Samuel HOVHANNISYAN

Xania ZARICINII

Melissa VITHIYANANTHAN

# Projet :
# Chasseur immobilier virtuel

TD Langage de programmation

## Table des matières

# *I. Web scraping*

## *A. Présentation web scraping*

### 1. Contexte

À l'ère où internet a une place centrale au sein de la société, les données sont devenues la base de tous les processus décisionnels, qu'il s'agisse d'une entreprise ou d'un organisme à but non lucratif. Ils sont de plus en plus dépendants des données. Désormais, il est impératif d'avoir accès aux données les plus récentes pour chaque problématique. Par conséquent, l'usage du web scraping est devenu indispensable pour extraire massivement des données.

### 2. Définition

Le « web scraping » est une technique pour extraire des données et des informations de pages web automatiquement à l'aide d'un logiciel qui simule la navigation Web humaine. En faisant cela de façon autonome, le web scraping ouvre un monde de possibilités dans l'exploration de données. Cette technique permet donc d'utiliser les données non structurées disponibles sur le web dans un autre contexte.

### 3.Exemples de domaines dans lesquels le web scraping est utilisé

**Finance :**

Les analystes ont besoin d'états financiers pour déterminer la santé d'une entreprise et conseiller leurs clients quant à l'opportunité d'investir ou non dans celle-ci. Cependant, il n'est pas possible d'obtenir les états financiers de plusieurs entreprises pendant de nombreuses années de façon manuelle. Le web scraping est utilisé pour extraire les états financiers de différents sites et pour différentes périodes afin de prendre des décisions d'investissement.

Marketing :

Les données sont essentielles à toute entreprise de marketing et de vente. Les données disponibles peuvent permettre de formuler des stratégies. Par exemple, dans le cadre du « content marketing », le web scraping est utilisé pour rassembler des données de différents sites tels que Twitter, Instagram, etc. Ces données peuvent alors être utilisées pour créer du contenu engageant. Un contenu attrayant est la clé de la croissance des entreprises et du trafic web.

Notre projet porte sur l'application du « web scraping » dans le secteur de l'immobilier. Nous aborderons les avantages de cette technique sur ce marché plus tard.

## 4.Web scraping dynamique ou statique ?

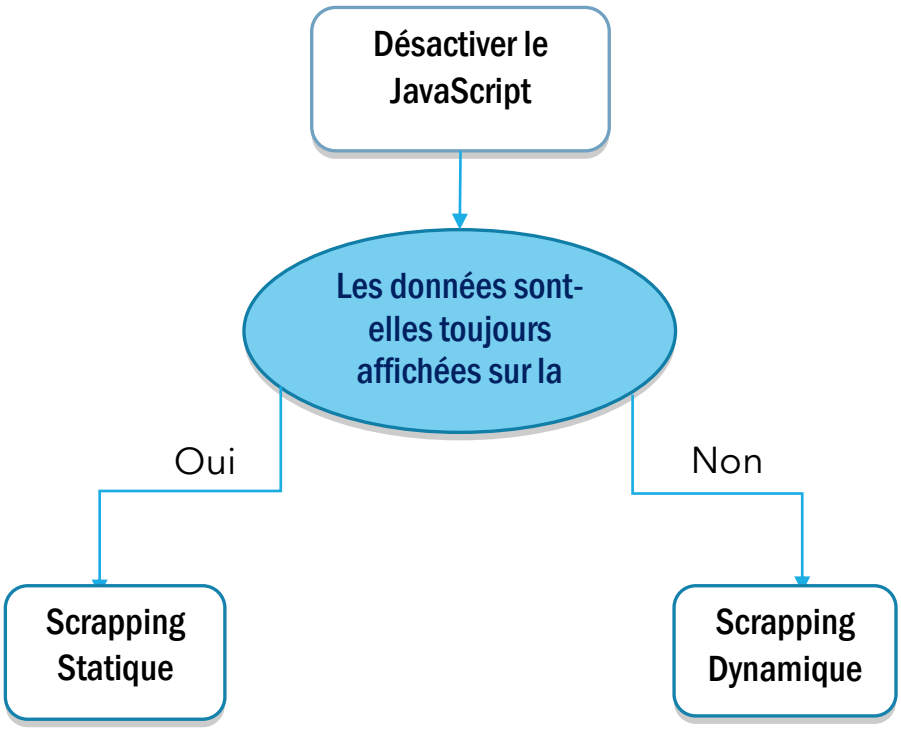
La méthode employée dépend de la structure de la page web.

web scraping statique

Le web scraping statique est utilisé pour des pages web statiques. Un contenu web statique fait généralement référence au contenu web qui est fixe et qui n'est pas capable de changement. Un site web statique ne peut fournir que des informations qui sont écrites dans le code source HTML et ces informations ne changeront pas à moins que le changement soit écrit dans le code source. Lorsqu'un navigateur web demande la page Web statique spécifique, un serveur renvoie la page au navigateur et l'utilisateur ne reçoit que les informations contenues dans le code HTML.Le web scrapping statique consiste à télécharger le code source d'une page et l'analyser. Cette méthode ignore Javascript. Beautiful Soup est une bibliothèque Python pour extraire des données de fichiers HTML et XML.

web scraping dynamique

Le web scaping dynamique est utilisé pour des pages web dynamiques. Une page web dynamique contient du contenu généré dynamiquement qui est retourné par un serveur à la demande d'un utilisateur. L'utilisateur peut demander l'information, qui peut être extraite de la base de données en fonction des paramètres d'entrée de l'utilisateur. Le web scraping dynamique consiste à prendre le contrôle d'un navigateur web réel et l'utiliser pour rechercher des données sur une page web. Pour afficher une page, le navigateur web effectuera un ensemble d'opérations. Parfois, les données que l'on souhaite sont affichées après l'exécution de JavaScript, ce qui signifie que l'on doit effectuer toutes les opérations qu'un navigateur web effectue normalement pour pouvoir obtenir les données souhaitées. Selenium est un outil d'automatisation de navigateur web.



Désormais, nous allons nous concentrer uniquement sur le web scraping dynamique car nous utiliserons des sites où des annonces apparaissent et disparaissent dans le temps, les sites web se mettent à jour constamment, et pour récupérer des données nous n'avons d'autre choix que de nous servir de Selenium.

### 5.Est ce légal ?

Au regard de la législation française, la pratique du web scraping et l'utilisation des données récupérées peuvent être considérée comme une pratique illégale et non éthique.

Droit pénal : La pratique du web scraping est condamnable. En effet, *« le fait d'introduire frauduleusement des données dans un système de traitement automatisé, d'extraire, de détenir, de reproduire, de transmettre, de supprimer ou de modifier frauduleusement les données qu'il contient est puni de cinq ans d'emprisonnement et de 150 000 € d'amende. Lorsque cette infraction a été commise à l'encontre d'un système de traitement automatisé de données à caractère personnel mis en œuvre par l'État, la peine est portée à sept ans d'emprisonnement et à 300 000 € d'amende ».*

Droit de la concurrence : Le web scraping peut être assimilé à un acte de concurrence déloyale selon l'article L. 121-1 du Code de la consommation et l'article 1240 du code civil. Toutefois, il est primordial de préciser que c'est l'utilisation des données récupérées qui est condamnable et non la pratique.

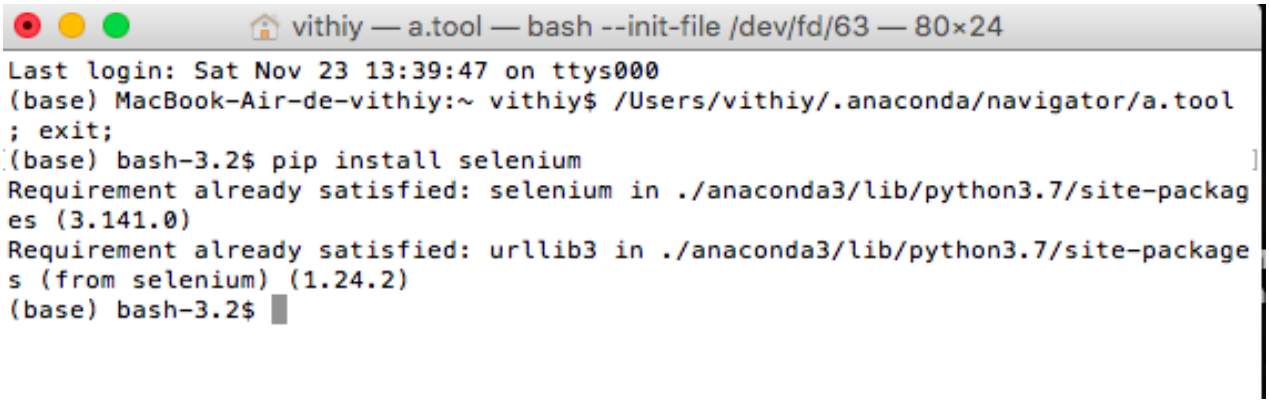
Droit de la propriété intellectuelle : Selon l'article L. 342-1 du Code la propriété intellectuelle « *Le producteur de base de données à le droit d'interdire l'extraction par transfert permanent ou temporaire de la totalité ou d'une partie qualitativement ou quantitativement substantielle du contenu d'une base de données sur un autre support, par tout moyen et sous toute forme que ce soit* ». C'est donc l'utilisation des données scapées sans modification qui est condamnable.

