Report for Data Management: supermarkets in Milan

University of Milano-Bicocca
Data Science master's degree
AY 2022-2023

Matteo Corona - 838138
Francesca Corvino - 898058
Costanza Pagnin - 898306

Contents

List of Figures

List of Codes

List of Tables

1 Introduction

This project focuses on data management and primarily leverages the technique of scraping to build a comprehensive dataset containing prices of a wide range of products from three different supermarkets located in the city of Milan. Scraping, or web scraping, is a methodology that allows for the automatic extraction of data from web pages, enabling the efficient collection of large amounts of information. The main purpose of this project is to create a dataset that can be used for comparative price analysis among supermarkets, as well as potentially evaluating price variations over time. This will provide valuable insights for both consumers and businesses to make more informed purchasing decisions or strategic choices. During the scraping process, several factors were taken into account, such as identifying the supermarkets to include, selecting the product categories to monitor, and implementing an algorithm to accurately and consistently extract prices from supermarket websites. Special attention was given to the quality of the collected data, ensuring the coherence and integrity of the obtained information, enabling a meaningful comparison among different supermarkets. In addition to simply collecting the data, the project also involved a phase of cleaning, transforming, and organizing the data to prepare it for subsequent analysis and processing.

2 Scraping

The data we want to collect includes unit prices, product names and categories. In addition to this, we have also collected (or extrapolated at a later stage) the quantities of individual products. The data has been collected from three major supermarket chains in Milan: Esselunga, Carrefour, and Conad. Some websites required entering a location or ZIP code to select specific store locations. However, the purpose of the project is solely to gather and compare prices from different supermarkets and not to investigate potential price differences among individual stores within a specific chain so random addresses were chosen.

2.1 Scraping on Esselunga

The preliminary step is certainly to import the necessary libraries. In order to perform scraping, *Selenium* and *Beautiful Soup 4* were used. Selenium is a popular open-source framework for automating web browsers: it can launch a web browser, interact with elements on web pages, and retrieve data by inspecting the HTML structure of the page. Selenium supports various programming languages including Python, Java, C#, and Ruby. This project was carried out using Python.

```python
# Importazione librerie
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
from bs4 import BeautifulSoup
import pandas as pd
import time
```

Codice 1: Libraries importation

2.1.1 Login

Firstly, it is necessary to access the Esselunga website. No authentication with an account is required, but it is sufficient to enter a ZIP code that falls within the areas covered by the "online shopping" service and then select the "proceed as guest" option[1]. These operations have been automated by writing a function called *access()* that performs exactly the steps just described.

```python
def access(driver):
    # Trovo l'elemento in cui inserire il CAP
    cap_input = driver.find_element(By.CSS_SELECTOR, "#postcode")
    # Inserisco un CAP
    cap_input.send_keys("20124")
    time.sleep(3)
    # Trovo l'elemento del pulsante "Verifica CAP" utilizzando il selettore CSS
    verifica_button = driver.find_element(By.CSS_SELECTOR,
                                          "button[type='submit']")
    # Click pulsante "Verifica CAP"
    verifica_button.click()
    time.sleep(6)
    # Trovo l'elemento del link "Continua come ospite"
    continua_ospite_link = driver.find_element(By.CSS_SELECTOR,
                                               "a[data-remodal-action='close']")
    # Click sul link "Continua come ospite"
    continua_ospite_link.click()
    time.sleep(6)
```

[1]On the day of delivery, 06/13/2023, the method of accessing the Esselunga web page has changed and it is also necessary to enter a specific address, therefore the code should be slightly modified.

```
19      return None
```

Codice 2: Login to Esselunga website code

As one can see from code 2, it is necessary to instruct Selenium to search for the appropriate CSS selectors in order to subsequently invoke the methods *.click()* and *.send_keys()*.

2.1.2 Loading products

Next, in code 4, we will illustrate how we were able to gather product prices. However, before collecting them, it's necessary to load all the products on the page after selecting a specific category. Figure 1 depicts the product screen on the Esselunga website. In order to achieve this purpose, we created a function called *scroll_down()* that scrolls the page downwards using the scroll sidebar, ensuring all the products are loaded. This function is implemented in code 3.

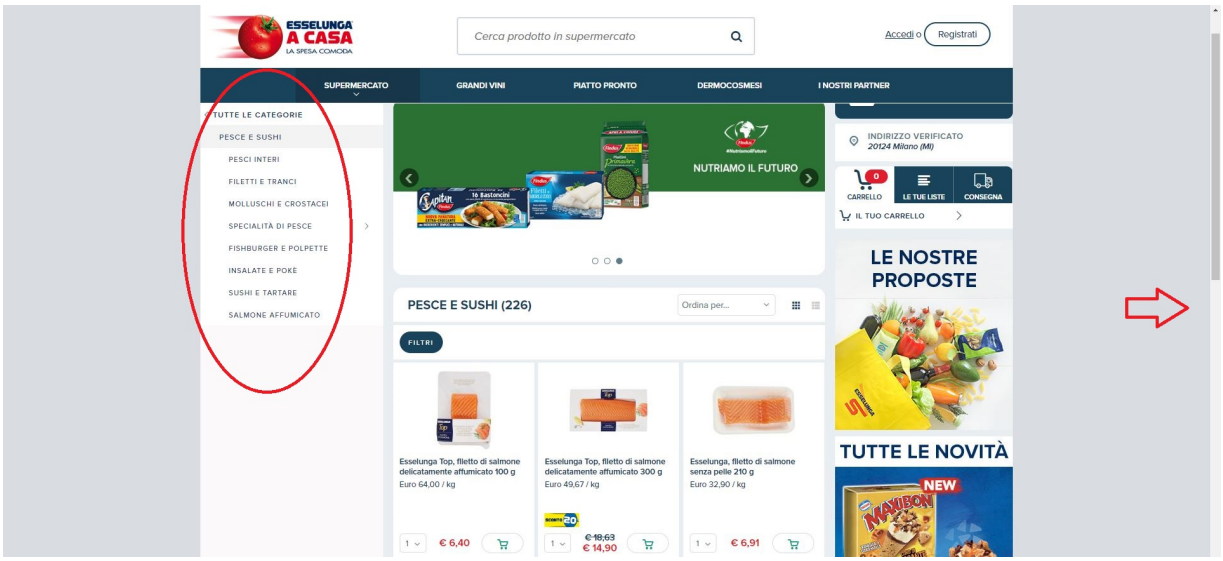

Figure 1: Esselunga website

```
1  # Funzione scorrimento pagina per caricare tutti i prodotti
2  def scroll_down(driver)
3      # Get scroll height
4      last_height = driver.execute_script(return document.body.scrollHeight)
5      while True
6          # Scroll down to bottom
7          driver.execute_script(window.scrollTo(0, document.body.scrollHeight);)
8          # Wait to load page
```

```
        time.sleep(3)
        # Calculate new scroll height and compare with last scroll height
        new_height = driver.execute_script(return document.body.scrollHeight)
        if new_height == last_height
            break
        last_height = new_height
```

Codice 3: Scroll downwards function

2.1.3 Collecting prices

It is now possible to build a function that allows gathering all the prices and names of the products. The *find_data()* function is implemented in code 4. As can be seen in figure 2, the data to be collected is located below the image of each product. By analyzing the HTML code, it has been observed that this information is contained within the classes *description* and *product-brand*. In order to find all the products on the page, it was sufficient to invoke the *.find_all()* method from the *BeautifulSoup* library. Afterwards, the price and name are extracted from each element of the *.find_all()* list using the *get_text* command. Finally, the obtained data is combined into a *Pandas* dataframe as the output. Some product names were not fully loaded, so a condition was implemented based on the length of the names. If a name has more than 55 characters, the link associated with that specific product is opened, and the name is collected from the new page where it is displayed in its entirety.

Figure 2: Esselunga products

```
# Funzione ricerca prodotti e prezzi
def find_data(soup):
    base = 'https://www.esselungaacasa.it/ecommerce/nav/auth/supermercato/'
    # Ricerca degli elementi desiderati (prezzi e nomi prodotti)
    desc = soup.find_all('div', class_='description')
    prod = soup.find_all('span', class_ = 'product-brand')
    # Creazione liste vuote
    prices = []
    products = []
    # Ricerca e salvataggio nomi prodotti
    for i in prod:
        # Nome prodotto
        text = i.get_text(strip=True)
        if len(text)>=55: # Condizione nome troppo lungo
            # Ricerca nodo parente
            a = i.find_parent('a')
            # Salvataggio link prodotto
            relativo = a.get('href')
            link = base + relativo
            time.sleep(3)
            # Switch e indirizzamento alla seconda pagina web
            driver.switch_to.window(driver.window_handles[1])
            driver.get(link)
            time.sleep(6)
            # Ricerca elementi desiderati salvando HTML con BeautifulSoup
            real_soup = BeautifulSoup(driver.page_source, 'html.parser')
            real_prod = real_soup.find('span', class_='product-brand')
            # Nome prodotto completo
            real_text = real_prod.get_text(strip=True)
            products.append(real_text)
            # Switch alla prima pagina web
            driver.switch_to.window(driver.window_handles[0])
        else:
            products.append(text)
    # Ricerca e salvataggio prezzi
    for element in desc:
        # Salvataggio prezzo
        price = element.find('p').text.strip()
```

```
        prices.append(price)
    # Costruzione output dataframe
    output = pd.DataFrame({'Product': products,
                           'Price': prices})
    return output
```

Codice 4: Find data function

The *time.sleep()* function is used with high durations of seconds. This decision was made because the loading of Esselunga pages was not immediate, and often they were not fully loaded before the scraping commands were executed. This could lead to errors, which were avoided by allowing sufficient time for the web pages to load completely.

