UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

MASTER 1 ECONOMÉTRIE STATISTIQUES

ANNÉE 2019/2020

Python - Projet de Web Scraping

Auteurs :
EL MEHDI AGUNAOU
ABDELHAK EL KASSIMI
MEHDI PADOVANI
JOVAN STOJANOVIĆ

Encadrant :
CLÉMENT GOULET

6 Décembre 2019

Table des matières

1 Présentation du Web Scraping

1.1 Le Web Scraping, un concept moderne en évolution

Le Web Scraping désigne une procédure qui permet d'extraire rapidement des données disponible sur Internet. Ces données peuvent ainsi être collectés, triés et par la suite traités pour en extraire des informations sur des sujets qui intéressent l'individu réalisant le Scraping. La particularité du Scraping est l'automatisation du processus lui-même : les données ne sont pas extraites "manuellement", en copiant puis collant les données qu'on veut, mais à travers un code spécifique qui permet de trouver un sens à la présentation du site en question et d'en extraire systématiquement les données désirés.
Par conséquent, des millions de données, et indépendamment de leur localisation sur le site pourront être scrappé uniquement à travers cette mise en liaison de caractéristiques communes du code HTML.
Le Scraping peut être statique ou dynamique en fonction de nos besoins : statiquement (e.g. : urllib, Beautiful Soup bs4), on ne conservera que la page HTML, celle que nous traiterions par la suite. Le Scraping dynamique nous permet par contre de naviguer à travers la page web (e.g. : Selenium) comme si nous l'ouvrions sur notre Web browser, et est donc une procédure plus automatique. Le scraping dynamique est beaucoup plus puissant mais aussi par conséquent plus coûteux en terme de mémoire que le scraping statique.

1.2 Un exemple pratique avec Selenium

Avant de se lancer dans le scraping, il faut avoir installer l'interface Python, Selenium et le driver du navigateur Internet que nous utiliserons pour naviguer sur le site.
Ainsi, après avoir choisi les données qu'on veut scrapper, la première étape revient à identifier des caractéristiques communes à l'ensemble de ces données en inspectant la page web. La page d'inspection s'ouvre avec CTRL-SHIFT-I sur Google Chrome, ou en faisant un clic droit puis appuyant sur inspect. Va t-on utiliser le nom des classes, le Xpath ou l'identifiant de l'élément ? La réponse dépendra du site qu'il faut inspecter et des données qui nous intéressent.

Nous allons travailler sur un exemple simple pour comprendre la méthode de scraping avec Selenium : nous allons extraire des informations sur les meilleures universités du monde. Pour ce faire, nous allons utiliser le site du Times magazine qui offre une nouvelle liste avec le ranking en 2020 des plus grandes universités du monde. Donc, pour commencer, il faut lancer Selenium, le driver et ouvrir la page internet où se trouve le ranking :

```
from selenium import webdriver
import pandas as pd
driver = webdriver.Chrome()
driver.get("https://www.timeshighereducation.com/world-university-rankings/
2020/world-ranking#!/page/0/length/-1/sort_by/rank/sort_order/cols/stats")
```

Listing 1 – Lancement de Selenium

La difficulté d'inspection dépend souvent de la constitution de la page web elle même et du langage de balisage : la plupart sont en HTML (formaté avec certaine règles précise) ou XML (avec des règles plus flexibles). Sur la page d'inspection, dans la partie "Elements", on retrouve les informations dont on a besoin. Lorsque nous avons identifié la balise pertinente, il suffit d'intégrer dans notre code l'élément qui contient l'information qui nous intéressent. En observant le code HTML dans notre exemple, nous pouvons nous rendre compte que les universités sont liées par une classe :

```
<a class="ranking-institution-title"
href="/world-university-rankings/university-oxford" data-position="title"
data-mz="">University of Oxford</a>
```

Listing 2 – Le code HTML du site

Il serait donc une bonne idée d'utiliser la classe pour scrapper toutes les universités avec les liens qui renvoient vers le site de description de l'université. Ceci peut être fait de la façon suivante :

```
a = driver.find_elements_by_class_name("ranking-institution-title")
sites = [ ]
for b in a:
    sites.append(b.get_attribute("href"))
```

Listing 3 – Scraping par classe

Ainsi, sont classés dans une liste a les webelement associés à toutes nos universités, dont il faut extraire les différents liens, ce qu'on fait dans la boucle en les classants dans la liste sites.
On a donc les liens avec le descriptif des meilleures universités !

