Web Scraping and Social Media Scraping Project
Group Members: Mustafa Ceyhun Vural (455158) & Onur Durmuş (455860)

NBA Player Data Extraction and Analysis from HoopsHype

Introduction:

This project aims to exploit the power of web scraping methodologies to acquire, analyze and make sense of relevant data about NBA players. The chosen platform for this endeavor is HoopsHype (https://hoopshype.com), a robust and comprehensive website dedicated to providing a wide spectrum of information on current and former NBA players.

HoopsHype has distinguished itself as a resource for NBA enthusiasts, providing detailed data including players' salaries, ages, performance measures, and even industry rumors. Beyond merely displaying statistics, the platform allows users to delve into in-depth comparisons between players, creating an interactive and engaging environment for its audience. Individual pages dedicated to each player serve as a rich source of data, making the site an ideal candidate for our web scraping project.

The primary information extracted from HoopsHype includes the players' names, links to their individual pages, salaries, and birthdates. To accomplish this, our team implemented a three-pronged approach utilizing BeautifulSoup, Scrapy, and Selenium - popular tools in the data scraping field. Each tool was used to extract the same set of data, ensuring an intricate understanding and mastery of these diverse methodologies.

Our choice of HoopsHype for this project was driven by our collective interest in the NBA, combined with the website's complex, data-rich structure, making it a challenging yet rewarding case study for data extraction and analysis.

Scraper Mechanism:

1. BeautifulSoup

This script uses BeautifulSoup to scrape NBA player data from the HoopsHype website. The script begins by setting up a Pandas DataFrame to hold the player data. It then runs a loop for each year in the specified range (in our case, 2020-2021, since we do not want to wait very long for computation time), crafting a unique URL for each year and sending a GET request to access that page.

Upon receiving a successful response, the script parses the page's HTML using BeautifulSoup and searches for the table containing player salary data. If the table exists, the script then iterates over each row of the table (excluding the header row). For each row, it extracts the player's name and the link to their individual page (prepended with 'https://hoopshype.com' if needed) and stores this data in a new row of the DataFrame.

Once the initial scrape is complete, the script then loops over each row in the DataFrame, and if a link to the player's individual page is available, sends another GET request to that page. The HTML of the player's page is parsed, and additional details - the player's salary and birth date - are extracted and stored in the DataFrame. To ensure the script doesn't overload the website's server with rapid requests, it pauses for one second after each request. If the counter has reached 100, indicating 100 records have been scraped, it breaks the loop and stops the process.(For submission, original data has more records for analysis part) Finally, after all the data is gathered and stored in the DataFrame, the script prints out the DataFrame and saves it as a CSV file named 'player_salaries_bs.csv'. This process ensures a comprehensive collection of data on NBA player salaries for the specified years, along with relevant details like links to player profiles and birth dates.

2. Scrapy

The script defines a Scrapy Spider class named PlayersSpider. The start_urls list includes the web page(s) that the spider begins scraping from. In this case, the start URL is the salaries page for NBA players for the year 2020-2021.

The parse method is called once for each URL in the start_urls list. It is used to extract data and find new URLs to follow. This method iterates over each row in the salaries table (excluding the header row) and collects the player's name and link to their individual page. If the player's link does not start with "http", the script prepends the base URL to it.

The parse_player method is a callback function used to parse the details of individual players' pages. It extracts the player's birth date and salary using CSS selectors. If the spider doesn't find this data on the page, it uses 'N/A' as a default value.

The parse method then makes a new request to each player's link and passes this parse_player method as a callback to be called with the response data of this request. It also passes some metadata (the player's name, the year, and the player's link) to this callback function using the meta argument.

The custom_settings dictionary is used to set a download delay of 2 seconds between consecutive requests to the website's server, to avoid overloading it.

The script also includes a counter to limit the number of players' pages to scrape. The spider stops extracting data after it has processed the details of 100 players.

The final output of the parse_player method is a Python dictionary that includes the player's name, salary, the year, the link to their individual page, and their date of birth. Scrapy handles this output and stores it in a convenient format for further analysis.

3. Selenium

This script uses Selenium, a powerful tool for automating web browsers, to scrape NBA player data from the HoopsHype website.

The script first sets up a Pandas DataFrame to store the data. It then initializes the Selenium WebDriver with the specified ChromeDriver path. A counter variable is set to 0 to keep track of the number of records scraped.

The script then iterates over the years in the specified range (in our case, 2020-2021, since we do not want to wait very long for computation time), constructing a unique URL for each year's player salaries page and using Selenium to navigate to that page.

Once on the page, the script uses the find_elements method and XPath expressions to locate elements containing player names, salaries, and the links to their individual pages. It iterates over each list of elements, extracting the relevant text (player name, salary, or URL) and storing it in its respective list.

Next, the script iterates over the list of links, navigating to each player's individual page and attempting to extract the player's birth date using another XPath expression. If it fails to find a birth date, it stores 'Not Found' in its place. The counter is incremented after each record.

The script then creates tuples of the data scraped for each player and appends them to the DataFrame. If the counter has reached 100, indicating 100 records have been scraped, it breaks the loop and stops the process.

After all data is gathered, Selenium closes the browser, and the script saves the DataFrame as a CSV file named 'player_salaries.csv'. This process achieves a detailed extraction of player data including names, salaries, profile links, and birth dates for the specified years.

