Stratégies d'arbitrage dans les paris sportifs

Léo Pouilly

CPGE Scientifique

Sommaire

- Définition & Intérêt d'un arbitrage
- La martingale derrière les paris
- Extraction des données
- Analyse asymptotique

Définition et Explication de l'intérêt d'un arbitrage

- **Dans le cas des paris sportifs les arbitrages ("surebet") permettent:**

Tirer profit des différences de cotes offertes par les ≠ bookmakers pour un = événement, pour assurer une victoire:

Considérons un match de tennis entre un joueur A et un joueur B: A chaque événement (i) on associe la cote Ci. Ainsi, le principe d'arbitrage pour un parieur nécessite:

$$\sum_{i=1}^{2} \frac{1}{C_i} \leq \frac{1}{\alpha + 1} \quad avec \quad \alpha > 0$$

- **<u>Enjeu</u>**: récolter un maximum de données pour établir le meilleur arbitrage

Avantages & Inconvénients

- +++:

Profit assuré: obtention d'un gain quelque soit l'issu du match

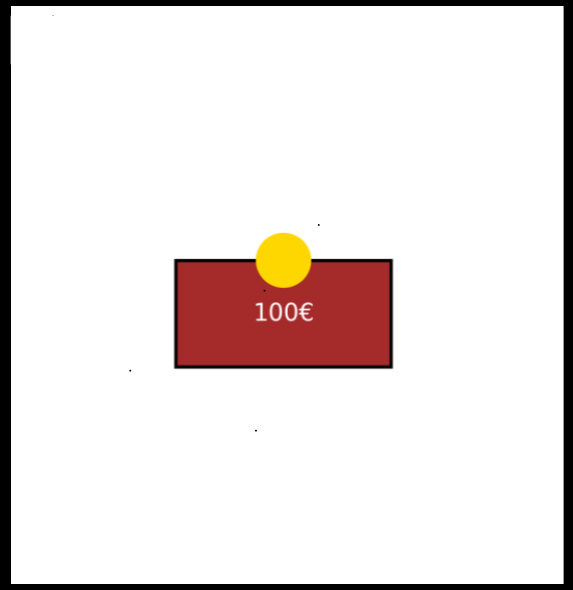
Risque minime: stratégie à faible risque car indépendante de l'issue de l'événement

- --- :

Exploitation difficile: fluctuations sans cesse des cotes

Limitation de compte: établit par les bookmakers --> réduisant l'utilisation de cette stratégie