1.3 Utilité et limites du Web Scraping : un enjeu grandissant

Le Web Scraping trouve son utilité dans de nombreuses applications dont :

— La comparaison des prix sur différents marchés dont les prix sont disponibles en ligne (par exemple marché immobilier, des voitures...)
— Peut inclure aussi des recherches plus qualitatives, par exemple sur l'avis des différents clients sur un produit, une campagne marketing, des décisions politiques etc.
— Le suivi en temps réel des nouvelles tendances, par exemple sur les marchés financiers qui sont particulièrement sensibles aux nouvelles informations.

Durant les années récentes, et dans un contexte où se pose de plus en plus la question de l'importance de la protection de données, le web scraping est devenu de plus en plus controversé. Se pose en particulier la question de la légalité de son utilisation dans certains contextes qui peuvent être fragile tel que le scraping de données personnelles, sur des sites tel que Linkedin ou Facebook. Certains sites essayent de se protéger contre le Web Scraping par divers moyen, par exemple avec :

— Une contrainte de temps, qui bloque l'extraction de données si elle est réalisé à une vitesse trop grande, considéré comme la vitesse d'un bot.
— Le fait de rendre difficile l'extraction en encapsulant les données avec Java API.

2 Scraping des sites de vente de voitures

2.1 L'idée du projet

L'idée de notre projet est de scraper différents sites de vente de voitures d'occasion pour en extraire les marques, les modèles, les prix et les liens directs vers les pages web de ces voitures. Nous allons, par la suite, classifier les voitures par leurs prix et utiliser ces données scrapées et classées pour fournir une plateforme de comparaison des prix de voitures des différents sites. Nous avons décidé de scraper 3 sites de ventes de voiture :

— Deux sites de voitures d'occasion à vente régulière en ligne : La Centrale et Carizy.
— Un site de vente en ligne de voitures d'occasion aux enchères : Vpauto.

2.2 Scraping du site "Vpauto" avec Beautiful Soup

Vpauto est un site spécialisé dans la vente aux enchères de véhicules. Dirigé par un commissaire-priseur fictif, le site propose des ventes en ligne et nous avons scrappé les voitures visibles dans la section "acheter une voiture".
Nous utilisons la librairie BeautifulSoup, qui peut être installé à l'aide du pip du gestionnaire de paquets Python ou du gestionnaire de paquets anaconda.

```
pip install BeautifulSoup4
#ou
```

```
conda install beautifulsoup4
```

Listing 4 – Installation de BeautifulSoup

Pour obtenir des informations sur les éléments auxquels nous voulons accéder, nous devons d'abord inspecter la page Web, ce qui nous permettra de visualiser le code HTML. Dans l'éditeur Python, la première étape est d'enregistrer l'URL de la page où se trouve les données dans une nouvelle variable :

```
url='https://vpauto.fr/vehicule/resultats?page=1'
import requests
from bs4 import BeautifulSoup
import pandas as pd
page=requests.get(url)
```

Listing 5 – Extraire le code HTML de la page 1

Nous utlilisons les librairies requests pour récupérer le code HTML de la page, et BeautifulSoup pour scanner ce code afin de nous faciliter la navigation, et pandas pour créer notre Dataframe. Maintenant que nous savons ce dont nous avons besoin pour extraire les données, nous pouvons commencer par analyser le code HTML,en inspectant la page web. On cible le modèle dans le developer tools : il est dans la classe "div".En effet, on peut extraire l'ensemble des modèles dans le site avec la fonction "find-all", ensuite pour ne retenir que le texte nous utilisons "getText" :

```
html=BeautifulSoup(page.text,'html.parser')
modele_elm=html.find_all('div',{'class':'elmt-modele'})
modele=[x.getText() for x in modele_elm]
```

Listing 6 – Récupérer les modèles

Le texte récupéré n'est pas parfait, on se retrouve ainsi avec des problèmes d'encodage et des caractères de plus :

```
 '\nKuga 1.5 TDCi 120 S&S 4x2 Powershift ST-Line\n\n2017 -\n
    50704 Km\n\n',
 '\nCX-5 2.2L Skyactiv-D 175 ch 4x4 BVA6 Selection\n\n2015 -\n
    75743 Km\n\n']
```