Product	Price	Category
esselunga bio cipolle dorate 500 g	3.16 € / kg	spesa bio
esselunga bio datterini biologici 300 g	8.27 € / kg	spesa bio
esselunga bio insalatina 100 g	14.80 € / kg	spesa bio
esselunga bio 6 uova fresche biologiche	0.49 € / pz	spesa bio
esselunga bio prezzemolo tritato biologico surgelato 50 g	23.00 € / kg	spesa bio
esselunga bio misto per soffritto biologico surgelato 150 g	7.67 € / kg	spesa bio
esselunga bio passata di pomodoro biologica 700 g	1.85 € / kg	spesa bio
esselunga bio farina tipo 00 di grano tenero biologica 1 kg	1.29 € / kg	spesa bio

Table 1: Scraping Esselunga dataset

2.2 Scraping on Conad

The scraping procedure was applied to the Conad website as well, in order to obtain product names, prices and categories. Firstly, the necessary libraries were imported

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
import time
from bs4 import BeautifulSoup
```

```
import requests
import pandas as pd
from selenium.common.exceptions import NoSuchElementException
import re
```

Codice 5: Libraries importation

2.2.1 Conad homepage

The following code was run to get from the home page to the products listed on the website.

```
driver = webdriver.Chrome()
url_conad = 'https://spesaonline.conad.it/home'
driver.get(url_conad)

#rifiuto i cookie
selettore_cookie = driver.find_elements(By.CSS_SELECTOR,
                                        '#onetrust-reject-all-handler')
selettore_cookie[0].click()

#clicca sul tuo indirizzo
selettore_indirizzo = driver.find_element(By.CSS_SELECTOR,
                                    "label[for='googleInputEntrypageLine1']")
driver.execute_script("arguments[0].click();", selettore_indirizzo)

input_box = driver.find_element(By.ID, "googleInputEntrypageLine1")
input_box.clear() # Clear any existing value
input_box.send_keys("20145")

#clicca su verifica
verifica_button = driver.find_element(By.CSS_SELECTOR, "button.submitButton")
verifica_button.click()
time.sleep(2)

seleziona_button= driver.find_element(By.CSS_SELECTOR, "#ordina-e-ritira >
                                                    div > div > button")
seleziona_button.click()
time.sleep(2)

```

```
conad_button= driver.find_element(By.CSS_SELECTOR,
                                  "#ordina-ritira-scelta-pdv >
                                  div.section-view-negozi > div >
                                  div.lista-negozi-section > div > ul >
                                  li:nth-child(1) > div")
conad_button.click()
time.sleep(1)

negozio_button= driver.find_element(By.CSS_SELECTOR,
         "#modal-onboarding-wrapper >
          div.modal-content.uk-align-center.uk-grid-margin.uk-first-column >
          div.bottom-section.btn-conferma-pdv > button")
negozio_button.click()
time.sleep(3)

tutti_prod_button = driver.find_element(By.CSS_SELECTOR,
                      "a[onclick*='Header.loadMiniBanners']")
tutti_prod_button.click()
time.sleep(3)
```

Codice 6: Product access

Code 6 refuses cookies, asks the user what area the he or she is located in and identifies the store the selected groceries will be collected from.

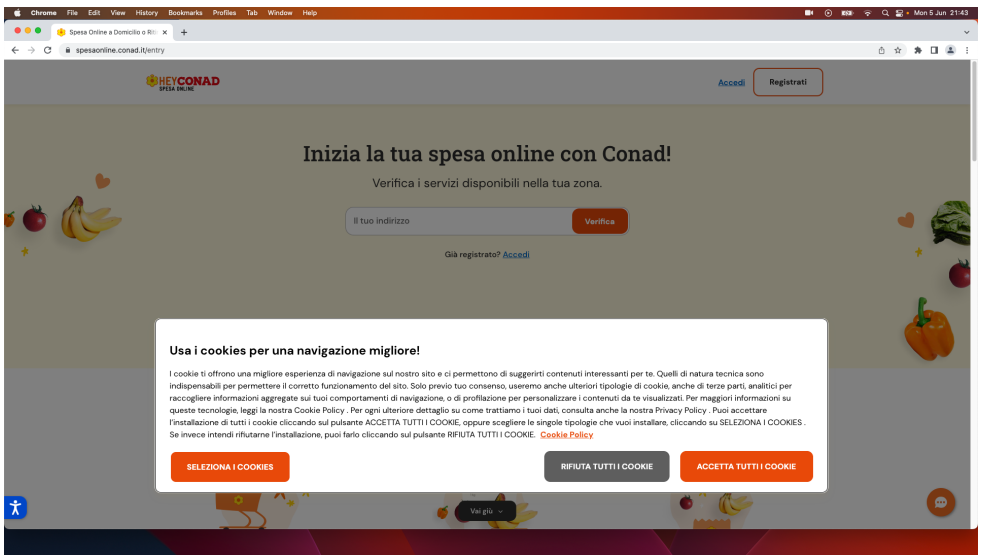

Figure 3: Conad initial website

The page we end up on after these steps in the following

Figure 4: Conad website

2.3 Accessing Conad products and categories

From here, we can access the products by clicking on "Tutti i prodotti" and inspect their relative subcategories.

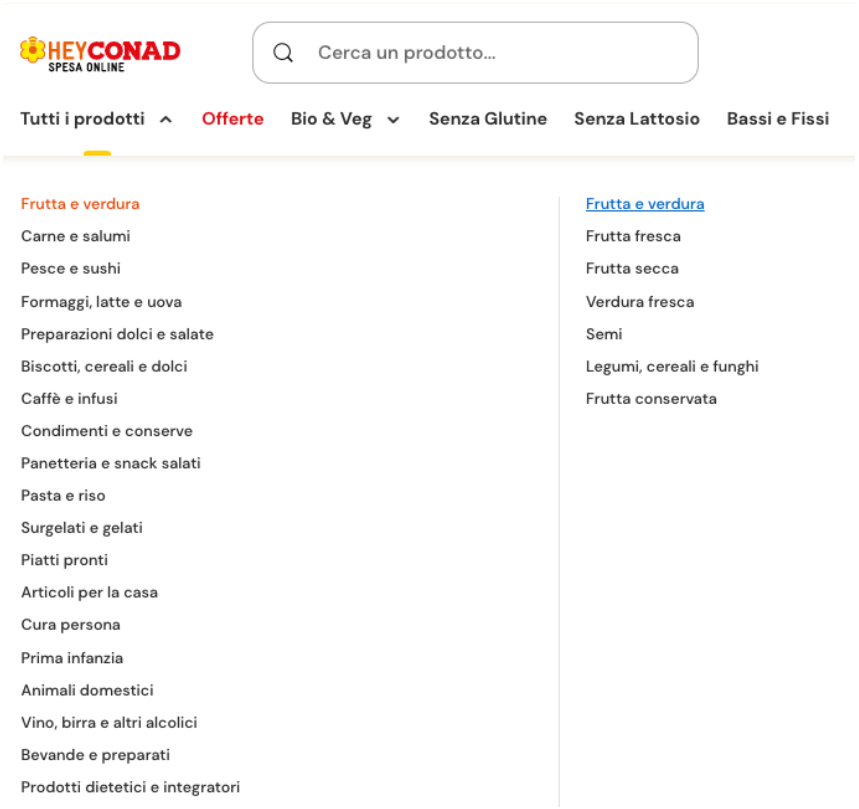

Figure 5: Conad categories

The following code is used to count the number of subcategories per category.

```
#counting the number of sottocategorie per ogni selettore
html = driver.page_source
soup = BeautifulSoup(html, 'html.parser')
megamenu_div = soup.find('div', class_='megamenu-second-level')
div_elements = megamenu_div.find_all('div', class_='uk-hidden')

numero_per_sottocat = []

for div in div_elements:
    li_elements = div.find_all('li')
    #faccio meno uno perche nelle sottocategorie, il primo non lo considero
    li_count = len(li_elements)-1
    numero_per_sottocat.append(li_count)

print(numero_per_sottocat)

#identifico i selettori
selettori=driver.find_elements(By.CSS_SELECTOR, "li.cat")
selettori_len=len(selettori)
```

Codice 7: subcat count

In the following step, we'll define functions to extract product names, their price and their price per unit of measurement. The price per unit of measurement is recalculated in case the product is discounted, in order to display only full prices on the final dataset. This way, the different pricing across stores is not due to discounts.

