Practica 1 Tipología y ciclo de vida de los datos

Realizada por:

Daria Gracheva
Zechao Jin

Enlace al repositorio:

https://github.com/zechao/scraperpiso

1. Contexto

Explicar en qué contexto se ha recolectado la información. Explique por qué el sitio web elegido proporciona dicha información.

En este proyecto decidimos investigar los precios de alquiler de Barcelona y recopilar datos relativos de anuncios de alquiler, como el precio, características del piso, tanto numéricas (número de habitaciones, planta, superficie, etc), como cualitativas binarias y textuales –la descripción del anuncio.

El interés para crear el dicho dataset es analizar los patrones sobre alquiler en la Ciudad Condal, y potencialmente responder a preguntas como "¿Dónde puede ser más rentable buscar un piso de alquiler?" o encontrar otros patrones.

La página elegida para el scraping es **habitaclia** (www.habitaclia.com) puesto que proporciona tanto la posibilidad de hacer el web scraping legalmente como la variedad de registros para enriquecimiento del dataset. Cada registro contiene la información clave para la construcción del dataset y su posterior análisis, por lo que la consideramos una buena fuente de información.

2. Definir un título para el dataset.

Las características de anuncios de alquiler en la ciudad de Barcelona de plataforma habitaclia.

3. Descripción del dataset.

Desarrollar una descripción breve del conjunto de datos que se ha extraído (es necesario que esta descripción tenga sentido con el título elegido).

El dataset contiene los datos numéricos, binarios y textuales más relevantes recopilados de cada uno de los anuncios de una ciudad dada, en este caso Barcelona: Precio, Titulo de anuncio, Superficie, Número de habitaciones, Número de baños, Barrio, Descripción completa, Amueblamiento (sí/no), Calefacción (sí/no), Aire acondicionado (sí/no), Ascensor (sí/no), Detalles de distribución y Características adicionales. Aunque los datos sobre calefacción, aire acondicionado, etc, también se recopilan dentro de la variable “features_detail” para tener el dato consistente y separar las características de más interés en las variables independientes directamente. Al momento de realización de scraping hay alrededor de 13.000 anuncios.

Los datos en muchos casos contienen la normalización al formato de la página e incluyen las palabras explicativas, por lo que algunos de los datos numéricos en dataset están representados en strings ya que se recopilan directamente del tag correspondiente, y están pensados para ser preprocesados durante los procesos de limpieza.

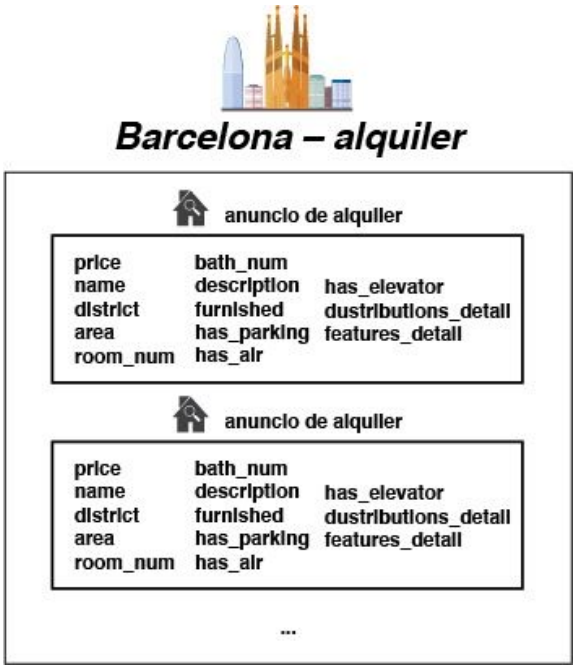
Otros datos son directamente numéricos por la metodología de recogida por scraper (sobre todo, en caso de siblings), y otros datos son de tipo booleano (True/False/None).

El dataset final presenta varios valores perdidos, ya que no todas las características están rellenadas en los anuncios –en este caso el valor de la celda es None.

El formato de fichero final es csv, que facilita el trabajo posterior de limpieza y análisis.