## *B. Tutoriel web scraping dynamique avec Selenium*

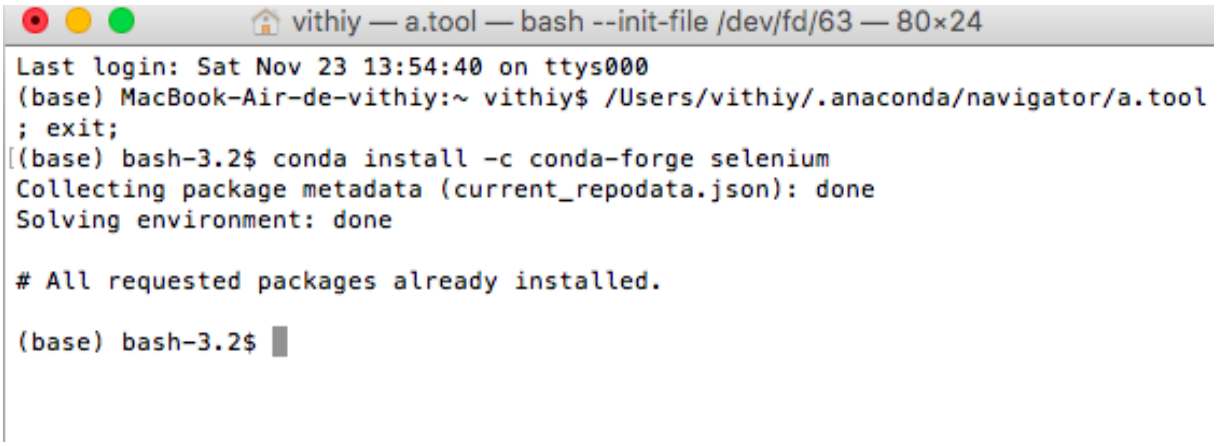
Dans cette partie, nous allons aborder comment récupérer les données d'une page web avec la librairie Selenium. C'est un outil d'automatisation de navigateur web. Il permet donc d'automatiser des actions dans un navigateur web et de récupérer les résultats de ces actions. Exemples d'actions qui peuvent être automatisées : visiter une page web, effectuer une recherche, cliquer sur un lien, remplir des formulaires, etc

### 1 .ETAPE 1 : Les prérequis

La façon la plus simple d'installer Selenium sur un environnement Python est par le pip. Le pip est un outil d'installation par prédiction. Sous l'invite de commande, tapez « pip install selenium ».




Une autre méthode est de taper **« conda install -c conda-forge selenium »** sous l'invite de commande d'Anaconda.




Maintenant que l'installation de Selenium rend la fonctionnalité de scraping disponible, nous avons besoin d'un web driver pour être en mesure d'interagir avec le navigateur web choisi. Web driver est un framework de tests fonctionnels issu de Selenium. Dans notre cas, le navigateur web choisi est Chrome. Chrome, nécessite Chromedriver, qui doit être installé avant de commencer à web scraper. Téléchargez le pilote approprié à Chrome via le lien suivant : https://sites.google.com/a/chromium.org/chromedriver/downloads. Veillez à trouver le chemin d'accès de votre Chrome driver.

## 2. ETAPE 2 : automatiser la navigation web humaine

Une fois l'étape 1 terminé, nous sommes prêts à automatiser les actions suivantes : visiter une page web, remplir un formulaire et effectuer une recherche. Dans d'autres termes, simuler un utilisateur réel travaillant avec un navigateur

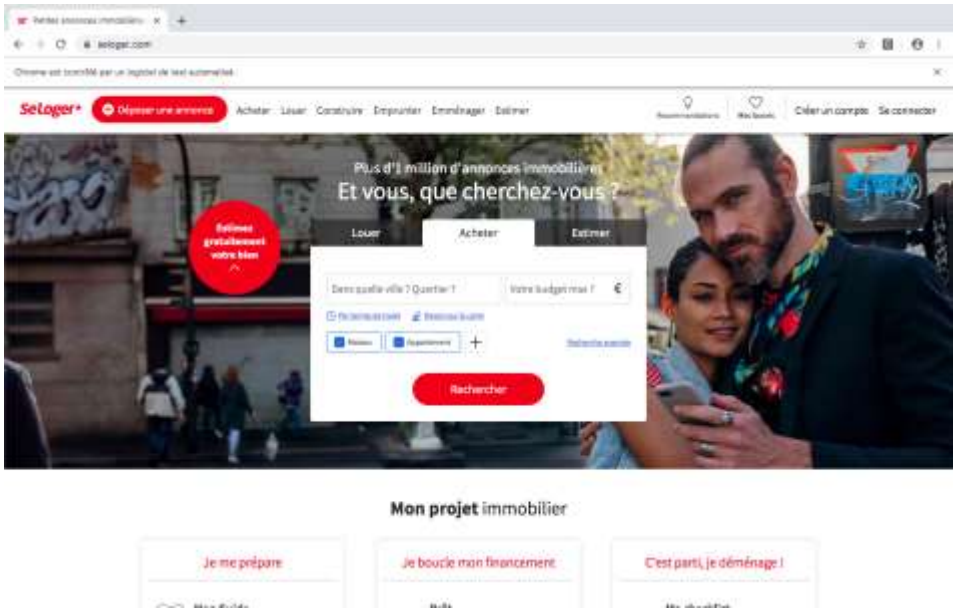
Dans un premier temps, importez les classes webdriver et Keys de Selenium. La classe webdriver permet de se connecter à l'instance d'un navigateur, de le lancer et de l'initialiser. Il peut interagir avec tous les types de navigateurs Web comme Firefox, Internet Explorer, Safari, Chrome, etc. La classe Keys simule l'action de taper sur les touches du clavier. Elle fournit des touches du clavier telles que RETURN, F1, ALT, ENTER, etc.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
```

Ensuite, spécifiez le chemin d'accès à Chrome driver et créez une instance de Chrome avec le chemin du pilote téléchargé. Chrome_path est le nom arbitraire que l'on donne au chromedriver (cela pourrait être Firefox, ou Internet Explorer, etc, mais nous on a choisit celui de Google Chrome). Avant de faire quoi que ce soit d'autre, ouvrez la page web que vous souhaitez scraper dans votre navigateur. Copiez l'URL. Ensuite, nous devons configurer le driver pour récupérer le contenu de l'URL. Utilisez la méthode **get ()** du driver pour charger le site web. Collez le lien dans la fonction **driver. get** (« votre URL ici ») et exécutez.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

chrome_path=r"/Users/vithiy/Downloads/Chromedriver"
driver = webdriver.Chrome(chrome_path)
driver.get('https://www.seloger.com/')
```

Le site chargé dans ce tutoriel est https://www.seloger.com/

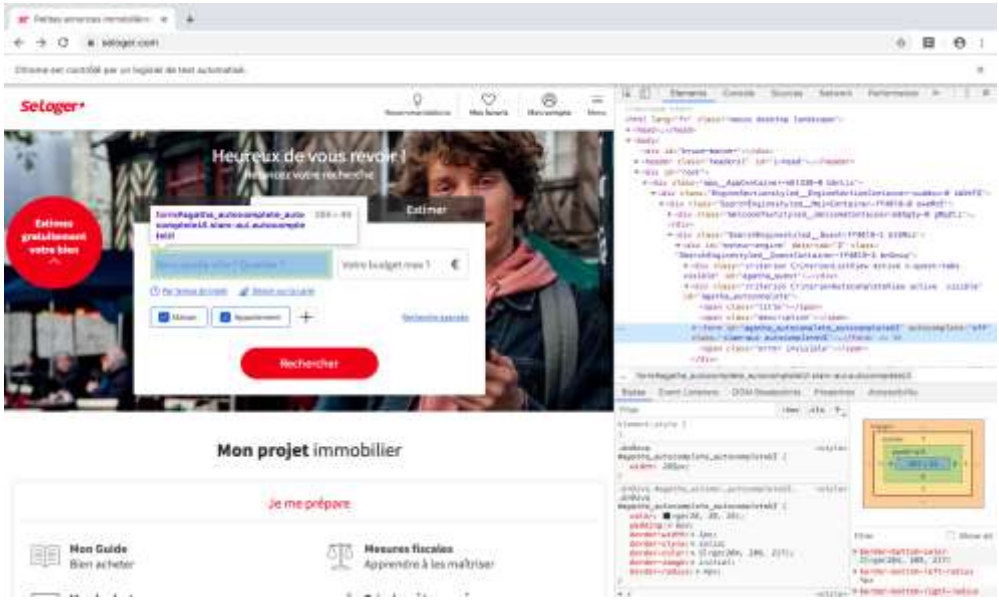



Désormais, nous souhaitons effectuer la recherche suivante : l'ensemble des biens immobiliers en vente à Paris. Nous allons donc automatiser l'action d'entrer **« Paris »** dans la barre de recherche de gauche puis de cliquer sur le bouton **« Rechercher ».**
Il existe diverses méthodes pour localiser des éléments sur une page web. Selenium fournit les méthodes suivantes pour localiser des éléments sur une page :

***-find_element_by_id***
***-find_element_by_name***
***-find_element_by_xpath***
***etc.***

Maintenant, allez dans le formulaire de recherche de biens immobiliers, cliquez sur le bouton droit de la souris sur la barre de recherche et sélectionnez « inspect element »

En inspectant le formulaire de recherche de bien sur la page d'accueil du site SeLoger, nous constatons que l'élément formulaire de recherche possède un attribut id "***agatha_autocomplete_autocompleteUI_input***". *À l'ai*de de la méthode **find_element_by_id,** nous localisons le champ de la ville du formulaire de recherche pour pouvoir soumettre la requête.