```
def find_prices(soup):
  price_elements = soup.find_all(class_='product-price-kg')
    # Extract the prices from the elements
    for element in price_elements:
      if 'override-color-red' in element.get('class', []):
        price1=element.get_text(strip=True)
        price = float(price1.split(' ')[0].replace(',', '.'))
        #calcolo la percentuale di sconto
        discount_span = element.find_previous(class_='badge-text')
        #trasformo lo sconto in numero intero
        discount=int(discount_span.get_text(strip=True).rstrip('%').lstrip('-'))
        #calcolo il prezzo non scontato
```

```
        new_price = round(price/((100-discount)/100),2)
        #lo riconverto in stringa
        string_risultato="{:,.1f}".format(new_price)
        #lo rimetto nel formato iniziale
        risultato = price1.replace(price1.split(' ')[0], string_risultato, 1)
        prices.append(risultato)
      else:
        price = element.get_text(strip=True)
        prices.append(price)
  return prices

def find_products(soup):
    div_elements = soup.find_all('div',
          class_='no-t-decoration product-description uk-position-relative')

    # Iterate over the <div> elements
    for div_element in div_elements:
        # Find the <h3> element within the current <div> element
        h3_element = div_element.find('h3')
        product_name = h3_element.text.strip()
        products.append(product_name)
        # Print or process the product name
    return products

def find_single_prices(soup):
    price_single=soup.find_all('div',class_="product-price")
    price_single
    for p in price_single:
        price_single_final= p.get_text(strip=True)
        prices_single.append(price_single_final)
    return prices_single
```

Codice 8: Defining functions

To retrieve the information needed from the products, the following code is run. For every category, we click on the first subcategory. Once we have accessed the first subcategory, a page with products of that subcategeory appear. The code is designed to retrieve the needed information, scroll down the page and select the following one in case it is present. If not, the system clicks on the following subcategory. Once all subcategories have been iterated upon, we go back to the 'Tutti i prodotti' page and apply the same procedure for the following category.

The output is a tabular dataset with product name, price per unit of measurement, category and subcategory

```
data = pd.DataFrame({'products': [], 'prices_per_unitmeasurement': [],
                     'unit_price': [], 'category': '','subcategory':''})

#ipotesi
total_prod=[]
count=[]


for i in (range(selettori_len-3)):
    driver.get(url_conad)
    time.sleep(0.5)
    tutti_prod_button = driver.find_element(By.CSS_SELECTOR,
               "a[onclick*='Header.loadMiniBanners']")
    tutti_prod_button.click()
    time.sleep(0.2)
    selettori=driver.find_elements(By.CSS_SELECTOR, "li.cat")
    time.sleep(0.2)
    selettori[i].click()
    category_links = driver.find_elements(By.CSS_SELECTOR,
                         "[onclick^='Header.trackSecondLevel']")
    category_links[sum(numero_per_sottocat[:i])].click()
    time.sleep(0.2)
    # itero per un numero di volte pari al numero di sottocategorie
    # per ogni selettore
    #remember: j starts from 0.
    for j in range(numero_per_sottocat[i]):
      #ora sono nella prima sottocategoria di un selettore
      #ottengo tutti i link delle sottocategorie
      category_list = driver.find_elements(By.CSS_SELECTOR,
                           "ul.category-list li a")
      category_selectors = [category.get_attribute("href")
                           for category in category_list]
      #per ogni sottocategoria, clicco sul suo link
      driver.get(category_selectors[j])
      # svuoto prodotti e prices perche' li rinnovo per ogni sottocategoria
```

```
     products = []
     prices_single=[]
     prices=[]
     time.sleep(0.5)
     #scorro tutte le pagine di una sottocategoria
     while True:
       try:
         html = driver.page_source #interazione beautiful soup e eselenium
         soup = BeautifulSoup(html, 'html.parser')
         find_prices(soup)
         find_products(soup)
         find_single_prices(soup)
         selettori_names[i]
         next_element = driver.find_element(By.CSS_SELECTOR,
                                    "a[aria-label='Pagina Successiva']")
         next_element.click()
         time.sleep(2.5)
       except NoSuchElementException:
         # No more pages to view
         break
    #qua ho finito di scorrere tutte le pagine di un selettore
    #conto il numero di item per questa sottocategoria
         results_element = soup.find('b', class_='results')
         conteggio= int(results_element.text.strip().split()[0])
         #salvo il conteggio in una lista
         count.append(conteggio)
         #salvo i prodotti di una sottocateogira in un df data_sottocat
         total_prod=total_prod + products
         data_sottocat = pd.DataFrame({'products': products,
                                    'prices_per_unitmeasurement': prices,
                                    'unit_price': prices_single,
                                    'category':selettori_names[i],
                                    'subcategory':
                                  subcat[sum(numero_per_sottocat[:i])+j]})
         data=data.append(data_sottocat)
data.to_csv('conad.csv', index=False)
```

Codice 9: Information retrieval

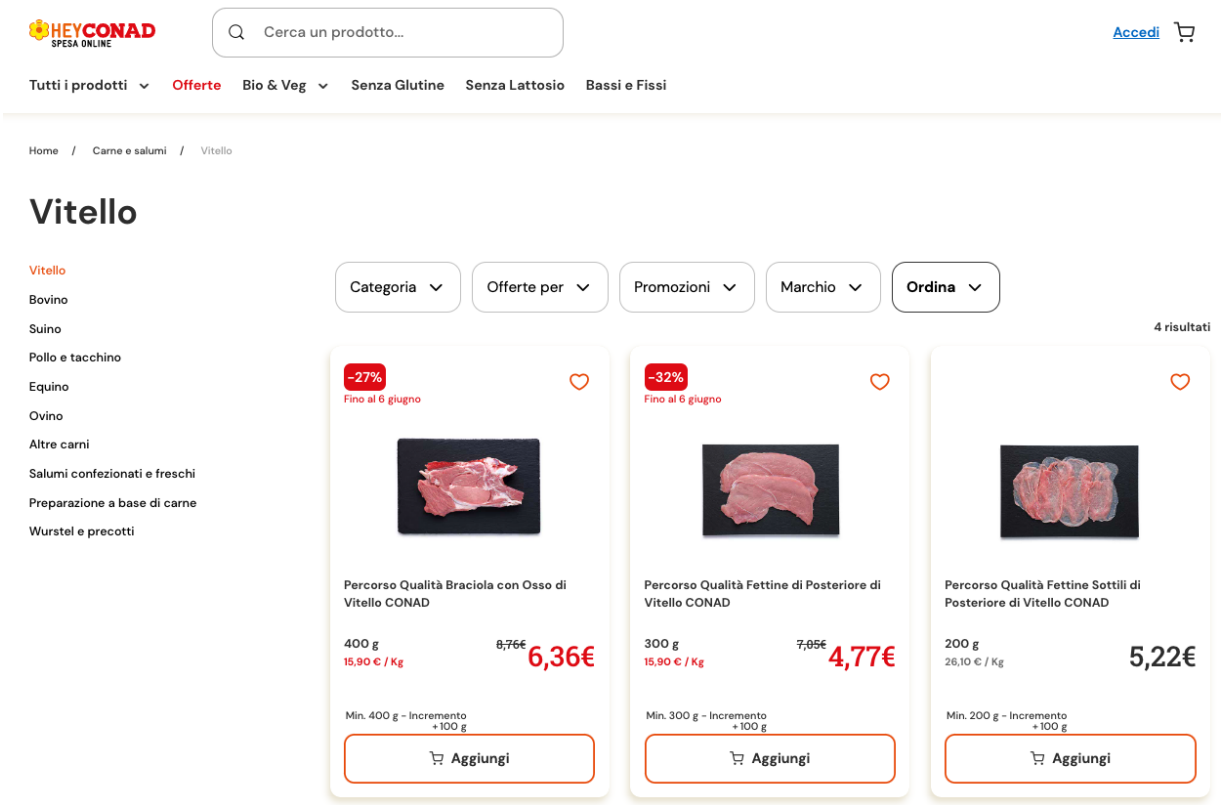

Figure 6: Conad products

Product	Price	Category	Subcategory
arance valencia italia cal. 7 1.5 kg percorso qualitá conad	2.65 € / Kg	frutta e verdura	frutta fresca
pompelmi rossi cal. 4/5 cat. i	2.48 € / Kg	frutta e verdura	frutta fresca
limone costa d'amalfi i.g.p. primofiore italia cal. 5 500 g sapori e dintorni conad	5.56 € / Kg	frutta e verdura	frutta fresca
limoni verdello italia 0.500 kg verso natura conad	3.96 € / Kg	frutta e verdura	frutta fresca
percorso qualitá limoni italia cal. 4 cat. i rete 500g conad	2.78 € / Kg	frutta e verdura	frutta fresca
mele golden delicious italia cal. 70/75 1500 g percorso qualitá conad	1.46 € / Kg	frutta e verdura	frutta fresca
mele red delicious italia cal. 70/75 1500 g percorso qualitá conad	1.46 € / Kg	frutta e verdura	frutta fresca
mele red delicious val venosta mela alto adige igp cal. 80-85 mm 0.90 kg sapori e dintorni conad	3.09 € / Kg	frutta e verdura	frutta fresca

Table 2: Scraping Conad dataset

2.4 Scraping on Carrefour

2.4.1 Selenium

To perform scraping, we must install Selenium. To install Selenium, we use Python's package management system called *pip*. The specific command to use is *!pip install Selenium*. When executed, this command installs the Selenium package in our Python development environment.

2.4.2 Webdriver initialization and home page loading

To begin browsing with Selenium and go to Carrefour website, code 10 was implemented. In code 10, a fresh Chrome browser session is launched with the *webdriver.Chrome()* function. The resulting browser object is then assigned to the driver variable, which will be used to manage and interact with the browser. Next, the *driver.get(Carrefour_home)* instruction will open the *Carrefour_home* URL webpage within the browser controlled by driver. In this particular instance, the URL takes you to the primary page of Carrefour's online store.

```
driver = driver = webdriver.Chrome()
Carrefour_home = 'https://www.carrefour.it/spesa-online/'
driver.get(Carrefour_home)
```

Codice 10: Initializing web driver for Carrefour

2.4.3 Categories

In the first stage, the function *extract_category(link)* is defined. Its purpose is to extract category names from product links using a regular expression. If the URL format is not as expected, it returns *None*. Next, code 11 fetches the HTML of the current webpage using the browser driver and creates a BeautifulSoup object named *soup* that allows HTML navigation and parsing. Using BeautifulSoup, we can identify all *li* HTML elements with the class *card glide_slide*, which represent different product categories available on the webpage. The code then iterates through these elements to extract relevant data, such as the link and category name associated with each. The extracted data is stored in the *categories_links* and *categories* lists. The code then cleans up the *categories_links* by appending *https://www.carrefour.it* to any links where necessary, removing duplicates, and replacing any hyphens in category names with spaces to ensure a consistent format.

