertizer_type,
            'image_url': image_url,
            'ts': datetime.utcnow(),
        }

    def _get_text(self, article, name, attrs, default=None):
        result = article.find(name, attrs)
        return result.text if result else default


class SeleniumBrowser:
    MAX_SCROLLS = 100

    def __init__(self, load_page_timeout=5, scroll_pause_time=1):
        self.load_page_timeout = load_page_timeout
        self.scroll_pause_time = scroll_pause_time

    def __enter__(self):
        self.session = uuid4()
        click.secho(f"Starting chrome session", fg='green')

        options = Options()
        options.add_argument('start-maximized')
        options.add_argument('disable-infobars')
        options.add_argument('--disable-extensions')

        self.browser = webdriver.Chrome(chrome_options=options)
        return self

    def __exit__(self, type, value, traceback):
        click.secho(f"Closing chrome session", fg='green')
        self.browser.quit()
```

```
    def read(self, url):

        self.browser.get(url)
        try:
            WebDriverWait(self.browser, self.load_page_timeout).until(lambda d:
d.find_element_by_tag_name("article"))
        except TimeoutException:
            click.secho(f"Timed out waiting for page to load: {url}", fg="red")
            return None

        scroll_counter = 1
        while scroll_counter < self.MAX_SCROLLS:
            ActionChains(self.browser).send_keys(Keys.PAGE_DOWN).perform()
            time.sleep(self.scroll_pause_time)
            self._accept_cookies()
            try:
                next_page_element = self.browser.find_element_by_class_name("ma-
NavigationPagination-nextButton")
                break
            except:
                continue

        return self.browser.page_source

    def _accept_cookies(self):
        try:
            buttons = self.browser.find_elements_by_class_name("sui-AtomButton--primary")
        except:
            return
        if buttons:
            buttons[0].click()
```