Listing 7 – Texte récupéré

Pour remédier à ce problème, et isoler les modèles des voitures il faut utiliser x.replace :

```
modele=[x.replace('\n','') for x in modele]
```

Listing 8 – Nettoyage du texte

On obtient :

```
  'Kuga 1.5 TDCi 120 S&S 4x2 Powershift ST-Line2017 -                    50704 Km'
    ,
 'CX-5 2.2L Skyactiv-D 175 ch 4x4 BVA6 Selection2015 -                    75743 Km
```

Listing 9 – Modèles des voitures

Nous appliquons la même procédure pour les prix et les marques des voitures :

```
marque_elm=html.find_all('div',{'class':'elmt-marque'})
marque=[x.getText() for x in marque_elm]
marque=[x.replace('\n','') for x in marque]
prix_elm=html.find_all('span',{'class':'prix'})
prix=[x.getText() for x in prix_elm]
prix=[x.replace('€','') for x in prix]
```

Listing 10 – Prix et marques des voitures

On obtient donc :

— pour les marques :

```
'FORD',
'MAZDA'
```

— pour les prix :

```
'16000',
'15800'
```

Après la construction de nos trois listes avec la marque, le modèle et le prix, nous allons ensuite les regrouper dans une dataframe nommée 'voiture' :

```
voiture=pd.DataFrame({'marque':marque,'modele':modele,'prix de vente':prix})
```

Marque	Modele	Prix de vente
AUDI	SQ5 V6 3.0 TFSI 354 Tiptronic 8 Quattro2018	59300

2.3 Scraping du site "La Centrale" avec Selenium

Nous commençons par importer les librairies nécessaires. Nous importons aussi 'Options' cette fois, qu'on utilisera pour scraper les sites sans ouvrir les pages. 'Options' est un attribut de driver important pour scraper "La Centrale" car on naviguera dans 55 pages du sites ce qui prend beaucoup de temps si on laisse Selenium ouvrir ces pages web automatiquement.

```
import pandas as pd
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
```

Listing 11 – Importation des librairies

Nous créons des listes vides où l'on mettra les données scrapées et une liste avec 55 marques les plus connues (choisis manuellement) qu'on utilisera pour changer le lien du site "Lacentrale".

```
marques_index=["ABARTH","BMW","MERCEDES","CITROEN","OPEL","VOLKSWAGEN","RENAULT","
    HYUNDAI","FORD","PEUGEOT","DACIA","NISSAN","AUDI","TOYOTA","MERCEDES-BENZ","SKODA"
    ,"ISUZU","MAZDA","FIAT","HONDA","ALFA ROMEO","MINI","SUZUKI","KIA","SAAB","
    CHEVROLET","SEAT","MITSUBISHI","LAND ROVER","JEEP","MAN","CHRYSLER","FERRARI","
    VOLVO","PORSHCE","ASTON MARTIN","BENTLEY","BUGATTI","CADILLAC","CHEVROLET USA","
    DAEWOO","DODGE","JAGUAR","LADA","LANCIA","LEXUS","LOTUS","TESLA","SMART","INFINITY
    ","SUBARU","ROLLS ROYCE","MASERATI","MAYBACH","MC LAREN"]
modeles=[]
prix=[]
hrefs =[]
```

Listing 12 – Création des listes

Nous créons ensuite une boucle principale qui permet de scrapper les prix et les liens de toutes les voitures avec les marques indiquées dans la liste précedemment.

```
for v in marques_index :
    options = Options()
    options.headless = True
    driver = webdriver.Chrome(options=options)
    driver.set_window_size(1440, 900)
    url = "https://www.lacentrale.fr/occasion-voiture-marque-"+v.lower()+".html"
    driver.get(url)
```