```
# Category extraction function
def extract_category(link):
    match = re.search(r'^((?:[^/]*/){4})([^/]*)', link)
    if match:
        return match.group(2)
    return None
# Get page HTML
html = driver.page_source
soup = BeautifulSoup(html, 'html.parser')
```

```
# Finding all categories tags
categories_tags = soup.find_all('li', class_ = 'card glide__slide')
# Blank list dor links
categories_links = []
# Iterating on all categories
for i in categories_tags:
    link = i.find('a').get('href')
    categories_links.append(link)
    category = i.find('span').get_text()
    categories.append(category)
# Fixing links
categories_links=['https://www.carrefour.it'+item for item in categories_links
       if 'www.carrefour' not in item]
categories_links = list(set(categories_links))
# Calling extract category function
categories = [extract_category(i) for i in categories_links]
categories = [i.replace('-', ' ') for i in categories]
```

Codice 11: Carrefour categories

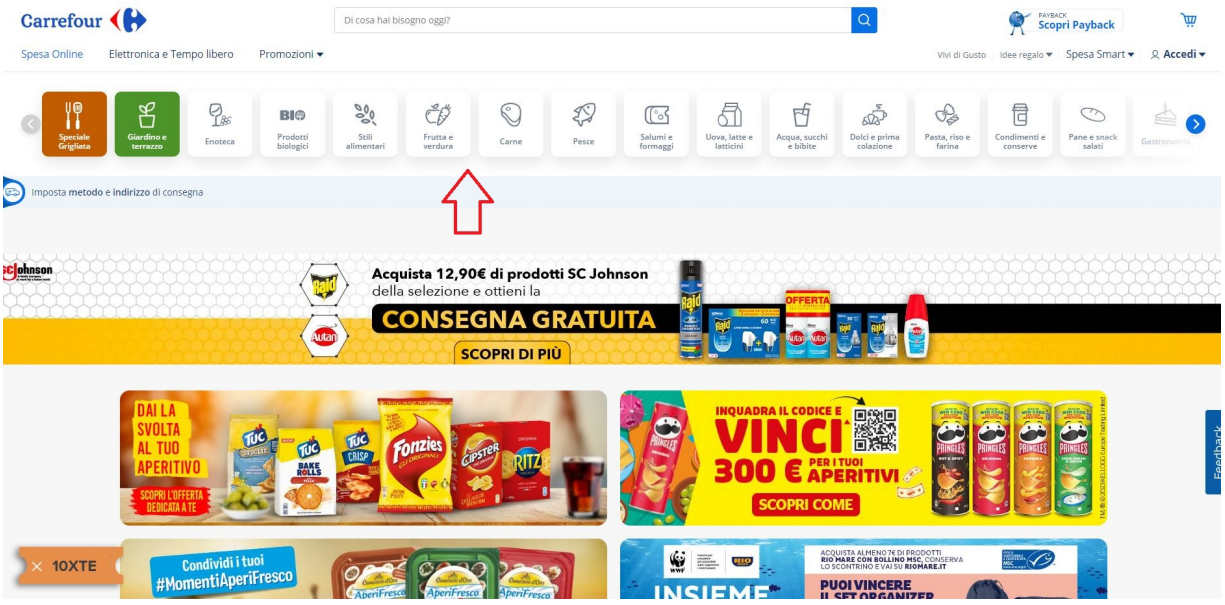

Figure 7: Carrefour categories

2.4.4 Products and prices

To identify the products displayed on a webpage, the *find_products(soup)* function utilizes a BeautifulSoup object that represents the webpage's HTML structure. The function begins by locating all *h3* tags with the *tile-description* class, which signify the product names. It then proceeds to extract the text from each of these elements, representing the product name, and adds it to the products list. Finally, the function returns the products list containing all the product names identified on the webpage. During our gathering of pricing data, we encountered a challenge with products that were discounted: the price tag would appear twice. Our solution is the *find_prices_not_discounted(soup)* function shown in code 12. This function filters out discounted prices, ensuring that each product has only one corresponding price.

```
# Product function
def find_products(soup):
    prods = soup.find_all('h3', class_ = 'tile-description')
    products = []
    for tag in prods:
        product = tag.get_text()
        products.append(product)
    return products
# Prices function
def find_prices_not_discounted(soup):
    all_prices = soup.find_all('span', class_ = 'unit-price')
    filtered_prices = []
    for tag in all_prices:
        if tag.find_parent(class_='sales discounted') is None:
            price = tag.get_text()
            filtered_prices.append(price)
    return filtered_prices
# Dealing with n.a. product function
def extract_not_available_product(soup):
    span = soup.find_all('span', class_='value', content='false')
    na_prods = []
    for i in span:
        a = i.parent.parent.parent.parent.parent
        b = a.find('h3', class_ = 'tile-description')
        c = b.get_text()
```

```
26          na_prods.append(c)
```

Codice 12: Carrefour products and prices

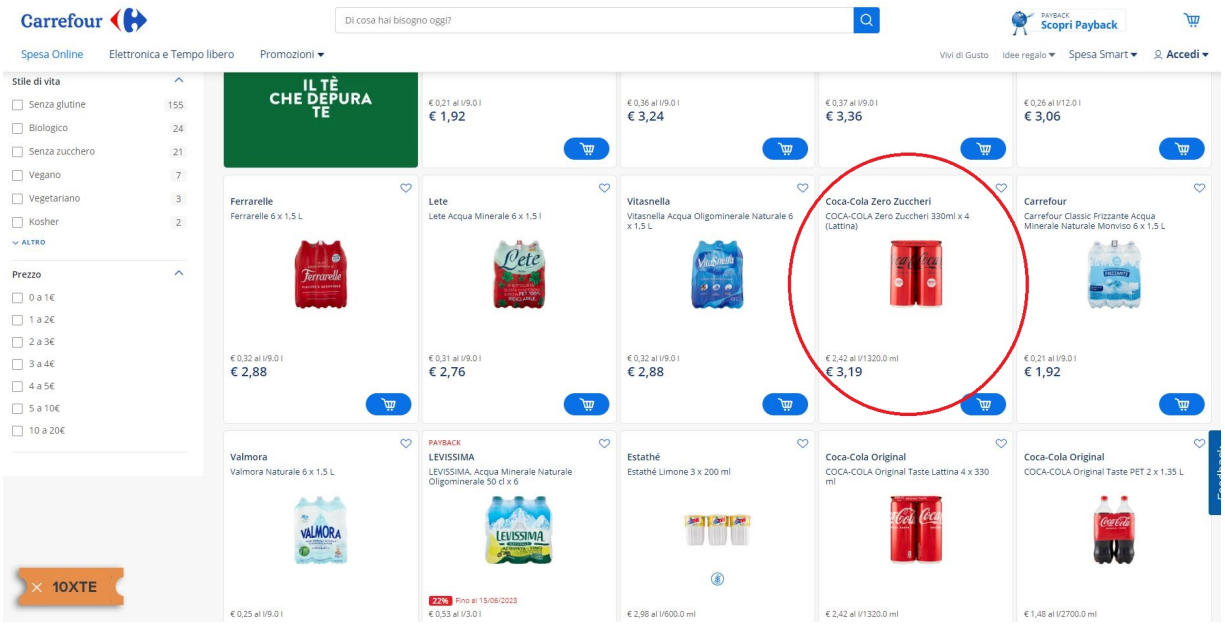

Figure 8: Carrefour products

2.4.5 Dealing with *Null*

During our data collection process, we experienced a second issue. At times, due to fast page loading, certain anomalies occurred. When the *scroll_down()* function was running, the price of some products would not load correctly, and instead, the HTML would display a *null* designation. This issue was sporadic and affected random products. However, reloading the page was usually sufficient to resolve the *null* error. To tackle this problem when it occurred, we created an additional list to track these products. We developed the *extract_not_available_product(soup)* function for this purpose (code 12). This function starts by identifying all *span* elements with the class *value* and content *false*, indicating a null product price. For each of these elements, the function navigates through parent elements to locate and extract the associated product name, found inside an *h3* tag with the class *tile-description*. The identified product name is then added to the *na_prods* list. Using this approach, we could maintain a record of products whose price couldn't be loaded. This record could aid in further investigation or alternative handling of such instances.

2.4.6 Creating Carrefour dataset

In this part of the code, we go over all the product categories in our pre-defined list of categories links. We access the webpage of each category and gather information about all the products within that category. To achieve this, we first instruct the web driver to navigate to the desired webpage and scroll down twice using the *scroll_down()* function. This scrolling action is necessary because some websites use dynamic loading, which means not all content is loaded at once. Scrolling down ensures that all products have a chance to load onto the page. After the page is fully loaded, we capture the entire page's HTML source code. We parse this HTML code into a BeautifulSoup object to create a structured representation of the page content, which allows us to extract information selectively. Using the structured data, we call all the functions we defined earlier. These functions sift through the HTML to find and store the product names, their corresponding non-discounted prices, and the names of any products that weren't available, respectively. We also create a list that replicates the category name for each product in the category, so we can keep track of which category each product belongs to. We then append the identified product names, prices, and categories to their respective lists. If any product was unavailable, its name gets added to the *not_availables* list. This comprehensive collection process ensures we capture all the necessary data for our project. To efficiently organize and export the acquired data into a CSV file, we employ the pandas library. Initially, we establish a blank DataFrame named CarrefourDataset, which will work as our structured dataset containing the collected information. Subsequently, we produce three columns within the DataFrame titled "Product," "Price" and "Category." The values for each column are filled in using the respective lists generated earlier. Once the DataFrame is constructed, we employ the *to_csv()* method to export it to a CSV file. This step is taken to enable easy access to the information for further analysis.