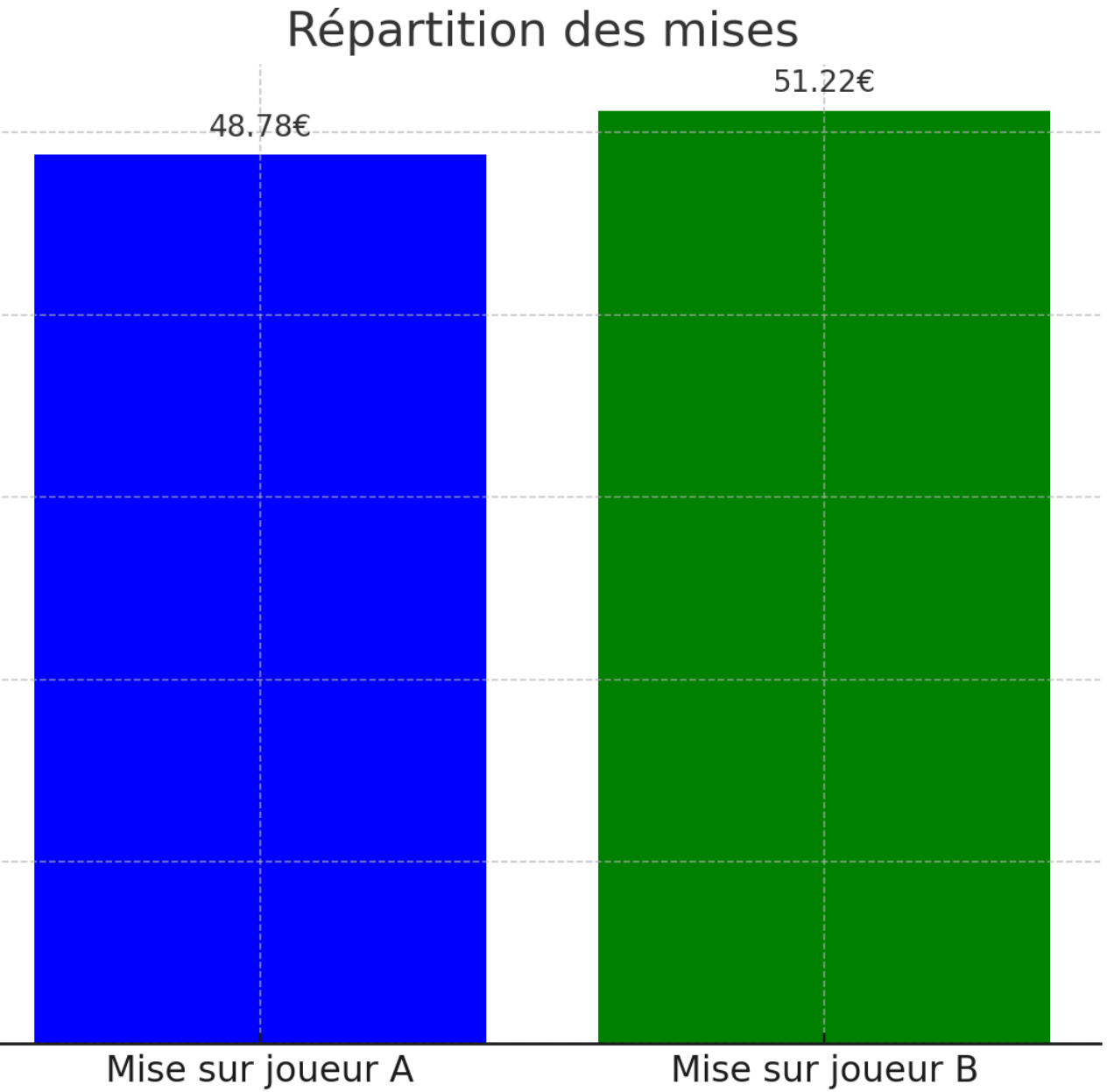
Illustration

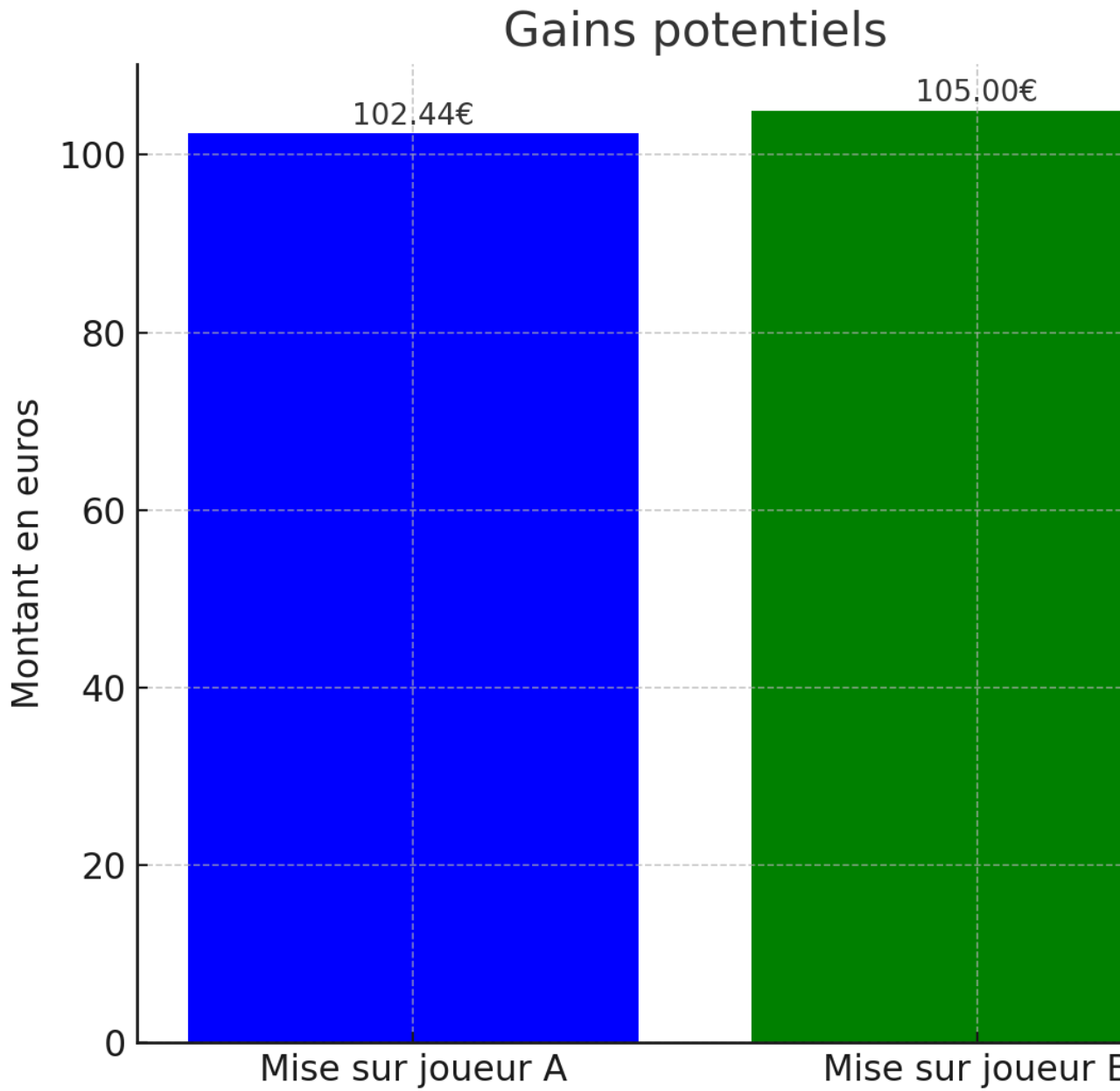


Cotes des Bookmakers pour les Joueurs A et B

Bookmaker	Cote Joueur A	Cote Joueur B
Bookmaker 1	2.1	N/A
Bookmaker 2	N/A	2.05

Exemple d'arbitrage dans les paris sportifs





Lien entre arbitrage & martingale

- **Objectif de gain** identique
- **Gestion du risque** assez différente selon la martingale employée
- **Exploitations des opportunités** (dans les marchés de paris):

→ Martingale: trouver des événements à parier de manière répétitive

→ Arbitrage: trouver des différences significatives de cotes

Définition

définition : Filtration

Une filtration A est une suite croissante $F = \{\mathcal{F}_n\}_{n \geq 0}$ *de sous − tribus de A*

$$\mathcal{F}_0 \subset \mathcal{F}_1 \subset \mathcal{F}_2 \subset \ldots \subset A$$

On dit que $(\Omega, A, F, \mathrm{P})$ *est un espace probabilisé filtré*

En particulier si, $\{X_n\}_{n \geq 1}$ *est un processus aléatoire. Alors la suite*

$$F_n = \sigma\left(X_i \ , \ i \leq n\right) \text{ avec } n \geq 0$$

est appelée la filtration naturelle du processus

Définition

- Une martingale est un processus aléatoire dont l'espérance conditionnelle par rapport au passé reste constante:

définition : Martingale

Soit $X = \{X_n\}_{n \geq 0}$ *un processus adapté à l'espace probabilisé filtré* (Ω, A, F, P)
Si X_n *est intégrable pour tout* n $(ie\ E(|X_n|) < +\infty)$, *on dit que* X *est :*

– une martingale si

$$E(X_n|F_{n-1}) = X_{n-1} \text{ avec } n \geq 1$$

– une sous – martingale si

$$E(X_n|\ F_{n-1}) \geq X_{n-1} \text{ avec } n \geq 1$$

Modélisation d'une situation de pari (match de tennis) par une sous martingale

Marche aléatoire dans $\mathbb{R}$

Soit $\{\xi_n\}_{n\geq 1}$ = *VAD dans* $\mathbb{R}$ (*gain ou perte au* $n-$ *ième pari*)

issu d'un match est incertaine donc $E(\xi_n) = 0$

La marche aléatoire $S = \{S_n\}_{n\geq 0}$ *telle que :*

$$\forall\ n \geq 1,\ S_n = \sum_{i=0}^{n} \xi_i \quad \text{avec } S_0 = 0 \text{ est une martingale}$$

$$\text{pour la filtration } \sigma\left(\xi_i\ ,\ i \leq n\right)$$

$$E(S_{n+1}|\ F_n) = E(S_n + \xi_{n+1}|\ F_n) = E(S_n|F_n) + E(\xi_{n+1}|F_n) = S_n + E(\xi_n)$$

Or, dans le cas d'un arbitrage : $E(\xi_n) \geq 0$

$$D'où\ E(S_{n+1}|F_n) \geq S_n$$

Extraction des données

- Pour "scraper" (= récolter) les données:

- Récupération du code source des différentes pages de statistiques qui nous interresse

