UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Web Scraping di dati statistici relativi alla NBA"

Relatore: Prof. Giorgio Maria Di Nunzio

Laureando: Sanguin Giacomo

ANNO ACCADEMICO 2021 – 2022

Data di laurea 14/11/2022

Abstract

Internet è ad oggi la più vasta collezione di informazioni e dati esistente, e il suo utilizzo ha subito negli ultimi anni una crescita esponenziale, portando questa tecnologia e l'immensa mole di dati che essa offre, alla portata di tutti.
Le informazioni disponibili nel web sono eterogenee: ogni sviluppatore decide come rappresentarle nel proprio sito e non viene utilizzato nessuno standard di rappresentazione dei dati. Per questo, la collezione di dati in grosse quantità è complicata da automatizzare. Il pratico copia-incolla non è un opzione quando i dati da prelevare sono dell'ordine delle centinaia, se non migliaia o addirittura milioni. Se ad esempio si dovessero salvare manualmente su un foglio di calcolo delle informazioni, il processo consisterebbe nel copiare e incollare ogni singolo dato dalla sua pagina web al documento da creare, richiedendo tempo che sarebbe preferibile risparmiare utilizzando un programma che svolga tutte le operazioni.
In tal senso, ci viene in aiuto uno strumento estremamente utile e funzionale, il web scraping, applicato da una serie di programmi, chiamati scraper, che in base al formato e alla struttura della pagina, possono adattarsi ed effettuare l'estrazione dei dati in maniera automatizzata.
In questo progetto di tesi, si andrà ad analizzare la tecnica del web scraping, a partire da una presentazione generale del suo funzionamento e degli strumenti maggiormente utilizzati per metterla in pratica, per arrivare poi ad applicarla concretamente in un particolare caso preso in esame. In particolare, verrà studiata ed analizzata, l'applicazione di tale strumento nel campo dell'analisi dei dati statistici relativi al mondo dello sport, nel nostro specifico caso quello del basket professionistico.

Indice

1 Introduzione

Nel campo delle Sports Analytics, la ricerca ed estrazione, con conseguente raccolta, dei dati relativi alle performance degli atleti, ha assunto un ruolo predominante.
In un contesto di questo tipo dunque, uno strumento come il web scraping, risulta di fondamentale importanza, per permettere un'estrazione automatizzata dei dati e delle informazioni che si vuole analizzare.

Il seguente progetto di tesi, si presenta come lo sviluppo di uno scraper, un programma che applica la tecnica del web scraping, per la ricerca e raccolta di dati statistici relativi ai giocatori della NBA, la più celebre ed importante lega di basket professionistico. Lo scopo principale, è quello di riuscire a raccogliere, all'interno di una struttura dati, le statistiche principali che riguardano la performance sul campo di ciascun giocatore, per un determinato anno in questione. Si tratta delle principali voci statistiche del mondo del basket, ovvero partite giocate, punti segnati, rimbalzi catturati e via dicendo.
Una volta collezionati tali dati, sarà possibile dunque procedere con una fase di analisi, nel quale l'obiettivo principale sarà quello di determinare la correlazione tra il salario che l'atleta ha guadagnato durante l'anno, e le sue reali prestazioni in campo.

La tesi è strutturata nel seguente modo:

- Nel secondo capitolo viene fornita un'introduzione sul web scraping.
- Nel terzo capitolo si fa una panoramica dei principali strumenti e librerie utilizzati nel processo di web scraping.
- Nel quarto capitolo viene fornita un'introduzione sul contesto preso in esame, la NBA, e sul sito web sul quale verrà applicato il web scraping.
- Nel quinto capitolo viene seguito nel dettaglio il processo di completamento del nostro scraper.
- Nel sesto capitolo si illustrano le analisi dei dati effettuate.
- Nel settimo capitolo, quello conclusivo viene fatto un resoconto del lavoro svolto e le ultime considerazioni.