```
# Blank list for output
all_products = []
all_prices = []
not_availables = []
categories_column = []
# Searching all products and prices
for i in range(len(categories_links)):
    link = categories_links[i]
    category = categories[i]
    driver.get(link)
    scroll_down(driver)
```

```
scroll_down(driver)
    html = driver.page_source
    soup = BeautifulSoup(html, 'html.parser')
    cat_products = find_products(soup)
    cat_prices = find_prices_not_discounted(soup)
    cat_not_available = extract_not_available_product(soup)
    cat = [category]*len(cat_products)
    for i in cat_products:
        all_products.append(i)
    for i in cat:
        categories_column.append(i)
    for i in cat_prices:
        all_prices.append(i)
    if cat_not_available:
        for i in cat_not_available:
            not_availables.append(i)

# Creating dataset
CarrefourDataset = pd.DataFrame()
CarrefourDataset['Product'] = all_products
CarrefourDataset['Price'] = all_prices
CarrefourDataset['Category'] = categories_column
```

Codice 13: Creating Carrefour dataset

Product	Price	Category
carrefour correttore a nastro maxi 5x12 mm	1.89 € / pz	articoli per la casa
carrefour confezione di 4 penne a sfera blu penna a sfera medio 4 pz	1.49 € / pz	articoli per la casa
carrefour forbici punte arrotondate 17cm	2.99 € / pz	articoli per la casa
carrefour confezione di 4 penne a sfera nero penna a sfera medio 4 pz	1.49 € / pz	articoli per la casa
carrefour quaderno a4 punto metallico con copertina in cartoncino e rigatura 5mm	1.59 € / pz	articoli per la casa
carrefour 4 penne a sfera colori assortiti	1.49 € / pz	articoli per la casa
carrefour 2 nastri trasparenti 25mx19mm	1.99 € / pz	articoli per la casa
carrefour 2 colle stick 20g-20g	3.19 € / pz	articoli per la casa

Table 3: Scraping Carrefour dataset

3 Data preprocessing

Before proceeding with the dataset integration part, it is necessary to standardize all the collected data so that they can be properly processed.

3.1 Esselunga

Regarding the Esselunga data, the first step taken was to fix the price column: the price data was not in the desired format. To fix this, we created a function called *format_price_unit*. This function properly formats the price information by splitting the price data based on the "/" character, separating the price and unit parts, and then reformatting them into a string that follows the format: "price € / unit". Before applying this function, we removed any rows with missing price data using the *dropna()* method on the "Price" column to prevent errors during the transformation process. Afterward, we applied the *format_price_unit function* to the "Price" column and converted the Product and Category columns to lowercase characters. Code 14 shows the function that format all prices in the Esselunga dataset. Next, we refined the extracted quantity data by deleting or converting specific units of measurement and substituting commas in the column with periods.

```
# Format prices Esselunga
def format_price_unit(row):
    row = str(row)
    parts = row.split("/")
    prezzo_part = parts[0].split()
    unita_part = parts[1].split()
    if "" in prezzo_part:
        prezzo = prezzo_part[0]
    if "Euro" in prezzo_part:
        prezzo = prezzo_part[1]
    unita = unita_part[0]
    return f"{prezzo}  / {unita}"
```

Codice 14: Standardizing prices in Esselunga

In addition to handling prices, we also had to standardize all string variables. This was achieved by calling the *.lower()* method to convert them to lowercase (code 15).

```
esselunga['Price'] = esselunga['Price'].apply(format_price_unit)
```

```
esselunga['Product'] = esselunga['Product'].str.lower()
esselunga['Category'] = esselunga['Category'].str.lower()
esselunga = esselunga.dropna(subset=['Price'])
```

Codice 15: Formatting Esselunga values

We extracted products quantities by exploiting regular expressions (code 16).

```
## 4. Estrazione quantita
pattern = r'((?:\d+,\d+|\d+|\d+\s*x\s*(?:\d+,\d+|\d+\.?\d+))\s*(?:g|kg|Kg|l|L|
    ml|ML|pz|ml|gr|buste|fogli|guanti|paio|pezzi|
    salviette|sacchi|m|cm|cl|litro|uova)\b)'
# Applying pattern
esselunga['Quantity'] = esselunga['Product'].apply(lambda x:
                          re.findall(pattern, x)[-1] if
           len(re.findall(pattern, x)) > 0 else None)
### Removing unwanted unit measurements
esselunga['Quantity'] = esselunga['Quantity'].apply(lambda x:
                                None if 'cm' in str(x) else x)
esselunga['Quantity'] = esselunga['Quantity'].apply(lambda x:
                                None if 'mm' in str(x) else x)
esselunga['Quantity'] = esselunga['Quantity'].apply(lambda x:
                                None if 'm' in str(x) else x)
### Uniforming quantities
esselunga['Quantity'] = esselunga['Quantity'].str.replace('pezzi', 'pz')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('Kg', 'kg')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('L', 'l')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('ML', 'ml')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('buste', 'pz')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('fogli', 'pz')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('Ml', 'ml')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('gr', 'g')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('fogli', 'pz')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('paio', 'pz')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('sacchi', 'pz')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('uova', 'pz')
esselunga['Quantity'] = esselunga['Quantity'].str.replace('sacchi', 'pz')
# Replacing comma
```

```
esselunga['Quantity'] = esselunga['Quantity'].str.replace(',','.')
```

Codice 16: Uniforming quantities in Esselunga

We utilized the process_quantity() and convert_to_SI() functions (code 17) to calculate the multiplication in the Quantity column and standardize the units of measurement. However, even after getting rid of unnecessary columns and unifying the required ones, we encountered issues with the price format. Specifically, for detergents, the price was listed in scoops with the total number of scoops included in the price. To address this, we developed a function that uses regex to search for the number of scoops in the Product name and perform a multiplication with the price per scoop listed in the price column (code 19). Additionally, we will substitute the term 'ms' with 'pz' to indicate the price per unit.

```
### definisco funzione per conversione unit di misura in: kg, l, pz
def convert_to_SI(row):
    unit = row['Quantity_unit di misura']
    value = row['Quantity_num_giusto']

    if unit == 'g':
        return value / 1000, 'kg' # convert grams to kilograms
    elif unit == 'ml':
        return value / 1000, 'l' # convert milliliters to liters
    elif unit in ['kg', 'l', 'pz']: # keep these as they are
        return value, unit
    else:
        return value, unit
```

Codice 17: Unit measurements conversion

```
esselunga[['Quantity_SI', 'Quantity_unit di misura_SI']] =
    sselunga.apply(convert_to_SI, axis=1, result_type='expand')
esselunga.head()
### unifico le colonne e elimino quelle che non servono
esselunga['Quantity_final'] = esselunga['Quantity_SI'].astype(str) +
       " " +
       esselunga['Quantity_unit di misura_SI']
esselunga = esselunga.drop('Quantity_unit di misura_SI', axis = 1)
esselunga = esselunga.drop('Quantity_num_giusto', axis = 1)
esselunga = esselunga.drop('Quantity_SI', axis = 1)
```

```
esselunga = esselunga.drop('Quantity_unit di misura', axis = 1)
esselunga = esselunga.drop('Quantity_processed', axis = 1)
esselunga = esselunga.drop('Quantity', axis = 1)
esselunga = esselunga.rename(columns={'Quantity_final':'Quantity'})
# Replacing comma
esselunga['Price'] = esselunga['Price'].str.replace(',','.')
esselunga.head()
```

Codice 18: Applying unit measurements conversion

```
## Risolvo problema dei detersivi per esselunga (prezzo = /misurino)
def adjust_detergents(row):
    if 'ms' in row['Price']:
        price = float(re.search('(\d+\.\d+)', row['Price']).group())
        quantity = float(re.search('(\d+)', row['Product']).group())
        price *= quantity
        return f'{price:.2f}  / pz'
    else:
        return row['Price']

esselunga['Price'] = esselunga.apply(adjust_detergents, axis=1)
# Deleting duplicates
esselunga = esselunga.drop_duplicates(subset=['Product'], keep='first')
esselunga = esselunga.reset_index()
```

Codice 19: Final steps for Esselunga preprocessing

Finally, we dropped the columns that contained NaN values in the price. Some price values were missing on the website and were filled with null values.

3.2 Carrefour

For Carrefour, the same steps were followed. A different price formatting function was written, which is shown in code 20. Next, we will proceed with the preprocessing of the Carrefour dataset. Similar to our previous approach, we will concentrate on effectively extracting and formatting the quantity and price per unit.

In order to accomplish this, we have created the following functions:

1. `format_price_unit_2(row)` function: we utilized this feature to arrange the details regarding 'price per unit'. The process includes splitting the given row (which is converted into

a string) into parts and extracting the price and the corresponding unit of measurement. The final output is in the format of "price € / unit".

2. `get_quantity_carrefour(row)` function: This function, like the previous one, divides the provided row into sections, but it only extracts the last two sections that indicate the amount and measurement unit. The output will be in the following format: "quantity unit".

3. `format_quantity_carrefour(row)` function: In the previous step, this feature examines the quantity string. Its purpose is to identify specific units of measurement within the string. If any units are present, it converts them to the standard measurement system (SI system). This includes converting grams (abbreviated as 'g' or 'gr') to kilograms ('kg'), and millilitres (abbreviated as 'ml' or 'cl') to liters ('l'). The resulting format is "quantity unit".