Ensuite, la méthode **send_keys** est utilisée pour valider l'entrée de la saisie **« Paris ».** La méthode **find_elment_by_xpath** permet de localiser le bouton **« Rechercher».** Le xpath est récupéré en inspectant le bouton **« Rechercher ».** L'une des principales raisons d'utiliser XPath est le fait que l'on a pas d'attribut *id* ou *name* approprié pour l'élément que l'on souhaite localiser. Enfin, la méthode **click()** permet de simuler l'action de cliquer.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

chrome_path=r"/Users/vithiy/Downloads/Chromedriver"
driver = webdriver.Chrome(chrome_path)
driver.get('https://www.seloger.com/')

#entrer Paris dans la barre de recherche
chercher_ville=driver.find_element_by_id("agatha_autocomplete_autocompleteUI__input")
chercher_ville.send_keys("Paris")
chercher_ville.send_keys(Keys.ENTER)

#appuyer sur le bouton recherche
chercher_ville.find_element_by_xpath('//*[@id="agatha_actionbuttons"]/div/div[2]/label/a').click()
```

Après avoir exécuté le code ci-dessus, nous obtenons la recherche souhaitée c'est-à-dire l'ensemble des biens immobiliers en vente à Paris.

## 3. ETAPE 3 : Extraction des données souhaitées

Nous souhaitons récupérer les données suivantes : prix des appartements, la surface, type de bien : maison, appartement etc. , nombre de pièces et arrondissement. Pour, trouver toutes les occurrences d'un élément recherché, Selenium fournit les méthodes suivantes :

***find_elements_by_name***

***find_elements_by_xpath***
***find_elements_by_class_name***
*etc.*
Ces méthodes retourneront une liste. Afin de localiser les éléments voulus, utilisons la méthode ***find_elements_by_class_name*** *et* ***find_elements_by_xpath.*** Inspectez la page web (click droit) et trouvez la classe name pour les éléments : prix des appartements et type de bien. Puis, spécifiez le xpath relatif pour les éléments suivants : nombre de pièces, la surface des biens et arrondissement. Xpath peut être utilisé pour localiser l'élément en termes absolus ou par rapport à un élément ayant un attribut id ou name. Les localisateurs xpath peuvent également être utilisés pour spécifier des éléments via des attributs autres que id et name.
Les XPaths absolus contiennent l'emplacement de tous les éléments de la racine (html) et risquent donc d'échouer avec le moindre ajustement du site web. En trouvant un élément proche avec un attribut id ou name (idéalement un élément parent), on peut localiser notre élément cible en fonction de la relation. Ceci est beaucoup moins susceptible de changer et peut rendre le scrapping plus robuste.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

chrome_path=r"/Users/vithiy/Downloads/Chromedriver"
driver = webdriver.Chrome(chrome_path)
driver.get('https://www.seloger.com/')

#entrer Paris dans la barre de recherche
chercher_ville=driver.find_element_by_id("agatha_autocomplete_autocompleteUI__input")
chercher_ville.send_keys("Paris")
chercher_ville.send_keys(Keys.ENTER)

#appuyer sur le bouton recherche
chercher_ville.find_element_by_xpath('//*[@id="agatha_actionbuttons"]/div/div[2]/label/a').click()

#localisation des données pertinentes
prix=driver.find_elements_by_class_name("lfyHpt")
type_de_bien=driver.find_elements_by_class_name("hXERKq")
nb_pieces=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[1]")
surface=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[3]")
arrondissement=driver.find_elements_by_xpath("//*[@class='Card__LabelGap-sc-7insep-5 ContentZone__Address-wghbmy-2 eAcNsS']/child::span[1]")
```

Maintenant, nous avons 5 listes contenant des éléments web.

| Name ▲ | Type | Size | Value |
|---|---|---|---|
| arrondissement | list | 25 | [WebElement, WebElement, WebElement, WebElement, WebElement, WebElemen ... |
| chrome_path | str | 1 | /Users/vithiy/Downloads/Chromedriver |
| nb_pieces | list | 25 | [WebElement, WebElement, WebElement, WebElement, WebElement, WebElemen ... |
| prix | list | 25 | [WebElement, WebElement, WebElement, WebElement, WebElement, WebElemen ... |
| surface | list | 25 | [WebElement, WebElement, WebElement, WebElement, WebElement, WebElemen ... |
| type_de_bien | list | 25 | [WebElement, WebElement, WebElement, WebElement, WebElement, WebElemen ... |

Nous souhaitons, afficher les textes des éléments web obtenus dans chacune des listes. Pour cela, nous devons connaître le nombre d'annonces par page. Or, nous savons qu'il y a le même nombre d'annonces que la longueur de chaque liste.

On définit donc items comme la longueur la liste type_de_bien. La méthode **len(type_de_bien)** compte le nombre d'élément dans la liste. Puis, pour l'ensemble des annonces (boucle for), on affiche (print) le texte des données récupérées pour chaque annonce avec comme séparateur **« , »** et on revient à la ligne à la fin avec **« \n ».**

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

chrome_path=r"/Users/vithiy/Downloads/Chromedriver"
driver = webdriver.Chrome(chrome_path)
driver.get('https://www.seloger.com/')

#entrer Paris dans la barre de recherche
chercher_ville=driver.find_element_by_id("agatha_autocomplete_autocompleteUI__input")
chercher_ville.send_keys("Paris")
chercher_ville.send_keys(Keys.ENTER)

#appuyer sur le bouton recherche
chercher_ville.find_element_by_xpath('//*[@id="agatha_actionbuttons"]/div/div[2]/label/a').click()

#localisation des données pertinentes
prix=driver.find_elements_by_class_name("lfyHpt")
type_de_bien=driver.find_elements_by_class_name("hXERKq")
nb_pieces=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[1]")
surface=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[3]")
arrondissement=driver.find_elements_by_xpath("//*[@class='Card__LabelGap-sc-7insep-5 ContentZone__Address-wghbmy-2 eAcNsS']/child::span[1]")

# afficher le texte des  éléments web de chacune des listes
items = len(type_de_bien)
for i in range(items):
 print(type_de_bien[i].text + "," + prix[i].text + "," + arrondissement[i].text + "," + nb_pieces[i].text  +  "," + surface[i].text +"\n")
```

## 4. ETAPE 4 : Enregistrer les données extraites sous forme Data frame et fichier CSV

Une fois que nous avons acquis les données, nous pouvons les stocker dans le format désiré, un fichier CSV et même dans un data frame. Dans ce tutoriel, nous allons aborder comment stocker les données dans une data frame puis exporter la data frame dans un fichier csv.
Importez, la librairie pandas as pd et csv.
Comme nous pouvons le voir ci-dessous, le code est assez explicite. Nous créons une liste vide total qui va contenir l'ensemble des annonces récoltées sur chaque page (ligne 26). Pour chaque annonce (boucle for), nous remplissons la liste par les données de chaque annonce.
Append permet d'ajouter un par un les caractéristiques de chaque bien immobilier (new) à la fin de la liste totale.
Puis,nous créons la data frame à partir d'une liste de tuple, **df = pd.DataFrame(liste, columns =['noms']).**
**df.to_csv('annonces_seloger.csv'),** permet d'exporter la data frame dans un fichier csv nommé 'annonces_seloger.csv.'
Enfin, le driver est fermé. **driver.close()** ferme la page URL à la fin du programme et notre fichier csv est prêt.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 23 15:31:33 2019

@author: vithiy
"""
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import csv
import pandas as pd

chrome_path=r"/Users/vithiy/Downloads/Chromedriver"
driver = webdriver.Chrome(chrome_path)
driver.get('https://www.seloger.com/')

#entrer Paris dans la barre de recherche
chercher_ville=driver.find_element_by_id("agatha_autocomplete_autocompleteUI__input")
chercher_ville.send_keys("Paris")
chercher_ville.send_keys(Keys.ENTER)

#appuyer sur le bouton recherche
chercher_ville.find_element_by_xpath('//*[@id="agatha_actionbuttons"]/div/div[2]/label/a').click()

#création d'une liste vide
total = []

#localisation des données pertinentes
prix=driver.find_elements_by_class_name("lfyHpt")
type_de_bien=driver.find_elements_by_class_name("hXERKq")
nb_pieces=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[1]")
surface=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[3]")
arrondissement=driver.find_elements_by_xpath("//*[@class='Card__LabelGap-sc-7insep-5 ContentZone__Address-wghbmy-2 eAcNsS']/child::span[1]")

items = len(type_de_bien)

# Remplissage de la liste total par les caractéristiques de chaque appartement
for i in range(items):

        new = ([type_de_bien[i].text,prix[i].text, surface[i].text, arrondissement[i].text,nb_pieces[i].text])
        total.append(new)
        print(total)

#dataframe et csv
df = pd.DataFrame(total,columns=['type_de_bien','prix', 'surface','arrondissement','nb_pieces'])
df.to_csv('annonces_seloger.csv')
driver.close()
```