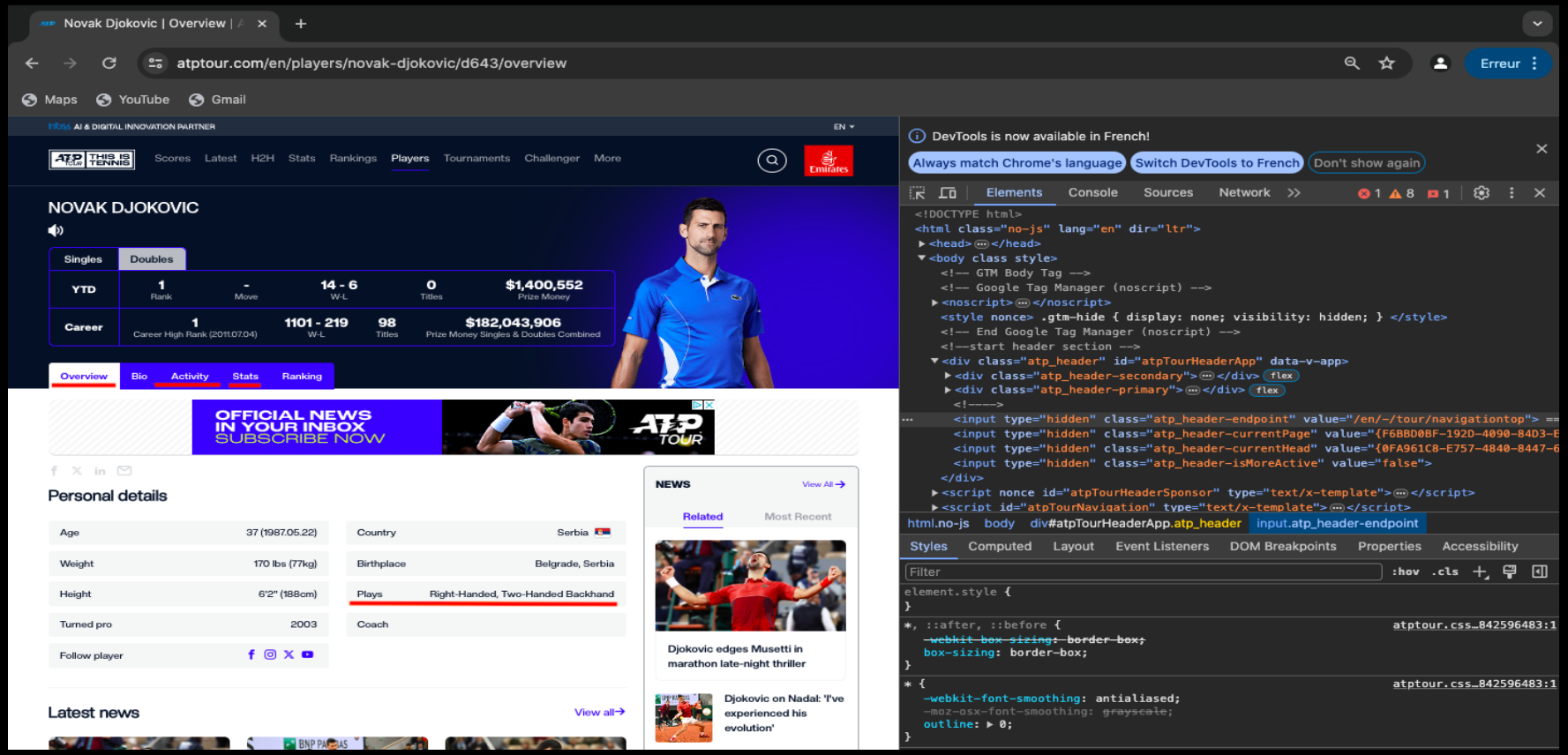
Problème : présence de fonction javascript (permettant aux gérants du site de faire le lien entre leurs données et celles présentes sur le site)

- **Solution:** supprimer la partie javascript

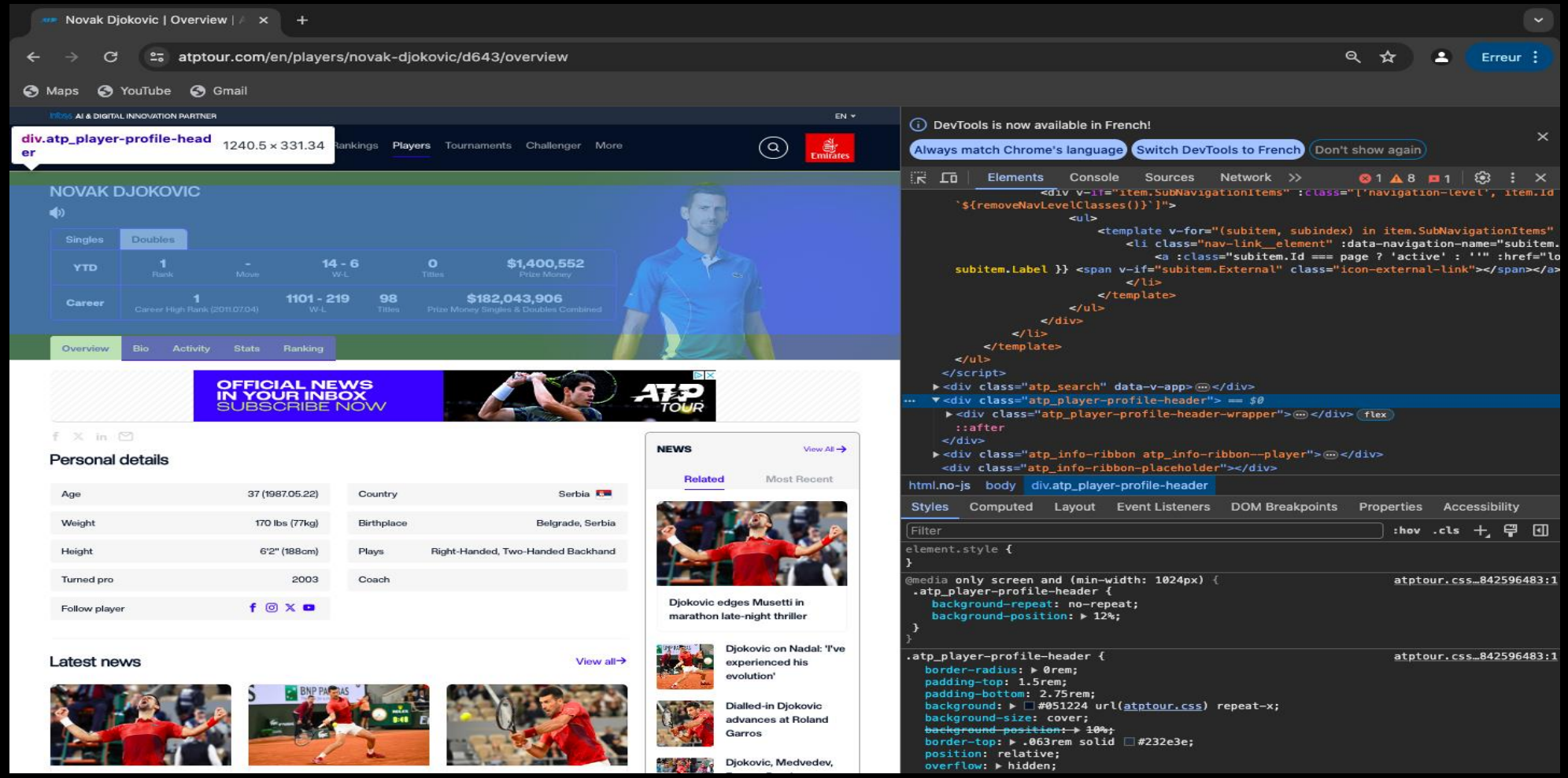
Selenium Python

Lancement de plusieurs instances de navigateurs pour les récupérer les codes sources pour récolter les données présentes sur le site: https://www.atptour.com/en dans :

Overview (Main) , Stats (Tout), et Activity (carrière):



