EUROPEAN UNION

Co-financed by the European Regional Development Fund

Inspire Policy Making with Territorial Evidence

FINAL REPORT //

Cross-border housing markets in Europe

Phyton Scripts

Annex No. 8: Technical Annex 2 // July 2022

This Final Report is conducted within the framework of the ESPON 2020 Cooperation Programme, partly financed by the European Regional Development Fund.

The ESPON EGTC is the Single Beneficiary of the ESPON 2020 Cooperation Programme. The Single Operation within the programme is implemented by the ESPON EGTC and co-financed by the European Regional Development Fund, the EU Member States and the Partner States, Iceland, Liechtenstein, Norway and Switzerland.

This delivery does not necessarily reflect the opinions of members of the ESPON 2020 Monitoring Committee.

Authors
Franziska Sielker – University of Cambridge, Department of Land Economy and Cambridge Centre for Planning and Housing Research (UK) and TU Wien (AT)

Brian Longobardi – University of Cambridge, Department of Land Economy (UK)

Efrain Larrea – MCRIT (ES)

Andreu Ulied – MCRIT (ES)

Luis Falcón Martínez de Marañón – inAtlas (ES)

Silvia Banchini – inAtlas (ES)

Elisabet Palma Gibert – inAtlas (ES)

Technical Support
Nati Franco – MCRIT (ES)

Alessandra Cappai – InAtlas (ES)

Mauri Arévalo Amaya – inAtlas (ES)

Advisory group
Marjan Van Herwijnen (project expert, ESPON EGTC)

Caroline Clause (financial expert, ESPON EGTC)

Information on ESPON and its projects can be found at www.espon.eu.
The website provides the possibility to download and examine the most recent documents produced by finalised and ongoing ESPON projects.

ISBN: 978-2-919816-50-7

Published in Luxembourg

Graphic design by BGRAPHIC, Denmark

Contact: info@espon.eu

Co-financed by the European Regional Development Fund

Inspire Policy Making with Territorial Evidence

FINAL REPORT //

Cross-border housing markets in Europe

Phyton Scripts

Annex No. 8: Technical Annex 2 // July 2022

Table of contents

List of Figures

1 Introduction

A major methodology used within this project was the use of web scraping techniques for data harvesting. This Annex presents an exemplary script to showcase the work that was undertaken as part of the project. In total, the project developed ten scripts for each website that was scrapped. In order to share the code for future studies, all scripts of the crawlers that have been created for this project are available in a GITHUB public repository accessible at https://github.com/inatlasorg/ESPON-BIG-DATA.

As stated in the Technical Annex to the ESPON Report, real estate websites of eleven different countries were scraped in order to obtain the required data for the project. In some countries one website was sufficient to obtain data for rent and for sale although in others, more than one portal needed to be scraped. In the case of Nestoria, as it was a common portal in some countries, the scraping code could be reused with little adjustments in each domain, resulting in ten different python scripts.

A customized script was designed for each site in order to gather the data with specific code depending on the structure of the portal. For example, in order to delimitate the geographical scope of the scraping, two ways were used: one for those portals which were allowed to be defined by a quadrant specified by coordinates and one for those which required the names of the regions containing the case study area. Moreover, as mentioned in the Technical Annex, in some case studies the coordinates of the listings could not be scraped and a geocoding process was needed to enrich the coordinates using the addresses.

A summary of the websites used and its scraping characteristics are detailed below:

CASE STUDY	COUNTRY	WEBSITE / DOMAIN	GEOGRAPHICAL SCOPE DEFINITION	GEOCODING PROCESS
Geneva-Annecy	France	https://www.nestoria.fr/	Quadrant	No
Geneva-Annecy	Switzerland	https://www.homegate.ch/	Regions	No
Northern Ireland – Ireland	Northern Ireland	https://www.nestoria.co.uk/	Quadrant	No
Northern Ireland – Ireland	Ireland	https://www.daft.ie/	Quadrant	No
Denmark – Sweden	Denmark	https://www.boligsiden.dk/ https://www.boliga.dk/	Quadrant Quadrant	No No
Denmark – Sweden	Sweden	https://www.blocket.se/ https://www.hemnet.se/	Regions Regions	No Yes
Slovakia – Austria	Slovakia	https://www.topreality.sk/	Regions	Yes
Slovakia – Austria	Austria	https://www.topreality.sk/ https://www.nestoria.at/	Regions Quadrant	Yes No
Romania-Bulgaria	Romania	https://www.publi24.ro/	Regions	No
Romania-Bulgaria	Bulgaria	https://en.realestates.bg/	Regions	No
France-Spain	France	https://www.nestoria.fr/	Quadrant	No
France-Spain	Spain	https://www.nestoria.es/	Quadrant	No

Figure 1: Websites scraped

2 Python Script Sample

2.1 Script workflow

Despite having all the scripts in a code repository, a sample of a script of one of the platforms (www.daft.ie) is detailed below with the aim of describing the workflow of a crawling process.