Fichier CSV obtenu :

| | type_de_bien | prix | surface | arrondissement | nb_pieces |
|---|---|---|---|---|---|
| 0 | Appartement | 9 900 000 € | 417,12 m² | Paris 16ème | 6 p |
| 1 | Appartement | 6 300 000 € | 230 m² | Paris 16ème | 5 p |
| 2 | Appartement | 1 988 000 € | 106 m² | Paris 1er | 4 p |
| 3 | Appartement | de 3M à 10M € | 333 m² | Paris 16ème | 10 p |
| 4 | Appartement | 629 000 € | 57 m² | Paris 11ème | 3 p |
| 5 | Appartement | 590 000 € | 70 m² | Paris 18ème | 3 p |
| 6 | Appartement | 2 800 000 € | 95 m² | Paris 4ème | 5 p |
| 7 | Appartement | 695 000 € | 72 m² | Paris 18ème | 3 p |
| 8 | Appartement | 870 000 € | 92 m² | Paris 13ème | 4 p |
| 9 | Appartement | 895 000 € | 93 m² | Paris 19ème | 5 p |
| 10 | Appartement en viager | bouquet 135 250 € | 70 m² | Paris 12ème | 2 p |
| 11 | Appartement | 2 547 000 € | 197,39 m² | Paris 17ème | 6 p |
| 12 | Appartement | 1 190 000 € | 120 m² | Paris 14ème | 5 p |
| 13 | Appartement | 469 000 € | 40 m² | Paris 18ème | 2 p |
| 14 | Appartement | 1 575 000 € | 110 m² | Paris 16ème | 5 p |
| 15 | Appartement | 830 000 € | 78,69 m² | Paris 19ème | 4 p |
| 16 | Appartement | 538 000 € | 54 m² | Paris 13ème | 3 p |
| 17 | Appartement | 1 790 000 € | 130,8 m² | Paris 16ème | 4 p |
| 18 | Appartement | 599 000 € | 50 m² | Paris 6ème | 2 p |
| 19 | Appartement | 1 550 000 € | 106 m² | Paris 16ème | 5 p |
| 20 | Appartement | 599 000 € | 54,79 m² | Paris 13ème | 3 p |
| 21 | Appartement | 918 000 € | 89 m² | Paris 18ème | 4 p |
| 22 | Appartement | 3 397 000 € | 264,6 m² | Paris 16ème | 7 p |
| 23 | Appartement | 2 390 000 € | 135,66 m² | Paris 8ème | 4 p |
| 24 | Appartement | 630 000 € | 2 etg | Paris 3ème | 1 p |

# *II. Projet : chasseur immobilier virtuel*

## *A. Présentation du projet*

Selon la récente étude « Les Français et l'investissement immobilier » réalisée par Opinionway en Janvier 2017, 60% des interrogés considèrent l'investissement immobilier comme le placement le plus sûr, devant l'assurance vie et loin devant les placements en bourse.
Notre projet résulte d'un constat simple : le marché de l'immobilier est opaque pour les particuliers et investisseurs, bien que des milliers d'annonces immobilières soient disponibles en ligne. Il est difficile d'obtenir des données immobilières, ce qui fait que la plupart des investisseurs font des investissements financiers à l'aveuglette. Avec le web scraping, un investisseur peut prendre des décisions basées sur des données qualitatives et pertinentes, plutôt que des informations incomplètes.
À la manière d'un comparateur de prix, notre chasseur immobilier virtuel parcourt différents sites d'agences immobiliers et repère les meilleures annonces en fonction des critères des potentiels acheteurs. Ainsi, le chasseur immobilier virtuel rend compte en direct de toutes les annonces susceptibles d'intéresser le particulier ou l'investisseur.
Notre projet permet un gain de temps en surveillant les nouvelles annonces et en rendant le marché de l'immobilier moins opaque.

### 1.Les grandes étapes de notre projet : chasseur immobilier virtuel

1 - Récupération des données immobilières uniquement dans Paris intra-muros.
Les données qui nous semble pertinents à extraire sont : prix des appartements, arrondissement, type de bien maison, appartement etc. , nombre de pièces, et la surface.
Nous avons chacun récupéré les données à partir de la page web d'une d'agence immobilière. Xenia a extrait les données du site Immo Logic, Samuel de Pap et Mélissa de Se Loger.

2 – Nettoyage les données collectées.

3 – Stocker les données dans une même data frame et fichier CSV afin d'utiliser les données qui étaient non structurées pour notre projet de chasseur immobilier virtuel.
Programmation de la requête affichant les biens immobiliers pertinents en fonction des critères des acheteurs : prix et surface.
Selon l'étude « Les Français et l'investissement immobilier » de Opinionway, les français attendent en priorité un rendement locatif intéressant pour 75% d'entre eux et ils choisiraient s'ils prenaient la décision d'investir pour 37% pour des petites surfaces. Le prix et la surface sont donc les critères décisifs lors d'un achat immobilier.

## 2.Les codes de notre projet et explications des codes

### a)Etape 1 & 2 : Script récupération des données immobilières et nettoyage des données pour chaque site web

Lors de la section I.2 Tutoriel du web scraping dynamique avec Selenium, nous avons vu comment extraire les données sur une page web. Or, pour notre projet nous avons besoin d'extraire les données sur l'ensemble des pages dynamiques d'un site. Nous avons construit trois scripts pour scraper les sites d'agences immobilières. Nous allons donc présenter les codes utilisés pour scraper : Se Loger, Immo Logic et Pap.

## Script SeLoger :

```
#
import csv
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import pandas as pd
with open('annonces.csv', 'w') as f:
    f.write("Type_de_bien;Prix;Surface;Arrondissement;Nb_pieces;Agence\n")

chrome_pather"/Users/vithiy/Downloads/Chromedriver"
driver = webdriver.Chrome(chrome_path)
driver.get('https://www.seloger.com/')


#entrer ville dans la barre de recherche
chercher_ville=driver.find_element_by_xpath('//*[@id="agatha_autocomplete_autocompleteUI__input"]')
chercher_ville.send_keys("Paris")
chercher_ville.send_keys(Keys.ENTER)

#appuyer sur la dernière recherche
chercher_ville.find_element_by_xpath('//*[@id="agatha_actionbuttons"]/div/div[2]/label/a').click()

#localisation des données pertinentes
prix=driver.find_elements_by_class_name("lfyNpt")
type_de_bien=driver.find_elements_by_class_name("hXERKq")
nb_pieces=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[1]")
surface=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[3]")
arrondissement=driver.find_elements_by_xpath("//*[@class='Card__LabelGap-sc-7insep-5 ContentZone__Address-wghbmy-2 eAcNsS']/child::span[1]")
items = len(type_de_bien)

# les données extraient sur la première page pour chaque annonce sont conservées dans le fichier csv"annonces".
with open('annonces.csv', 'a') as f:
    for i in range(items):
        prix=driver.find_elements_by_class_name("lfyNpt")
        type_de_bien=driver.find_elements_by_class_name("hXERKq")
        nb_pieces=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[1]")
        surface=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[3]")
        arrondissement=driver.find_elements_by_xpath("//*[@class='Card__LabelGap-sc-7insep-5 ContentZone__Address-wghbmy-2 eAcNsS']/child::span[1]")
        f.write(type_de_bien[i].text + ";" + prix[i].text + ";" + surface[i].text+ ";" + arrondissement[i].text + ";" +nb_pieces[i].text+";"+"Seloger"+"\n")
        print(type_de_bien[i].text + ";" + prix[i].text + ";" + surface[i].text + ";" + arrondissement[i].text + ";" + nb_pieces[i].text+";"+"Seloger"+"\n")

# récupération des données sur chaque page du site web
while True:

    link_next_page=driver.find_element_by_xpath('//*[@id="root"]/div/div/div[3]/div/div[2]/section/ul/li[@class="next"]/a')
    if not link_next_page:
        break
    next_page_url = link_next_page.get_attribute("href")
    print(next_page_url)
    driver.get(next_page_url)
    #driver.implicitly_wait(3)
    prix=driver.find_elements_by_class_name("lfyNpt")
    type_de_bien=driver.find_elements_by_class_name("hXERKq")
    nb_pieces=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[1]")
    surface=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[3]")
    arrondissement=driver.find_elements_by_xpath("//*[@class='Card__LabelGap-sc-7insep-5 ContentZone__Address-wghbmy-2 eAcNsS']/child::span[1]")
    items = len(type_de_bien)
# les données récupérées sur chaque page sont ajoutées au fichier csv" annonces "
    with open('annonces.csv', 'a') as f:
        for i in range(items):
            prix=driver.find_elements_by_class_name("lfyNpt")
            type_de_bien=driver.find_elements_by_class_name("hXERKq")
            nb_pieces=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[1]")
            surface=driver.find_elements_by_xpath("//*[@class='ContentZone__Tags-wghbmy-7 fNlgGF']/child::li[3]")
            arrondissement=driver.find_elements_by_xpath("//*[@class='Card__LabelGap-sc-7insep-5 ContentZone__Address-wghbmy-2 eAcNsS']/child::span[1]")
            f.write(type_de_bien[i].text + ";" + prix[i].text + ";" + surface[i].text+ ";" + arrondissement[i].text + ";" +nb_pieces[i].text+";"+"Seloger"+"\n")
            print(type_de_bien[i].text + ";" + prix[i].text + ";" + surface[i].text + ";" + arrondissement[i].text + ";" + nb_pieces[i].text+";"+"Seloger"+"\n")

driver.close()
```