Technical description of the output:

The output from our script provides valuable, real-time feedback on the web scraping process and ultimately generates a comprehensive, well-structured CSV file containing the collected data.

When our script is executed, it prints the HTTP response status code to the console. A 200 status code signifies a successful request, indicating that the script was able to connect to the webpage.

As the script continues to run, it prints the name of each player being processed. This serves as a progress update, allowing us to monitor the scraper's progress in real-time. It also helps in debugging, as we can determine at which point (i.e., which player's data) any potential error might occur.

Upon completion of the scraping process, our script generates a CSV file named 'player_salaries_bs.csv'. This file contains the data extracted from the website, neatly organized into five columns: Player, Salary, Year, Link, and Born.

Each row corresponds to a single player and contains their name (Player), salary for the specified year (Salary), the year of the salary (Year), the link to their individual page on the HoopsHype website (Link), and their date of birth (Born). For example, the first row contains data for Stephen Curry, revealing his salary for 2020, a link to his profile page, and his date of birth.

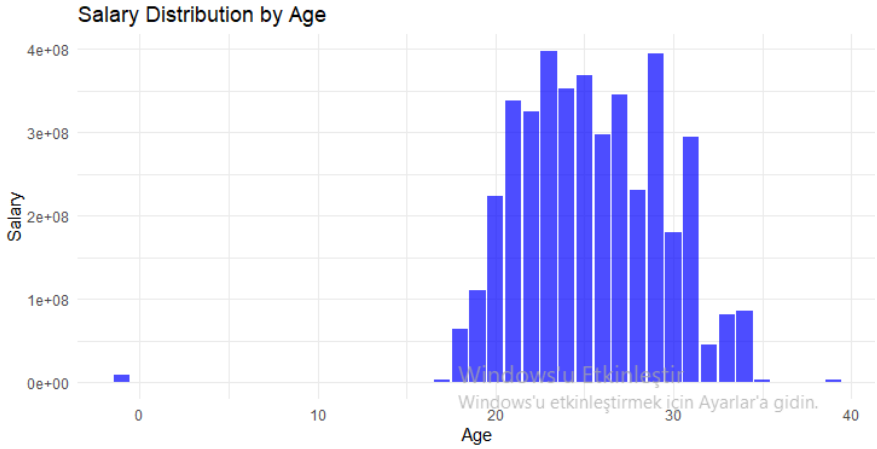
Although our original code is capable of extracting data for 569 players(or more if we increase the number of years involve in the loop), the submitted version of the script includes a limitation to scrape data only for the first 100 players.

Web Scraping and Social Media Scraping Project
Group Members: Mustafa Ceyhun Vural (455158) & Onur Durmuş (455860)

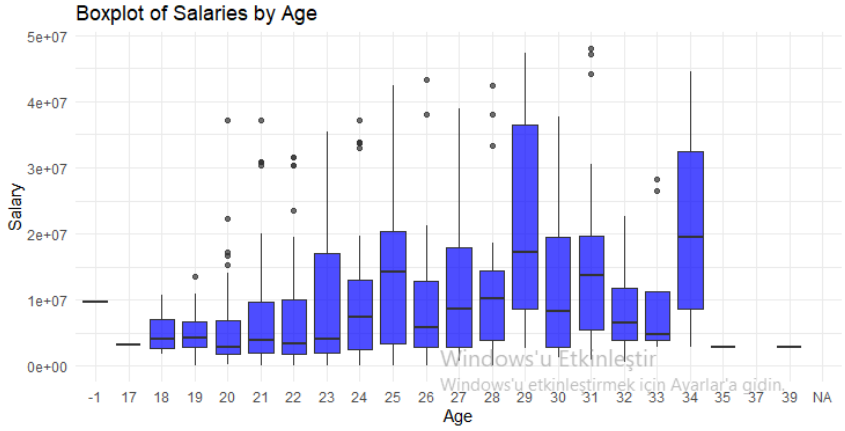
Elementary data analysis:

Upon completion of the scraping process, our script generates a CSV file named 'player_salaries_bs.csv' with five columns: Player, Salary, Year, Link, and Born. With this file we made some data analysis using R language. In the and we check the relationship between salary and age and get: Histogram of Salary Distribution by Age and Boxplot of Salaries by Age. You can find the codes and results below:

```
#Histogram of Salary Distribution by Age:
ggplot(player_salaries, aes(Age, Salary)) +
  geom_histogram(stat = "identity", fill = "blue", alpha = 0.7) +
  labs(x = "Age", y = "Salary", title = "Salary Distribution by Age") +
  theme_minimal()
```




```
#Boxplot of Salaries by Age:
ggplot(player_salaries, aes(x = factor(Age), y = Salary)) +
  geom_boxplot(fill = "blue", alpha = 0.7) +
  labs(x = "Age", y = "Salary", title = "Boxplot of Salaries by Age") +
  theme_minimal()
```



Web Scraping and Social Media Scraping Project
Group Members: Mustafa Ceyhun Vural (455158) & Onur Durmuş (455860)

Distribution of Work:

Task	Contributors
Beautiful Soup	Mustafa Ceyhun Vural
Scrapy	Onur Durmuş
Selenium	Together
Github	Together
Project Report	Together