40_NP_data_aquisition

December 15, 2023

1 Data aquisition

1.1 Data aquisition

Un modèle de data science se décompose comme suit:

1. Collecte des données
2. Nettoyage des données
3. Modélisation des données
4. Prédiction des données

Ce jupyter porte uniquement sur le point 1, la collecte de données

1.2 Techniques de collecte de données

Il existe plusieurs techniques de collecte de données:

- Scraper
- Web scraping
- Web scraping avec Selenium
- REST API
- DataSets (Kaggle, UCI, yfinance, seaborn, etc.)

1.2.1 Scrapper

Le scrapper est une technique de collecte de données qui permet de collecter les données d'une page web ou autre support.

1.2.2 Web scraping

Le web scraping est une technique de collecte de données qui permet de collecter les données d'une page web statique.

1.2.3 Web scraping avec Selenium

Le web scraping avec Selenium est une technique de collecte de données qui permet de collecter les données d'une page web dynamique. Contrairement au web scraping, le web scraping avec Selenium permet de collecter les données d'une page web dynamique. Il agit comme un humain et permet de collecter des données qui sont visible seulement après certaines actions (clic, scroll, etc.).

1.2.4 REST API

Les API REST sont des API en ligne qui permettent de retourner des données à la suite d'une requête HTTP. Les données sont retournées dans un format standardisé (JSON, XML, etc.).

1.2.5 DataSets

Les DataSets sont des jeux de données qui sont disponibles en ligne. Ils sont souvent utilisés pour l'entrainement de modèles de machine learning.

1.3 L'objet requests

L'objet requests permet de faire des requêtes HTTP. Il permet de récupérer le contenu d'une page web. Nous utiliseront cet objet pour faire du web scraping. Car il faut récupérer le contenu d'une page web à l'aide d'une requête avant de pouvoir faire du web scraping.

Pour installer l'objet requests, il faut utiliser la commande suivante:

```
conda install -c anaconda requests
```

[ ]:
```python
import requests

wiki = requests.get("https://en.wikipedia.org/wiki/
 ↪Python_(programming_language)")

print(wiki.status_code)
print(wiki.text)
print(wiki.json())
```

1.3.1 Web scraping

Pour faire du web scraping **simple**, on peut utiliser la librairie BeautifulSoup.

pour installer executer la commande suivante dans l'environement conda:

```
conda install -c anaconda beautifulsoup4
```

BeautifulSoup BeautifulSoup est une librairie qui permet de faire du web scraping. Elle permet de récupérer des données d'une page web statique. Elle permet de `parser` le code HTML d'une page web et de récupérer les données qui nous intéresse.

[ ]:
```python
from bs4 import BeautifulSoup
import requests
```

[ ]:
```python
url = "https://quotes.toscrape.com/page/2/"

# Get the HTML page
response = requests.get(url)
#print(response.text)

# Parse the HTML page
```

```
soup = BeautifulSoup(response.text, "html.parser")
print(soup.prettify())
```

[ ]:
```
# Get the quotes
quotes = soup.find_all("div", class_="quote")
#quotes = soup.select(".quote")

# Get the author and the quote
for quote in quotes:
    author = quote.find("small", class_="author").text
    quote_text = quote.find("span", class_="text").text
    print(f"{author} said: {quote_text}")
```

[ ]:
```
tags_dict = {}
# count all tags
for quote in quotes:
    tags = quote.find_all("a", class_="tag")
    for tag in tags:
        tag_text = tag.text
        if tag_text in tags:
            tags_dict[tag_text] += 1
        else:
            tags_dict[tag_text] = 1

print(tags_dict)
tags_dict["heartbreak"] += 1
print(max(tags_dict, key=tags_dict.get))
```

BeautifulSoup Methods voici une liste des methodes de BeautifulSoup:

Methode	Description
`find()`	Retourne le premier élément qui correspond au filtre
`find_all()`	Retourne tous les éléments qui correspondent au filtre
`find_parent()`	Retourne le parent de l'élément
`find_next_sibling()`	Retourne le prochain élément du même niveau
`find_previous_sibling()`	Retourne le précédent élément du même niveau
`find_next()`	Retourne le prochain élément
`find_previous()`	Retourne le précédent élément
`find_all_next()`	Retourne tous les éléments suivants
`find_all_previous()`	Retourne tous les éléments précédents
`select()`	Retourne tous les éléments qui correspondent au filtre CSS
`select_one()`	Retourne le premier élément qui correspond au filtre CSS
`prettify()`	Retourne le code HTML formaté

Methode	Description

BeautifulSoup Attributes voici une liste des attributs de BeautifulSoup:

Attribut	Description
`name`	Retourne le nom de la balise
`attrs`	Retourne les attributs de la balise
`string`	Retourne le contenu de la balise
`text`	Retourne le contenu de la balise
`contents`	Retourne le contenu de la balise
`children`	Retourne les enfants de la balise
`parent`	Retourne le parent de la balise
`next_sibling`	Retourne le prochain élément du même niveau
`previous_sibling`	Retourne le précédent élément du même niveau
`next`	Retourne le prochain élément
`previous`	Retourne le précédent élément

1.3.2 Web scraping with Selenium

Pour faire du web scraping **avancé**, on peut utiliser la librairie Selenium. Elle permet de faire du web scraping sur des pages web dynamiques. Selenium permet de simuler un navigateur web et d'interagir avec une page web comme un humain. On pourra donc récupérer des données qui sont visible seulement après certaines actions (clic, scroll, etc.).