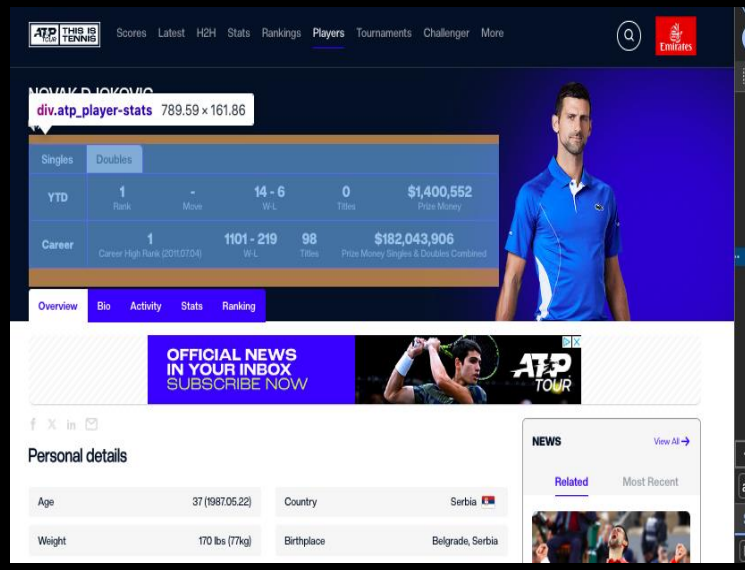
- Module Beautiful Soup: html.parser permet de garder en mémoire les contenus des conteneurs:

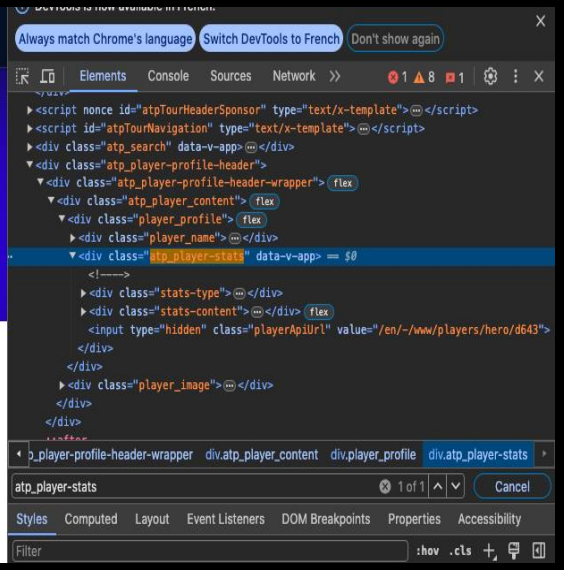


Récupération des données intéressantes

- Ouverture code source
- Mise en mémoire pour garder les codes stats et sources
- À l'aide de soup(1,2,3).find: trouve les données présentes dans le conteneurs div **stylé à** l'attribut "atp_player-stats"

```python
def scrape_atp(url: str) -> dict:
    driver = webdriver.Chrome()
    driver.get(f"https://www.atptour.com{url}overview")
    html_overview = driver.page_source
    driver.quit()

    driver = webdriver.Chrome()
    driver.get(f"https://www.atptour.com{url}player-stats?year=all&surface=all")
    html_stats = driver.page_source
    driver.quit()

    driver = webdriver.Chrome()
    driver.get(f"https://www.atptour.com{url}atp-win-loss?tourType=Tour")
    html_activity = driver.page_source
    driver.quit()

    soup1 = BeautifulSoup(html_overview, 'html.parser')
    soup2 = BeautifulSoup(html_stats, 'html.parser')
    soup3 = BeautifulSoup(html_activity, 'html.parser')

    atp_player_stats = soup1.find('div', class_='atp_player-stats')
    career_stats = soup2.find('div', class_='statistics_content')
    activity_stats = soup3.find('div', class_='atp_player-win_loss-index')
    infos = {}

    infos["Personnal Details"] = scrape_personal_details(soup1)
    infos["YTD&Career Player Stats"] = scrape_atp_player_stats(atp_player_stats) if atp_player_stats else 0
    infos["Career Stats"] = scrape_career_stats(career_stats) if career_stats else 0
    infos["Activity Stats"] = scrape_activity_stats(activity_stats) if activity_stats else 0

    return infos
```





- .text = récupérer le texte
- .text.strip() = enlève tous les espaces en trop

- L 30: split car pour le rang on avait par exemple : « 1 espace rank » et on voulait juste le 1

- on split également pour la même raison mais par contre le [1] était le tiret qui ne nous intéresse pas d'où le [0] et [2]

-Update: ne supprime pas : ajoute ou modifie même la valeur de la clé

```python
def scrape_atp_player_stats(datas: BeautifulSoup) -> dict:
    player_stats = {
        'YTD': {},
        'Career': {}
    }

    all_stats_details = datas.find_all('div', class_='player-stats-details')

    for stat_detail in all_stats_details:
        type_stat = stat_detail.find('div', class_='type').text.strip()

        rank = stat_detail.find('div', class_='stat').text
        w_l = stat_detail.find('div', class_='wins').text
        titles = stat_detail.find('div', class_='titles').text

        stats = {
            'Wins': int(w_l.split()[0]),
            'Loses': int(w_l.split()[2]),
            'Rank': int(rank.split()[0]),
            'Titles': int(titles.split()[0])
        }

        if type_stat == 'YTD':
            player_stats['YTD'].update(stats)
        elif type_stat == 'Career':
            player_stats['Career'].update(stats)

    return player_stats
```