```
#Reading the files and ordering columns
cols = ['Product','Price','Quantity','Category']
carrefour = pd.read_csv('https://raw.githubusercontent.com/CoroTheBoss/
                    DM-project/main/carrefour_final.csv', index_col=0)

## funzioni
def format_price_unit_2(row):
    row = str(row)
    parts = row.split()
    prezzo = parts[1]
    unita = parts[3]
    return f"{prezzo}  / {unita}"
def get_quantity_carrefour(row):
    row = str(row)
    parts = row.split()
    quantity = parts[-2]
    unita = parts[-1]
    return f"{quantity} {unita}"
def format_quantity_carrefour(row):
    row = str(row)
    parts = row.split()
    if 'g' in parts:
        parts[0] = float(parts[0])/1000
        parts[1] = 'kg'
    if 'ml' in parts:
        parts[0] = float(parts[0])/1000
```

```
        parts[1] = 'l'
    if 'gr' in parts:
        parts[0] = float(parts[0])/1000
        parts[1] = 'kg'
    if 'cl' in parts:
        parts[0] = float(parts[0])/100
        parts[1] = 'l'
    out = str(parts[0]) + ' ' + parts[1]
    return out
```

Codice 20: Standardizing prices and quantities in Carrefour

We then executed the data functions and cleaned the Carrefour dataset. To make the data more manageable, we reset the DataFrame's index and removed unnecessary columns like 'Unit Price [€]', 'Unit Type', 'Quantity [units]', 'Quantity', 'Unit of measure', and 'Product Price [€]'. This step helped us narrow down the dataset to only the most pertinent information.

After that, we focused on transforming the 'unit Price' column. We eliminated newline characters and slashes from the strings, substituted them with spaces, and renamed the column to 'Price'.

After cleaning the 'Price' column, we utilized the `get_quantity_carrefour` function to extract quantity information from the price details. Next, we processed the quantities with the `format_quantity_carrefour` function to ensure consistent formatting in the Standard International (SI) system of units.

To standardize the 'Product' and 'Category' columns, we converted them to lowercase. This standardization enables easier analysis by avoiding discrepancies in case.

Continuing with the 'Price' column, we used the `format_price_unit_2` function to format the price information as desired. We also replaced commas with dots in the 'Price' strings to enable accurate numeric computations in later analysis with Python.

To improve the DataFrame's readability and usability, we reordered the columns according to the predefined order stored in the 'cols' variable. The Carrefour dataset is now fully preprocessed and standardized for comparison and analysis alongside other supermarkets' data.

Finally, we removed duplicates and reset the index.

```

# Dropping unwanted columns, resetting index and applying functions
carrefour = carrefour.reset_index(drop=True)
carrefour = carrefour.drop(['Unit Price []'], axis=1)
carrefour = carrefour.drop(['Unit Type'], axis=1)
```

```
carrefour = carrefour.drop(['Quantity [units]'], axis=1)
carrefour = carrefour.drop(['Quantity'], axis=1)
carrefour = carrefour.drop(['Unit of measure'], axis=1)
carrefour = carrefour.drop(['Product Price []'], axis=1)
carrefour['unit Price'] = carrefour['unit Price'].str.replace('\n',' ')
carrefour['unit Price'] = carrefour['unit Price'].str.replace('/',' ')
carrefour = carrefour.rename(columns={'unit Price': 'Price'})
carrefour['Quantity'] = carrefour['Price'].apply(get_quantity_carrefour)
carrefour['Quantity'] = carrefour['Quantity'].apply(format_quantity_carrefour)
# Lowering strings
carrefour['Product'] = carrefour['Product'].str.lower()
carrefour['Category'] = carrefour['Category'].str.lower()
carrefour['Price'] = carrefour['Price'].apply(format_price_unit_2)
# Replacing comma
carrefour['Price'] = carrefour['Price'].str.replace(',','.')
carrefour = carrefour[cols]
carrefour.head()
# Deleting duplicates
carrefour = carrefour.drop_duplicates(subset=['Product'], keep='first')
carrefour = carrefour.reset_index()
```

Codice 21: Formatting Carrefour values

3.3 Conad

Our current task is to preprocess data from the Conad supermarket's website. Our starting point was a dataset comprising details about Conad's products, such as their price, category, and subcategory. However, the original column names were inconsistent with other datasets. Therefore, we renamed these columns using pandas rename() method to ensure uniformity.

```
conad = pd.read_csv('https://raw.githubusercontent.com/CoroTheBoss/
                    DM-project/main/conad_final.csv')
conad = conad.drop(['unit_price'], axis=1)
conad = conad.rename(columns={'products': 'Product',
                    'prices_per_unitmeasurement': 'Price',
                  'category': 'Category',
                    'subcategory': 'Subcategory'})
```

Codice 22: Formatting Conad values

Next, we needed to determine the quantity of each product from its description. We accomplished this by utilizing a regular expression (regex) to identify and extract quantities.

```
pattern = r'((?:\d+,\d+|\d+|\d+\s*x\s*(?:\d+,\d+|\d+\.?\d+))\s
                (?:g|kg|Kg|l|L|ml|ML|pz|ml|gr|buste|fogli
                |guanti|paio|pezzi|salviette|sacchi|m|cm|cl|litro|uova)\b)'
conad['Quantity'] = conad['Product'].apply(lambda x:
                re.findall(pattern, x)[-1] if
                len(re.findall(pattern, x)) > 0 else None)
```

Codice 23: Regex pattern

This design captures various numerical formats that indicate quantities such as "1,5", "2", or "3 x 2,5". It is then followed by a measurement unit or packaging type such as "g", "kg", "L", "ml", "pz", and others.

We discovered that different units of measurement were utilized to express quantities after analysing the data. To ensure ease of comparison and standardize the units of measure, we opted to standardise them. For instance, we substituted terms like 'buste', 'fogli', 'guanti', 'paio', 'pezzi', 'sacchi', 'salviette', 'uova' with 'pz' (pieces) and 'gr' with 'g' (grams) using the pandas str.replace() method. Additionally, we replaced commas with points using the same method.

```
## Stampo il set di unit di misura presenti nel dataset e inizio a uniformare
conad['Quantity'] = conad['Quantity'].str.replace('buste', 'pz')
conad['Quantity'] = conad['Quantity'].str.replace('fogli', 'pz')
conad['Quantity'] = conad['Quantity'].str.replace('guanti', 'pz')
conad['Quantity'] = conad['Quantity'].str.replace('paio', 'pz')
conad['Quantity'] = conad['Quantity'].str.replace('pezzi', 'pz')
conad['Quantity'] = conad['Quantity'].str.replace('sacchi', 'pz')
conad['Quantity'] = conad['Quantity'].str.replace('salviette', 'pz')
conad['Quantity'] = conad['Quantity'].str.replace('uova', 'pz')
conad['Quantity'] = conad['Quantity'].str.replace('gr', 'g')
conad['Quantity'] = conad['Quantity'].str.replace('litro', 'l')
## Sostituisco virgola con punto (serve dopo per convertire in float)
conad['Quantity'] = conad['Quantity'].str.replace(',', '.')
```

Codice 24: Substitution

We first cleaned and standardised the "Quantity" column. Afterwards, we created a function to further process the quantities. During this process, we discovered that certain quantities were

expressed in a multiplicative format, such as "3 x 500g". Additionally, we observed that some numerical values lacked spacing between them and the unit of measure. In order to address such scenarios, we created the *process_quantity()* function which performs the following actions:

- Inserts a space between a number and any accompanying letters if it is absent
- If the string 's' includes the letter 'x', which indicates multiplication, the function separates the string into two segments, multiplies the two numbers together, and then adds the unit of measurement to the outcome

```
## Creo funzione che esegua le moltiplicazioni
def process_quantity(s):
    if s is None:
        return None
    # Aggiungi spazio tra numero e lettere se non c'
    s = re.sub(r'(\d)([a-zA-Z])', r'\1 \2', s)
    # Risolvi la moltiplicazione se presente
    if 'x' in s:
        parts = s.split('x')
        result = float(parts[0].strip()) * float(parts[1].split()[0].strip())
        unit = parts[1].split()[1]
        return str(result) + " " + unit
    else:
        return s

conad['Quantity_processed'] = conad['Quantity'].apply(process_quantity)
```

Codice 25: Processing quantities

In our data analysis, we used pandas' apply() method to apply a function on the 'Quantity' column. We wanted to refine the 'Quantity' information by separating the numerical quantity from the measurement unit and converting some units to the SI system.

```
## Creo colonne con quantit e unita di misura
conad[['Quantity_num_giusto', 'Quantity_unit di misura']] =
       conad['Quantity_processed'].str.rsplit(' ', 1, expand=True)
# Leggo numeri come float e sostituisco virgola con punto
conad['Quantity_num_giusto'] = conad['Quantity_num_giusto'].apply(lambda x:
```

```
6         float(x.replace(',','.')) if x is not None else x)
```

Codice 26: Cleaning

To do this, we split the 'Quantity_processed' column into two columns: one for the numeric value and another for the unit of measurement. We then converted the value to float and applied the convert_to_SI() function, focusing on 'kg', 'l', and 'pz' units.