Commençons par le site Se loger, la méthode est la suivante : les bibliothèques nécessaires sont déjà importées (voir tutoriel). Premièrement, nous créons un fichier csv « annonces » en mode écriture en spécifiant les colonnes : Type de bien, Prix, Surface, Arrondissement, Nb_pieces et Agence.
**with open('annonces.csv', 'w') as f:**
**f.write("Type_de_bien;Prix;Surface;Arrondissement;Nb_pieces;Agence\n")** . →
Ce code nous permet de créer un nouveau fichier qui n'existe pas. Pour écrire dans ce nouveau

fichier, on doit ajouter un paramètre à la fonction ***open ()*** → On a le nom du fichier csv que l'on choisit arbitrairement, et 'w' pour write c'est-à-dire qu'il va écrire dans le fichier.
« as f » signifie « comme f », que l'on choisit arbitrairement aussi, pour ne pas réécrire le nom du fichier à chaque fois. Le deuxième code ***f.write***(« ») va écrire ce que l'on veut dans le fichier que l'on a crée, avec le séparateur que l'on veut (virgule, point-virgule,etc) et on termine par « \n » pour indiquer un saut d'une ligne.
Nous avons automatisé la recherche des biens immobiliers en vente à Paris. Puis, nous récupérons les données de la première page en fonction du résultat dynamique. La procédure est la même que lors du tutoriel. Pour chaque annonce de la première page de résultat de la recherche, le fichier csv est remplit par les données récoltées. ***with open('annonces.csv', 'a') as f:*** → ici au lieu de 'w' on 'a' pour append c'est à dire les données récupérées pour chaque annonce sont ajoutées à la fin du fichier.

Afin de récupérer les données sur l'ensemble des pages, nous créons une boucle infinie à l'aide de ***While True*** . Nous localisons le bouton « Suivant » à l'aide du xpath pour parcourir l'ensemble des pages. Lorsque, il y a le bouton suivant le driver récupère le lien de la page suivante, y accède, récupère les données et remplit le fichier csv par les données récupérées sur la page suivante. Le driver récupère le lien de la page suivante par son attribut : ***« href ».*** L'attribut ***« href »*** indique la cible du lien sous la forme d'une l'url. À la dernière page, la boucle s'arrête dû à l'instruction « Break » car le bouton suivant n'est plus présent. Enfin, nous fermons le driver : ***driver.close()***

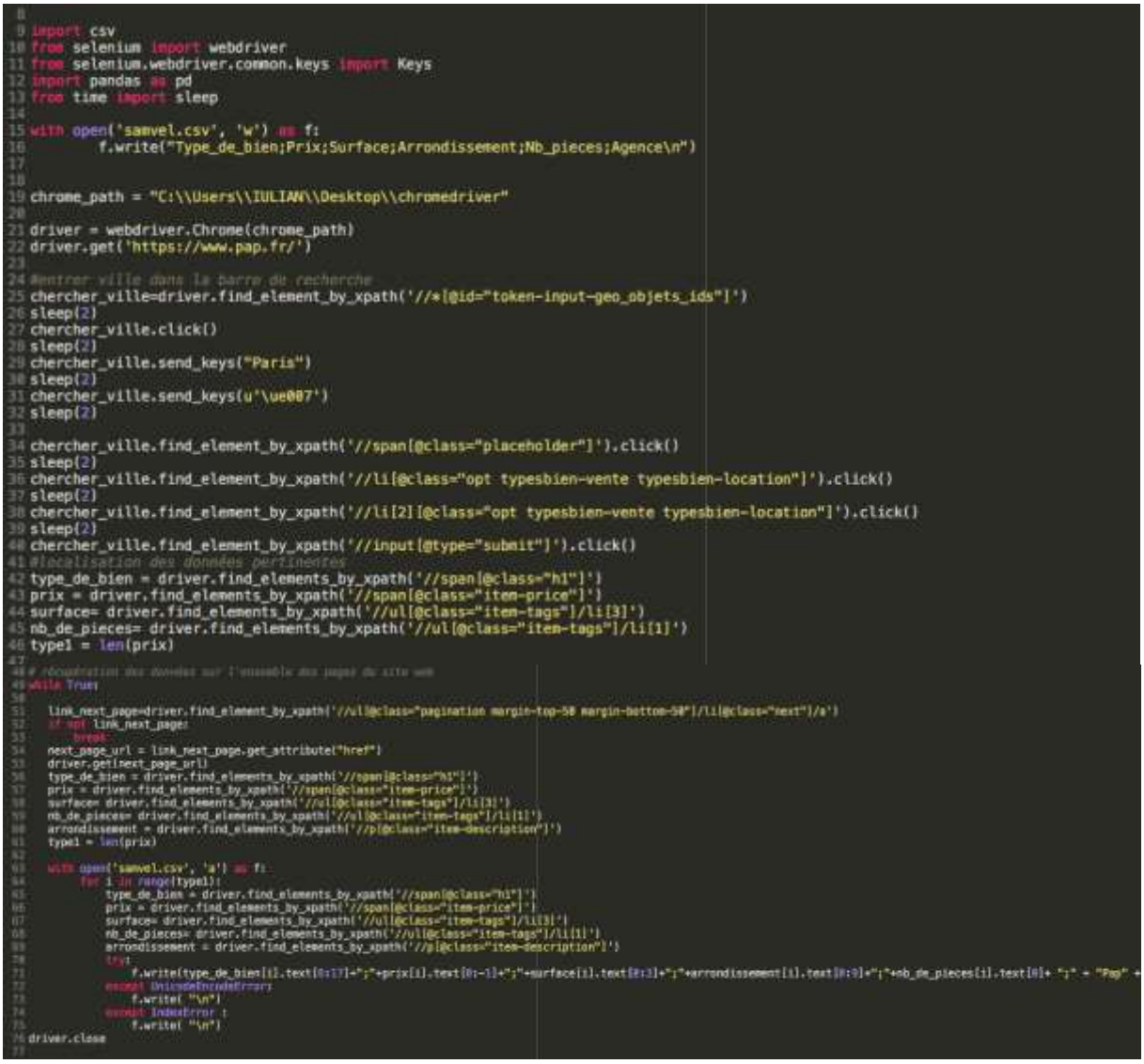
**Explication script nettoyage des données Seloger:**

```
import csv
import pandas as pd

#nettoyage des données de SeLoger

df1= pd.read_csv('annonces.csv',sep=';',engine='python')
df1['Prix'] = df1['Prix'].str.replace(" ","")
df1['Surface'] = df1['Surface'].str.replace(" ","")
df1['Nb_pieces'] = df1['Nb_pieces'].str.replace(" ","")
df1['Prix'] = df1['Prix'].str[:-1]
df1['Surface'] = df1['Surface'].str[:-2]
df1['Nb_pieces'] = df1['Nb_pieces'].str[:-1]
#enleve les lignes si les cellules :prix, surface, et nb pieces  sont non numeriques
df1 = df1[df1['Prix'].apply(lambda x: str(x).isdigit())]
df1 = df1[df1['Surface'].apply(lambda x: str(x).isdigit())]
df1 = df1[df1['Nb_pieces'].apply(lambda x: str(x).isdigit())]

df1.to_csv('Seloger_annonces_final.csv')

```

Nous créons une data frame à partir du fichier csv qu'on a obtenu grâce au web scraping du site Seloger, avec comme séparateur « ; » . *Python est le processeur d'analyse à utiliser lors de la lecture du fichier csv.*
Remplaçons les espaces par vide dans les colonnes Prix, Surface, Nb_pieces afin d'effectuer des corrections. De la colonne des prix nous ôtons le symbole « € » c'est à dire le dernier élément de des cellules des prix. « m2 » est retiré de la colonne des surfaces et « p » de la colonne des pièces.
Puis nous éliminons les annonces pour lesquels les cellules prix, surface et nombre de pièces sont non numériques. Si il manque une des données précédentes alors l'annonce est supprimée. ***df1.to_csv('Seloger_annonces_final.csv')*** → permet de créer un fichier csv avec cette nouvelle data frame corrigée.

La méthode générale pour web scraper le site de Pap et Logic Immo est semblable à la précédente. Cependant, quelques parties du code sont spécifiques à ces page web. Nous allons uniquement expliciter les parties du script qui diffèrent du premier.