Même procédé pour carrière et activity

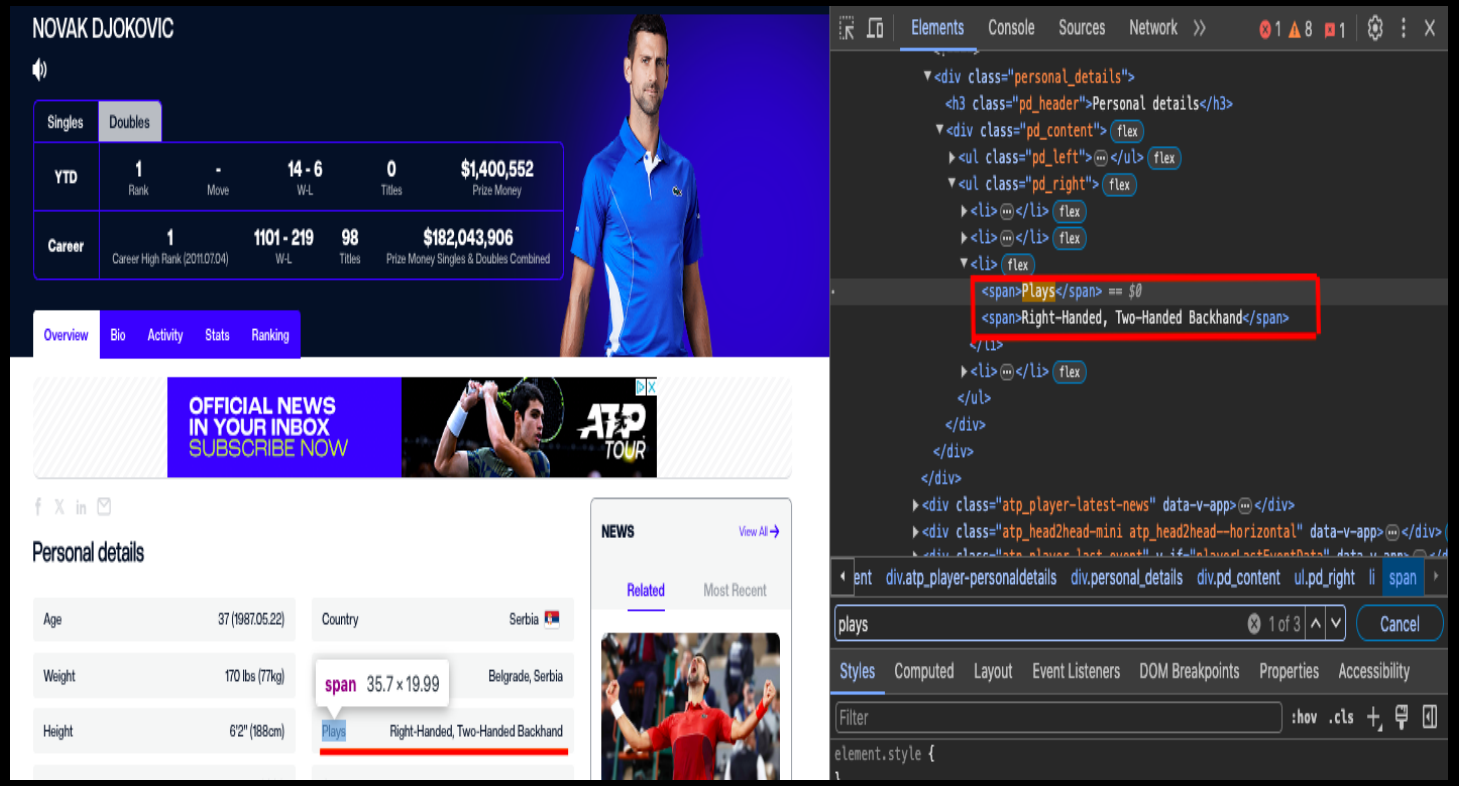
```python
def scrape_career_stats(datas: BeautifulSoup) -> dict:
    datas_extraites = {}

    stats = datas.find_all('li', class_='stats_items')

    for stat in stats:
        categorie = stat.find('span', class_='stats_record').text
        valeur = stat.find('span', class_='stats_percentage').text
        datas_extraites[categorie] = valeur

    return datas_extraites
```

```python
def scrape_activity_stats(datas: BeautifulSoup) -> dict:
    hands = {}
    surface = {}

    table = datas.find('table')
    rows = table.find_all('tr')

    for row in rows:
        cells = row.find_all('th') + row.find_all('td')
        if cells[0].get_text(strip=True) in ['Clay', 'Grass', 'Hard']:
            career_index = cells[4].get_text(strip=True)
            surface[cells[0].get_text(strip=True)] = float(career_index)
        elif cells[0].get_text(strip=True) in ['vs. Right Handers*', 'vs. Left Handers*']:
            career_index = cells[4].get_text(strip=True)
            hands[cells[0].get_text(strip=True)] = float(career_index)

    return {"vs Hands": hands, "Surface": surface}
```

```python
from bs4 import BeautifulSoup
from selenium import webdriver
import time


def scrape_personal_details(datas: BeautifulSoup) -> dict:
    hands = {"Plays": datas.find('span', text='Plays').find_next_sibling('span').text}
    return hands
```

- .next_sibling = on voulait juste la ligne suivante (=plus proche)
- et savoir : droitier ou gaucher

Même procédé sur le site : https://www.oddsportal.com/matches/tennis/ (annexe)



Après cela, le dictionnaire info est de cette forme suivante:


```
{'Activity Stats': {'Clay': '0.500',
                    'Grass': '0.000',
                    'Hard': '0.688',
                    'vs. Left Handers': '0.667',
                    'vs. Right Handers': '0.647'},
 'Career Stats': {'1st Serve': '58%',
                  '1st Serve Points Won': '72%',
                  '1st Serve Return Points Won': '24%',
                  '2nd Serve Points Won': '51%',
                  '2nd Serve Return Points Won': '50%',
                  'Aces': '166',
                  'Break Points Converted': '42%',
                  'Break Points Faced': '135',
                  'Break Points Opportunities': '93',
                  'Break Points Saved': '67%',
                  'Double Faults': '56',
                  'Return Games Played': '238',
                  'Return Games Won': '16%',
                  'Return Points Won': '34%',
                  'Service Games Played': '237',
                  'Service Games Won': '81%',
                  'Total Points Won': '49%',
                  'Total Service Points Won': '63%'},
 'Personnal Details': {'Age': '18 (2005.09.01)',
                       'Birthplace': 'Prostejov',
                       'Coach': 'Tomas Josefus',
                       'Country': 'Czechia ',
                       'Follow player': '',
                       'Height': '6'4" (193cm)',
                       'Plays': 'Right-Handed, Two-Handed Backhand',
                       'Turned pro': '2022',
                       'Weight': '184 lbs (83kg)'},
 'YTD&Career Player Stats': {'Career': {'Loses': 7,
                                        'Rank': 65,
                                        'Titles': 0,
                                        'Wins': 13},
                             'YTD': {'Loses': 6,
                                     'Rank': 76,
                                     'Titles': 0,
                                     'Wins': 10}}}
```

Filtration des données collectées

```python
import re


def filter_player(infos: dict) -> dict:
    hands_infos = infos["Personnal Details"]["Plays"].split(", ")
    hand = "Right-Handed" if "Right-Handed" in hands_infos else "Left-Handed"
    surface_dict = infos["Activity Stats"]["Surface"]

    infos_f = {"Rank (YTD)": int(infos["YTD&Career Player Stats"]["YTD"]["Rank"]),
               "Titles (Career)": int(infos["YTD&Career Player Stats"]["Career"]["Titles"]),
               "Wins (YTD)": int(infos["YTD&Career Player Stats"]["YTD"]["Wins"]),
               "Loses (YTD)": int(infos["YTD&Career Player Stats"]["YTD"]["Loses"]),
               "Wins (Career)": int(infos["YTD&Career Player Stats"]["Career"]["Wins"]),
               "Loses (Career)": int(infos["YTD&Career Player Stats"]["Career"]["Loses"]),
               "Preferred Surface": max(surface_dict, key=surface_dict.get),
               "Dominant Hand": hand,
               "1st Serve Points Won": int(infos["Career Stats"]["1st Serve Points Won"][:-1]),
               "1st Serve Return Points Won": int(infos["Career Stats"]["1st Serve Return Points Won"][:-1]),
               "2nd Serve Points Won": int(infos["Career Stats"]["2nd Serve Points Won"][:-1]),
               "2nd Serve Return Points Won": int(infos["Career Stats"]["2nd Serve Return Points Won"][:-1]),
               "Break Points Converted": int(infos["Career Stats"]["Break Points Converted"][:-1]),
               "Break Points Saved": int(infos["Career Stats"]["Break Points Saved"][:-1]),
               "Clay Index (Career)": float(infos["Activity Stats"]["Surface"]["Clay"]),
               "Grass Index (Career)": float(infos["Activity Stats"]["Surface"]["Grass"]),
               "Hard Index (Career)": float(infos["Activity Stats"]["Surface"]["Hard"]),
               "VS Right-Handers Index (Career)": float(infos["Activity Stats"]["vs Hands"]["vs. Right Handers*"]),
               "VS Left-Handers Index (Career)": float(infos["Activity Stats"]["vs Hands"]["vs. Left Handers*"]),
               }
    return infos_f


def player_name(url: str) -> str:
    extract_name = re.search(r"/players/([a-zA-Z-]+-[a-zA-Z-]+)/", url)
    names = extract_name.group(1).split('-')
    return f"{names[-1][0].upper()}{names[-1][1:].lower()} {names[0][0].upper()}."
```

.split("-"): devient 2 éléments d'une liste créée à partir de la virgule

ça retournerait toujours Hard car H est la plus grande de l'alphabet par rapport aux surfaces ("Grass", "Clay"): d''où surface_dict = dico avec les clé Hard grass et clay

- recherche du nom dans l'URL

met la première lettre du nom de famille en majuscule, etc.. : slice (pour mettre en forme)

```
"/en/players/tomas-martin-etcheverry/ea24/",
```

Ex: on a **Etcheverry M.**

Avant la filtration, le Json..