Listing 13 – Boucle principale : ouvertures des liens

Options.headless permet de scraper sans ouvrir de page web, c'est plus rapide car ici nous allons ouvrir 55 pages web. Cette procédure sera expliquée plus en détail dans la partie 4. Window size permet d'éviter des problèmes de taille de la page web, car plusieurs fois il ne trouve pas l'élément web si la page ouverte par Selenium ne prend pas en compte l'intégralité de la page. Concernant la définition de la variable url, on procéde ainsi car dans l'URL la seule chose qui change pour passer d'une marque à une autre c'est le nom de la marque dans l'URL écrite en miniscule, d'où le v.lower().

```
    modeles_elm = driver.find_elements_by_class_name("brandModel")
    prix_elm = driver.find_elements_by_class_name("fieldPrice")
    href_elm = driver.find_elements_by_class_name("linkAd")

    for i in range(len(modeles_elm)):
        hrefs.append(href_elm[i].get_attribute('href'))
        modeles.append(modeles_elm[i].text)
        prix.append(prix_elm[i].text)
```

Listing 14 – Boucle principale : récupération des données

En inspectant la page web, nous avons remarqué que les noms des marques et les modèles sont écrites en HTML dans une balise qui a comme nom de classe :"brandModel", les prix sont dans la classe :"fieldPrice" et les liens sont dans l'attribut href de la classe :"linkAd".
Le code "find_elements_by_class_name" permet de trouver les éléments web par leur nom de classe. Nous avons utiliser "by_class_name" au lieu de "by_xpath" ici car les divisions où les voitures sont mises dans le code HTML ne sont pas ordonnées, ce qui rend difficile de les scraper par leur chemin HTML.
Après avoir rassemblé tous les éléments web, nous allons en extraire les textes ou les hrefs dans le cas de liens à l'aide de la deuxième boucle.
Le code ".get_attribute('href')" permet d'extraire le lien du webelement, ".text" permet d'extraire le texte du webelement. En utilisant ces deux techniques nous avons pu rassembler les modèles, prix et liens des voitures dans 3 listes.

```
def filtrer_to_float (s):
    s=s.replace("€","")
    s=s.replace(" ","")
    s=float(s)
    return s
cars_df = pd.DataFrame({'Modeles':modeles,'Prix':list(map(filter_to_float,prix)),'
    Liens':hrefs})
```

Listing 15 – Nettoyage et rassemblement des données

Dans cette partie nous avons rassemblé les trois listes dans une Data Frame et nettoyé les données de la liste "prix", nous avions besoin de supprimer les symboles €et les espaces puis transformer en float. Nous avons nettoyer les données grâce à la fonction map qui permet d'appliquer une fonction à tous les éléments de la liste précisée.

```
cars_df[['Marques','Modeles']] = cars_df["Modeles"].str.split(" ", n = 1, expand =
    True)
cars_df = cars_df[['Marques','Modeles','Prix','Liens']]

cars_df.to_csv(r'C:\Users\mehdi\Desktop\M1 ecotr\Langage de programmation 1\Web
    scraping\csv files\LaCentrale.csv',index = None, header=True,sep=";")
```

Listing 16 – Exportation des données

La première partie du code permet d'extraire la marque de la colonne 'Modeles' (qui contient en ce moment la marque et le modele) et la mettre dans une nouvelle colonne 'Marques'.
La deuxième partie du code permet d'exporter les données vers le chemin précisé, il créera un fichier csv au nom mentionné "LaCentrale" dans le dossier au chemin précisé. Header=True permet d'utiliser les titres dans la Data Frame comme titres de colonnes dans le tableaux csv et sep = ";" permet de séparer les colonnes du tableau csv de la même façon dont elle le sont dans la Data Frame.

2.4 Scraping du site "Carizy" avec Selenium

Nous commençons par créer des listes qui vont contenir les données qu'on va récupérer :

```
modeles=[]
marques=[]
prix=[]
liens=[]
```

Listing 17 – Création des listes

Nous créons ensuite une première boucle pour scrapper plusieurs pages avec "i" allant de 1 à 35, (i=numéro de la page) :

```
for i in range (1,35):
driver = webdriver.Chrome()
driver.get("https://www.carizy.com/voiture-occasion?q=&hPP=21&idx=CarAlgoliaIndex_prod
    &p="+str(i))
```

Listing 18 – Création de la première boucle

Puis une deuxième boucle avec "i" de 1 à 21, (avec i=nombre de voitures par pages) incluse dans la première. Cette boucle est crée en lien avec la spécificité du site scrapé.
Dans le cas du site carizy on a observé comment le Xpath (le chemin de localisation) de la voiture qu'on a extrait en inspectant le site, change avec les voitures choisies :

```
//*[@id="hits-container"]/div/div[1]/div
//*[@id="hits-container"]/div/div[2]/div
```

