SCRAPPING PRODUCT DATA FROM WEBSITE E-ROBERTO.EU

2022-05

Project: 2400-DS1WSMS Web scraping and Social Media Scraping

Instructor: Maciej Wysocki

Students:
Nhat Tung Le (426246)
Khon Nguyen (444135)
Adnan Sevinç (437971)

Contents

Abstract

This project was made to complete the Web scraping and Social Media Scraping course. In this study, the authors use the knowledge we have learned in the course to experimentally scrape data from the website e-rubeto.eu - a eCommerce website in Poland. The experiment will use all three tools: Beautiful Soup, Scrapy and Selenium. Product data after scraping will be saved as tables on SQLite database. After the experiments are done successfully, we conclude that each tool has its own pros and cons. Depending on the priority at specific moments that developer can choose the appropriate tool. For the data collected, they can be utilized to improve the decision making process of the business owners in order to have the appropriate pricing strategy and furthermore win market shares on the market.

1 Introduction

1.1 Web-scrapping

For the field of data science, data sources are very important for analysts. Usually data will be collected through surveys, or collected manually. With the development of the global Internet, there are many sources of data on websites, especially e-commerce websites or social networks. To collect data from this giant, we cannot do it manually, web scrapping techniques and many tools that assist researchers in collecting this data.

Web scraping, web harvesting, or web data extraction is data scraping used for extracting data from websites. The web scraping software may directly access the World Wide Web using the Hypertext Transfer Protocol or a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis [8].

1.2 Website e-roberto.eu

E-roberto has many years of experience in wholesale footwear, they run an online store and a stationery store, they give their customers access to a wide range of men's, women's and children's footwear. For many years they have gathered observations and studied the market in detail, so that, by offering online shoe wholesalers, they are able to offer their customers corresponding products. with trending trends, attractive designs and colors [3].

In this project, we try to analyzes E-roberto's online store to scrape its data, the website can be accessed at the link: https://e-roberto.eu.

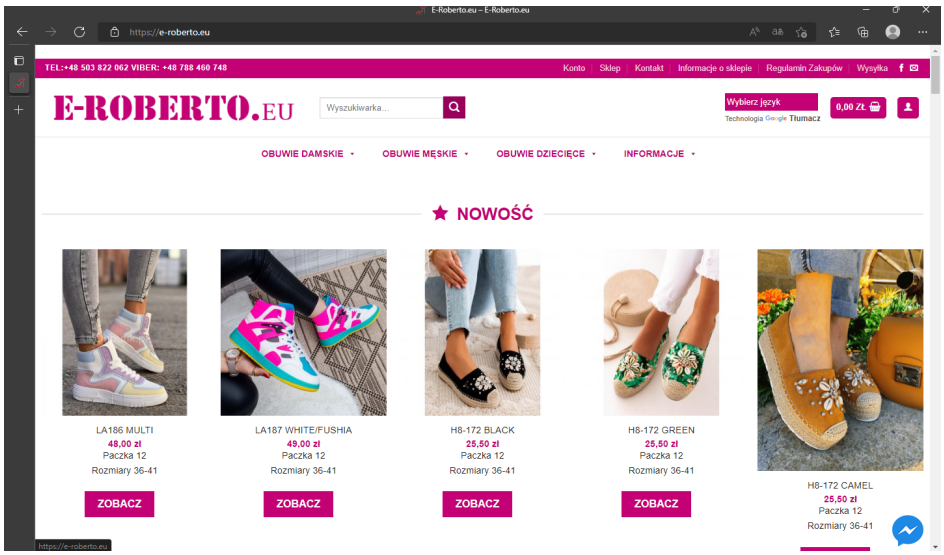


Figure 1: roberto.eu website

- The basic parameters of the website:
  - The online store is organized according to the dynamic web model.
  - URL: e-roberto.eu
  - Access protocol: Hypertext Transfer Protocol Secure (https)
  - CMS Framework: Wordpress
  - Robots.txt: The information is hidden and cannot be extracted from robots.txt
  - The website structure consists of many web pages with the purpose of introducing the store and displaying products, providing users with an interactive ability to order online. The two most important websites are the product listing page and the product detail page.
- Product list page: is a place to display a list of products, usually aggregated by product category, and divided into many pages. We can access the product list page through the main menu of the website. The link structure of a product listing page is as follows:
  - URL structure: https://e-roberto.eu/product-category/**CATEGORY-ID**/page/**PAGE-NUMBER**/
  - Example: https://e-roberto.eu/product-category/obuwie-damskie/page/2/

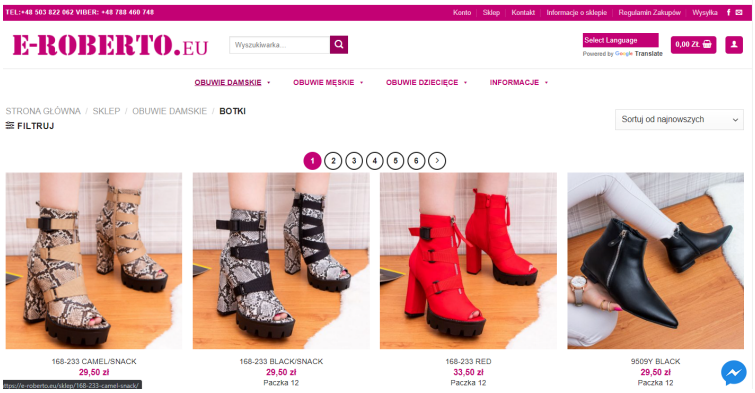


Figure 2: Product list page

- Product detail page: This is the place to display detailed information of a product. The link structure of a page is as follows:
  - URL structure: https://e-roberto.eu/sklep/**PRODUCT-ID**/
  - Example: https://e-roberto.eu/sklep/tu133-grey/

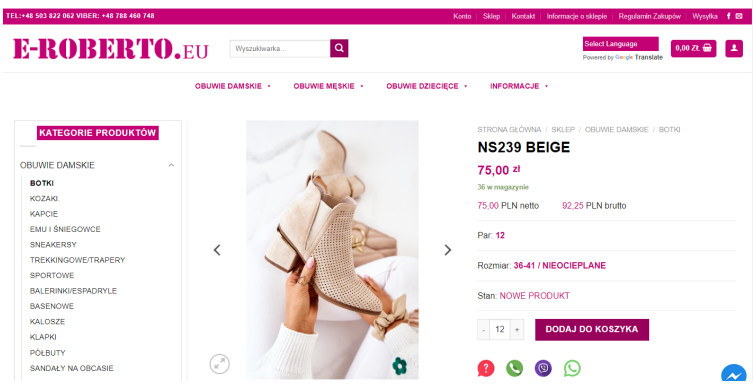


Figure 3: Product detail page

1.3 Data scraping goal

The goal of data scraping at the e-roberto.eu website is to collect most of the information on the products being sold on this website.

- Specific information to be collected about a product includes:
  - Product title (it's also the product ID)
  - Price Net
  - Price Gross
  - Quantity in stock
  - Number of product in a package
  - Size
  - Status
  - Category
  - Product url
  - Image url
- Although there are many product categories on the website, we decided to choose two categories with the following links for testing:
  - Botki: https://e-roberto.eu/product-category/obuwie-damskie/botki/
  - Sneakersy: https://e-roberto.eu/product-category/obuwie-damskie/sneakersy/

1.4 Technology and libraries

1.4.1 Python

Scraping data from online websites can be done in many different programming languages such as: Java, C# or Python. In which Python is an open source language with many support libraries for data scraping.

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed [9].

1.4.2 SQLite

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. SQLite database files are a recommended storage format by the US Library of Congress [1].

1.4.3 Beautiful Soup

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work [2].

The current version of Beautiful Soup is 4.11.0, as it is a library that is not built into Python core, users will need to install this library when they want to use it. Quite a few data scraping projects have used Beautiful Soup, but there are also many websites with dynamic structure, using many complex technologies, leading to inefficient use of Beautiful Soup.

1.4.4 Scrapy

Scrapy is a fast high-level web crawling and web scraping framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing[7].

Scrapy is real titan in case of efficiency. It performs much better than Beautiful Soup and Selenium [4]. The current version of Scrapy is 2.6.

1.4.5 Selenium

Selenium is a powerful web scraping tool developed originally for website testing. These days, it's also used when the accurate portrayal of websites—as they appear in a browser—is required. Selenium works by automating browsers to load the website, retrieve the required data, and even take screenshots or assert that certain actions happen on the website [6].

- Advantages of using Selenium [5]:
  - It can be used in automated testing and task automation.
  - It fairly easy allows to do powerful stuff.