Force = True, peu importe ce qu'il y a dans le dico ça va le réécrire dans le fichier

Avantage du Json: données sous forme de dictionnaire

Problème: tous les joueurs n'avaient pas toutes les stats

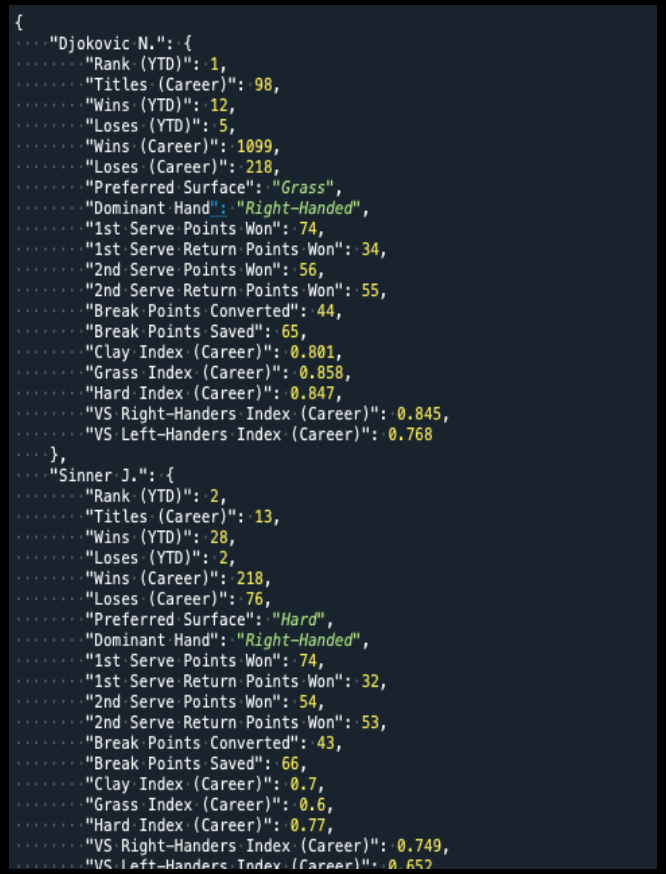
```python
import json
import logging


def save_json(infos: dict, force=False, file='save.json') -> None:
    if force:
        with open(file, 'w') as f: # ouvre le fichier + écriture write
            json.dump(infos, f, indent=4) # on importe la librairie , et dump ça réécrit dans le fichier f , indentation pour gérer l'espace
        return None

    try:
        with open(file, 'r') as f: # lecture
            data = json.load(f) # pour le charger
    except (FileNotFoundError, json.JSONDecodeError): #s'il y a une erreur
        data = {} # d'ou le dico vide

    data.update(infos)

    with open(file, 'w') as f:
        json.dump(data, f, indent=4) # on réécrit comme en haut


def read_json(file='save.json') -> dict:
    with open(file, 'r') as f:
        data = json.load(f)
        return data


def is_player_in_json(player_name: str, file='save.json') -> bool:
    try:
        with open(file, 'r') as f:
            return player_name in json.load(f) # vérifie dans si ton joueur est dans ton fichier , toujours sous forme de dico !
    except:
        return False # cas où il y a un problème de lecture


logging.basicConfig(filename='log_player.txt', level=logging.INFO, format='%(asctime)s - %(message)s') #pour avoir un journal de tout ce qui s'est passé


def log_player_status(name: str, status: str = 'waiting'):
    logging.info(f'{name} - Status: {status}') # vérification dans le fichier log
```

Analyse des match

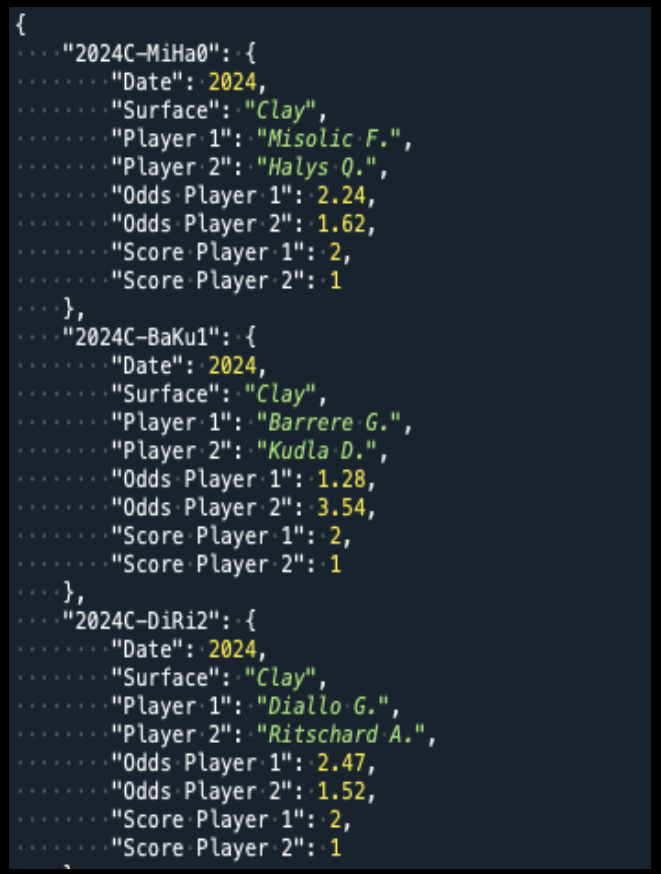
Récupération du score du dernier match

```python
from tipe_save import read_json


def get_match(player: str, surface=("Grass", "Clay", "Hard"), only_last=False) -> dict:
    matches = read_json('save_matches.json')
    played_matches = {}

    for id, match in matches.items(): #lit le id associé à la valeur du match en question
        if player in (match["Player 1"], match["Player 2"]) and match["Surface"] in surface:
            played_matches[id] = match

    if played_matches:
        if only_last:
            last_id = sorted(played_matches.keys())[-1] # sortir dans l'ordre croissant, dernier année (prendra un match forcément en terre battu car 'Clay')
            return {last_id: played_matches[last_id]}
        return played_matches

    return None
```