Listing 19 – Xpaths des Voitures

Le choix des voitures dépend du nombre dans le div[] qui sera donc rempli à chaque fois par un autre nombre de la boucle.
Selenium dispose d'une fonction appelée "find-elements-by-xpath", Nous allons passer notre XPath dans cette fonction et obtenir un élément selenium pour chaques liste :

```
  for j in range(1,21):
        modele_element = driver.find_elements_by_xpath('//*[@id="hits-container"]/div/
    div['+str(j)+']/div/div/div[14]/div/div[1]/div[2]')[0]
        prixvoiture_element = driver.find_elements_by_xpath('//*[@id="hits-container
    "]/div/div['+str(j)+']/div/div/div[14]/div/div[2]/p[2]')[0]
        marque_element=  driver.find_elements_by_xpath('//*[@id="hits-container"]/div/
    div['+str(j)+']/div/div/div[14]/div/div[1]/a/h2')[0]
        liens_element =driver.find_elements_by_xpath('//*[@id="hits-container"]/div/
    div['+str(j)+']/div/div/div[14]/div /div[1]/a')[0]
```

Listing 20 – Xpath des Voitures

Une fois que nous avons l'élément, nous pouvons extraire le texte à l'intérieur de notre XPath en utilisant la fonction 'text' et 'get-attribute' :

```
marques.append(marque_element.text)
modeles.append(modele_element.text)
prix.append(prixvoiture_element.text)
liens.append(liens_element.get_attribute('href'))
```

Listing 21 – Retirer le texte

Finalement, après la construction de nos trois listes nous allons ensuite les regrouper dans une data frame 'voiture2' :

```
voiture2= pd.DataFrame({'Marques':marques,'Modeles':modeles,'Prix':prix,'Liens':liens
    })
```

Listing 22 – Transformer en Dataframe

3 Manipulation des données scrapées

3.1 Rassemblement des données

Après avoir scrapé et exporté en dossier csv les trois sites, le code qui suit rassemble en une seule dataframe ces données qu'on utlisera principalement pour la présentation et pour l'interface de comparaison.
'carizy_df.dropna()' est utilisé pour supprimer les lignes qui contiennent des données manquantes, 'pd.concat' est utilisé pour concaténer les dataframes en une seule dataframe.

```
import os
os.getcwd()
os.chdir("C:\\Users\\mehdi\\Desktop\\M1 ecotr\\Langage de programmation 1\\Web
    scraping\\csv files")
import pandas as pd
cars_df = pd.read_csv("LaCentrale.csv",sep=";")
vpauto_df=pd.read_csv("vpauto.csv",sep=";")
carizy_df = pd.read_csv("carizy.csv",sep=";")

carizy_df = carizy_df.dropna()
allcars_df= pd.concat([cars_df, vpauto_df, carizy_df])
allcars_df.to_csv(r'C:\Users\mehdi\Desktop\M1 ecotr\Langage de programmation 1\Web
    scraping\csv files\Allcars.csv',index = None, header=True,sep=";")
```

Listing 23 – Rassemblement des données

3.2 Analyse de donnée sur les voitures

Nous avons décidé de classifier les voitures par prix à l'aide de la classification non supervisée hiérarchique. Pour ce faire, il fallait commencé par créé une dataFrame contenant la moyenne des prix de chaque marque.

```
brands = Allcars.groupby('Marques').mean()
brands = brands.set_index('Marque')
```

Listing 24 – Moyennes de prix par marque

Le résultat est une DataFrame de 53 marques :

Index	Prix
DAEWOO	2542.86
SSANGYONG	3150
LADA	7126.67
FIAT	10280.1
CITROEN	10448.3
RENAULT	10758.9
PEUGEOT	11776.9
OPEL	11785.4
NISSAN	12076.3
SMART	12282.6
FORD	12493
DACIA	12528.2
SUZUKI	12752.9
LANCIA	13205

FIGURE 1 – DataFrame utilisée

L'algorithme de classification est présenté par le code qui suit :

```
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

X = linkage(brands, 'ward')
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Marques')
plt.ylabel('distance (Ward)')
dendrogram(X, labels=brands.index, leaf_rotation=90)
```

Listing 25 – Classification hiérarchique

Le code donne le graphique suivant :