2 Web Scraping

Nel capitolo seguente, andremo ad illustrare il concetto protagonista della nostra tesi, quello del web scraping. Forniremo dunque una definizione esaustiva, che permetta di comprendere l'utilità di tale strumento, introducendone il funzionamento. Andremo poi ad analizzare le possibili problematiche dal punto di vista legale, delle quali è importanti essere a conoscenza per garantire un utilizzo del web scraping in piena legalità.

2.1 Definizione

Il web scraping, detto anche "web harvesting" o "web data extraction", è una forma di data mining, e come intuibile dalla traduzione letterale del termine (grattare, raschiare), è una tecnica di estrazione di dati non strutturati da diversi siti web, per mezzo di programmi software. Si tratta di un sistema in grado di estrapolare una grande varietà di informazioni: dati di contatto, indirizzi di posta elettronica, numeri di telefono, così come singoli termini di ricerca o URL.
Anche se l'estrazione di dati da un sito web può essere eseguita manualmente dall'utente (in tal caso si parla di scraping manuale, che consiste in un processo di copia e incolla di singole informazioni mirate), si parla di web scraping sopratutto quando il processo di estrazione dei dati è automatizzato grazie all'uso di un software che emula la visita di un utente ad un sito web. Tali dati, possono così successivamente essere salvati all'interno di un database o di una tabella locale, per poi essere analizzati nel complesso.

2.2 Le fasi del web scraping

I programmi che si occupano di eseguire lo scraping, sono chiamati "scraper", e il loro intero funzionamento può essere suddiviso in due fasi sequenziali:

- Acquisizione delle risorse dal web:

  Nella fase iniziale, un software di web scraping si occuperà di effettuare una richiesta HTTP, in modo da ricevere in risposta le risorse che si vuole raccogliere da una pagina web di interesse.

  Il tipo di richiesta può differire a seconda del caso specifico. Può trattarsi di una richiesta GET contenente l'URL del sito web al quale si sta cercando di accedere, metodo tipicamente più comune ed utilizzato, dal momento che chiede al server una

specifica risorsa o informazione. O può trattarsi di una richiesta di tipo POST, quando ad esempio si vuol fare in modo che il server riceva determinate informazioni.
Una volta ricevuta la richiesta con successo da parte del sito web, questi invierà al software una risposta, la cui prima riga conterrà al suo interno un codice numerico che indica il risultato della HTTP request (200 indica che la richiesta è stata correttamente accettata). Se correttamente ricevuta, il software potrà dunque gestire la risposta HTTP nella maniera che ritiene più opportuna. Le risorse contenute all'interno della risposta HTTP, possono essere in formati differenti, dal momento che le pagine web sono spesso costituite da vari linguaggi di programmazione.

- Estrazione dell'informazione:

  Nella fase successiva all'acquisizione delle risorse dal web, il software si occupa di gestire i dati grezzi ricevuti, facendo tutte le operazioni necessarie per renderli fruibili nella maniera più efficiente possibile, a seconda del tipo di utilizzo che si vuole farne. Il software può quindi eseguire un parsing dei dati, così da estrapolare le informazioni di interesse, può formattarli, organizzandoli e attribuendo loro un determinato formato, e copiarli in un foglio di calcolo, o alternativamente caricarli in un nuovo database. Per fare un esempio, si potrebbe voler estrapolare esclusivamente dei valori presenti in determinate celle di una tabella, quindi il codice dovrà essere in grado di "catturare" solo quelle informazioni, escludendo tutto il resto dei dati ricevuti dalla risposta HTTP.

Lo scopo del web scraping quindi, è sostanzialmente quello di raccogliere dei dati da una determinata pagina web, per poi poterli utilizzare a proprio piacimento in un'altro tipo di applicazione.
Il web scraping, viene utilizzato in svariati scenari, quali ad esempio il monitoraggio dell'andamento dei prezzi presenti su un sito web di e-commerce, l'analisi delle variazioni delle quote di mercato in borsa, o lo studio dell'opinione degli utenti riguardo ad un determinato argomento all'interno dei social network.