The whole code is organized in 5 different scripts:

- Main_Spyder.py
- Quadrants.py
- Get_Listings.py
- Helper.py
- Postgres.py

These 5 scripts are related among them to achieve the scraping steps defined in the Technical Annex:

- STEP 1: Defining the geographical scope
- STEP 2: Downloading URL
- STEP 3: Downloading content
- STEP 4: Downloading latitude and longitude

The following diagram shows the scraping workflow and the relationship between the 5 scripts and the 4 scraping steps:

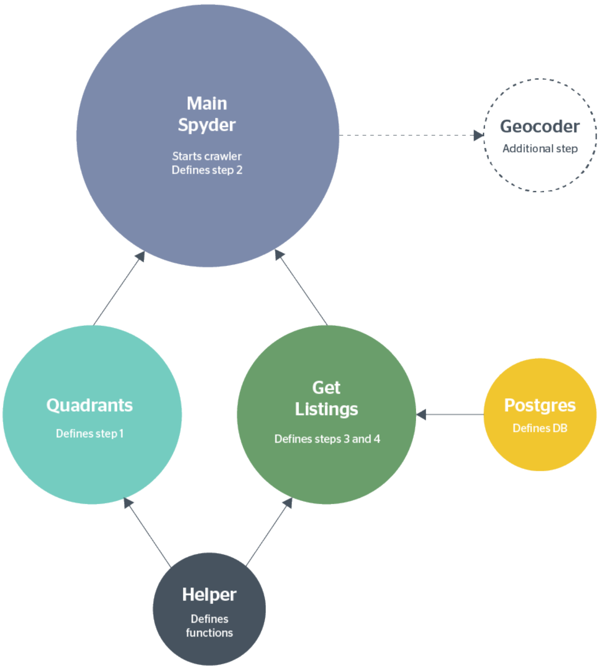

Figure 2: Script Workflow

As it can be observed in the diagram, the MAIN_SPYDER is the script which starts the crawler to achieve the 4 steps of the scraping. It defines step 2 inside it, but it calls QUADRANTS script which defines step 1 and GET_LISTINGS script which defines steps 3 and 4. GET_LISTINGS also requires POSTGRES script which defines de DB to store the data. Moreover, GET_LISTINGS and QUADRANTS scripts call the HELPER script which defines some functions that are being used in both of them.

Finally, as mentioned before, in those cases where the script was not able to gather latitude and longitude data, an additional step was performed using a geocoding process.

2.2 Main_spyder.py

```
from daft.request import quadrants
from daft.request.headers import daft_h
from daft.helper import helper
from daft.logger.formatter import MyFormatter
from daft.selectors import get_listings
from daft.DB import postgresql


import scrapy
from scrapy.http import Response
import simplejson as json
import logging
import sys
import datetime


class MainSpiderSpider(scrapy.Spider):
    """
    This is Spider Object that gets called when executing the command 'scrapy crawl'.
    At creation time we set some important variables/constants needed for the proper download process.
    """
    name = 'main_spider'
    single_type = None
    quadrant = None
    search_code = datetime.date.today().strftime('%y%m%d')
    fmt = MyFormatter()
    handler = logging.StreamHandler(sys.stdout)
    handler.setFormatter(fmt)
    logging.root.addHandler(handler)
    psqObject = postgresql.PostgresObject()


    def start_requests(self) -> scrapy.Request:
        """
        This method is called by default at starting the crawler (you don't need to explicitly call it). It expects
        some URLs to start requesting.
        """


        # >> Here it basically goes through the types to be downloaded, either specified by command line or the default
        #    ones
        for type_ in ([self.single_type] if self.single_type else ['residential-for-sale', 'residential-to-rent']):
            logging.log(8, f'STARTING TYPE -> {type_}')

            # >> Next, for every time, it goes to the actual quadrants to be
downloaded, again specified by command
            # line or by default.
            for quadrant_ in helper.get_quadrants(self.quadrant,
quadrants.quadrants):
                logging.log(8, f'Crawler set to -> {quadrant_}')

                # >> Here it sends the request to the URL of the quadrant
                yield scrapy.Request(
                    'https://gateway.daft.ie/old/v1/listings',
                    method='POST',
                    callback=self.parse,
                    headers=daft_h.headers,
                    body=json.dumps(helper.format_body(quadrant_, type_)),
                    meta={'type_': type_, 'quadrant': quadrant_, 'study_case':
'ireland'}
                )

    def parse(self, response: Response, **kwargs):
        """
         This is the callback method for when receives the server response. It
expects to load a JSON object (delivered)
         from the API. Counts the total number of items as so does for the total
pagination needed.

         Processes and stores the first 'X' (x = the items delivered at page 0)
items and starts pagination.
        """
        big_data = json.loads(response.body.decode('utf-8'))
        total_count = big_data['paging']['totalResults']
        laps = big_data['paging']['totalPages']
        limit = big_data['paging']['pageSize']
        quadrant = response.meta['quadrant']
        type_ = response.meta['type_']
        logging.log(1, f'>> Starting selected zone with a total of
{total_count}')
        logging.log(5, f'Total Laps of {laps}')
        logging.log(5, 'N° 0')

        # >> Processes the first 'X' items, by calling the specified method
        get_listings.get_listings_data(response, psq=self.psqObject,
sc=self.search_code)

        # >> Starts pagination
        offset = limit
        for lap in range(1, laps):
            response.meta['cookiejar'] = helper.get_cookie()
            response.meta['lap'] = lap
            yield scrapy.Request(
                'https://gateway.daft.ie/old/v1/listings',
                method='POST',
                callback=self.parse_page,
                headers=daft_h.headers,
                body=json.dumps(helper.format_body(quadrant, type_, offset)),
                meta=response.meta,
                cb_kwargs={'psq': self.psqObject, 'sc': self.search_code}
            )
            offset += limit

    def parse_page(self, response: Response, **kwargs):
        """
        Final internal method that forwards the response to a selector function
that extracts all data needed from
        every single item returned, and stores it into the DB
        """
        get_listings.get_listings_data(response, **kwargs)
```