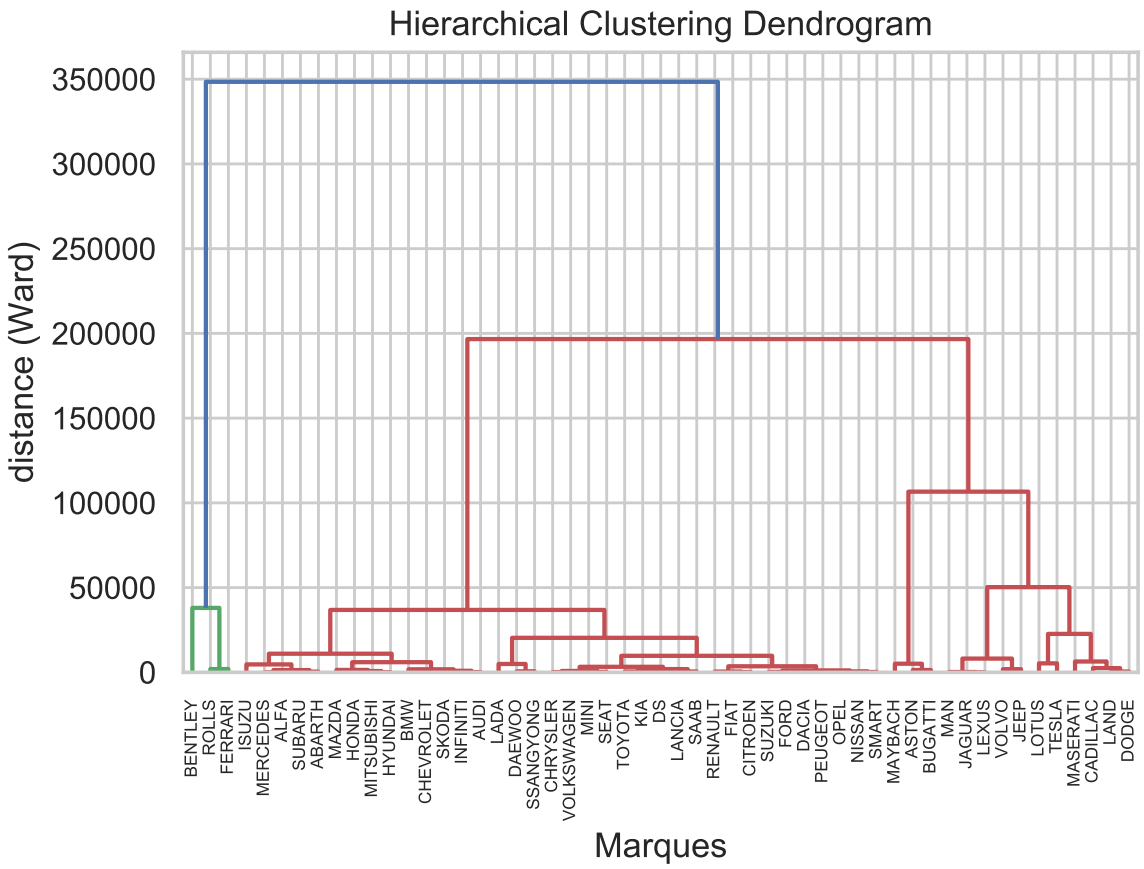

FIGURE 2 – Output du code

Si on coupe d'une ligne horizentale à 150000 le dendrogramme, on récupère 3 classes :

- Des voitures très chères : Bentley / Rolls Royce / Ferrari
- Des voitures chères : Aston Martin / Bugatti / Maserati / Jaguar / Jeep / Tesla / (etc...)
- Des voitures moins chères : Dacia / Renault / Toyota / Skoda / Mazda / (etc...)