4. Representación gráfica

Presentar una imagen o esquema que identifique el dataset visualmente



5. Contenido

Explicar los campos que incluye el dataset, el periodo de tiempo de los datos y cómo se ha recogido.

Los datos se recogen con un web scraper escrito en lenguaje Python y obtiene los campos para cada anuncio actualmente publicado para la ciudad de Barcelona (apr. 13000 anuncios).

Para ello, primero se accede a la búsqueda por ciudad, donde se recorre por todas las páginas resultantes, y desde cada url correspondiente a un anuncio se extraen las características individuales.

Las características extraídas son:

- **price** - precio mensual del alquiler
- **name** - título del anuncio
- **district** - barrio donde se encuentra la vivienda
- **area** - superficie de la vivienda
- **room_num** - número de habitaciones
- **bath_num** - número de baños
- **description** - descripción proporcionada por el anunciante
- **furnished** - amueblamiento del piso
- **has_parking** - si hay plaza de aparcamiento incluida en el alquiler
- **has_air** - si hay aire acondicionado
- **has_elevator** - si la comunidad dispone de un ascensor
- **feature_detail**: Información adicional sobre la característica del piso que puede afectar el precio, como por ejemplo, el piso que está, el año de construcción, si tiene buena vista, etc.
- **distributions_detail**: Información adicional sobre la distribución de la casa que podría afectar el precio, como por ejemplo,sí tiene lavadero, sí tiene terraza,el estado de la cocina, etc.

De esta manera, se crea un dataset sobre una única ciudad.

Finalmente, se guardan los datos en un fichero en formato csv llamado dataset.csv.

6. Agradecimientos

Presentar al propietario del conjunto de datos. Es necesario incluir citas de investigación o análisis anteriores (si los hay).

La página web pertenece al grupo Adventia. Adevinta Spain es una gran familia de marketplaces con páginas como Fotocasa, coches.net, etc. Por lo tanto es una compañía tech de referencia en nuestro mercado.

Esta página es una de las pocas páginas con los anuncios de alquiler que permite (o mas bien no prohíbe) hacer web scraping según el contenido de su fichero robots.txt. Cada anuncio, naturalmente, contiene los datos introducidos por los usuarios y normalizados según el diseño de la página.

En cuanto a los proyectos similares (fase investigación y análisis), no hay proyectos de un alcance parecido publicados abiertamente, sino otros tipos de analiticas, por ejemplo sobre alquileres turísticos o anuncios de venta, como se presenta a continuación:

https://www.hlaboa.com/post/Compra_tu_casa_de_forma_inteligente_1_web_scraping/

7. Inspiración

Explique por qué es interesante este conjunto de datos y qué preguntas se pretenden responder.

El dataset se inspiró en la idea de poder determinar y analizar los patrones en alquileres de larga duración (no turísticos) en Barcelona. No solo pretendemos perfilar los alquileres de la ciudad por sus barrios, sino también intentar encontrar patrones que puedan ser explicados dentro de este conjunto de datos. Consideramos que puede resultar interesante cara a la búsqueda de un piso en Barcelona, sobre todo para la gente de fuera.

Las preguntas que se pretendan responder, entre otras, son:

- ¿Dónde puede ser más rentable buscar un piso de alquiler?
- ¿Cuales son los precios medios de alquiler en Barcelona y cual es la dispersión en los precios?
- ¿Hay correlaciones entre algunas características y precios de alquiler?
- ¿Qué zonas son las más representadas en el mercado de alquiler?

8. Licencia

Seleccione una de estas licencias para su dataset y explique el motivo de su selección:

- Released Under CC0: Public Domain License
- Released Under CC BY-NC-SA 4.0 License
- Released Under CC BY-SA 4.0 License
- Database released under Open Database License, individual contents under Database Contents License
- Other (specified above)
- Unknown License

La licencia seleccionada para el dataset es Creative Commons (CC) Attribution 4.0 y precisamente la licencia CC BY-SA. Esta licencia combina los componentes de open access bajo el concepto Creative Commons (CC) y la protección necesaria / reconocimiento de trabajo según las cláusulas BY (Atribución) y SA (Compartir igual). Además, permite el uso comercial que puede generar valor añadido para el scraper y aumentar su potencial usabilidad en el marco empresarial.

