Blog Post 2: Automating Football Data Collection with Selenium

Aim

This week, we moved from theory to practice. Our goal was to automate the process of collecting historical and contextual football match data from a trusted source. To that end, we developed a powerful web scraping and browser automation tool to pull data from SofaScore using Selenium. In this blog post, we share the steps we followed to create an efficient and reusable web scraping pipeline.

Major Accomplishments

This week's work focused on laying the foundations for efficient and comprehensive data collection. Here are the key milestones achieved:

1 Selenium Installation

- Chrome WebDriver Launch We set up an automated browser environment using Selenium and the WebDriver Manager. This included options such as maximizing browser windows, disabling notifications, and headless mode for faster and more efficient scraping.

```python
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager

def start_driver():
options = webdriver.ChromeOptions()
options.add_argument("--start-maximized")
options.add_argument("--disable-notifications")
options.add_argument("--headless")
```

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)

return driver

```

- We have configured error handling mechanisms to handle scenarios where web elements are not available or pages are not loading.

## 2. Accessing SofaScore

- We have automatically logged in to the SofaScore homepage using WebDriver.

- We have implemented error handling mechanisms for page load delays and dynamic content.

```python

def sofascore_login():

driver = start_driver()

url = "https://www.sofascore.com/tr/"

driver.get(url)

print(f"{url} address logged in.")

return driver

```

3. League Search and Navigation

- Automatic Search: We developed a function to find specific leagues like "Trendyol Super League" using the search bar and select the league we want.

```python

def search_league(driver, league_name="Trendyol Super League"):

search_box_xpath = '//*[@id="search-input"]'

search_box = WebDriverWait(driver, 10).until(

EC.presence_of_element_located((By.XPATH, search_box_xpath))

)

```
search_box.send_keys(league_name)

first_result_xpath =
'//*[@id="__next"]/header/div[1]/div/div/div[2]/div/div/div[2]/div[1]/div/div/a/div'

first_result = WebDriverWait(driver, 10).until(

EC.element_to_be_clickable((By.XPATH, first_result_xpath))

)

first_result.click()
```

- We ensured accurate and fast navigation by automatically selecting the first result.

4. Data Scraping Workflow

- We extracted critical information like teams, scores, date, and match results by navigating through dynamically loaded league data on the website.

- We developed helper functions to manage pop-ups and smoothly navigate between multiple pages.

5. Interaction with Dynamic Web Elements

- We designed functions to navigate through seasonal data and ensured that weekly match details were captured for each available league.

- Reusable Data Extraction: We created modular functions to extract tabular data, ensuring scalability for future needs.

```python
def navigate_to_matches_section(driver):

matches_section_xpath =
'//*[@id="__next"]/main/div/div[3]/div/div[1]/div[1]/div[3]/div[1]/div'

matches_section = WebDriverWait(driver, 10).until(

EC.presence_of_element_located((By.XPATH, matches_section_xpath))
```

)

driver.execute_script("arguments[0].scrollIntoView(true);", matches_section)

```

## Challenges Encountered

- Dynamic Loaded Elements: Navigating dynamically generated content on SofaScore required careful use of explicit waits and XPath selectors.

- Pop-Up Management: Pop-up elements interfered with the workflow and required the development of a robust closing logic.

- Dynamic Page Structure: Frequent changes in page structure required adaptive coding techniques and testing to maintain the pipeline's reliability.

## Takes Away

This week allowed us to explore the complexities of web scraping and emphasize clean coding practices and modularity. The automated scraping pipeline provides a solid foundation for collecting historical match data and then feeds data into predictive models to be developed later in the project. The practical challenges encountered deepened our understanding of browser automation and effectively managing real-world data collection scenarios.
```