3.3 Présentation du comparateur de prix

Le but de cette partie est de créer une intéraction avec le client qui profitera de notre base de données obtenue en scrappant les différents sites de voiture. L'intéraction consiste à demander au client d'insérer des marques et un budget. Le code cherchera automatiquement les voitures de ces marques spécifiées disponibles dans notre base de donnée à un prix inférieur à son budget (avec les liens directs vers le site pour les acheter). Le code montrera aussi le nombre d'autres voitures de la même marques choisie par le client mais à un prix supérieur à son budget.
Nous commençons par importer notre base de donnée et créer des listes vides qu'on utilisera ensuite.

```
import os
os.getcwd()
```

```
os.chdir("C:\\Users\\mehdi\\Desktop\\M1 ecotr\\Langage de programmation 1\\Web
    scraping\\csv files")
import pandas as pd
allcars=pd.read_csv("Allcars.csv",sep=";")

marques_available=[]
modeles_available=[]
prix_available=[]
liens_available=[]
```

Listing 26 – Importation des données

La fonction "input" de Python permet à l'utilisateur d'intéragir avec le code, elle mettra en mémoire ce qui a été écrit dans la variable désignée. Dans cette partie on l'utilise pour enregistrer les marques choisies par le client et son budget.
Nous avons besoin de marques séparés par des espaces, donc si l'utilisateur a laisser un espace à la fin des marques choisies, le code verra l'espace comme une marque en elle même. Pour éviter ce problème, on utilise une condition pour supprimer l'espace s'il est en fin de chaine de caractere. Ensuite on crée une liste à partir de cette chaine de caractere à l'aide de la fonction split.
Nous avons besoin de budget en numérique sans espaces, nous utilisons : "try/except" pour redemander le budget s'il n'est pas écrit en chiffres (numérique), "replace" pour enlever les espaces et float pour transformer les chiffres enregistrés comme chaine de caractere en float. Ceci permet d'éviter toutes erreurs associés au format du contenu rentré.

```
client_marques = input("Choisissez des marques :  ").upper()
if client_marques[-1:]== " ":
    client_marques = client_marques.replace(client_marques[-1:],"")
client_marques = client_marques.split(" ")
try:
    client_budget = float(input("Inserez votre budget : ").replace(" ",""))
except :
   client_budget = float(input("Veuillez inserer votre budget en numerique: ").replace
    (" ",""))
```

Listing 27 – Importation des données

Dans le code qui suit, nous avons crée une boucle qui recupère chacune des marques précisées par le client, et une autre avec une condition pour les comparer avec les marques dans notre base de données. La condition est vérifiée si la marque choisie par le client existe parmis les marques de voitures présentes dans notre base de donnée et si son budget est supérieur au prix de ces voitures.
Nous avons rajouté les attributs (Marque, Modele, Prix et Lien) des voitures qui respectent ces conditions aux listes prédéfinies dans l'étape précédente, pour ensuite les rassembler dans une DataFrame nommée ("cars_available").
Néanmoins, une deuxième condition qui doit être vérifiée : si la marque choisie par le client existe parmis les marques de voitures dans notre base de donnée et si son budget est **inférieur** au prix de ces voitures. Cette deuxième condition et la variable "k" sont là pour compter le nombre de voitures vérifiant la deuxième condition .
Nous avons rassemblé ensuite ces nombres dans une liste "p", pour afficher, à l'aide d'une boucle et d'une condition, le nombre de voitures de la marque choisie trouvées dans notre base de donnée qui ont un prix supérieur au budget du client.
Nous affichons enfin la DataFrame qui contient les voitures.

```
p = []
for i in range(len(client_marques)):
    k = 0
    for j in range(len(allcars['Marques'])):
        if client_marques[i] in allcars['Marques'][j] and client_budget >= allcars['
    Prix'][j]:
            marques_available.append(allcars['Marques'][j])
            modeles_available.append(allcars['Modeles'][j])
```

```
            prix_available.append(allcars['Prix'][j])
            liens_available.append(allcars['Liens'][j])
        elif client_marques[i] in allcars['Marques'][j] and client_budget < allcars['Prix'][j]:
            k +=1
    p.append(k)
cars_available = pd.DataFrame({'Marques Disponible':marques_available,'Modeles Disponible':modeles_available,'Prix':prix_available,'Liens':liens_available})

for z in range(len(client_marques)):
    if p[z] > 0:
        print("On retrouve "+str(p[z])+" autres "+client_marques[z]+" mais à plus de "+str(client_budget)+"€")
print(cars_available)
```

4 Problèmes rencontrés et solutions

Parmis les problèmes rencontrés, on peut mentionner les suivant :

4.1 Xpaths non-ordonnés

Les divisions dans l'Xpaths des prix des voitures n'étaient pas ordonnés. (e.g. le prix de la première voiture est dans la division li[2], la deuxième voiture dans la division li[5], la troixième voiture dans la division li[1], etc...)