2.3 Quadrants.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""moduleName.py Module

Module where you specify all the quadrants you want for the crawler to download
(you can also provide them via
command line arg, but THIS IS NOT RECOMMENDED. ONLY FOR DEVELOPMENT).

When typing quadrants, follow next convention:
    » Insert on a list every single point separated by comma. I.e. [ NE_LNG,
NE_LAT, SW_LNG ,SW_LAT ]
Created by: Davis Yoel Armas Ayala
"""

quadrants = [
    # NE_LNG, NE_LAT, SW_LNG ,SW_LAT
    #  ['53.333125188661114',  '-6.196114053045818',  '53.31126377750158',  '-
6.295070946962028'],
    ['53.18042442', '-7.5260941', '54.25922611', '-5.97115239'],
]
```

2.4 Get_listings.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""moduleName.py Module

Explanation goes here.

Created by: Davis Yoel Armas Ayala
"""
from daft.helper import helper
from scrapy.http import Response
import simplejson as json
import logging
from daft.DB.postgresql import PostgresObject
from colorama import Fore, Style



def get_listings_data(response: Response, **kwargs):
    """
    Function that goes for every item returned by the response API, extracts all
the data needed, and stores the data
    into a DB as a new entry.
    """
    psqObject: PostgresObject = kwargs.get('psq', None)
    search_code = kwargs.get('sc', 000000)
    data = json.loads(response.body.decode('utf-8'))
    if "lap" in response.meta:
        logging.log(5, f'N° {response.meta["lap"]}')

    # >> Goes for every item returned
    for listing_data in data['listings']:
        # >> From here, it extracts the data
        listing = listing_data['listing']

        listing_id = listing['id']

        listing_title = listing['title']
        listing_price = listing['price']
        if listing_price:
            listing_price = listing_price\
                .replace('€', '').replace(',', '').replace('AMV:',
'').replace(' ', '').replace('From', '')\
                .replace('permonth', '').replace('perweek', '')
            try:
                listing_price = int(listing_price)
            except ValueError as _:
                logging.log(4, f'Error while converting price. Actual value ->
{listing_price}')
                listing_price = 0

        listing_bedrooms = listing.get('numBedrooms', None)
        if listing_bedrooms:
            listing_bedrooms = helper.get_beds_baths(listing_bedrooms)

        listing_bathrooms = listing['numBathrooms'] if 'numBathrooms' in listing
else '0'
        if listing_bathrooms:
            listing_bathrooms = helper.get_beds_baths(listing_bathrooms)

        listing_prop_type = listing['propertyType']

        listing_adv_code = listing['seller']['sellerId']

        listing_adv_title = listing['seller']['name']

        listing_lng, listing_lat = listing['point']['coordinates']

        listing_url = 'https://www.daft.ie' + listing['seoFriendlyPath']

        listing_area = 0
        if 'floorArea' in listing:
            listing_area = listing['floorArea']['value']
        elif 'propertySize' in listing:
            listing_area = listing['propertySize'].replace('m²', '').replace('
', '')

        # >> And here it stores into the DB
        psqObject.insert_data([
            listing_url, listing_title, listing_adv_title, listing_adv_code,
listing_lng, listing_lat,
            response.meta['type_'], listing_id, listing_prop_type,
listing_price, listing_bedrooms, listing_bathrooms,
            search_code, listing_area, 'daft', 'ireland'
        ])

    logging.log(6, Fore.LIGHTGREEN_EX + f">> {len(data['listings'])} Items
processed" + Style.RESET_ALL + '\n')
```

2.5 Helper.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""moduleName.py Module

Explanation goes here.