1.5 Tasks and members

Table 1: Members

#	Student Name	ID	Group	Contribution rate for this report
1	Nhat Tung Le	426246	GR.4 Lab	34%
2	Khon Nguyen	444135	GR.3 Lab	33%
3	Adnan Sevinç	437971	GR.4 Lab	33%

Table 2: Tasks

#	Tasks	Students	Deadline	Status
1	Analysis and design	All members	25-04-2022	Done
2	Analyze web pages, identify data scraping steps and scenarios, programmatically scrape data with BeautifulSoup	Khon Nguyen	05-05-2022	Done
3	Analyze web pages, identify data scraping steps and scenarios, programmatically scrape data with Selenium	Adnan Sevinç	05-05-2022	Done
4	Analyze web pages, identify data scraping steps and scenarios, programmatically scrape data with Scrapy	Nhat Tung Le	07-05-2022	Done
5	Synthesize and write reports	All members	12-05-2022	Done

2 Experiment

2.1 Model and Database

2.1.1 Product Model

Python is an object-oriented programming language, to facilitate the management of product information. We build a Product class, which includes constructors and information printing. The Product class will be used in the next parts of this project.

```
class Product:
    # Constructor
    def __init__(self="", product_title="", price_net=0, price_gross=0,
                 quantity_in_stock=0, number_of_product_in_a_package=0,
                 size="", status="", category="", product_url="", image_url=""):
        self.product_title = product_title
        self.price_net = price_net
        self.price_gross = price_gross
        self.quantity_in_stock = quantity_in_stock
        self.number_of_product_in_a_package = number_of_product_in_a_package
        self.size = size
        self.status = status
        self.category = category
        self.product_url = product_url
        self.image_url = image_url

    def printMe(self):
        strMe = """product_title: {0}, price_net: {1}, price_gross: {2},
        quantity_in_stock: {3}, number_of_product_in_a_package: {4}, size: {5},
        status: {6}, category: {7}, product_url: {8}, image_url: {9}
        """.format(
            self.product_title,
            self.price_net,
            self.price_gross,
            self.quantity_in_stock,
            self.number_of_product_in_a_package,
            self.size,
            self.status,
            self.category,
            self.product_url,
            self.image_url)
        print(strMe)

    def toString(self):
        strMe = """product_title: {0}, price_net: {1}, price_gross: {2},
        quantity_in_stock: {3}, number_of_product_in_a_package: {4}, size: {5},
        status: {6}, category: {7}, product_url: {8}, image_url: {9}
        """.format(
            self.product_title,
            self.price_net,
            self.price_gross,
            self.quantity_in_stock,
            self.number_of_product_in_a_package,
            self.size,
            self.status,
```

```
            self.category,
            self.product_url,
            self.image_url)
        return strMe
```

2.1.2 Database Structure

In this project we use SQLite as the archive database that we scraped from the website. The reason for us choosing to store to database instead of storing to CSV file is because the scraped data can be very large, we want to split the data scraping stages. Avoid Repeat the operation and try not to duplicate the data that has been scraped.

With three different technologies, we built 3 tables with similar structure. Data scraped from any tool will be stored in the corresponding table. We will have a comparison and evaluation step for each type of tool we test in this report.

- List of Tables:
  - product_beautifulsoup
  - product_scrapy
  - product_selenium

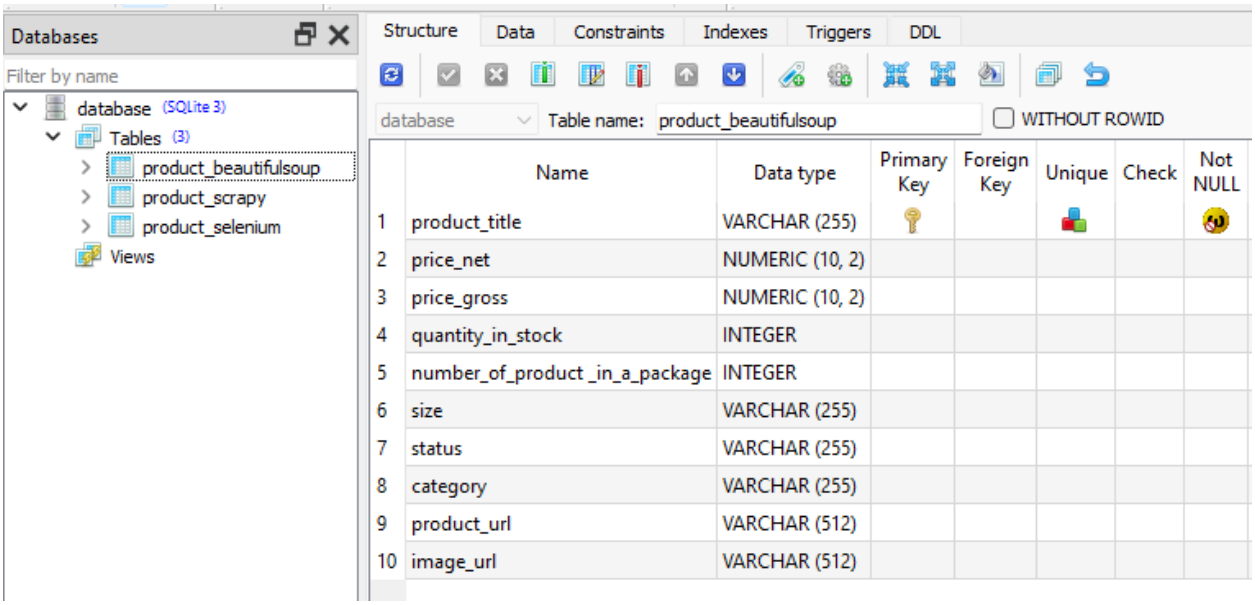


Figure 4: Product list page

2.1.3 Connection

To interact with the connection to the SQLite database, we build a Connection class, it supports other functions that can connect to the database quickly, it helps the user to create cursors to be able to execute SQL statements.

```
import sqlite3

class Connection:
    @staticmethod
    def getConnection():
```

```
        conn = sqlite3.connect('database.db', timeout=10)
        return conn

    @staticmethod
    def getCursor():
        conn = sqlite3.connect('database.db', timeout=10)
        c = conn.cursor()
        return c
```

2.1.4 Data Access Object

With operations Add, Delete, Update, Or query data related to each table. We create a corresponding Data Access Object Class, each function in this class will be built as static, helping to interact with the Product object, and the operations related to it in the database. For this project, we mainly use the Insert method to add data to the table after we scrape the data from the website; In addition, data query methods help us to query data so that we can compare results or perform simple analyses.

Table 3: table interaction

#	Table name	DAO class
1	product_beautifulsoup	ProductBeautifulsoupDAO
2	product_scrapy	ProductScrapyDAO
3	product_selenium	ProductSeleniumDAO

- List of functions: createObject(), listByQuery(), listAll(), listByCondition(), insert(), update(), delete(), findOne()

```
from DBConnection import Connection
from ProductModel import Product

# ProductBeautifulsoup Data access object ##################################


class ProductBeautifulsoupDAO:
    @staticmethod
    def createObject(result):
        rs = result
        obj = Product(rs[0], rs[1], rs[2], rs[3], rs[4],
                      rs[5], rs[6], rs[7], rs[8], rs[9])
        return obj

    @staticmethod
    def listByQuery(sql):
        listOut = []
        conn = Connection.getConnection()
        c = conn.cursor()
        c.execute(sql)
        results = c.fetchall()
        for result in results:
            listOut.append(ProductBeautifulsoupDAO.createObject(result))
        for obj in listOut:
            print(obj.toString())
        return listOut
```

```
    @staticmethod
    def listAll():
        return ProductBeautifulsoupDAO.listByQuery("SELECT * FROM product_beautifulsoup")

    @staticmethod
    def listByCondition(column, value):
        if (isinstance(value, str)):
            return ProductBeautifulsoupDAO.listByQuery(
                """SELECT * FROM product_beautifulsoup WHERE {0}='{1}'"""
                .format(column, value)
            )
        else:
            return ProductBeautifulsoupDAO.listByQuery(
                """SELECT * FROM product_beautifulsoup WHERE {0}='{1}'"""
                .format(column, str(value))
            )

    @staticmethod
    def insert(obj):
        if (isinstance(obj, Product)):
            conn = Connection.getConnection()
            c = conn.cursor()
            sql = """
                INSERT INTO product_beautifulsoup
                VALUES ('{0}', {1}, {2}, {3}, {4}, '{5}', '{6}','{7}', '{8}','{9}')
            """
            c.execute(sql.format(
                obj.product_title,
                obj.price_net,
                obj.price_gross,
                obj.quantity_in_stock,
                obj.number_of_product_in_a_package,
                obj.size,
                obj.status,
                obj.category,
                obj.product_url,
                obj.image_url))
            rowcount = c.rowcount
            conn.commit()
            conn.close()
            return rowcount

    @staticmethod
    def update(obj):
        if (isinstance(obj, Product)):
            conn = Connection.getConnection()
            c = conn.cursor()
            sql = """
                UPDATE product_beautifulsoup
                SET price_net={1},
                    price_gross={2},
                    quantity_in_stock={3},
                    number_of_product_in_a_package={4},
```

```
                    size='{5}',
                    status='{6}',
                    category='{7}',
                    product_url='{3}',
                    image_url='{9}',
                WHERE ProductId='{0}'
            """
            c.execute(sql.format(
                obj.price_net,
                obj.price_gross,
                obj.quantity_in_stock,
                obj.number_of_product_in_a_package,
                obj.size,
                obj.status,
                obj.category,
                obj.product_url,
                obj.image_url,
                obj.product_title))
            rowcount = c.rowcount
            conn.commit()
            conn.close()
            return rowcount

    @staticmethod
    def delete(obj):
        if (isinstance(obj, Product)):
            conn = Connection.getConnection()
            c = conn.cursor()
            sql = """
                DELETE FROM product_beautifulsoup
                WHERE product_title='{0}'
            """
            # print(sql.format(obj.product_title))
            c.execute(sql.format(obj.product_title))
            rowcount = c.rowcount
            conn.commit()
            conn.close()
            # print(rowcount)
            return rowcount

    @staticmethod
    def findOne(id):
        listObj = ProductBeautifulsoupDAO.listByQuery(
            "select * from product_beautifulsoup where product_title='{0}'".format(id))
        if (len(listObj) > 0):
            return listObj[0]
        else:
            return None
```

2.2 Scrapping

2.2.1 Scrapping using Beautiful Soup

a) Data scraping steps:

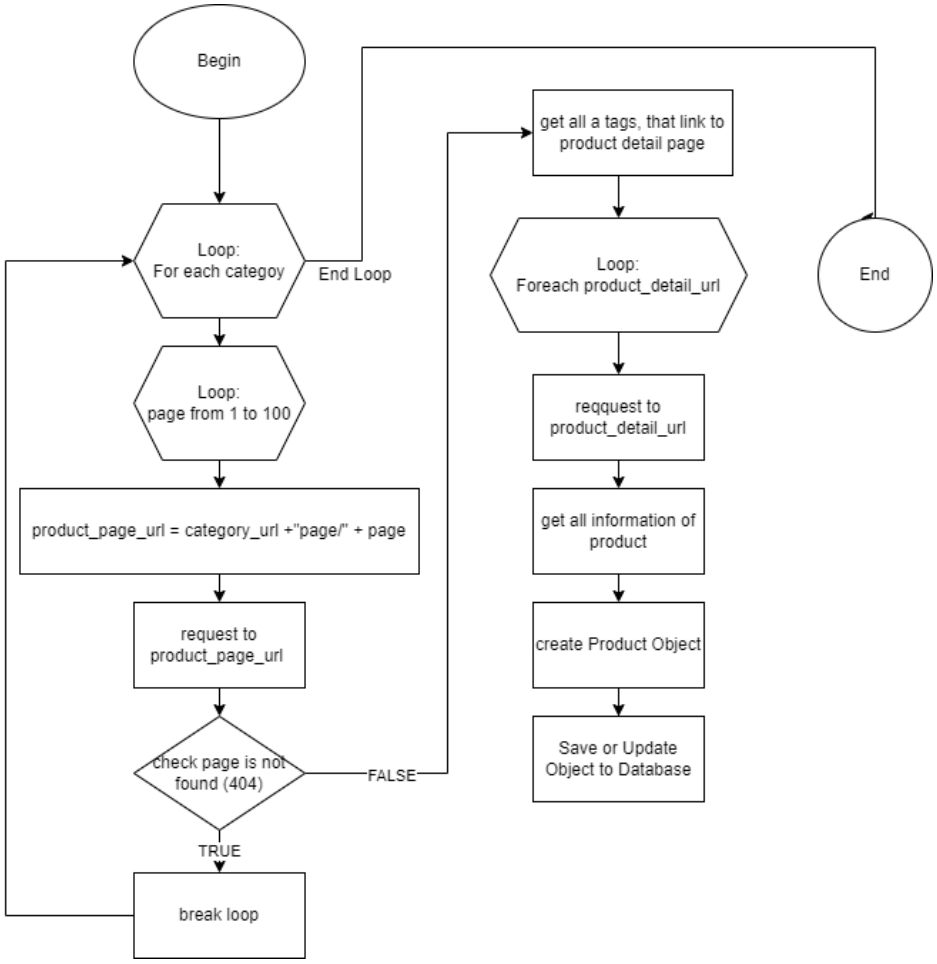


Figure 5: Data scraping steps (using Beautiful Soup)

b) Python code:

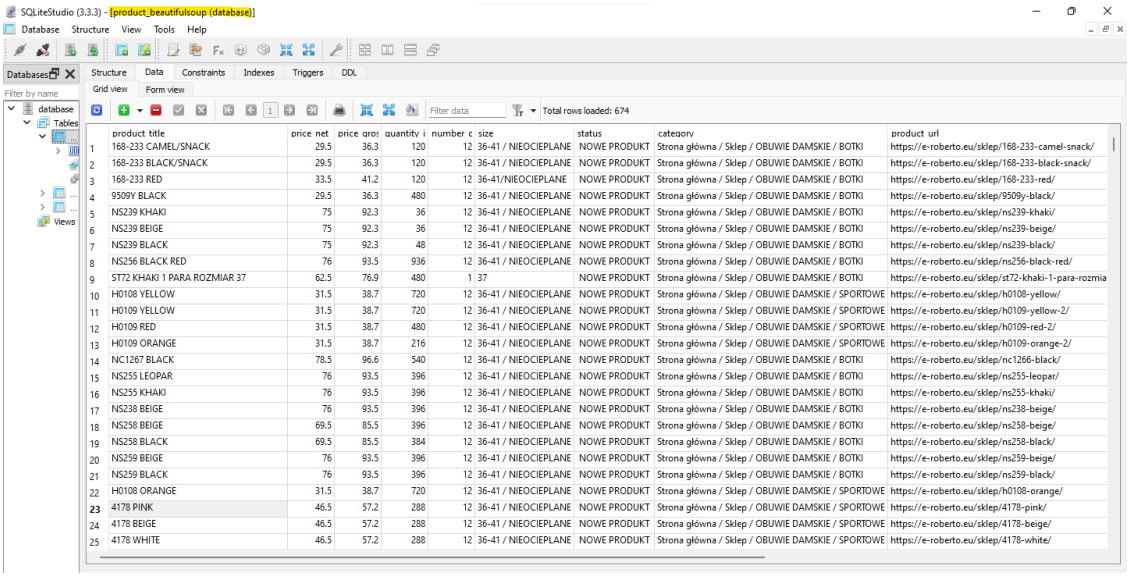
```
# set limit number of products to be scraped
limit = True
number_of_limit = 100
###############################################

from urllib import request
from bs4 import BeautifulSoup as BS
import re
import pandas as pd
from ProductModel import Product
from ProductDAO import *

listCategories = ['https://e-roberto.eu/product-category/obuwie-damskie/botki/',
                  'https://e-roberto.eu/product-category/obuwie-damskie/sneakersy/']
```

```
# count scaped products
count = 0

for c in listCategories:
    for page in range(1, 101):
        # create link of product page
        url_product_page = c+"page/"+str(page)+"/"
        # print(url_product_page)
        try:
            html = request.urlopen(url_product_page)
            bs = BS(html.read(), 'html.parser')

            # check page is exits
            # check the span has 404: page not found
            check_404 = ''
            try:
                check_404 = bs.find('span', string='404').txt
            except:
                check_404 = ''

            # print("check_404: " + check_404)
            if(check_404 == ''):
                # Get all a link product
                link_temp_list = []
                a_tags = bs.find_all(
                    'a', {'class': '''woocommerce-LoopProduct-link
                                    woocommerce-loop-product__link'''})

                for link_product_detail in a_tags:
                    # check limit and exit if count > number_of_limit
                    count = count + 1
                    if(limit):
                        if(count > number_of_limit):
                            exit()

                    # scraping product detail

                    html = request.urlopen(link_product_detail['href'])
                    bs = BS(html.read(), 'html.parser')

                    # Product Title
                    try:
                        product_title = bs.find('h1').text
                        product_title = (product_title.rstrip()).lstrip()
                    except Exception as e:
                        product_title = ''

                    # Price Net
                    try:
                        price_net = bs.find('span', {'id': 'gianet'}).text
                    except Exception as e:
                        price_net = ''
```

```
                    # Price Gross
                    try:
                        price_gross = bs.find(
                            'span', {'id': 'giabrut'}).text
                    except Exception as e:
                        price_gross = ''

                    # Qquantity in stock
                    try:
                        string_quantity_in_stock = bs.find(
                            'p', {'class': 'stock in-stock'}).text
                        array_quantity_in_stock = string_quantity_in_stock.split(
                            " ", 1)
                        quantity_in_stock = array_quantity_in_stock[0]
                    except Exception as e:
                        quantity_in_stock = 0

                    # Number of product in a package
                    try:
                        number_of_product_in_a_package = bs.find(
                            'span', {'id': 'paczka'}).text
                    except Exception as e:
                        number_of_product_in_a_package = 0

                    # Size
                    try:
                        size = bs.find(
                            'span', {'id': 'size'}).text
                    except Exception as e:
                        size = 0

                    # Status
                    try:
                        status = bs.find(
                            'span', {'id': 'XXC'}).next_sibling.next_sibling.text
                    except Exception as e:
                        status = 0

                    # Category
                    try:
                        category = bs.find(
                            'nav', {'class': '''woocommerce-breadcrumb
                                    breadcrumbs uppercase'''}).text
                    except Exception as e:
                        category = ''

                    # Product Url
                    try:
                        product_url = link_product_detail['href']
                    except Exception as e:
                        product_url = ''

                    # Image Url
```

```
                try:
                    image_url = bs.find(
                        'img', {'class': 'wp-post-image skip-lazy'})['src']
                except Exception as e:
                    image_url = ''

                # Insert or Update Product to Database
                try:
                    product = Product(product_title, price_net, price_gross,
                              quantity_in_stock, number_of_product_in_a_package,
                              size, status, category, product_url, image_url)

                    poduct_check_exit = ProductBeautifulsoupDAO.findOne(
                        product_title)

                    if(poduct_check_exit is None):
                        ProductBeautifulsoupDAO.insert(product)
                    else:
                        ProductBeautifulsoupDAO.update(product)
                except Exception as e:
                    print(e)

        except Exception as e:
            print(e)
```

c) Data scraping results:



	product title	price net	price gro	quantity i	number c	size	status	category	product url
1	168-233 CAMEL/SNACK	29.5	36.3	120	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/168-233-camel-snack/
2	168-233 BLACK/SNACK	29.5	36.3	120	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/168-233-black-snack/
3	168-233 RED	33.5	41.2	120	12	36-41/NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/168-233-red/
4	9509Y BLACK	29.5	36.3	480	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/9509y-black/
5	NS239 KHAKI	75	92.3	36	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns239-khaki/
6	NS239 BEIGE	75	92.3	36	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns239-beige/
7	NS239 BLACK	75	92.3	48	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns239-black/
8	NS256 BLACK RED	76	93.5	936	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns256-black-red/
9	ST72 KHAKI 1 PARA ROZMIAR 37	62.5	76.9	480	1	37	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/st72-khaki-1-para-rozmia
10	H0108 YELLOW	31.5	38.7	720	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/h0108-yellow/
11	H0109 YELLOW	31.5	38.7	720	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/h0109-yellow-2/
12	H0109 RED	31.5	38.7	480	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/h0109-red-2/
13	H0109 ORANGE	31.5	38.7	216	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/h0109-orange-2/
14	NC1267 BLACK	78.5	96.6	540	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/nc1266-black/
15	NS255 LEOPAR	76	93.5	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns255-leopar/
16	NS255 KHAKI	76	93.5	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns255-khaki/
17	NS238 BEIGE	76	93.5	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns238-beige/
18	NS258 BEIGE	69.5	85.5	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns258-beige/
19	NS258 BLACK	69.5	85.5	384	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns258-black/
20	NS259 BEIGE	76	93.5	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns259-beige/
21	NS259 BLACK	76	93.5	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/ns259-black/
22	H0108 ORANGE	31.5	38.7	720	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/h0108-orange/
23	4178 PINK	46.5	57.2	288	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/4178-pink/
24	4178 BEIGE	46.5	57.2	288	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/4178-beige/
25	4178 WHITE	46.5	57.2	288	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	Strona główna / Sklep / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/4178-white/