```
## Converto le unit di misura
def convert_to_SI(row):
    unit = row['Quantity_unit di misura']
    value = row['Quantity_num_giusto']

    if unit == 'g':
        return value / 1000, 'kg' # convert grams to kilograms
    elif unit == 'ml':
        return value / 1000, 'l' # convert milliliters to liters
    elif unit in ['kg', 'l', 'pz']: # keep these as they are
        return value, unit
    else:
        return value, unit # if you want to keep other units as they are
conad[['Quantity_SI', 'Quantity_unit di misura_SI']] =
         conad.apply(convert_to_SI, axis=1, result_type='expand')
```

Codice 27: Unit measurement standardization function

Next, we combined the processed quantity and its unit to create a new column 'Quantity_final'. We then removed unnecessary columns, including temporary ones used during preprocessing and the 'Subcategory' column.

```
## Unisco le nuove colonne ed elimino quelle non pi necessarie
conad['Quantity_final'] = conad['Quantity_SI'].astype(str) + " " +
       conad['Quantity_unit di misura_SI']
conad = conad.drop('Quantity_unita di misura_SI', axis = 1)
conad = conad.drop('Quantity_num_giusto', axis = 1)
conad = conad.drop('Quantity_SI', axis = 1)
conad = conad.drop('Quantity_unita di misura', axis = 1)
conad = conad.drop('Quantity_processed', axis = 1)
conad = conad.drop(['Quantity', 'Subcategory'], axis = 1)
conad['Price'] = conad['Price'].str.replace(',','.')
```

```
conad = conad.rename(columns={'Quantity_final':'Quantity'})

conad = conad.drop_duplicates(subset=['Product'], keep='first')
```

Codice 28: Final steps for Conad preprocessing

Finally, we renamed the final quantity column and replaced commas with points in the Price column.

4 Data integration

Regarding the data integration, it was necessary to find an efficient way to match the products from the three different supermarkets. The matches were made using cosine similarity. Cosine similarity is a metric used to measure the similarity between two vectors. This metric calculates the cosine of the angle between the vectors, which represents their orientation or direction in relation to each other. In the context of matching products, cosine similarity can be applied by representing each product as a vector, where each dimension corresponds to a specific feature or attribute of the product. By calculating the cosine similarity between two product vectors, we can determine how similar or closely related they are based on the words that compose each product description. Higher cosine similarity values indicate a greater degree of similarity between the vectors. A cosine similarity value of 1 indicates that the two vectors are identical or perfectly similar, while a value of 0 indicates no similarity or orthogonality.

4.1 Preliminary steps

The first step (code 29 and code 30) was to import libraries and read the three datasets from their *.csv* files.

```
# Importazione librerie
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
import pandas as pd
import re
```

Codice 29: Libraries for integration

```
Esselunga = pd.read_csv('https://raw.githubusercontent.com/CoroTheBoss/
                DM-project/main/esselunga_preprocessed.csv')
Carrefour = pd.read_csv('https://raw.githubusercontent.com/CoroTheBoss/
                DM-project/main/carrefour_preprocessed.csv', index_col=0)
Conad = pd.read_csv('https://raw.githubusercontent.com/CoroTheBoss/
                DM-project/main/conad_preprocessed.csv')
```

Codice 30: Reading datasets for integration

4.2 Matching all possible combinations

In order to achieve successful data integration, we needed to take into account data from various sources. As we were working with data from three different supermarkets, we required a strategy that would enable us to efficiently merge and analyse these distinct datasets. To accomplish this and find matching products, code 31 creates combinations of possible pairs between the supermarkets (Carrefour - Esselunga, Carrefour - Conad and Conad - Esselunga). It utilizes the TF-IDF vectorization technique to calculate cosine similarity matrices for each pair of supermarkets.

```
# Prendo le colonne "Product" dai dataset
products1 = Carrefour['Product'].tolist()
products2 = Conad['Product'].tolist()
products3 = Esselunga['Product'].tolist()

# Calcolo lunghezze prodotti dei singoli supermercati
len_ca = len(products1)
len_co = len(products2)
len_es = len(products3)

# Creo liste possibili coppie
ca_co = products1 + products2
ca_es = products1 + products3
co_es = products2 + products3

# Calcolo la matrice di similarita'
vectorizer = TfidfVectorizer()
ca_co_vectors = vectorizer.fit_transform(ca_co)
sim_ca_co = cosine_similarity(ca_co_vectors)
```

```
ca_es_vectors = vectorizer.fit_transform(ca_es)
sim_ca_es = cosine_similarity(ca_es_vectors)
co_es_vectors = vectorizer.fit_transform(co_es)
sim_co_es = cosine_similarity(co_es_vectors)

# Defining column order for output
cols = ['Esselunga', 'Carrefour', 'Conad',
       'Esselunga price', 'Carrefour price', 'Conad price',
       'Esselunga category', 'Carrefour category', 'Conad category',
       'Quantity']
```

Codice 31: Computing cosine similarity for all possible combinations

Using a similarity threshold of 0.7, code 32 identifies the most similar products between each pair and selects the matching products along with their corresponding prices.[2]

```
# Blank list for carrefour-conad matches
match_ca_co = []
# Iteration for carrefour products
for i in range(len_ca):
    temp = [] # temporary list
    # Iteration for conad products
    for j in range(len_co):
        # Similarity threshold and price equality condition
        if sim_ca_co[i][len_ca+j]>=0.7 and Carrefour.loc[i, "Quantity"] ==
                                                Conad.loc[j, "Quantity"]:
            # Appending all match
            temp.append((sim_ca_co[i][len_ca+j],products2[j],j))
    if temp:
        # Selecting the highest similarity match
        prod = max(temp, key=lambda x: x[0])[1]
        index_price = max(temp, key=lambda x: x[0])[2]
        # Appending match pair for final output
        match_ca_co.append((products1[i],prod,
                            Carrefour.loc[i, "Price"],
                            Conad.loc[index_price,"Price"],
                            Carrefour.loc[i, "Category"],
                            Conad.loc[index_price, "Category"],
```

[2]To avoid redundancies since the code is the same we report only the code for the first couple of supermarkets

```
                        Carrefour.loc[i, "Quantity"]))
    else:
        match_ca_co.append((products1[i],np.nan,
                        Carrefour.loc[i, "Price"],
                        np.nan,
                        Carrefour.loc[i, "Category"],
                        np.nan,
                        Carrefour.loc[i, "Quantity"]))
output_ca_co = pd.DataFrame(match_ca_co, columns=['Carrefour', 'Conad',
                                        'Carrefour price', 'Conad price',
                                        'Carrefour category', 'Conad category',
                                        'Quantity'])
```

Codice 32: Selecting matching products based on cosine similarity threshold

4.3 Combining results

The final result is a consolidated dataframe that contains the matched products and prices from all three supermarkets, including those products that only appears in just one two of the three supermarkets. This integrated dataset provides a comprehensive view of similar products across different supermarkets, allowing for analysis and comparison.

Code 33 shows how it was possible to combine the three datasets coming from the cosine similarity condition of code 32. The main idea idea is to compare each couple of those dataframe and see weather there is a match in the remaining dataframe. If a match is found in the third dataframe, it means that the product is common to all three supermarkets. Otherwise, a NaN value is appropriately inserted, indicating that a particular product is not present in a supermarket.[3]

```
final = output_ca_co.copy()
# Iteration over the first dataset
for i in range(len(output_ca_co)):
    # Selecting product
    elem = output_ca_co["Carrefour"][i]
    # Search for product in the second dataset
    if elem in output_es_ca["Carrefour"].tolist():
        # Append product and price
        index = output_es_ca["Carrefour"].tolist().index(elem)
```

[3]To avoid redundancies since the code is the same we report the code only for the first couple of dataframes

```
        final.loc[i, "Esselunga"] = output_es_ca.loc[index, "Esselunga"]
        final.loc[i, "Esselunga price"] = output_es_ca.loc[index,
                                                           "Esselunga price"]
        final.loc[i, "Esselunga category"] = output_es_ca.loc[index,
                                                           "Esselunga category"]
    else:
        # Append NaN if there is no match
        final.loc[i, "Esselunga"] = np.nan
        final.loc[i, "Esselunga price"] = np.nan
        final.loc[i, "Esselunga category"] = np.nan
final = final[cols]
```

Codice 33: Creating integrated dataset - part 1

Carrefour	Conad	Esselunga	Carrefour price	Conad price	Esselunga price
carrefour bio salmone norvegese affumicato 75g	salmone affumicato norvegese 100 g conad	salmone norvegese affumicato 100 g	61.20 € / g	64.90 € / kg	76.90 € / kg
carrefour bio ceci 400 g	ceci 400 g conad	esselunga bio ceci 500 g	"3.96 € / g	4.97 € / kg	4.98 € / kg
carrefour bio fagioli cannellini 400 g	fagioli cannellini 400 g conad	esselunga bio fagioli cannellini 500 g	4.13 € / g	4.97 € / kg	7.58 € / kg
carrefour bio latte parzialmente scremato 1 l	granarolo bio latte biologico parzialmente scremato 1 l	granarolo biologico latte bio parzialmente scremato 1 l	1.35 € / ml	2.19 € / l	2.19 € / l
carrefour bio lenticchie 400 g	lenticchie 400 g conad	NaN	3.96 € / g	2.22 € / kg	NaN
carrefour bio fagioli borlotti 400 g	fagioli borlotti 3 x 400 g conad	esselunga bio fagioli borlotti 500 g	4.13 € / g	1.82 € / kg	5.74 € / kg
carrefour bio latte parzialmente scremato microfiltrato 1 l	più gusto latte uht microfiltrato parzialmente scremato 1 l conad	smart latte microfiltrato parzialmente scremato 1 l	1.75 € / ml	1.29 € / l	1.08 € / l
carrefour bio crescenza 100 g	crescenza 100 g conad	esselunga crescenza 100 g	17.50 € / g	11.50 € / kg	11.90 € / kg

Table 4: Integrated dataset

Finally, we concatenated the three final dataframes to obtain the integrated dataset. After this step, we called the *drop_duplicates* command to remove any duplicates, and then saved the dataframe (code 34).