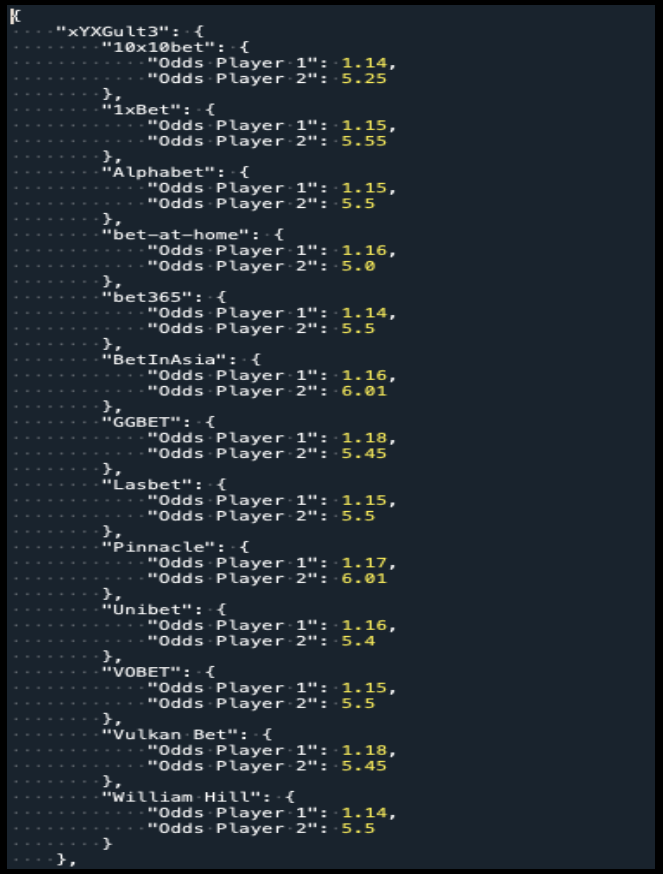
Dictionnaires finaux



```
{
    "Djokovic N.": {
        "Rank (YTD)": 1,
        "Titles (Career)": 98,
        "Wins (YTD)": 12,
        "Loses (YTD)": 5,
        "Wins (Career)": 1099,
        "Loses (Career)": 218,
        "Preferred Surface": "Grass",
        "Dominant Hand": "Right-Handed",
        "1st Serve Points Won": 74,
        "1st Serve Return Points Won": 34,
        "2nd Serve Points Won": 56,
        "2nd Serve Return Points Won": 55,
        "Break Points Converted": 44,
        "Break Points Saved": 65,
        "Clay Index (Career)": 0.801,
        "Grass Index (Career)": 0.858,
        "Hard Index (Career)": 0.847,
        "VS Right-Handers Index (Career)": 0.845,
        "VS Left-Handers Index (Career)": 0.768
    },
    "Sinner J.": {
        "Rank (YTD)": 2,
        "Titles (Career)": 13,
        "Wins (YTD)": 28,
        "Loses (YTD)": 2,
        "Wins (Career)": 218,
        "Loses (Career)": 76,
        "Preferred Surface": "Hard",
        "Dominant Hand": "Right-Handed",
        "1st Serve Points Won": 74,
        "1st Serve Return Points Won": 32,
        "2nd Serve Points Won": 54,
        "2nd Serve Return Points Won": 53,
        "Break Points Converted": 43,
        "Break Points Saved": 66,
        "Clay Index (Career)": 0.7,
        "Grass Index (Career)": 0.6,
        "Hard Index (Career)": 0.77,
        "VS Right-Handers Index (Career)": 0.749,
        "VS Left-Handers Index (Career)": 0.652
```



```
{
    "2024C-MiHa0": {
        "Date": 2024,
        "Surface": "Clay",
        "Player 1": "Misolic F.",
        "Player 2": "Halys Q.",
        "Odds Player 1": 2.24,
        "Odds Player 2": 1.62,
        "Score Player 1": 2,
        "Score Player 2": 1
    },
    "2024C-BaKu1": {
        "Date": 2024,
        "Surface": "Clay",
        "Player 1": "Barrere G.",
        "Player 2": "Kudla D.",
        "Odds Player 1": 1.28,
        "Odds Player 2": 3.54,
        "Score Player 1": 2,
        "Score Player 2": 1
    },
    "2024C-DiRi2": {
        "Date": 2024,
        "Surface": "Clay",
        "Player 1": "Diallo G.",
        "Player 2": "Ritschard A.",
        "Odds Player 1": 2.47,
        "Odds Player 2": 1.52,
        "Score Player 1": 2,
        "Score Player 2": 1
```



```
{
    "xYXGult3": {
        "10x10bet": {
            "Odds Player 1": 1.14,
            "Odds Player 2": 5.25
        },
        "1xBet": {
            "Odds Player 1": 1.15,
            "Odds Player 2": 5.55
        },
        "Alphabet": {
            "Odds Player 1": 1.15,
            "Odds Player 2": 5.5
        },
        "bet-at-home": {
            "Odds Player 1": 1.16,
            "Odds Player 2": 5.0
        },
        "bet365": {
            "Odds Player 1": 1.14,
            "Odds Player 2": 5.5
        },
        "BetInAsia": {
            "Odds Player 1": 1.16,
            "Odds Player 2": 6.01
        },
        "GGBET": {
            "Odds Player 1": 1.18,
            "Odds Player 2": 5.45
        },
        "Lasbet": {
            "Odds Player 1": 1.15,
            "Odds Player 2": 5.5
        },
        "Pinnacle": {
            "Odds Player 1": 1.17,
            "Odds Player 2": 6.01
        },
        "Unibet": {
            "Odds Player 1": 1.16,
            "Odds Player 2": 5.4
        },
        "VOBET": {
            "Odds Player 1": 1.15,
            "Odds Player 2": 5.5
        },
        "Vulkan Bet": {
            "Odds Player 1": 1.18,
            "Odds Player 2": 5.45
        },
        "William Hill": {
            "Odds Player 1": 1.14,
            "Odds Player 2": 5.5
        }
    },
```