Figure 6: Data ressults (using Beautiful Soup)

- Total products: 679 (rows)

2.2.2 Scrapping using Selenium

a) Data scraping steps:

(Flowchart is shown in figure 7)

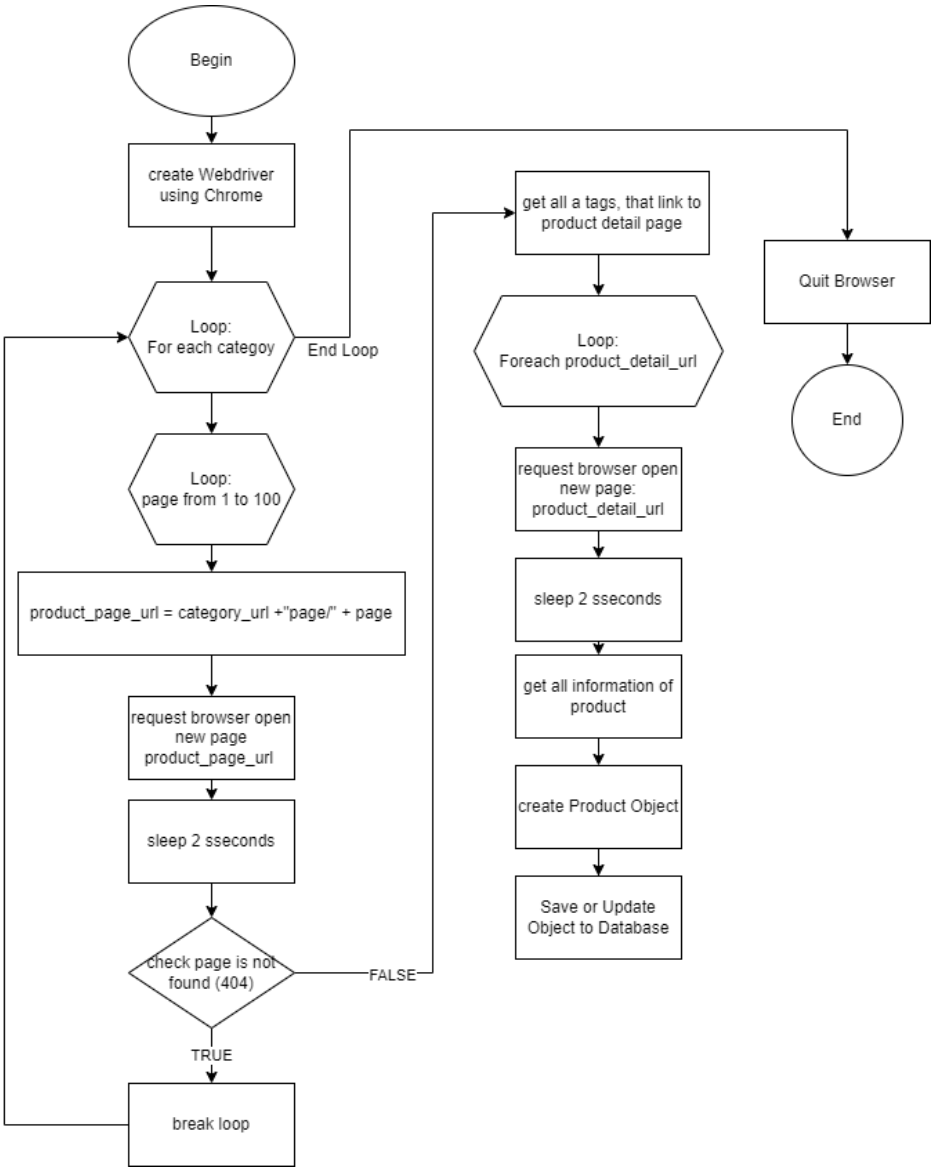


Figure 7: Data scraping steps (using Selenium)

b) Python code:

```
# set limit number of products to be scraped
limit = True
number_of_limit = 100
###############################################

from dataclasses import replace
from xml.dom.minidom import Element
from boto import BUCKET_NAME_RE
from selenium import webdriver
```

```
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
import datetime
import os
from ProductModel import Product
from ProductDAO import *


driver = webdriver.Chrome()

# sending file to professor
listCategories = ['https://e-roberto.eu/product-category/obuwie-damskie/botki/',
                  'https://e-roberto.eu/product-category/obuwie-damskie/sneakersy/']

for url in listCategories:
    for page in range(1, 101):
        try:
            url_product_page = url+"page/"+str(page)+"/"
            driver.get(url_product_page)
            time.sleep(2)

            error_404 = False
            try:
                driver.find_element(By.XPATH, '''//section[@class="error-404
                            not-found mt mb"]''')
                error_404 = True
            except:
                error_404 = False

            print(error_404)

            if (error_404 == False):
                # Get all a link product has class cotains
                # 'woocommerce-LoopProduct-link woocommerce-loop-product__link'
                a_tags = driver.find_elements_by_xpath(
                    '''//a[@class=woocommerce-LoopProduct-link
                                woocommerce-loop-product__link"]''')

                list_link_product_details = []
                for atag in a_tags:
                    list_link_product_details.append(atag.get_attribute('href'))

                print(list_link_product_details)

                for link_product_detail in list_link_product_details:
                    # check limit and exit if count > number_of_limit
                    count = count + 1
                    if(limit):
                        if(count > number_of_limit):
                            driver.quit()
                            exit()
```

```
                # scraping product detail
                driver.get(link_product_detail)
                time.sleep(2)

                # Product Title
                try:
                    product_title = driver.find_element(By.XPATH, '//h1').text
                    product_title = (product_title.rstrip()).lstrip()
                except Exception as e:
                    product_title = ''

                # Price Net
                try:
                    price_net = float((driver.find_element(By.XPATH,
                        '//span[@id="gianet"]').text).replace("," , '.'))
                except Exception as e:
                    price_net = 0

                # Price Gross
                try:
                    price_gross = float((driver.find_element(By.XPATH,
                      '//span[@id="giabrut"]').text).replace("," , '.'))
                except Exception as e:
                    price_gross = 0

                # Qquantity in stock
                try:
                    string_quantity_in_stock = driver.find_element(By.XPATH,
                              '//p[@class="stock in-stock"]').text
                    array_quantity_in_stock = string_quantity_in_stock.split(
                        " ", 1)
                    quantity_in_stock = int(array_quantity_in_stock[0])
                except Exception as e:
                    quantity_in_stock = 0

                # Number of product in a package
                try:
                    number_of_product_in_a_package = int(driver.find_element(By.XPATH,
                    '//span[@id="paczka"]').text)
                except Exception as e:
                    number_of_product_in_a_package = 0

                # Size
                try:
                    size = driver.find_element(By.XPATH, '//span[@id="size"]').text
                except Exception as e:
                    size = ''

                # Category
                try:
                    category = driver.find_element(By.XPATH,
                    '''//nav[@class="woocommerce-breadcrumb
                    breadcrumbs uppercase"]''').text
```

```
                    except Exception as e:
                        category = ''

                    # Status
                    try:
                        status = driver.find_element(By.XPATH,
                        '//span[@id="XXC"]/following-sibling::*[1]').text
                    except Exception as e:
                        status = ''

                    # Product Url
                    product_url = link_product_detail

                    # Image Url
                    try:
                        image_url = driver.find_element(By.XPATH,
                        '''//img[@class="wp-post-image skip-lazy"]''').get_attribute('src')
                    except Exception as e:
                        image_url = ''

                    # Insert or Update Product to Database
                    try:
                        product = Product(product_title, price_net, price_gross,
                             quantity_in_stock, number_of_product_in_a_package,
                            size, status, category, product_url, image_url)

                        poduct_check_exit = ProductSeleniumDAO.findOne(
                            product_title)

                        if(poduct_check_exit is None):
                            ProductSeleniumDAO.insert(product)
                        else:
                            ProductSeleniumDAO.update(product)
                    except Exception as e:
                        print(e)
            else:
                break
        except Exception as e:
            print(e)
            print("error")
driver.quit()
```

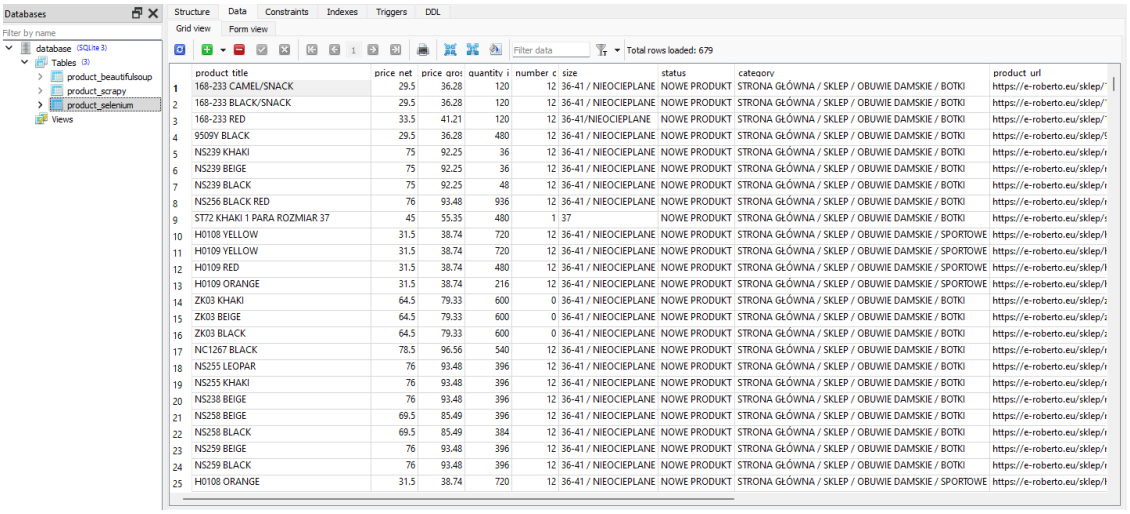
c) Data scraping results:

- Total products: 674 (rows)

2.2.3 Scrapping using Scrapy

a) Preparing the environment:

E-robeto.eu is a dynamic website, implemented on the Wordpress platform, so that some elements of the website are rendered based on JavaScript code. Therefore, with the pure Scrapy library, we cannot get



	product title	price net	price gros	quantity i	number c	size	status	category	product url
1	168-233 CAMEL/SNACK	29.5	36.28	120	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
2	168-233 BLACK/SNACK	29.5	36.28	120	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
3	168-233 RED	33.5	41.21	120	12	36-41/NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
4	9509Y BLACK	29.5	36.28	480	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
5	NS239 KHAKI	75	92.25	36	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
6	NS239 BEIGE	75	92.25	36	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
7	NS239 BLACK	75	92.25	48	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
8	NS256 BLACK RED	76	93.48	936	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
9	ST72 KHAKI 1 PARA ROZMIAR 37	45	55.35	480	1	37	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
10	H0108 YELLOW	31.5	38.74	720	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/
11	H0109 YELLOW	31.5	38.74	720	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/
12	H0109 RED	31.5	38.74	480	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/
13	H0109 ORANGE	31.5	38.74	216	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/
14	ZK03 KHAKI	64.5	79.33	600	0	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
15	ZK03 BEIGE	64.5	79.33	600	0	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
16	ZK03 BLACK	64.5	79.33	600	0	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
17	NC1267 BLACK	78.5	96.56	540	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
18	NS255 LEOPAR	76	93.48	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
19	NS255 KHAKI	76	93.48	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
20	NS238 BEIGE	76	93.48	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
21	NS258 BEIGE	69.5	85.49	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
22	NS258 BLACK	69.5	85.49	384	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
23	NS259 BEIGE	76	93.48	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
24	NS259 BLACK	76	93.48	396	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / BOTKI	https://e-roberto.eu/sklep/
25	H0108 ORANGE	31.5	38.74	720	12	36-41 / NIEOCIEPLANE	NOWE PRODUKT	STRONA GŁÓWNA / SKLEP / OBUWIE DAMSKIE / SPORTOWE	https://e-roberto.eu/sklep/

Figure 8: Data ressults (using Selenium)

the most important data of each product. In this project, we use some additional tools that make it possible for Scrapy to get more information.

- Additional tools we use are as follows:
  - WSL: Windows Subsystem for Linux
  - Docker: a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers.
  - Splash: Splash is a javascript rendering service with an HTTP API. It's a lightweight browser with an HTTP API, implemented in Python 3 using Twisted and QT5. It's fast, lightweight and state-less which makes it easy to distribute.
- Some commands in preparing the environment for scrapy-splash:

```
# INSTALL DOCKER & LINUX WSL
# Install Docker Desktop: https://docs.docker.com/desktop/windows/install/
# Linux-kernel:
# https://docs.microsoft.com/en-us/windows/wsl/install-manual
# step-4---download-the-linux-kernel-update-package
# Install linux: user: wme, password: 123456
# Rundocker: & "C:\Program Files\Docker\Docker\DockerCli.exe" -SwitchDaemon

# INSTALL AND RUN SCRAPY-SPLASH DOCKER
# https://github.com/scrapy-plugins/scrapy-splash
# pip install scrapy-splash
# docker pull scrapinghub/splash
# docker run -p 8050:8050 scrapinghub/splash
# docker pull scrapinghub/splash
# docker run -p 8050:8050 scrapinghub/splash


# INSTALL scrapy-splash library
# $ pip install scrapy scrapy-splash
```

- Configuration for the Scrappy project (settings.py):

```
# Splash
SPLASH_URL = 'http://localhost:8050/'
```

```
DOWNLOADER_MIDDLEWARES = {
    'scrapy_splash.SplashCookiesMiddleware': 723,
    'scrapy_splash.SplashMiddleware': 725,
    'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware': 810,
}
```

b) Data scraping steps:

We divide the data scraping process with Scrapy into several different stages, each of which will have detailed implementation steps related to data scraping.

- *Stages:*
  - *Stage 0:* Preparing and running splash Docker
  - *Stage 1:* Search all product list pages for the categories provided from the file *link_categories.csv*. The results of this phase will be stored in the file *link_pages.csv*.
  - *Stage 2:* Based on the links provided from the *link_pages.csv* file, find all the links leading to the detailed description of each product. The results of this phase will be stored in the file *link_products.csv*.
  - *Stage 3:* Based on the links provided from the *link_products.csv* file, find all information of a product, then write to CSV. The results of this phase will be stored in the file *product_details.csv*.
  - *Stage 4:* Transfer all data from *product_details.csv* file to database, in this step we will also remove data lines that do not meet the basic information of the product.

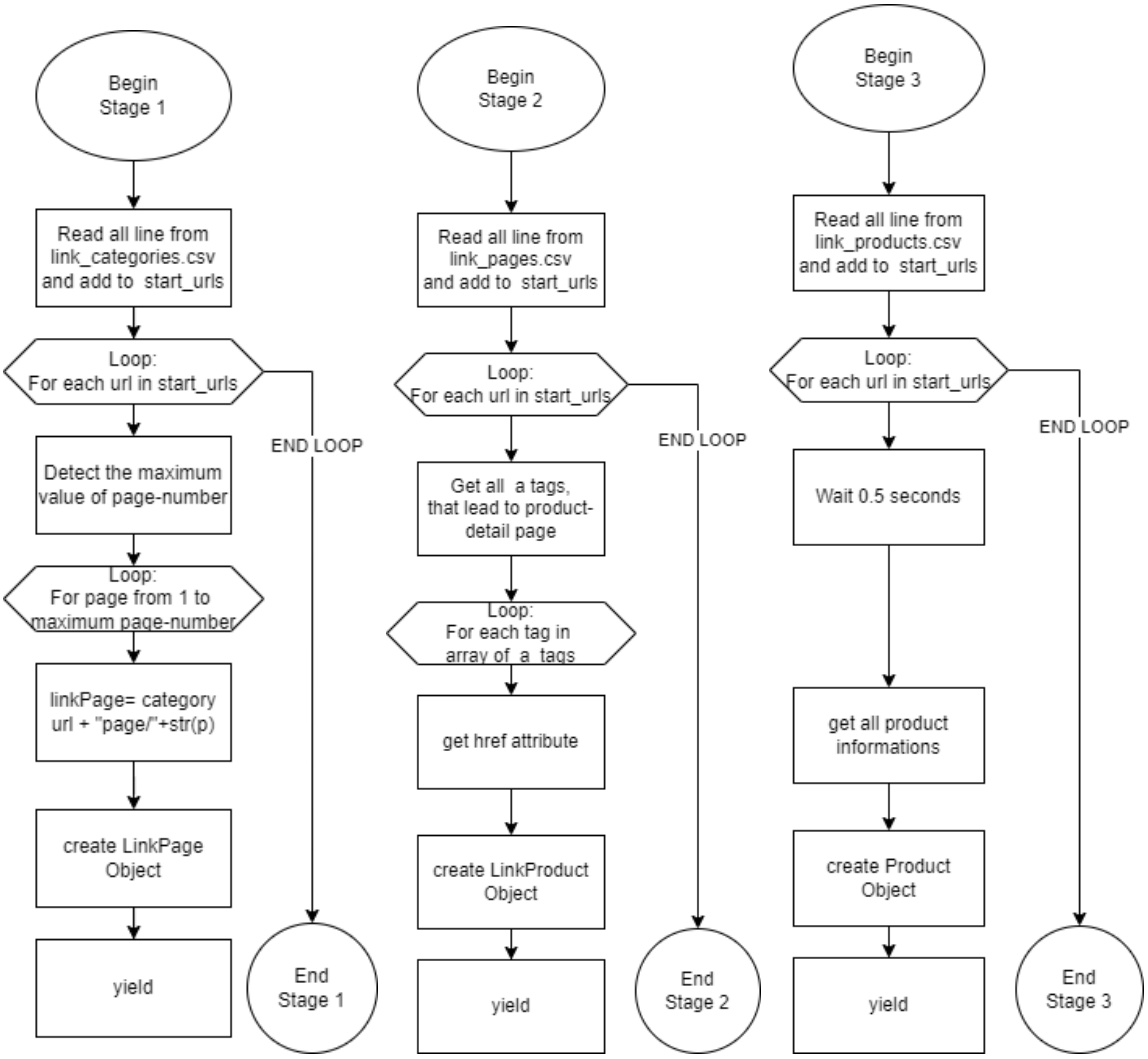


Figure 9: Flowchart (using Scrapy)

c) Python code:

- *Stage 1:*

```
import scrapy

class LinkPage(scrapy.Item):
    page = scrapy.Field()


class LinkPageSpider(scrapy.Spider):
    name = 'link_page'
    allowed_domains = ['https://e-roberto.eu/']
    try:
         with open("link_categories.csv", "rt") as f:
             start_urls = [url.strip() for url in f.readlines()][1:]
    except:
         start_urls = []

    def parse(self, response):
        # xpath: get all a tags
        max_page = 1
        xpath = '//a[contains(@class, "page-number")]/text()'
        for link in response.xpath(xpath):
            max_page = max(int(link.get()), max_page)

        for p in range(1, max_page + 1):
            linkPage = LinkPage()
            linkPage['page'] = response.request.url + "page/"+str(p)
            print(linkPage['page'])
            yield linkPage


#scrapy crawl link_page -o link_pages.csv
```

- *Stage 2:*

```
import scrapy

class LinkProduct(scrapy.Item):
    productLink = scrapy.Field()


class LinkProductSpider(scrapy.Spider):
    name = 'link_product'
    allowed_domains = ['https://e-roberto.eu/']
    try:
         with open("link_pages.csv", "rt") as f:
             start_urls = [url.strip() for url in f.readlines()][1:]
    except:
         start_urls = []

    def parse(self, response):
        # xpath: get all a tags
        xpath = '//div[contains(@class, "image-zoom")]/a/@href'
        for link in response.xpath(xpath):
```

```
            linkProduct = LinkProduct()
            linkProduct['productLink'] = link.get()
            print(linkProduct['productLink'])
            yield linkProduct

#scrapy crawl link_product -o link_products.csv
```

- *Stage 3:*

```
# set limit number of products to be scraped
limit = True
number_of_limit = 100
##############################################

import scrapy
from scrapy_splash import SplashRequest

class Product(scrapy.Item):
    product_title = scrapy.Field()
    price_net = scrapy.Field()
    price_gross = scrapy.Field()
    quantity_in_stock = scrapy.Field()
    number_of_product_in_a_package = scrapy.Field()
    size = scrapy.Field()
    status = scrapy.Field()
    category = scrapy.Field()
    product_url = scrapy.Field()
    image_url = scrapy.Field()

class ProductDetailSpider(scrapy.Spider):
    name = 'product_detail'
    allowed_domains = ['https://e-roberto.eu/']
    try:
         with open("link_products.csv", "rt") as f:
            if(limit):
                # get limit number of products to be scraped
                start_urls = [url.strip() for url in f.readlines()][1:number_of_limit]
            else:
                start_urls = [url.strip() for url in f.readlines()][1:]
    except:
         start_urls = []

    def start_requests(self):
        for url in self.start_urls:
            yield SplashRequest(url, self.parse,
                endpoint='render.html',
                args={'wait': 0.5},
        )

    def parse(self, response):
        product = Product()

        # xpath: get all h1 tags
        try:
```

```
            xpath = '//h1[contains(@class, "product-title")]/text()'
            product_title = response.xpath(xpath).get()
            product_title = (product_title.rstrip()).lstrip()
            product['product_title'] = product_title
        except:
            product['product_title'] = ''

        # xpath: get price net
        try:
            xpath = '//span[contains(@id, "gianet")]/text()'
            product['price_net'] = response.xpath(xpath).get()
        except:
            product['price_net'] = 0

        # xpath: get price gross
        try:
            xpath = '//span[contains(@id, "giabrut")]/text()'
            product['price_gross'] = response.xpath(xpath).get()
        except:
            product['price_gross'] = 0

        # xpath: get quantity in stock
        try:
            xpath = '//p[contains(@class, "stock in-stock")]/text()'
            array_quantity_in_stock = (response.xpath(xpath).get()).split(" ", 1)
            product['quantity_in_stock'] = array_quantity_in_stock[0]
        except:
            product['quantity_in_stock'] = 0

        # xpath: get number of product in a package
        try:
            xpath = '//span[contains(@id, "paczka")]/text()'
            product['number_of_product_in_a_package'] = response.xpath(xpath).get()
        except:
            product['number_of_product_in_a_package'] = 0

        # xpath: get size
        try:
            xpath = xpath = '''//span[contains(@id, "gianet")]
            /following-sibling::*[3]/text()'''
            product['size'] = response.xpath(xpath).get()
        except:
            product['size'] = ''

        # xpath: get status
        try:
            xpath = '//span[@id="XXC"]/following-sibling::*[2]/text()'
            product['status'] = response.xpath(xpath).get()
        except:
            product['status'] = ''

        # xpath: get category
        try:
```

```
            xpath = '''//nav[@class="woocommerce-breadcrumb
            breadcrumbs uppercase"]/a[last()]/text()'''
            product['category'] = response.xpath(xpath).get()
        except:
            product['category'] = ''

        # xpath: get product url
        try:
            # product['product_url'] = response.request.url
            product['product_url'] = "https://e-roberto.eu/sklep/{0}/"
            .format((product_title.replace(" ", "-")).lower())
        except:
            product['product_url'] = ''

        # xpath: get image url
        try:
            xpath = '//img[contains(@class, "wp-post-image")]/@src'
            product['image_url'] = response.xpath(xpath).get()
        except:
            product['image_url'] = ''

        yield product

#scrapy crawl product_detail -o product_details.csv
```

- *Stage 4:*

```
'''
The scraping of data is done on the basis of scrapy framework.
This file supports the transfer of data from the "product_details.csv" file into
the database in preparation for further evaluation steps.
'''

from ProductModel import Product
from ProductDAO import *

try:
    with open("scrapy_project/product_details.csv", "rt") as f:
        lines = [url.strip() for url in f.readlines()][1:]
        for line in lines:
            try:
                data = line.split(",")
                category = "" + data[0]
                image_url = "" + data[1]
                number_of_product_in_a_package = int("0"+data[2])
                price_gross= data[3].replace("\"", "")+"."+data[4].replace("\"", "")
                price_gross = price_gross.replace(",", ".")
                price_net = data[5].replace("\"", "")+"."+data[6].replace("\"", "")
                price_net = price_net.replace(",", ".")
                product_title= "" + data[7]
                product_url = "" + data[8]
                quantity_in_stock= int("0" +data[9])
                size = "" + data[10]
                status= data[11] + ""
```

```
                product = Product(product_title, price_net, price_gross,
                                  quantity_in_stock, number_of_product_in_a_package,
                                      size, status, category, product_url, image_url)

                print(product.toString())
                poduct_check_exit = ProductScrapyDAO.findOne(
                    product_title)

                if(poduct_check_exit is None):
                    ProductScrapyDAO.insert(product)
                else:
                    ProductScrapyDAO.update(product)
            except Exception as e:
                print(e)
except Exception as e:
    print(e)
```

d) Data scraping results:

	A	B	C	D	E	F	G	H	I	J
1	category	image_url	number_of_product	price_gross	price_net	product_title	product_url	quantity_in_stock	size	status
2	SPORTOWE	https://e-roberto.eu/wp-content/uploac		59,66	48,5	NB537 BLACK	https://e-roberto.eu	960		
3	SPORTOWE	https://e-roberto.eu/wp-content/uploac		34,44	28	F44 BLACK	https://e-roberto.eu	120		
4	SPORTOWE	https://e-roberto.eu/wp-content/uploac		59,66	48,5	NB537 BEIGE	https://e-roberto.eu	960		
5	BOTKI	https://e-roberto.eu/wp-content/uploac		48,59	39,5	7897 BLACK	https://e-roberto.eu	936		
6	SPORTOWE	https://e-roberto.eu/wp-content/uploac		34,44	28	F44 KHAKI	https://e-roberto.eu	1200		
7	BOTKI	https://e-roberto.eu/wp-content/uploac		48,59	39,5	TX-1873 BLACK	https://e-roberto.eu	948		
8	SPORTOWE	https://e-roberto.eu/wp-content/uploac		20,29	16,5	WB806 ORANGE	https://e-roberto.eu	324		
9	SPORTOWE	https://e-roberto.eu/wp-content/uploac		51,66	42	UB07 WINE	https://e-roberto.eu	552		
10	SPORTOWE	https://e-roberto.eu/wp-content/uploac		44,9	36,5	TCYR-76 KHAKI	https://e-roberto.eu	252		
11	SPORTOWE	https://e-roberto.eu/wp-content/uploac		36,28	29,5	PG06 BLACK	https://e-roberto.eu	1776		
12	SPORTOWE	https://e-roberto.eu/wp-content/uploac		38,74	31,5	85-751 BEIGE	https://e-roberto.eu	480		
13	SPORTOWE	https://e-roberto.eu/wp-content/uploac		36,28	29,5	AB630 WH/RED/BLU	https://e-roberto.eu	600		
14	SPORTOWE	https://e-roberto.eu/wp-content/uploac		38,74	31,5	85-751 BLACK	https://e-roberto.eu	480		
15	SPORTOWE	https://e-roberto.eu/wp-content/uploac		51,66	42	UB07 NAVY	https://e-roberto.eu	564		
16	SPORTOWE	https://e-roberto.eu/wp-content/uploac		36,28	29,5	JT902 WHITE	https://e-roberto.eu	600		
17	SPORTOWE	https://e-roberto.eu/wp-content/uploac		38,74	31,5	F071 BEIGE	https://e-roberto.eu	360		
18	SPORTOWE	https://e-roberto.eu/wp-content/uploac		47,97	39	NB528 PINK	https://e-roberto.eu	600		
19	SPORTOWE	https://e-roberto.eu/wp-content/uploac		47,97	39	NB528 BEIGE	https://e-roberto.eu	600		
20	SPORTOWE	https://e-roberto.eu/wp-content/uploac		44,9	36,5	TCYR-76 PINK	https://e-roberto.eu	252		

Figure 10: Data ressults (using Scrapy Splash - Stage 1)