9. Código

Adjuntar el código con el que se ha generado el dataset, preferiblemente en Python o, alternativamente, en R.

El código fue escrito en el lenguaje Python con el uso de librerías `BeautifulSoup` y `requests`, además, se introduce el uso de multihilo para acelerar la ejecución, hay un hilo que se encarga de recuperar el url de cada anuncio, y varios hilos resuelven los urls, al final un hilo escribe los resultados en el archivo csv.

Aunque el scraper está diseñado para poder recuperar otra ciudad cambiando la variable city_name = 'barcelona' en el main, también se puede recuperar datos de pisos en venta cambiando search_type=sale en el main.

El fichero con el codigo completo `main.py` se encuentra en el repositorio con el enlace https://github.com/zechao/scraperpiso/blob/main/main.py y el codigo se inlcuye a continuacion.

10. Dataset

Publicación del dataset en formato CSV en Zenodo (obtención del DOI) con una breve descripción.

El dataset resultante en formato CSV se ha publicado en Zonde con el DOI:"**10.5281/zenodo.4139894**" en el siguiente enlace:https://zenodo.org/record/4139894

Contribuciones	Firma
Investigación previa	D.G.; Z.J.
Redacción de las respuestas	D.G.; Z.J.
Desarrollo código	D.G.; Z.J.