Created by: Davis Yoel Armas Ayala
"""

from daft.request import daft_body
import random
import logging
import re


def get_quadrants(input_quadrant: str or None, default_quadrants: [[str]]) ->
[[str]]:
    if input_quadrant:
        return [input_quadrant.replace(' ', '').split(',')]

    return default_quadrants


def format_body(quadrant: [str], type_: str, offset: int = 0) -> daft_body.body:
    """
    Function that formats a raw/generic dictionary on provided data, which is
gonna end up being the body of an
    outgoing request
    """
    body = daft_body.body

    body['section'] = type_
    body['paging']['from'] = str(offset)
    body['geoFilter']['top'] = quadrant[0]
    body['geoFilter']['right'] = quadrant[1]
    body['geoFilter']['bottom'] = quadrant[2]
    body['geoFilter']['left'] = quadrant[3]
    body['geoFilter']['section'] = type_

    if offset > 0:
        body['geoFilter']['showMap'] = "false"
        body['geoFilter']['mapView'] = "false"

    return body


def get_cookie() -> int:
    """ Function that returns a random aggregate number to be use as a cookie
tracker by Scrapy """
    return random.randint(0, 20) + random.randint(0, 20) + random.randint(0, 20)
+ random.randint(0, 20)


def get_beds_baths(text: str) -> int or float:
    aux = text.replace('Bed', '').replace('Bath', '').replace(' ',
'').replace('bed', '')

    if '.' in aux:
        try:
            aux = float(aux)
        except ValueError as _:
            logging.log(4, f'Error while converting baths/beds to float. Current
value -> {aux}')
            aux = 0
    elif '&' in aux or ',' in aux:
        aux = get_complicated_beds(aux)
    else:
        try:
            aux = int(aux)
        except ValueError as _:
            logging.log(4, f'Error while converting baths/beds to float. Current
value -> {aux}')
            aux = 0

    return aux


def get_complicated_beds(bedrooms: str):
    aux = re.findall(r'(\d+)', bedrooms)

    if aux:
        try:
            add_up = sum(list(map(lambda x: int(x), aux)))
            return add_up
        except ValueError:
            logging.log(4, f'Error while converting baths/beds to float. Current
value -> {aux}')
            return 0
    return 0
```

2.6 Postgres.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""moduleName.py Module

Explanation goes here.

Created by: Davis Yoel Armas Ayala
"""

import psycopg2
import logging
from colorama import Fore, Back, Style
import os

err = Back.RED + Fore.GREEN + Style.BRIGHT + "[BD {} ERROR]" + Style.RESET_ALL
+ ": {}"


class PostgresObject(object):
    conn = cursor = ""

    def __init__(self):
        """ DataBase Connection """
        self.conn = psycopg2.connect(
            user=os.getenv('DB_USER'),
            password=os.getenv('DB_PWD'),
            host=os.getenv('DB_HOST'),
            port=os.getenv('DB_PORT'),
            database=os.getenv('DB_DB'),
        )
        self.conn.autocommit = True
        self.cursor = self.conn.cursor()

    def insert_data(self, data: list):
        """
        Insert on DataBase Table (The schema always presented)
        """

        try:
            # >> Executes the query for INSERT
            self.cursor.execute(
                f"""INSERT INTO espon.espon (listing_url, listing_title,
advertiser, advertiser_code, lng,lat,
                listing_type, listing_id, listing_prop_type, listing_price,
listing_rooms, listing_bathrooms,
                search_code, listing_area, listing_crawled_platform,
listing_study_case)
                VALUES ({("%s, " * 16)[:-2]})""", data)
            self.conn.commit()
        except BaseException as e:
            logging.log(4, err.format("INSERT", e))
            logging.exception(e)
            logging.log(4, Fore.WHITE + Style.BRIGHT + ">> With URL -> " +
Style.RESET_ALL + "{}".format(data[0]))
            self.conn.rollback()
```

EUROPEAN UNION

Co-financed by the European Regional Development Fund

Inspire Policy Making with Territorial Evidence

espon.eu

ESPON 2020

ESPON EGTC
4 rue Erasme, L-1468 Luxembourg
Grand Duchy of Luxembourg
Phone: +352 20 600 280
Email: info@espon.eu
www.espon.eu

The Single Operation within the programme is implemented by the ESPON EGTC and co-financed by the European Regional Development Fund, the EU Member States, the United Kingdom and the Partner States, Iceland, Liechtenstein, Norway and Switzerland.

Disclaimer
This delivery does not necessarily reflect the opinion of the members of the ESPON 2020 Monitoring Committee.