2.3 Questioni legali

Le operazioni di web scraping, sono ormai utilizzate con frequenza, data l'estrema utilità nei più svariati campi di interesse. Ciò nonostante, non vi è ancora una regolamentazione precisa o un codice etico ben definito che permetta di regolarne la sua applicazione. Una tale

ambiguità dunque, può risultare piuttosto scomoda per il programmatore, dal momento che può essere estremamente complicato sviluppare un software di web scraping volto alla ricerca di informazioni utili al suo obiettivo, mantenendo la certezza di non incorrere in possibili controversie etiche e legali, che potrebbero compromettere il lavoro, sporcarne la reputazione, e sfociare in dispendiose cause legali.

In generale, è possibile affermare con certezza che il web scraping non sia illegale [2], a patto che i dati catturati siano liberamente accessibili sui siti web e siano usati per scopi statistici o di monitoraggio dei contenuti. In particolare, sotto questo primo aspetto, è fondamentale assicurarsi che all'interno del contratto dei "termini di utilizzo", il proprietario del sito web in questione non abbia specificato il divieto di accesso automatizzato al suo sito. Nel caso i termini vengano accettati dal visitatore difatti, si incappa in una violazione contrattuale.
Si rischia inoltre di oltrepassare la soglia della legalità, quando i dati estrapolati dallo scraper vengono impiegati per altri usi, quali la pubblicazione di contenuti in violazione del diritto d'autore, l'utilizzo per scopi di lucro e in violazione delle regole sulla concorrenza, o nel caso di raccolta di dati personali per scopi commerciali (quali l'e-mail marketing). Tali violazioni sono passibili di denuncia.
In un caso simile ad esempio, avvenuto nel 2016, il garante della privacy è intervenuto inibendo ad una società l'utilizzo dei dati personali, quali nomi, cognomi, indirizzi e-mail e recapiti telefonici di dodici milioni di utenti, che erano stati individuati e raccolti mediante scraping da diverse pagine web. Tale società, aveva poi pubblicato tali informazioni in un proprio portale online, consultabile liberamente anche da altre società per finalità di marketing.
In conclusione dunque, è importante che il programmatore sia aggiornato ed informato riguardo ai termini e alle condizioni dei siti web ai quali vuole accedere sistematicamente, così da evitare di incappare in problematiche come quelle descritte sopra.

3 Strumenti e librerie per il web scraping

Non è facile reperire ed esportare i dati in modo semplice ed immediato dal web, poiché la sua forma non strutturata complica il lavoro di estrazione e analisi dei dati e delle informazioni che si vogliono ricercare. Alcuni siti (Facebook, Google, Twitter, ad esempio) mettono a disposizione delle API (Application Programming Interface) che permettono di accedere più facilmente ai dati, ma ciò non vale nella totalità dei casi. Diventa dunque necessario l'utilizzo di strumenti e tecniche più complessi per accedere ai dati, salvarli e analizzarli in modo che siano utilizzabili concretamente per raggiungere il nostro obiettivo. Nel seguente capitolo dunque, andremo ad esporre i principali strumenti e librerie utilizzate. Inoltre andremo ad analizzare la struttura delle pagine web HTML, di cui è fondamentale essere a conoscenza per poter eseguire delle operazioni di estrazione di dati in maniera veloce ed efficiente.

3.1 Document Object Model

Il "Document Object Model", detto anche "DOM", è un'interfaccia di programmazione standardizzata per la strutturazione di documenti HTML e XML. È stato sviluppato e pubblicato dal World Wide Web Consortium (W3C), organizzazione fondata nel 1994 dall'inventore del web Tim Berners-Lee per la progettazione e la definizione di standard per il World Wide Web.

Lo scopo del Document Object Model è quello di rendere più semplice ed immediato l'accesso da parte dei programmatori ai componenti di un progetto web, e quindi aggiungere, modificare e cancellarne i contenuti. Il DOM è, in generale, una struttura ad albero che rappresenta la struttura del documento in questione, dove ogni nodo è un oggetto indipendente e gestibile.