Código completo:

```
import requests
from bs4 import BeautifulSoup
import csv
import sys
import os
import re
from multiprocessing import Lock
import queue
import threading

headers = {
    "accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
bp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
    "accept-encoding": "gzip, deflate, br",
    "accept-language": "es",
    "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36"
}

# field names of our csv file.
variables = ['price',  # Mandatory
             'district',  # Mandatory
             'area',  # Mandatory
             'room_num',  # Mandatory
             'bath_num',  # Mandatory
             'furnished',  # Important info that affect the price
             'has_parking',  # Important info that affect the price
             'has_elevator',  # Important info that affect the price
             'has_air',  # Important info that affect the price
             # Contain detail information of the distribution of the
rooms, kitchen, etc.
             'distributions_detail',
             'features_detail',  # detail info contains featureas of the
floor which affect the price
             'name',  # contains useful information
```

```
            'description'  # contains a description written by the owner
which contains useful information
            ]

# you can search rent or sale informaticon
rent = 'alquiler'
sale = 'viviendas'


bf4parser = 'lxml'


def valid_url(url):
    """some url are invalid because the data are not belong to the
original web, they belong to the partner web."""
    if re.match(r"^https:.*?.com/f[a|v]\d+$", url):
        return False
    return True


def request_page_number(search_type, city_name):
    """given the search_type and city_name return the number of
pagination"""
    try:
        url = 'https://www.habitaclia.com/{}-{}.htm'.format(
            search_type, city_name)
        page = requests.get(url, headers=headers)
        soup = BeautifulSoup(page.text, features=bf4parser)
        aside = soup.find(id='js-nav')
        li_next = aside.find('li', {'class': 'next'})
        if li_next == None:
            return 1
        else:
            max_page_text =
li_next.previous_element.find_previous_sibling(
                'li').text.strip('\n')
            if max_page_text.isdigit():
                return int(max_page_text)
```

```
            else:
                raise Exception(
                    'incorrect page number:[{}]'.format(max_page_text))
    except Exception as e:
        print('Unknow error while getting max page number:', str(e))
        sys.exit(0)


def build_page_url(search_type, city_name, page_idx):
    url = 'https://www.habitaclia.com/{}-{}{}.htm'.format(search_type,
city_name,
                                                          '' if page_idx
== 0 else '-'+str(page_idx))
    return url


def requests_pages(search_type, city_name, page_idx=None):
    """request url of each pagination"""
    try:
        # first page without index, the rest we need concat the page index
        page = requests.get(build_page_url(search_type,
                                           city_name, page_idx),
headers=headers)
        soup = BeautifulSoup(page.text, features=bf4parser)
        section = soup.find('section', {'class': 'list-items'})
        all_articles = section.find_all('article')

        result = []
        # only articles with data-href
        for article in all_articles:
            if 'data-href' in article.attrs and
valid_url(article['data-href']):
                result.append(article['data-href'])

        return result
    except Exception as e:
        print('Unknow error while requesting pages:', str(e))
```

```
def true_false_none(true_str, false_str, *search_texts):
    """search in the text if contain some string and return TRUE,FALSE or
None"""
    result = None
    for text in search_texts:
        if true_str in text.lower():
            result = True
        if false_str in text.lower():
            result = False
    return result


def get_features(detail_container):
    """get list of feature of each floor"""
    features = []
    general_feature_detail = detail_container.find(
        'h3', string='Características generales')

    if general_feature_detail != None:
        ul_node = general_feature_detail.find_next('ul')

        feature_list = ul_node.find_all('li', attrs={'class': None})

        for each in feature_list:
            text = each.string.strip()
            features.append(text)

    community_equipment = detail_container.find(
        'h3', string='Equipamiento comunitario')

    if community_equipment != None:
        ul_node = community_equipment.find_next('ul')
        equipment_list = ul_node.find_all('li')
        for each in equipment_list:
            text = each.string.strip()
            features.append(text)
```

```
    return features


def get_distribution(detail_container):
    """get list of distribution of each floor"""
    distribution = []
    distribution_detail = detail_container.find(
        'h3', string='Distribución')
    if distribution_detail != None:
        ul_node = distribution_detail.find_next('ul')
        distribution_list = ul_node.find_all('li')

        for each in distribution_list:
            text = each.text.strip()
            distribution.append(text)

    return distribution


def resquest_each_page(url):
    page = requests.get(url, headers=headers)
    return page.text


def resolve_each_page(text):
    """the main function of this scraper resolve the page and return the
result"""
    result = None
    soup = BeautifulSoup(text, features=bf4parser)
    summary = soup.find('div', {'class': 'summary-left'})
    price = summary.find('div', {'class': 'price'}).find(
        'span', {'class': 'font-2'}).string
    name = summary.h1.text.replace('\n', '.').replace('\r', '.')

    if summary.find(id='js-ver-mapa-zona') == None:
        return None

    district = summary.find(id='js-ver-mapa-zona').string.strip()
```

```
    feature_container = summary.find(
        'ul', {'class': 'feature-container'}).find_all('li')

    area = None
    roomNum = None
    bathNum = None

    for feature in feature_container:
        if 'm2' in feature.text and '€/m2' not in feature.text:
            area = re.findall('[0-9]+', feature.text)[0]
        if 'hab.' in feature.text:
            roomNum = re.findall('[0-9]+', feature.text)[0]
        if 'baño' in feature.text:
            bathNum = re.findall('[0-9]+', feature.text)[0]

    detail_container = soup.find('section', {'class': 'detail'})

    description =
'{}.{}'.format(detail_container.find(id='js-detail-description-title').tex
t,

detail_container.find(id='js-detail-description').text)
    description = description.replace('\r', '.').replace('\n', '.')
    features = get_features(detail_container)
    distributions = get_distribution(detail_container)

    # The following variables can be TRUE,FALSE or None
    furnished = None
    has_parking = None
    has_air = None
    has_elevator = None

    for text in features:
        text_lower = text.lower()
        if 'plaza parking' in text_lower:
            has_parking = true_false_none(
                'plaza parking', 'sin plaza parking', text)
```

```
        if 'amueblado' in text_lower or 'sin amueblar' in text_lower:
            furnished = true_false_none('amueblado', 'sin amueblar', text)
        if 'aire acondicionado' in text_lower:
            has_air = true_false_none(
                'aire acondicionado', 'sin aire acondicionado', text)
        if 'ascensor' in text_lower:
            has_elevator = true_false_none('ascensor', 'sin ascensor',
text)

    features_detail = "%;%".join(features)
    distributions_detail = "%;%".join(distributions)
    result = {
        'price': price,
        'district': district,
        'area': area,
        'room_num': roomNum,
        'bath_num': bathNum,
        'furnished': furnished,
        'has_parking': has_parking,
        'has_elevator': has_elevator,
        'has_air': has_air,
        'distributions_detail': distributions_detail,
        'features_detail': features_detail,
        'name': name,
        'description': description
    }
    return result


def get_pages_url_worker(max_page_number, resolve_threads_number,
search_type, city_name, pages_url_queue):
    """worker that get url from each page"""
    count = 0
    for page_idx in range(max_page_number):
        pages = requests_pages(search_type, city_name, page_idx)
        for page in pages:
            count = count + 1
            pages_url_queue.put([count, page])
```

```
    for _ in range(resolve_threads_number):
        pages_url_queue.put('stop')


def page_resolve_worker(pages_url_queue,  result_queue, print_lock):
    """worker that resolve each page_url in the pages_url_queue and store
result in result_queue"""
    while True:
        data = pages_url_queue.get()
        if data == 'stop':
            break

        cout, page_url = data
        with print_lock:
            print('Resolving Page:{},Count:{}, URL:{},'.format(
                cout//16+1, cout, page_url))
        try:
            html_text = resquest_each_page(page_url)
            result = resolve_each_page(html_text)

            if result == None:
                with print_lock:
                    print('ERROR!!!!NOT ENOUGH
DATA!!!:{},\n'.format(page_url))
            else:
                result_queue.put(result)
        except Exception as e:
            with print_lock:
                print('UNEXPECTED EROOR:[{}]!!:{}\n'.format(str(e),
page_url))
        finally:
            pages_url_queue.task_done()
    result_queue.put('stop')


def write_file_worker(writer, file_lock, result_queue,
resolve_threads_number):
    """worker that store date in csv file"""
```

```
    done_count = 0
    while True:
        try:
            result = result_queue.get()
            if result == 'stop':
                done_count = done_count+1
                if resolve_threads_number == done_count:
                    break
                else:
                    continue
        except Exception:
            print("No more element to write in file")
            break
        with file_lock:
            writer.writerow(result)
        result_queue.task_done()


def main(city_name, search_type):

    # Get max page to resolve
    max_page_number = request_page_number(search_type, city_name)
    print('Max page number is:[{}], estimate url to resolve:[{}]'.format(
        max_page_number, max_page_number*15))

    # lock to avoid race condition
    file_lock = Lock()
    print_lock = Lock()
    # Queue for store the page result to store in the csv file
    result_queue = queue.Queue()
    # Queue for store pages that need be resolved
    pages_url_queue = queue.Queue(max(os.cpu_count()*20, 10))

    csvfile = open('dataset.csv', 'w', newline='', encoding='utf-8')
    writer = csv.DictWriter(csvfile, fieldnames=variables)
    writer.writeheader()

    resolve_threads_number = min(15, os.cpu_count()*4)
```

```
    resolve_threads_list = []
    # Thread that get pages
    get_pages_thread = threading.Thread(target=get_pages_url_worker,
args=(
        max_page_number, resolve_threads_number, search_type, city_name,
pages_url_queue), name='Thread get page worker')
    get_pages_thread.start()

    # Threads that store data in csv file
    write_file_thread = threading.Thread(
        target=write_file_worker, args=(writer, file_lock, result_queue,
resolve_threads_number))
    write_file_thread.start()

    threads = []
    # Threads that resolve all pages
    for x in range(resolve_threads_number):
        t = threading.Thread(target=page_resolve_worker, args=(
            pages_url_queue,  result_queue, print_lock, ), name='Thread
resolver '+str(x))
        threads.append(t)
        t.start()
        resolve_threads_list.append(t)

    # block until all tasks are done
    pages_url_queue.join()
    result_queue.join()
    get_pages_thread.join()
    for t in threads:
        t.join()
    write_file_thread.join()


def test_page(page_url):
    html_text = resquest_each_page(page_url)
    result = resolve_each_page(html_text)
    print(result)
```

```
if __name__ == "__main__":
    # you can search rent or sale informaticon by setting rent or sale
    city_name = 'barcelona'
    search_type = rent
    main(city_name, search_type)
```