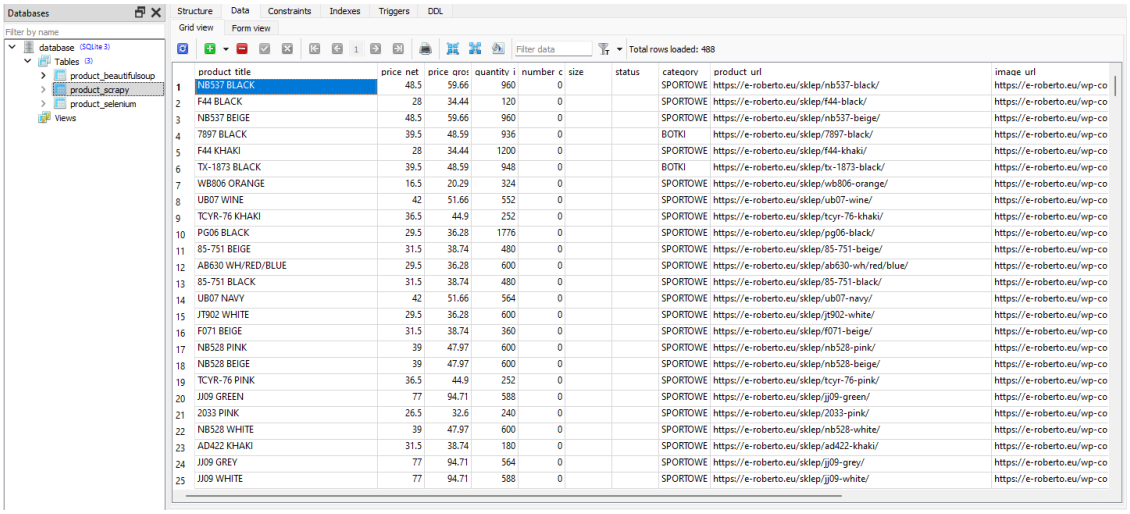
	A	B	C	D	E	F	G	H	I	J
1	category	image_url	number_of_product	price_gross	price_net	product_title	product_url	quantity_in_stock	size	status
2	SPORTOWE	https://e-roberto.eu/wp-content/uploac		59,66	48,5	NB537 BLACK	https://e-roberto.eu	960		
3	SPORTOWE	https://e-roberto.eu/wp-content/uploac		34,44	28	F44 BLACK	https://e-roberto.eu	120		
4	SPORTOWE	https://e-roberto.eu/wp-content/uploac		59,66	48,5	NB537 BEIGE	https://e-roberto.eu	960		
5	BOTKI	https://e-roberto.eu/wp-content/uploac		48,59	39,5	7897 BLACK	https://e-roberto.eu	936		
6	SPORTOWE	https://e-roberto.eu/wp-content/uploac		34,44	28	F44 KHAKI	https://e-roberto.eu	1200		
7	BOTKI	https://e-roberto.eu/wp-content/uploac		48,59	39,5	TX-1873 BLACK	https://e-roberto.eu	948		
8	SPORTOWE	https://e-roberto.eu/wp-content/uploac		20,29	16,5	WB806 ORANGE	https://e-roberto.eu	324		
9	SPORTOWE	https://e-roberto.eu/wp-content/uploac		51,66	42	UB07 WINE	https://e-roberto.eu	552		
10	SPORTOWE	https://e-roberto.eu/wp-content/uploac		44,9	36,5	TCYR-76 KHAKI	https://e-roberto.eu	252		
11	SPORTOWE	https://e-roberto.eu/wp-content/uploac		36,28	29,5	PG06 BLACK	https://e-roberto.eu	1776		
12	SPORTOWE	https://e-roberto.eu/wp-content/uploac		38,74	31,5	85-751 BEIGE	https://e-roberto.eu	480		
13	SPORTOWE	https://e-roberto.eu/wp-content/uploac		36,28	29,5	AB630 WH/RED/BLU	https://e-roberto.eu	600		
14	SPORTOWE	https://e-roberto.eu/wp-content/uploac		38,74	31,5	85-751 BLACK	https://e-roberto.eu	480		
15	SPORTOWE	https://e-roberto.eu/wp-content/uploac		51,66	42	UB07 NAVY	https://e-roberto.eu	564		
16	SPORTOWE	https://e-roberto.eu/wp-content/uploac		36,28	29,5	JT902 WHITE	https://e-roberto.eu	600		
17	SPORTOWE	https://e-roberto.eu/wp-content/uploac		38,74	31,5	F071 BEIGE	https://e-roberto.eu	360		
18	SPORTOWE	https://e-roberto.eu/wp-content/uploac		47,97	39	NB528 PINK	https://e-roberto.eu	600		
19	SPORTOWE	https://e-roberto.eu/wp-content/uploac		47,97	39	NB528 BEIGE	https://e-roberto.eu	600		
20	SPORTOWE	https://e-roberto.eu/wp-content/uploac		44,9	36,5	TCYR-76 PINK	https://e-roberto.eu	252		

Figure 11: Data ressults (using Scrapy Splash - Stage 2)

- Total products: 488 (rows)

	A	B	C	D	E	F	G	H	I	J
1	category	image_url	number_of_product	price_gross	price_net	product_title	product_url	quantity_in_stock	size	status
2	SPORTOWE	https://e-roberto.eu/wp-content/uploac		59,66	48,5	NB537 BLACK	https://e-roberto.eu	960		
3	SPORTOWE	https://e-roberto.eu/wp-content/uploac		34,44	28	F44 BLACK	https://e-roberto.eu	120		
4	SPORTOWE	https://e-roberto.eu/wp-content/uploac		59,66	48,5	NB537 BEIGE	https://e-roberto.eu	960		
5	BOTKI	https://e-roberto.eu/wp-content/uploac		48,59	39,5	7897 BLACK	https://e-roberto.eu	936		
6	SPORTOWE	https://e-roberto.eu/wp-content/uploac		34,44	28	F44 KHAKI	https://e-roberto.eu	1200		
7	BOTKI	https://e-roberto.eu/wp-content/uploac		48,59	39,5	TX-1873 BLACK	https://e-roberto.eu	948		
8	SPORTOWE	https://e-roberto.eu/wp-content/uploac		20,29	16,5	WB806 ORANGE	https://e-roberto.eu	324		
9	SPORTOWE	https://e-roberto.eu/wp-content/uploac		51,66	42	UB07 WINE	https://e-roberto.eu	552		
10	SPORTOWE	https://e-roberto.eu/wp-content/uploac		44,9	36,5	TCYR-76 KHAKI	https://e-roberto.eu	252		
11	SPORTOWE	https://e-roberto.eu/wp-content/uploac		36,28	29,5	PG06 BLACK	https://e-roberto.eu	1776		
12	SPORTOWE	https://e-roberto.eu/wp-content/uploac		38,74	31,5	85-751 BEIGE	https://e-roberto.eu	480		
13	SPORTOWE	https://e-roberto.eu/wp-content/uploac		36,28	29,5	AB630 WH/RED/BLUI	https://e-roberto.eu	600		
14	SPORTOWE	https://e-roberto.eu/wp-content/uploac		38,74	31,5	85-751 BLACK	https://e-roberto.eu	480		
15	SPORTOWE	https://e-roberto.eu/wp-content/uploac		51,66	42	UB07 NAVY	https://e-roberto.eu	564		
16	SPORTOWE	https://e-roberto.eu/wp-content/uploac		36,28	29,5	JT902 WHITE	https://e-roberto.eu	600		
17	SPORTOWE	https://e-roberto.eu/wp-content/uploac		38,74	31,5	F071 BEIGE	https://e-roberto.eu	360		
18	SPORTOWE	https://e-roberto.eu/wp-content/uploac		47,97	39	NB528 PINK	https://e-roberto.eu	600		
19	SPORTOWE	https://e-roberto.eu/wp-content/uploac		47,97	39	NB528 BEIGE	https://e-roberto.eu	600		
20	SPORTOWE	https://e-roberto.eu/wp-content/uploac		44,9	36,5	TCYR-76 PINK	https://e-roberto.eu	252		

Figure 12: Data ressults (using Scrapy Splash - Stage 3)