⇒ Au lieu d'utiliser les Xpaths pour ce site nous avons utilisé class-name pour récupérer les éléments web.

4.2 Detecteur de bots

Le site "leboncoin" utilise un detecteur de bot qui s'active quand il reçoit un grand nombre de requêtes en un court laps de temps. Il demande ensuite de résoudre un code captcha, ce qui bloque notre Automate.

⇒ Il est donc essentiel d'importer la librairie Time et d'utiliser l'attribut sleep qui force l'automate à faire des pauses entre ses requêtes.L'argument donné à sleep est le nombre de secondes de pause qu'on souhaite demander à l'automate. Ceci permet de contourner le contrôle éffectué par le host, le temps de pause dépend du site utilisé.

⇒

```
import time
time.sleep(2)
```

Listing 28 – Time.sleep

4.3 List index out of range

Parfois, il est possible qu'on souhaite naviguer dans les différentes pages d'un site et en récupérer toutes les voitures. Toutefois, le nombre de voiture par page n'est pas le même, ce qui fait que l'automate essaie de récupérer une voiture qui n'existe pas dans la page et renvoi le message d'erreur suivant : list index out of range. Ce message signifie que l'élément qu'on essaye de scrapper n'existe pas, et tout le processus de scraping est donc abbandoné. On a trouver une solution pour ce problème par la méthode suivante :

⇒ Nous avons utilisé (try :/except :) pour dire à l'automate de continuer à scraper les données même s'il reçoit une erreur. Ainsi on pu être scrappé toutes les données d'une page sans se préocupper du nombre d'élément qui sont présent dans une page individuelle.

4.4 La fonction Map

Dans une partie de notre projet, nous avons dû nettoyer une liste après l'avoir construite.(e.g. les prix des voitures étaient enregistrés de la façon suivante : 10 000 €.). Pour supprimer les €et les espaces, nous avons crée une boucle pour appliquer la fonction "replace" à chaque élément de la liste comme ceci :

```
for i in range(len(carizy_df['Prix'])):
    carizy_df['Prix'][i]=carizy_df['Prix'][i].replace(" ","")
    carizy_df['Prix'][i]=carizy_df['Prix'][i].replace("€","")
```

Listing 29 – Nettoyer les données

⇒ Or, une meilleure façon de faire était de définir la fonction qu'on souhaite appliquer à chaque élément de la liste, puis l'insérer comme premier argument de la fonction map qui permet de l'appliquer à chaque élément de la liste insérée comme deuxième argument de map. C'est ce qu'on fait comme suit :

⇒

```
def filtrer_to_float (s):
    s=s.replace(" ","")
    s=s.replace("€","")
    s=float(s)
    return s
carizy_df['Prix']=list(map(filtrer_to_float,carizy_df['Prix']))
```

Listing 30 – Map

4.5 Scraping lent

Un problème qui peut être commun lorsqu'on veut extraire beaucoup de données avec Selenium et que nous avons eu pour scraper le site "LaCentrale" est que nous avions besoin d'ouvrir 55 pages du sites (une page pour chaque marque) ce qui nous a pris beaucoup de temps, car l'automate ouvre toutes les fenêtres une après l'autre.

⇒ Pour récupérer les données plus rapidement, il fallait donc scraper les différentes pages du site sans ouvrir les fenêtres. Parmis les options de Selenium on retrouve "Headless" qui permet de naviguer sans ouvrir les fenêtres. Les données sont donc scrappé beaucoup plus rapidement.

⇒

```
from selenium.webdriver.chrome.options import Options
options = Options()
options.headless = True
driver = webdriver.Chrome(options=options)
driver.set_window_size(1440, 900)
```

Listing 31 – Headless scraping

5 Conclusion

En conclusion, nous pouvons indiquer que le Web Scraping est une procédure d'exctraction de données en ligne qui dépend de nombreux facteurs tel que :

— la structure du code HTML du site.
— Le nombre de données que nous souhaitons récupérer (par exemple le nombre de pages).
— La présence de bloqueurs de bots sur le site (Captcha).

En fonction de ces facteurs et de ses besoins, l'utilisateur devra donc adapter ses méthodes de scraping pour contourner tout types d'autres problèmes qui peuvent survenir.