## Script Pap :



```
import csv
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import pandas as pd
from time import sleep

with open('samvel.csv', 'w') as f:
        f.write("Type_de_bien;Prix;Surface;Arrondissement;Nb_pieces;Agence\n")


chrome_path = "C:\\Users\\IULIAN\\Desktop\\chromedriver"

driver = webdriver.Chrome(chrome_path)
driver.get('https://www.pap.fr/')

#entrer ville dans la barre de recherche
chercher_ville=driver.find_element_by_xpath('//*[@id="token-input-geo_objets_ids"]')
sleep(2)
chercher_ville.click()
sleep(2)
chercher_ville.send_keys("Paris")
sleep(2)
chercher_ville.send_keys(u'\ue007')
sleep(2)

chercher_ville.find_element_by_xpath('//span[@class="placeholder"]').click()
sleep(2)
chercher_ville.find_element_by_xpath('//li[@class="opt typesbien-vente typesbien-location"]').click()
sleep(2)
chercher_ville.find_element_by_xpath('//li[2][@class="opt typesbien-vente typesbien-location"]').click()
sleep(2)
chercher_ville.find_element_by_xpath('//input[@type="submit"]').click()
#localisation des données pertinentes
type_de_bien = driver.find_elements_by_xpath('//span[@class="h1"]')
prix = driver.find_elements_by_xpath('//span[@class="item-price"]')
surface= driver.find_elements_by_xpath('//ul[@class="item-tags"]/li[3]')
nb_de_pieces= driver.find_elements_by_xpath('//ul[@class="item-tags"]/li[1]')
type1 = len(prix)

# récupération des données sur l'ensemble des pages du site web
while True:

    link_next_page=driver.find_element_by_xpath('//ul[@class="pagination margin-top-50 margin-bottom-50"]/li[@class="next"]/a')
    if not link_next_page:
        break
    next_page_url = link_next_page.get_attribute("href")
    driver.get(next_page_url)
    type_de_bien = driver.find_elements_by_xpath('//span[@class="h1"]')
    prix = driver.find_elements_by_xpath('//span[@class="item-price"]')
    surface= driver.find_elements_by_xpath('//ul[@class="item-tags"]/li[3]')
    nb_de_pieces= driver.find_elements_by_xpath('//ul[@class="item-tags"]/li[1]')
    arrondissement = driver.find_elements_by_xpath('//p[@class="item-description"]')
    type1 = len(prix)

    with open('samvel.csv', 'a') as f:
        for i in range(type1):
            type_de_bien = driver.find_elements_by_xpath('//span[@class="h1"]')
            prix = driver.find_elements_by_xpath('//span[@class="item-price"]')
            surface= driver.find_elements_by_xpath('//ul[@class="item-tags"]/li[3]')
            nb_de_pieces= driver.find_elements_by_xpath('//ul[@class="item-tags"]/li[1]')
            arrondissement = driver.find_elements_by_xpath('//p[@class="item-description"]')
            try:
                f.write(type_de_bien[i].text[6:17]+";"+prix[i].text[0:-1]+";"+surface[i].text[0:3]+";"+arrondissement[i].text[0:9]+";"+nb_de_pieces[i].text[0]+ ";" + "Pap" +
            except UnicodeEncodeError:
                f.write( "\n")
            except IndexError :
                f.write( "\n")
driver.close
```

***from time import sleep*** : ici on importe le module de temps de Python qui a une fonction pratique appelée ***sleep ().*** Comme son nom l'indique, il met en pause notre programme Python. Python. **Sleep(2*)***, marque une pause de 2 secondes entre chaque action pour ralentir les actions.

**Explication script nettoyage des données PAP (ligne 70 à 75):**

***try :***
***f.write(type_de_bien[i].text[6:17]+";"+prix[i].text[0:-1]+";"+surface[i].text[0:3]+";"+arrondissement[i].text[0:9]+";"+nb_de_pieces[i].text[0]+ ";" + "Pap" + "\n")***

→ Ici nous demandons d'essayer d'écrire chaque élément récupéré par variable en format texte l'un après l'autre tant que l'on est dans la boucle, avec comme séparateur un « ; » et on revient à la ligne à la fin avec « \n »

***type_de_bien[i].text[6:17]*** → entre crochets nous avons les index 6 à 17, ceci pour récupérer seulement l'information qu'il nous faut dans cet élément à savoir le mot « appartement », le reste étant inutile.
***prix[i].text[0:-1]*** → l'index va de 0 à -1, -1 qui va éliminer le dernier élément de la liste, à savoir le symbole €, pour uniquement garder les chiffres.
***surface[i].text[0:3]*** → index de 0 à 3 pour conserver les chiffres uniquement, mais cela n'étant pas suffisant pour les surfaces n'ayant qu'un chiffre, nous opérons une autre correction dans un autre code que nous expliciterons le moment venu.
***arrondissement[i].text[0:9]*** → ici, nous avons les index 0 à 9 pour afficher l 'arrondissement.
Le xpath permettant de récupérer la donnée arrondissement récupère également d'autres informations dont nous n'avons pas l'utilité. Nous isolons l'information arrondissement en choisissant les index 0 à 9.

***nb_de_pieces[i].text[0]*** → ici 0 pour conserver uniquement le chiffre aussi.
***except UnicodeEncodeError:***

***f.write( "\n")***
→ nous demandons que si ***UnicodeEncodeError*** s'affiche, il ne doit rien écrire dans la ligne car il y a une annonce atypique qui ne possède pas les mêmes codes que les autres bien donc cette annonce est igonrée..

***except IndexError :***
***f.write( "\n")***
→ *Si* ***IndexError*** → si IndexError s'affiche, il ne doit rien écrire dans cette ligne car il y a forcément une information manquante donc inutile de récupérer les données pour ce bien.

Voici le script pour faire les dernières corrections nécessaires pour avoir un fichier propre et le plus homogène possible.

```
#nettoyage des données de PAP
df3= pd.read_csv('samvel.csv',sep=',',engine='python')
df3['Prix'] = df3['Prix'].str.replace(".","")
df3['Surface'] = df3['Surface'].str.replace("m","")
df3['Arrondissement'] = df3['Arrondissement'].str.replace("e","")

df3 = df3[df3['Prix'].apply(lambda x: str(x).isdigit())]
df3 = df3[df3['Surface'].apply(lambda x: str(x).isdigit())]
df3 = df3[df3['Nb_pieces'].apply(lambda x: str(x).isdigit())]
df3.to_csv('sam_final.csv')
```

***df3['Prix'] = df3['Prix'].str.replace(".","")*** → dans les prix, il y a des points pour les milliers, ce que n'ont pas les deux autres site, ceci est propre au site Pap.fr. Nous avons donc remplacé les points par vide donc les points disparaissent.

***df3['Surface'] = df3['Surface'].str.replace("m","")*** → Nous avons effectué cette correction car pour les surface à un chiffre, le « m » s'affichait toujours donc nous le supprimons .

***df3['Arrondissement'] = df3['Arrondissement'].str.replace("e","")*** → Nous enlevons le « e » pour conserver le format *Paris 17* par exemple au lieu de *Paris 17e.*

***df3.to_csv('sam_final.csv')*** → permet d'exporter cette nouvelle data frame corrigée dans un fichier csv.

## Script Logic Immo :

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import pandas as pd
chrome_path=r"C:\Users\Xenia\Downloads\chromedriver_win32\chromedriver.exe"
driver=webdriver.Chrome(chrome_path)
driver.get("https://www.logic-immo.com/")#open the web site
driver.find_element_by_xpath('/html/body/div[6]/div/div/div/div/div[2]/button[2]').click()
input_paris=driver.find_element_by_xpath('//*[@id="s2id_autogen1"]')
input_paris.send_keys('Paris')
input_paris.send_keys(Keys.ENTER)
driver.find_element_by_xpath('//*[@id="js-submitBtnHD"]').click()
total=[]
vente_app=driver.find_elements_by_class_name("offer-details-type")
items=len(vente_app)
for i in range(items):
        prices=driver.find_elements_by_class_name('offer-details-price')
        rooms=driver.find_elements_by_class_name("offer-details-caracteristik--rooms")
        location=driver.find_elements_by_class_name("offer-details-location")
        vente_app=driver.find_elements_by_class_name("offer-details-type")
        area=driver.find_elements_by_class_name("offer-details-caracteristik--area")

        new=((vente_app[i].text,prices[i].text,area[i].text,location[i].text,rooms[i].text))
        total.append(new)
while True:
    link_next_page=driver.find_element_by_xpath('//*[@id="js-content"]/section[5]/div/div/div/div/
    if not link_next_page:
       break
    next_page_url = link_next_page.get_attribute("href")
    print(next_page_url)
    driver.get(next_page_url)
    prices=driver.find_elements_by_class_name('offer-details-price')
    rooms=driver.find_elements_by_class_name("offer-details-caracteristik--rooms")
    location=driver.find_elements_by_class_name("offer-details-location")
    vente_app=driver.find_elements_by_class_name("offer-details-type")
    area=driver.find_elements_by_class_name("offer-details-caracteristik--area")
    items=len(vente_app)
    for i in range(items):
        prices=driver.find_elements_by_class_name('offer-details-price')
        rooms=driver.find_elements_by_class_name("offer-details-caracteristik--rooms")
        location=driver.find_elements_by_class_name("offer-details-location")
        vente_app=driver.find_elements_by_class_name("offer-details-type")
        area=driver.find_elements_by_class_name("offer-details-caracteristik--area")