```
# List of dataframes
pdList = [final, final2, final3]
# Concat dataframes
df = pd.concat(pdList)
# Dropping duplicates
df = df.drop_duplicates(keep='first')
df = df.reset_index()
df.to_csv("integrated.csv")
df
```

Codice 34: Creating integrated dataset - part 2

The final dataframe contains the product matches along with the corresponding prices, categories and quantity.

4.4 Queries

We converted our integrated dataset in order to execute some queries (code 35)

```
# Libraries
import sqlite3
import csv
import pandas as pd
# Converting dataset to SQL
conn = sqlite3.connect('/Users/Chicca/Desktop/Integrated_final.db')
df.to_sql('Integrated', conn, if_exists='replace', index=False)
conn.close()
```

Codice 35: Converting dataset into SQL table

4.4.1 Query 1

A first simple query consist of retrieving all rows from the 'Integrated' table where the columns 'Carrefour', 'Conad', or 'Esselunga' contain the word 'bonduelle' (case-insensitive) anywhere within their respective values.

```
import csv

conn = sqlite3.connect('/Users/Chicca/Desktop/Integrated_final.db')
cursor = conn.cursor()
query = "SELECT * FROM Integrated WHERE
  LOWER(Carrefour) LIKE '%bonduelle%' OR
  LOWER(Conad) LIKE '%bonduelle%' OR
  LOWER(Esselunga) LIKE '%bonduelle%'"
cursor.execute(query)
results = cursor.fetchall()
csv_file = '/Users/Chicca/Desktop/Query1.csv'


with open(csv_file, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow([i[0] for i in cursor.description])
    writer.writerows(results)
```

```
18
19 conn.close()
```

Codice 36: First query

4.4.2 Query 2

Our second query selects rows from the 'Integrated' table where the columns 'Carrefour', 'Conad', or 'Esselunga' contain the word 'noberasco' (case-insensitive). It also filters out rows where any of these columns are null. Additionally, it computes a column called 'Most_Convenient' based on price comparisons between 'Esselunga', 'Carrefour', and 'Conad'. The selected columns and the computed 'Most_Convenient' column are returned as the result of the query.

```
1  conn = sqlite3.connect('/Users/Chicca/Desktop/Integrated_final.db')
2  cursor = conn.cursor()
3
4  query = """
5  SELECT *,
6         CASE
7             WHEN "Esselunga price" <= "Carrefour price" AND
8      "Esselunga price" <= "Conad price" THEN "Esselunga price"
9             WHEN "Carrefour price" <= "Esselunga price" AND
10     "Carrefour price" <= "Conad price" THEN "Carrefour price"
11            ELSE "Conad price"
12        END AS "Most_Convenient"
13 FROM Integrated
14 WHERE (LOWER(Carrefour) LIKE '%noberasco%' OR
15   LOWER(Conad) LIKE '%noberasco%' OR
16   LOWER(Esselunga) LIKE '%noberasco%')
17   AND (Carrefour IS NOT NULL AND
18   Conad IS NOT NULL AND
19   Esselunga IS NOT NULL)
20   AND (LOWER("Carrefour category") LIKE '%frutta%' OR
21   LOWER("Conad category") LIKE '%frutta%' OR
22   LOWER("Esselunga category") LIKE '%frutta%')
23 """
24
25 cursor.execute(query)
26 results = cursor.fetchall()
```

```
csv_file = '/Users/Chicca/Desktop/Query2.csv'

with open(csv_file, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow([i[0] for i in cursor.description])
    writer.writerows(results)

conn.close()
```

Codice 37: Second query

Esselunga	Carrefour	Conad	Esselunga price	Carrefour price	Conad price	Esselunga category	Carrefour category	Conad category	Quantity	Price um	Most Convenient
noberasco bio uva sultanina biologiche 250 g	carrefour uva sultanina 250 g	Uva Sultanina 250 g Conad	11.96	6.32	6.2	spesa bio	frutta e verdura	Frutta e verdura	0.25 kg	€ / kg	6.2
esselunga mandorle pelate 40 g	mandorle pelate 40 g	Noberasco € 0.99 Mandorle Pelate 40 g	24.75	24.75	24.75	frutta e verdura	frutta e verdura	Frutta e verdura	0.04 kg	€ / kg	24.75
esselunga nocciole sgusciate 40 g	nocciole sgusciate 40 g	Noberasco € 0.99 Nocciole Sgusciate 40 g	24.75	24.75	24.75	frutta e verdura	frutta e verdura	Frutta e verdura	0.04 kg	€ / kg	24.75
noberasco bio uva sultanina biologiche 250 g	carrefour bio uva sultanina biologica 250 g	Uva Sultanina 250 g Conad	11.96	10.72	6.2	spesa bio	prodotti biologici	Frutta e verdura	0.25 kg	€ / kg	6.2
noberasco bio uva sultanina biologiche 250 g	carrefour uva sultanina 250 g	Uva Sultanina 250 g Conad	11.96	6.32	6.2	frutta e verdura	frutta e verdura	Frutta e verdura	0.25 kg	€ / kg	6.2

Table 5: Second query

4.4.3 Query 3

Finally, we made a representative query of our initial idea. The query aims to see in which supermarket you would spend less if you needed to buy bananas, pears, and apples. Indeed our third query calculates the minimum prices for each store ("Carrefour price", "Conad price", "Esselunga price") and their sum, considering rows where the columns "Carrefour", "Conad", or "Esselunga" contain the words 'mela', 'pera', and 'banana' (case-insensitive). The results are returned as "min_carrefour", "min_conad", "min_esselunga", and "total".

```
conn = sqlite3.connect('/Users/Chicca/Desktop/Integrated_final.db')
cursor = conn.cursor()

query = """
SELECT MIN("Carrefour price") AS min_carrefour,
       MIN("Conad price") AS min_conad,
       MIN("Esselunga price") AS min_esselunga,
       MIN("Carrefour price") +
       MIN("Conad price") +
       MIN("Esselunga price") AS total
FROM Integrated
WHERE (LOWER("Carrefour") LIKE '%mela%' OR
  LOWER("Conad") LIKE '%mela%' OR
  LOWER("Esselunga") LIKE '%mela%')
```

```
  AND (LOWER("Carrefour") LIKE '%pera%' OR
 LOWER("Conad") LIKE '%pera%' OR
 LOWER("Esselunga") LIKE '%pera%')
  AND (LOWER("Carrefour") LIKE '%banana%' OR
 LOWER("Conad") LIKE '%banana%' OR
 LOWER("Esselunga") LIKE '%banana%');

"""
cursor.execute(query)
results = cursor.fetchall()
csv_file = '/Users/Chicca/Desktop/Query3.csv'
# Saving reslut
with open(csv_file, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow([i[0] for i in cursor.description])
    writer.writerows(results)

conn.close()
```

Codice 38: Third query

min carrefour	min conad	min esselunga	total
8.17	6.63	6.97	21.77

Table 6: Third query

4.5 Data quality

It is now convenient to get a measure of data quality coming from the integrated dataset. In order to evaluate this parameter we decided to take a sample of random observation in the integrated dataset and cont the number of correct and incorrect matches. This procedure has been repeated for three different categories and, for each one, we selected a sample of 50 products. Code 39 shows how we selected the samples.

```
# Selecting sample
quality_esse = df[df['Esselunga category'] == 'latticini, salumi e formaggi']
                .dropna(subset=['Conad', 'Carrefour'],thresh=1).sample(50)
quality_esse.reset_index(drop=True, inplace=True)
```

```
quality_esse
# Stampo le osservazioni per contare eventuali errori nei match
for i in range(len(quality_esse)):
    print(quality_esse.iloc[i]['Esselunga'])
    print(quality_esse.iloc[i]['Carrefour'])
    print(quality_esse.iloc[i]['Conad'])
    print('-------')
```

Codice 39: Evaluating data quality

After counting and verifying the matches for three different categories, we averaged the obtained percentages and derived a data quality value of $87,34$

5 Conclusions

In conclusion, our data scraping and integration project aimed to create an integrated dataset of products from three different supermarkets, focusing on product names, prices, and categories. By employing cosine similarity with a threshold of 70%, we successfully identified matches between products from different supermarkets, allowing us to merge the relevant information into a unified dataset.

Upon querying the integrated dataset, we found that 87% of the identified matches were indeed accurate, demonstrating the effectiveness of our approach. However, it is important to note that some incorrect matches were encountered, primarily stemming from flavor differences. For instance, a vanilla yogurt product was occasionally matched with coconut yogurt product. While these discrepancies represent a challenge in achieving a perfect match, the overall success rate and usefulness of the integrated dataset were satisfying.

Our project can be useful for other users on the web who seek comprehensive product information across multiple supermarkets. By sharing our integrated dataset, we aim to enhance the accessibility and usability of product data, enabling users to make more informed decisions and comparisons.

Moving forward, further refinements could be made to improve the accuracy of matching, particularly when dealing with subtle variations in flavors or other product attributes. Implementing more advanced natural language processing techniques or exploring alternative similarity measures may be worthwhile avenues for future exploration.

Overall, our data scraping and integration project represents a successful effort in consolidating product information from multiple supermarkets. By achieving a high match accuracy rate and considering potential areas for improvement, we are confident that our integrated dataset will prove to be a valuable asset for other users, promoting more effective decision-making.