Note: `Selenium` est souvent utilisé pour faire du web unit testing

Selenium Installation Pour utiliser Selenium, il faut d'abord installer la librairie:

```
conda install -c conda-forge selenium
```

Ensuite il faut installer le driver du navigateur web que l'on souhaite utiliser:

- Chrome : https://googlechromelabs.github.io/chrome-for-testing/#stable
- Edge : https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
- Firefox : https://github.com/mozilla/geckodriver/releases

Selenium - XPATH XPATH est un langage de requête qui permet de sélectionner des éléments dans un document XML ou HTML. Il permet de sélectionner des éléments dans une page web. voir ici. Il est possible d'utiliser XPATH dans Selenium.

XPATH Syntax

Expression	Description
`nodename`	Selects all nodes with the name "nodename"
`/`	Selects from the root node

Expression	Description
`//`	Selects nodes in the document from the current node that match the selection no matter where they are
`.`	Selects the current node
`..`	Selects the parent of the current node
`@`	Selects attributes

XPATH Examples

Expression	Description
`//span[@class='a-size-medium a-color-base a-text-normal']`	Selects all span elements with the class "a-size-medium a-color-base a-text-normal"
`//span[@class='a-size-medium a-color-base a-text-normal']/text()`	Selects all span elements with the class "a-size-medium a-color-base a-text-normal" and returns the text
`//div[@id='nav-xshop']/a`	Selects all a elements in the div with the id "nav-xshop"

Selenium - CSS CSS est un langage qui permet de définir le style d'un document HTML. Il permet de sélectionner des éléments dans une page web. voir ici. Il est possible d'utiliser CSS dans Selenium.

CSS Syntax

Expression	Description
`*`	Selects all elements
`#id`	Selects the element with the id "id"
`.quote`	Selects all elements with the class "class"
`element`	Selects all elements with the tag "element"
`element.class`	Selects all elements with the tag "element" and the class "class"
`element,element`	Selects all elements with the tag "element" or "element"
`element element`	Selects all elements with the tag "element" inside the element with the tag "element"

CSS Examples

Expression	Description
`span.a-size-medium.a-color-base.a-text-normal`	Selects all span elements with the class "a-size-medium a-color-base a-text-normal"
`span.a-size-medium.a-color-base.a-text-normal`	Selects all span elements with the class "a-size-medium a-color-base a-text-normal"

Expression	Description
`div#nav-xshop a`	Selects all a elements in the div with the id "nav-xshop"

1.4 Selenium example

Selenium Methods voici une liste des methodes de Selenium:

Methode	Description
`get()`	Ouvre une page web
`find_element_by_id()`	Retourne le premier élément qui correspond à l'id
`find_element_by_name()`	Retourne le premier élément qui correspond au name
`find_element_by_xpath()`	Retourne le premier élément qui correspond au xpath
`find_element_by_link_text()`	Retourne le premier élément qui correspond au link text
`find_element_by_partial_link_text()`	Retourne le premier élément qui correspond au partial link text
`find_element_by_tag_name()`	Retourne le premier élément qui correspond au tag name
`find_element_by_class_name()`	Retourne le premier élément qui correspond au class name
`find_element_by_css_selector()`	Retourne le premier élément qui correspond au css selector
`find_elements_by_id()`	Retourne tous les éléments qui correspondent à l'id
`find_elements_by_name()`	Retourne tous les éléments qui correspondent au name
`find_elements_by_xpath()`	Retourne tous les éléments qui correspondent au xpath
`find_elements_by_link_text()`	Retourne tous les éléments qui correspondent au link text
`find_elements_by_partial_link_text()`	Retourne tous les éléments qui correspondent au partial link text
`find_elements_by_tag_name()`	Retourne tous les éléments qui correspondent au tag name
`find_elements_by_class_name()`	Retourne tous les éléments qui correspondent au class name
`find_elements_by_css_selector()`	Retourne tous les éléments qui correspondent au css selector
`click()`	Clique sur l'élément
`clear()`	Efface le contenu de l'élément
`send_keys()`	Envoie des données à l'élément
`submit()`	Soumet le formulaire

1.4.1 REST API

Les API REST sont des API en ligne qui permettent de retourner des données à la suite d'une requête HTTP. Les données sont retournées dans un format standardisé (JSON, XML, etc.).

L'url d'une API REST est appelé `endpoint`. Il permet de faire une requête HTTP à l'API REST.

L'API rest se comporte differement en fonction de la méthode HTTP utilisée.

you can find an example of a mock rest api here

[ ]:
```python
import requests

url = "https://jsonplaceholder.typicode.com/users"

users = requests.get(url)
users = users.json()
for user in users:
    print(user["name"])
```

[ ]:
```python
posts = requests.get("https://jsonplaceholder.typicode.com/posts").json()

post_count = {}
for post in posts:
    if post["userId"] in post_count:
        post_count[post["userId"]] += 1
    else:
        post_count[post["userId"]] = 1
post_count
```

1.4.2 DATASETS

Les datasets sont des fichiers qui contiennent des données. Ils sont crées par d'autres personnes et mis à disposition sur internet. Ils sont souvent utilisés pour l'entrainement de modèles de machine learning.

Différents moyen de récupérer des datasets: - Google Colab - Kaggle - UCI - Seaborn - yfinance - pandas_datareader - scikit-learn - tensorflow_datasets - pytorch_datasets - pytorch_datasets

Datasets - Seaborn, titanic Le dataset suivant est une base de données qui contient les informations relatives à la survie des passagers du titanic.

[ ]:
```python
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

titanic = sns.load_dataset('titanic')
print(titanic.head())
```

[ ]:
```
# Plot the distribution of the age
sns.distplot(titanic['age'])
```