	product title	price net	price gros	quantity i	number c	size	status	category	product url	image url
1	NB537 BLACK	48.5	59.66	960	0			SPORTOWE	https://e-roberto.eu/sklep/nb537-black/	https://e-roberto.eu/wp-co
2	F44 BLACK	28	34.44	120	0			SPORTOWE	https://e-roberto.eu/sklep/f44-black/	https://e-roberto.eu/wp-co
3	NB537 BEIGE	48.5	59.66	960	0			SPORTOWE	https://e-roberto.eu/sklep/nb537-beige/	https://e-roberto.eu/wp-co
4	7897 BLACK	39.5	48.59	936	0			BOTKI	https://e-roberto.eu/sklep/7897-black/	https://e-roberto.eu/wp-co
5	F44 KHAKI	28	34.44	1200	0			SPORTOWE	https://e-roberto.eu/sklep/f44-khaki/	https://e-roberto.eu/wp-co
6	TX-1873 BLACK	39.5	48.59	948	0			BOTKI	https://e-roberto.eu/sklep/tx-1873-black/	https://e-roberto.eu/wp-co
7	WB806 ORANGE	16.5	20.29	324	0			SPORTOWE	https://e-roberto.eu/sklep/wb806-orange/	https://e-roberto.eu/wp-co
8	UB07 WINE	42	51.66	552	0			SPORTOWE	https://e-roberto.eu/sklep/ub07-wine/	https://e-roberto.eu/wp-co
9	TCYR-76 KHAKI	36.5	44.9	252	0			SPORTOWE	https://e-roberto.eu/sklep/tcyr-76-khaki/	https://e-roberto.eu/wp-co
10	PG06 BLACK	29.5	36.28	1776	0			SPORTOWE	https://e-roberto.eu/sklep/pg06-black/	https://e-roberto.eu/wp-co
11	85-751 BEIGE	31.5	38.74	480	0			SPORTOWE	https://e-roberto.eu/sklep/85-751-beige/	https://e-roberto.eu/wp-co
12	AB630 WH/RED/BLUE	29.5	36.28	600	0			SPORTOWE	https://e-roberto.eu/sklep/ab630-wh/red/blue/	https://e-roberto.eu/wp-co
13	85-751 BLACK	31.5	38.74	480	0			SPORTOWE	https://e-roberto.eu/sklep/85-751-black/	https://e-roberto.eu/wp-co
14	UB07 NAVY	42	51.66	564	0			SPORTOWE	https://e-roberto.eu/sklep/ub07-navy/	https://e-roberto.eu/wp-co
15	JT902 WHITE	29.5	36.28	600	0			SPORTOWE	https://e-roberto.eu/sklep/jt902-white/	https://e-roberto.eu/wp-co
16	F071 BEIGE	31.5	38.74	360	0			SPORTOWE	https://e-roberto.eu/sklep/f071-beige/	https://e-roberto.eu/wp-co
17	NB528 PINK	39	47.97	600	0			SPORTOWE	https://e-roberto.eu/sklep/nb528-pink/	https://e-roberto.eu/wp-co
18	NB528 BEIGE	39	47.97	600	0			SPORTOWE	https://e-roberto.eu/sklep/nb528-beige/	https://e-roberto.eu/wp-co
19	TCYR-76 PINK	36.5	44.9	252	0			SPORTOWE	https://e-roberto.eu/sklep/tcyr-76-pink/	https://e-roberto.eu/wp-co
20	JJ09 GREEN	77	94.71	588	0			SPORTOWE	https://e-roberto.eu/sklep/jj09-green/	https://e-roberto.eu/wp-co
21	2033 PINK	26.5	32.6	240	0			SPORTOWE	https://e-roberto.eu/sklep/2033-pink/	https://e-roberto.eu/wp-co
22	NB528 WHITE	39	47.97	600	0			SPORTOWE	https://e-roberto.eu/sklep/nb528-white/	https://e-roberto.eu/wp-co
23	AD422 KHAKI	31.5	38.74	180	0			SPORTOWE	https://e-roberto.eu/sklep/ad422-khaki/	https://e-roberto.eu/wp-co
24	JJ09 GREY	77	94.71	564	0			SPORTOWE	https://e-roberto.eu/sklep/jj09-grey/	https://e-roberto.eu/wp-co
25	JJ09 WHITE	77	94.71	588	0			SPORTOWE	https://e-roberto.eu/sklep/jj09-white/	https://e-roberto.eu/wp-co

Figure 13: Data ressults (using Scrapy Splash)

2.2.4 Comparing scarpping results

After scraping data with 3 different tools for E-roberto.eu website, we compare and make the following observations:

- About the accuracy of data scraping:
  - BeatifulSoup and Selenium have the most accurate data scraping results, these tools can get all the data fields of a product as outlined in section 1.3.
  - Although Scrapy has combined with Splash, we still cannot get the information fields: size, status, quantity product in a package.
  - We have scraped the data at roughly the same time, but the number of products that can be seen for each tool is different: Selenium with 679 products, BeautifulSoup with 674 products and finally Scrapy with 488 products (actually thethe number of lines scraped when using Scrapy is 679 , but there is a lot of blank data, including not getting the title of the product).
- About data scraping speed:
  - BeautifulSoup has the fastest scraping speed in this project.
  - For Selenium, when we open the product category page, open the detailed description page of the product, we have to let the browser pause for 2 secondes to make sure that the data is rendered completely.
  - For Scrapy, through the Splash server, we have to let the program wait for 0.5 seconds, to ensure that the JavaScript finishes rendering the data. Because based on our experience, the speed of scraping data can vary depending on the computer configurations.
  - (Because the implementation of data scraping technologies on personal computers with different configurations, our comments are based on actual experience.)
- About the experimental implementation:
  - In our opinion, BeautifulSoup is the easiest approach for this website scaping. Because it has relevant library to utilize. Although E-roberto.eu is a dynamic website, BeatifulSoup still fully meets the requirements of the data scraping process
  - Selenium ranks 2nd in term of ease when doing experimental setup. However we must ensure that the chromedriver is compatible with the Chrome browser being used.
  - Scrapy requires the most difficult setup. We have to search for additional tools like Docker or Splash in order to work successfully in this project.

Table 4: Rating by levels (1 is best)

#	Lib	Accuracy rank	Speed rank	Easy to deploy rank
1	Beautifulsoup	2	1	1
2	Selenium	1	3	2
3	Scrapy	3	2	3

2.2.5 Extremely elementary data analysis

Data scraped from the webiste can be applied to data science related problems. For businesses, they can scrape data and observe competitors' product prices. Thanks to those, they business owners can have more inputs in order to decide the effective prices for their own products. It can give them more advantage to compete in the market. | In addition, getting updates on the market situations allows the managers having the big picture about the market. They can foresee and predict the trend when seeing the products from competitors' website. From that, they can choose the appropriate strategy win in the competition and get more market shares.

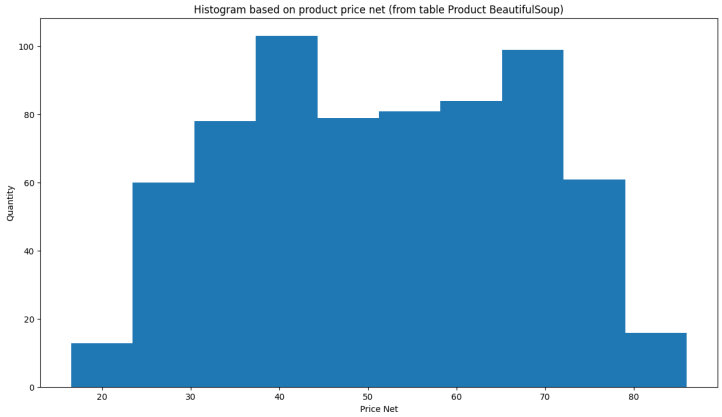


Figure 14: Histogram based on product price net (from table Product BeautifulSoup)

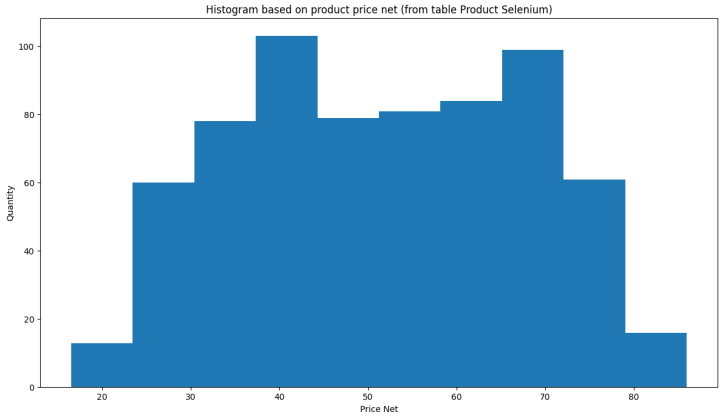


Figure 15: Histogram based on product price net (from table Product Selenium))

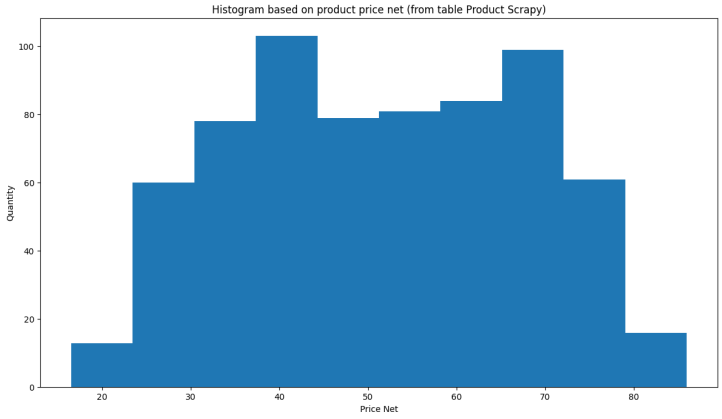


Figure 16: Histogram based on product price net (from table Product Scrapy)

3 Conclusion

In this project, we have had success using 3 libraries BeautifulSoup, Selenium and Scrapy (with Splash) to experimentally scrape data from E-roberto.eu website. Each tool has its own advantages and disadvantages, we have made comments in the comparison of results.

Through this project, we have gained more knowledge related to data scraping, some related tools are also presented in this report.

We will continue to use these tools to scrape data for data science research. We are also interested in learning about technologies to help scrape data for high-security websites, capable of detecting bots or requiring image or captcha authentication.

References

[1] *About SQLite*. URL: https://www.sqlite.org/about.html (visited on 05/03/2022).

[2] *Beautiful Soup Documentation — Beautiful Soup 4.9.0 Documentation*. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (visited on 05/03/2022).

[3] *Informacje o sklepie – E-Roberto.eu*. URL: https://e-roberto.eu/informacje/ (visited on 05/02/2022).

[4] Przemysław Kurek and Maciej Wysocki. "Web Scraping and Social Media Scraping: Scrapy". In: (), p. 10.

[5] Przemysław Kurek and Maciej Wysocki. "Web Scraping and Social Media Scraping: Selenium". In: (), p. 9.

[6] Ryan Mitchell. *Web Scraping with Python: Collecting More Data from the Modern Web*. 2nd edition. Sebastopol, CA: O'Reilly Media, May 8, 2018. 306 pp. ISBN: 978-1-4919-8557-1.

[7] *Scrapy 2.6 Documentation — Scrapy 2.6.1 Documentation*. URL: https://docs.scrapy.org/en/latest/ (visited on 05/03/2022).

[8] *Web Scraping*. In: *Wikipedia*. Feb. 9, 2022. URL: https://en.wikipedia.org/w/index.php?title=Web_scraping&oldid=1070889749 (visited on 05/02/2022).

[9] *What Is Python? Executive Summary*. Python.org. URL: https://www.python.org/doc/essays/blurb/ (visited on 05/03/2022).