        new=((vente_app[i].text,prices[i].text,area[i].text,location[i].text,rooms[i].text))
        total.append(new)
        print(total)
df = pd.DataFrame(total,columns=['Type_de_bien','prix','surface','arrondissement','nb_pieces'])
df['prix']=df['prix'].str.replace(' ','')
df[['Prix','AUtre1']]=df['prix'].str.split('€',n=1,expand=True)
df[['Surface','autre']]=df['surface'].str.split(' ',n=1,expand=True)
df[['Arrondissement','Autre2']]=df['arrondissement'].str.split('(',n=1,expand=True)
df[['Nb_pieces','autre3']]=df['nb_pieces'].str.split(' ',n=1,expand=True)
df[['type','Type_de_bien']]=df['type_de_bien'].str.split(' ',n=1,expand=True)
df=df.drop(columns=['AUtre1','prix','autre3','Autre2','arrondissement','nb_pieces','surface','autr
df=df[['Type_de_bien','Prix','Surface','Arrondissement','Nb_pieces']]
df.to_csv('immo_logic_results.csv')
driver.close()
quit()
```

**Explication script nettoyage des données Logic Immo (ligne 55-63):**

Sur immo-logic.com les prix sont récupérés avec le prix par mois et le texte supplémentaire.
***df['prix']=df['prix'].str.replace(' ','')***
***df[['Prix','AUtre1']]=df['prix'].str.split('€',n=1,expand=True)***
La méthode str.split() permet de séparer la colonne en 2 colonnes différentes où la première contient l'information nécessaire et dans la deuxième est l'information inutile. De cette façon il est possible de garder uniquement la partie voulue.
**Str.replace** nous permet de supprimer les espaces. Cette méthode permet de supprimer les espaces entres les chiffres. En utilisant ces deux méthodes, il est possible de manipuler les données afin de les nettoyer.
Cependant, ***str.split*** crée des colonnes inutiles pour le fichier final. Pour cette raison il faut utiliser le script suivant :
***df=df.drop(columns=['nom_de_colonne1','nom_de_colonne2'])***
Et ***df=df[['l'ordre des colonnes dans fichier final']]*** => pour placer les colonnes dans l'ordre désirable.
Quand toutes les manipulations sont faites, il faut réunir les data frame qui contiennent les données. La concaténation des tables est faite à l'aide de la fonction :
***df=pd.concat([df1,df2],ignore_index=True)***
C'est une fonction de concaténation de pandas qui regroupe les data frame **df1** et **df2** dans une seule data frame ***df***. ***Ignore_index =True*** va ignorer les indexes des tables et attribue des nouveaux indexes.
Le fichier final regroupant les annonces de Logic Immo est prêt à être utilisé.

### b)Etape 3 - Programmation de la requête affichant les biens immobiliers pertinents en fonction des critères des acheteurs : prix et surface.

```
import csv
import pandas as pd

df1= pd.read_csv('Seloger_annonces_final.csv',sep=',',engine='python')
# Les colonnes Prix,Surface et Nb_pieces de la data frame sont desormais des int
df1[['Prix','Surface','Nb_pieces']] = df1[['Prix','Surface','Nb_pieces']].apply(pd.to_numeric)

df2= pd.read_csv('immo_results_clean.csv',engine='python')
df2= df2[df2['Prix'].apply(lambda x: str(x).isdigit())]
df2 = df2[df2['Surface'].apply(lambda x: str(x).isdigit())]
df2 = df2[df2['Nb_pieces'].apply(lambda x: str(x).isdigit())]
# Les colonnes Prix,Surface et Nb_pieces de la data frame sont desormais des int
df2[['Prix','Surface','Nb_pieces']] = df3[['Prix','Surface','Nb_pieces']].apply(pd.to_numeric)

df3= pd.read_csv('sam_final.csv',sep=',',engine='python')
# Les colonnes Prix,Surface et Nb_pieces de la data frame sont desormais des int
df3[['Prix','Surface','Nb_pieces']] = df3[['Prix','Surface','Nb_pieces']].apply(pd.to_numeric)

#concaténation des 3 data frame
df4=df1.append(df2, ignore_index=True)
df=df4.append(df3, ignore_index=True)

#verification des types de Prix, Surface et Nb_pieces
print (df.dtypes)
#on exporte la data frame final dans un fichier csv
df.to_csv('cleaned_annonces.csv')

#requete dans la data frame
budget_maximum=input(" Entrez votre budget maximum: ")
budget_maximum=int(budget_maximum)
Surface_client= input(" Entrez la surface souhaitée: ")
Surface_client=int(Surface_client)
print ("bugdet maximum=", budget_maximum, "Surface_client=" , Surface_client)
print(df.loc[(df['Prix'] <= budget_maximum) & (df['Surface'] >= Surface_client),: ])


```

Nous importons les données collectées des sites Seloger, Logic Immo et Pap.

***df1= pd.read_csv('Seloger_annonces_final.csv',sep=',',engine='python')*** => Lecture d'un data frame à partir d'un fichier csv, le separateur est « , » et le fichier cvs est lu avec le moteur d'analyse de Python.

Les colonnes Prix, Surface et Nb_pieces de df1, df2 et df3 sont converties en type int64 afin de pouvoir effectuer des requêtes sur ces trois colonnes.
Puis, nous concaténons les 3 data frame et cette data frame est exportée dans un fichier csv : « cleaned_annonces.csv ».

Nous pouvons désormais programmer la requête affichant les biens immobiliers pertinents en fonction des critères prix et surface.

La fonction input permet d'interagir avec l'utilisateur. Ainsi, l'utilisateur spécifie budget_maximum et surface_client. Les types de budget_maximum et surface_client sont transformés en int pour pourvoir effectuer des comparaisons avec les données de la colonne Prix et Surface. Nous pouvons isoler les sous-ensembles d'observations répondant à des critères définis sur les champs. Nous utiliserons la **méthode .loc[,]** dans ce cadre. L'opérateur logique & permet de combiner les conditions

***print(df.loc[(df['Prix'] <= budget_maximum) & (df['Surface'] >= Surface_client),: ])*** affiche les biens immobiliers dont les prix sont inférieurs ou égaux à budget_maximum et dont les surfaces sont supérieurs ou égaux à surface_client.

## 3)Les difficultés rencontrées

La première difficulté rencontrée est la gestion des données manquantes ou mal placées. Après avoir extrait les données à partir du site Logic Immo, nous avons comparé les longueurs des listes surface et des autres listes. Les longueurs étant différentes, nous avons donc conclu que certaines annonces ont des données manquantes. Sans des données complètes nous ne pouvons pas poursuivre notre projet. En scrapant les données sur l'ensemble des pages, il n'est pas possible d'inspecter manuellement tous les éléments manquants. Un autre exemple, sur le site Se loger, à quelque moment la donnée surface est manquante et lorsque celle-ci est manquante d'autres données telles que étage ou ascenseur sont insérées à l'emplacement dédié à la donnée surface. Nous avons donc décidé de poursuivre notre projet en travaillant uniquement sur un échantillon d'annonces pour lesquelles les données extraient sont complètes

En récoltant les données, nous n'avions pas mesurer l'importance de l'étape nettoyage des données et le temps nécessaire. Le web scraping est beaucoup plus que l'obtention de données brutes à partir d'un site web. L'étape nettoyage des données était fastidieuse, les données brutes n'étant pas homogènes sur chacun des sites. Il est nécessaire de nettoyer les données pour l'étape 3 du projet. Malgré, avoir nettoyé les données aux mieux nous n'avons pas réussis à retirer certaines adresses de la colonne arrondissement.

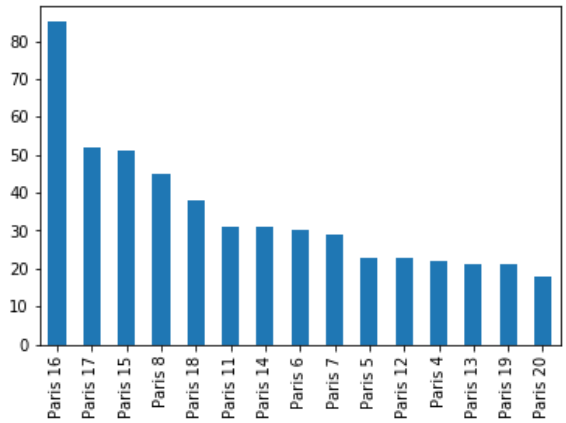
# III. Etude statistique à partir des données collectées

Nous avons effectué une étude statistique descriptive succincte sur les données extraites à partir du site Logic Immo. Notre échantillon est composé de n= 755 annonces. Nous utilisons la librairie Pandas afin d'effectuer l'étude. Pandas est une librairie Python spécialisée dans l'analyse des données

## A. Graphique à barre : Arrondissement

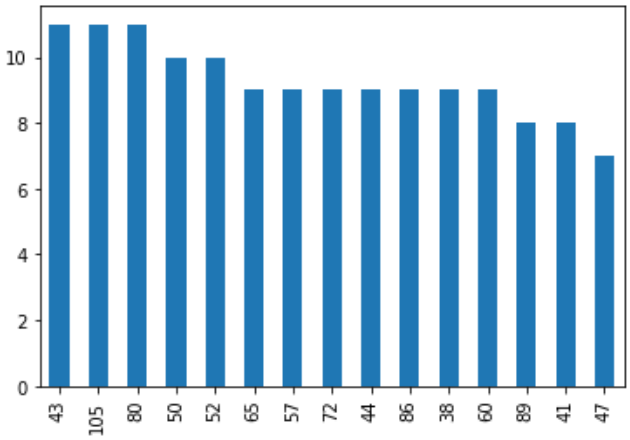
```
arrondi_counts=pg['Arrondissement'].value_counts()
arrondi_counts[:15].plot(kind='bar')
```

Cette commande calcule le nombre d'occurrence de chaque arrondissement puis effectue l'histogramme des 15 arrondissements les plus redondantes.



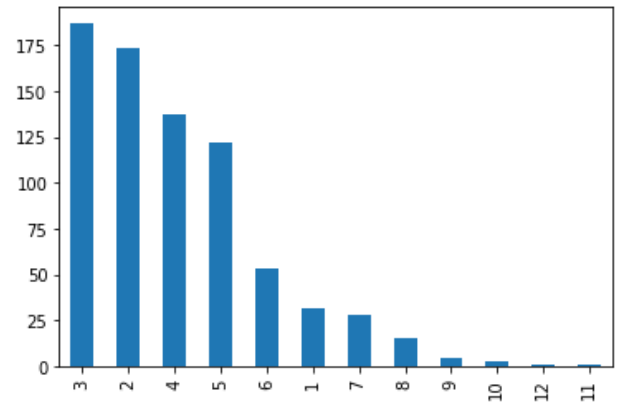

Les offres de ventes dans le 16e arrondissement sont nettement supérieures aux offres de ventes dans les autres arrondissements de Paris. Le 16e regroupe 84 offres de ventes.
Puis, le 17e et le 15e regroupe également de nombreuses offres de ventes.
Le contraste entre le 16e et les autres arrondissements peut s'expliquer par le fait que le 16e se distingue du reste de Paris par sa forte proportion de logements de grande taille et le prix du m² est d'en moyenne 10.700 €. Le prix des biens dans le 16ème arrondissement se trouve donc au-dessus de la moyenne à Paris rendant les biens immobiliers du 16e inaccessible pour la plupart des français.

## B. Graphique à barre : Surface




Le graphique ci-dessus représente les 15 surfaces les plus représentés au sein de notre échantillon. La majorité des biens immobiliers en vente sont des biens à grande surface ou à moyenne surface ( 43 m2,105m2 et 80 m2). Nous constatons que les biens à petites surfaces sont très peu nombreux au sein de l'échantillon voire absent. Tous les profils de logements n'ont pas la même cote à Paris. Les petites surfaces sont très peu représenté car ils sont très convoité et ces biens sont vendu en un clin d'œil. Ils ne restent donc pas longtemps sur le marché.

## C. *Graphique à barre : Nombre de pièces*




Les classes les plus représentées sont les biens possédant 3 pièces, 2 pièces et 4 pièces. Les studios sont peu représentés et les biens de 9,10,11,12 pièces sont les plus rares.

## D. *Médiane : Prix, Nombre de pièces et Surface*

**df['Prix'].median()**

La commande ci-dessus permet de calculer la médiane pour les colonnes Prix, Surface et Nb_pieces de la data frame

## *E. Les médianes*

| | |
|---|---|
| Surface | **82** |
| Prix | **969 000** |
| Nb_pieces | **3** |

**Interprétation :** Pour les données ordonnées, la médiane des prix est de 969 000€. Autrement dit, la moitié des biens immobiliers au sein de notre échantillon à un prix inférieure ou égale à 969 000€ et l'autre moitié est supérieure ou égale. La médiane des surfaces et de 82 m2 et la médiane des nombres de pièces est de 3 pièces.

## *F. Analyse statistique basique*