info joueur

matches

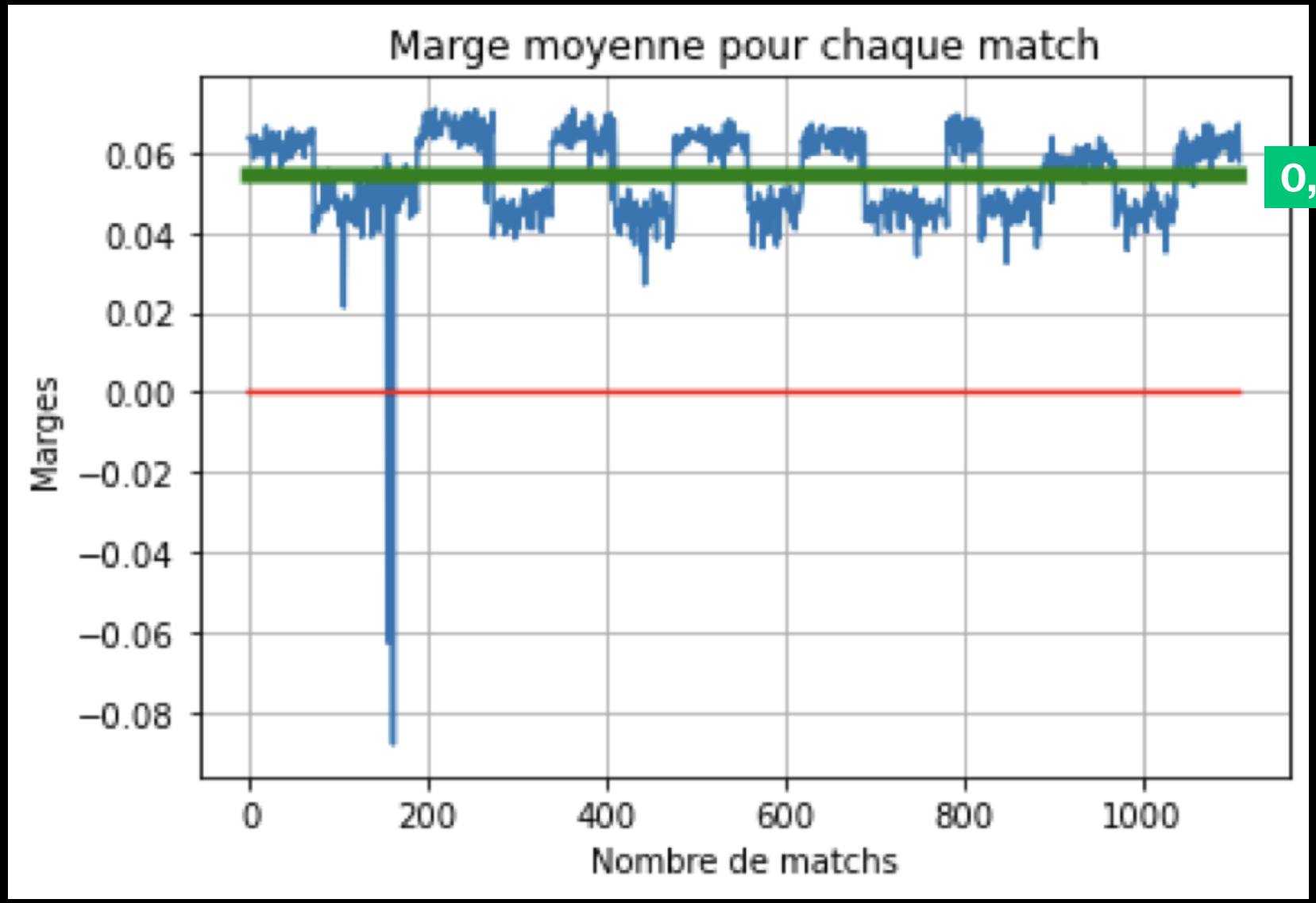
paris

Total: + 300 joueurs , 1108 matches

Réalité dans un pari

$$on\ a\ \frac{1}{C_1} + \frac{1}{C_2} = 1 + \gamma\ avec\ \gamma > 0$$

Gamma est la marge que se fait le bookmaker quelque soit l'issu du match



Marge établis par les bookmakers pour 1108 match de tennis

Cas d'un arbitrage

dans le cas d'un arbitrage :

$$M_{n,i} = \frac{S_n}{1+\alpha} \times C_i \text{ avec } M_{n,i} \text{ la mise lors du } n-\text{ième pari dur le joueur } i \text{ , pour } i \in \{1,2\}$$

avec $\alpha < 0$

De cette façon si on gagne alors :

$$S_{n+1} = M_{n,i} \times C_i = \frac{S_n}{1+\alpha} > S_n$$

Rareté d'un arbitrage

Analyse asymptotique

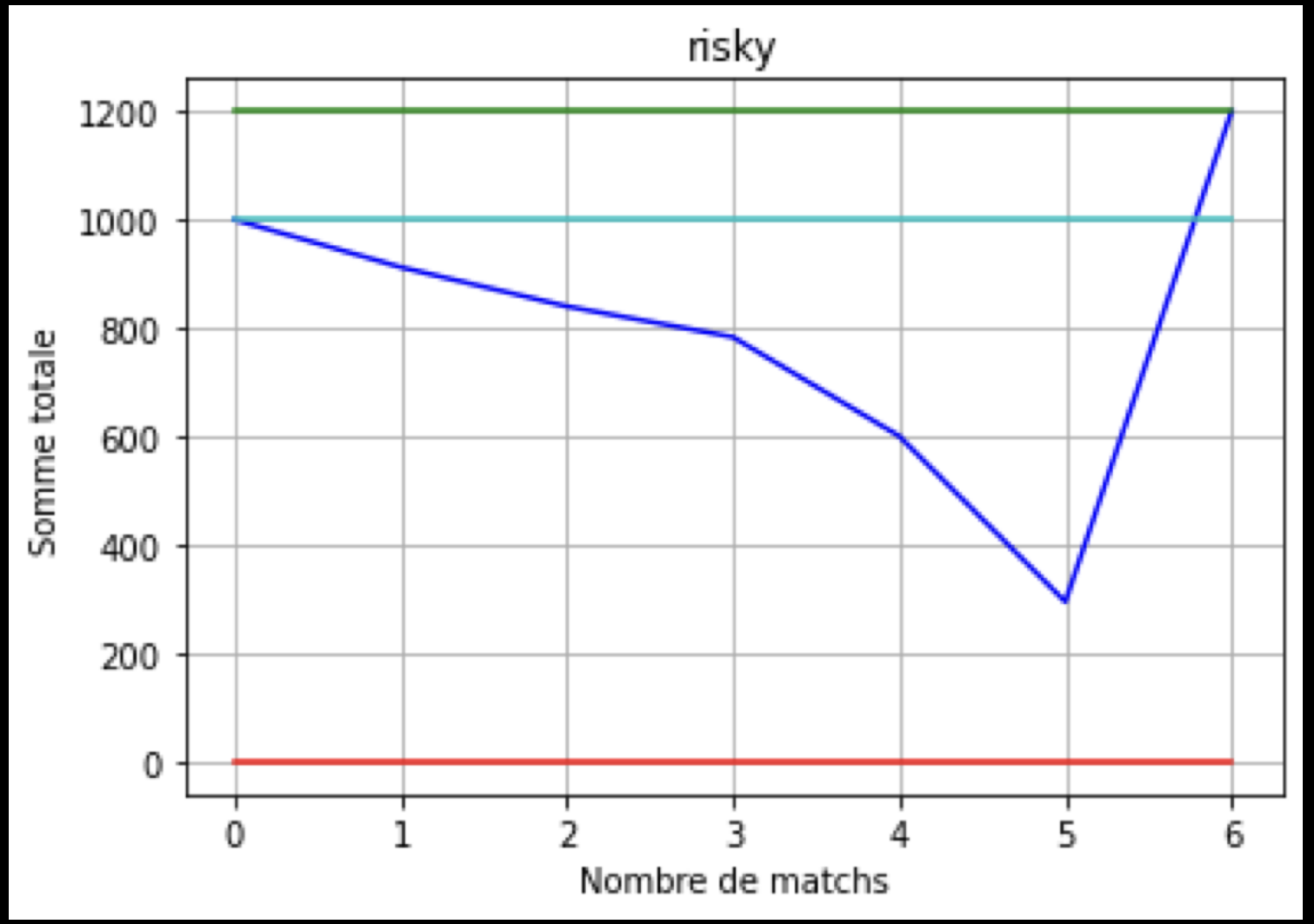
Dans le cas d'une stratégie "risky': le joueur arrête de parier dès qu'il atteint une somme voulue:

On peut donc utiliser la notion de temps d'arrêt

Soit $(\Omega,\ A,\ P)$ *un espace probabilisé muni d'une filtration* $\{Fn\}_{n\geq 0}$.

Une variable alétoire $T : \Omega \to \mathbb{N} \cup \{+\infty\}$ *est appelé un temps d'arret si :*

$$\{T \leq n\} \in F_n$$



Cela nécessite d'utiliser un temps d'arret : $T = \inf\{n \geq 0,\ S_n = 1100\}$

Longévité du temps d'arrêt