```
top=[]
top.append(df.describe(include='all'))
print(flats.describe(include='all')
```

Pandas describe() est utilisé pour obtenir une analyse statistique de base comme percentile, moyenne, std etc. d'une base de données ou d'une série de valeurs numériques.
Pour les colonnes avec des chaînes de caractéres c'est à dire pour les colonnes Type_de_bien et Arrondissement, Nan a été retourné pour les opérations numériques.

| | Type_de_bien | Prix | Surface | Arrondissement | Nb_pieces |
|---|---|---|---|---|---|
| **count** | 755 | 7.550000e+02 | 755.000000 | 755 | 755.000000 |
| **unique** | 8 | NaN | NaN | 57 | NaN |
| **top** | Appartement | NaN | NaN | Paris 16 | NaN |
| **freq** | 414 | NaN | NaN | 84 | NaN |
| **mean** | NaN | 1.406610e+06 | 100.159987 | NaN | 3.744371 |
| **std** | NaN | 1.230767e+06 | 68.343807 | NaN | 1.735747 |
| **min** | NaN | 5.450000e+04 | 6.000000 | NaN | 1.000000 |
| **25%** | NaN | 6.270000e+05 | 54.010000 | NaN | 2.000000 |
| 50% | NaN | 9.690000e+05 | 82.000000 | NaN | 3.000000 |
| 75% | NaN | 1.855000e+06 | 122.000000 | NaN | 5.000000 |
| max | NaN | 1.390000e+07 | 700.000000 | NaN | 12.000000 |

**Interprétation :**

Ligne **Count** représente le nombre d'observations, ici 755 observations.

La fonction **unique** renvoie les valeurs uniques présentes dans une structure de données Pandas. Pour Type de bien ce sont les appartement, loft, maison etc. Pour arrondissement il y a 57 valeurs uniques. Or, Il devrait y avoir 20 valeurs une pour chaque arrondissement. Les autres 35 valeurs s'expliquent par le fait que certains biens ont des addreses au lieu d'un arrondissement.

**Top** est une valeur de unique qui est la plus populaire, qui se répète plus de fois que les autres. Pour Type de bien c'est évidement appartement comme à Paris les ventes d'appartements étant supérieurs aux autres biens. Pour arrondissements c'est Paris 16. Ce qui signifie qu'il y a plus d'appartements qui se vendent dans arrondissement 16.

**Freq** donne la fréquence de la valeur **top.** Nous avons 84 appartements dans le 16e arrondissement.

Pour les colonnes numeriques les lignes unique, top et frequence ne peuvent pas être mesurées.

**Mean** est la moyenne, c'est-à-dire elle représente la somme de toutes les observations, divisée par le nombre d'observations.
Par exemple, pour la colonne Prix c'est la somme des prix divisé par **count** qui est égal à 755. Le prix moyen des biens immobiliers en vente est de 1.406610e+06 au sein de notre échantillon. La surface moyenne est d'environs de 100 m2 avec 3.74 pièces en moyenne.

**Std** est standard deviation ou encore écart-type en français qui est une racine de variance. Std est une mesure de la dispersion des valeurs d'un échantillon. Pour les prix, L'écart type est de 1 230 767 €. Pour la colonne Surface la dispersion est de 68.34 m^2. Pour le nombre de pieces la dispersion est de 1.74 pieces.

**Min** est une valeur minimale dans la colonne. Pour le prix nous voyons que le bien le moins cher coûte 54 500€. La surface la plus petite est de 6 m^2. Le minimum de nombre de pièces 1 pièce.

**25%,** est le 1e quartile qui désigne que 25 % des données sont inférieures ou égales à cette valeur. Pour les prix, 25% des prix sont inferieurs ou égaux à 627 000€. Pour la surface ,25% des surfaces sont inferieurs ou égales à 54 m^2. Pour la dernière colonne, 25% des biens en ventes ont moins ou exactement 2 pièces.

**75%** est le 3e quartile indique que 75 % des données sont inférieures ou égales à cette valeur. 75% des appartements ont un prix inférieures ou égaux à 1 855 000€. 75% des surfaces sont inférieures ou égales à 122 m^2. Finalement , 75% des biens ont 5 pieces ou moins.

**Max** est une valeur maximale. Le prix maximal est de 13 900 000€. La surface maximale est 700 m^2 et au maximum nous avons 12 pieces pour les biens en vente de notre échantillon.

# *IV. Conclusion*

Le web scraping est un processus par lequel des sites web particuliers sont crawlés et des données importantes sont extraites. Ces informations sont ensuite formatées et stockées dans une base de données pour un usage ultérieur. Il peut être un outil utile, et son efficacité et ses avantages pour le marché immobilier ne peuvent être ignorés. Les données recueillis peuvent être exploitées de maintes façons.

Cet outil, peut permettre à un investisseur de prendre des décisions basées sur des données qualitatives et pertinentes, plutôt que des informations incomplètes ( II) Projet chasseur immobilier virtuel).

Pour les sociétés immobilières, le web scraping de sites web immobiliers peut être un moyen inestimable de trouver des informations pertinentes et exploitables. Pour les professionnels, le web scraping peut représenter un moyen de surveiller le marché et la concurrence en effectuant des analyses statistiques sur les données collectées ( III) Etude statistique).