Il motivo per il quale il DOM è centrale all'interno delle operazioni di web scraping, è che tipicamente le librerie utilizzate per fare web scraping, che vedremo nei prossimi paragrafi, funzionano principalmente in due passi:

1. Si costruiscono il DOM rappresentante il documento che si vuole analizzare.
2. Offrono una serie di API per muoversi all'interno del DOM e raccogliere solamente i dati di nostro interesse.

Per comprendere meglio questi passaggi, è utile andare a visualizzare concretamente un classico esempio di albero DOM generato da un browser web, nell'interpretazione di un documento HTML come quello che mostriamo qui di seguito:

```html
<!DOCTYPE html>
<head>
    <title> Titolo Pagina </title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>

<body>
    <div id="content">
        <div id="main-content">
            <h1> Headline </h1>
            <p> Paragrafo </p>
        </div>

        <div id="footer">
            <p class="author"><b>Author</b>:Giacomo Sanguin</p>
        </div>
    </div>
</body>
<html>
```

Figura 3.1 - Esempio di documento HTML

Tale documento, avrà un albero DOM di questo tipo (grafico prodotto grazie all'applicazione Graphviz):

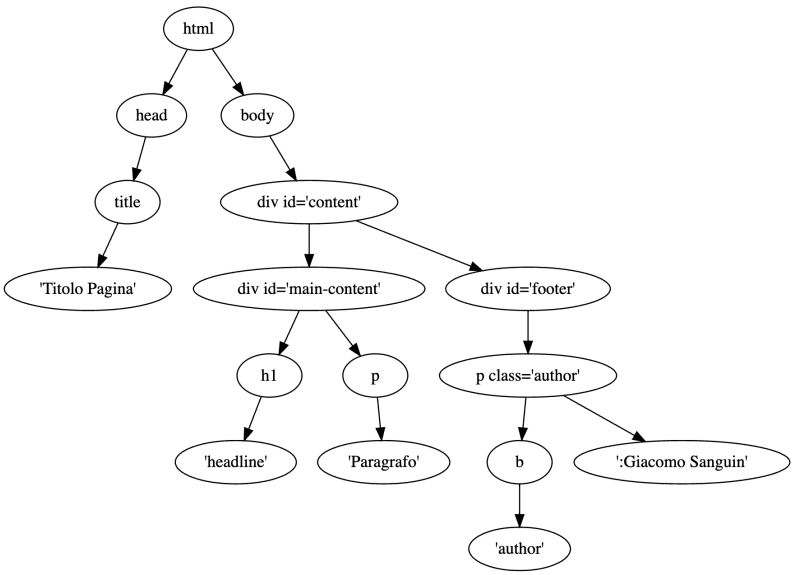

Figura 3.2 - Dom del documento HTML d'esempio

Si può vedere chiaramente come il nodo "radice" sia composto dal tag iniziale <html>, che si dirama nei due sotto alberi principali <head> e <body>, con il primo che conterrà le informazioni inerenti al titolo della pagina, e il secondo che ne conterrà il corpo, con i vari paragrafi.

3.2 Requests e BeautifulSoup

Requests [3] è la libreria di python che permette di effettuare una richiesta HTTP, ed elaborarne una risposta, così da poter stabilire una linea di comunicazione tra il software e la pagina web di cui si fornisce l'URL.
Requests dunque, ritornerà un Response Object contenente tutti i dati della risposta, che potranno poi essere utilizzati nella successiva estrazione.
Requests richiede come vincolo l'installazione del proprio pacchetto da riga di comando.
BeautifulSoup [4] è invece la libreria che permette di eseguire il parsing della pagina HTML, sfruttando, come spiegato nel paragrafo sopra, la struttura alborea del DOM. Grazie a BeautifulSoup, è possibile dunque ricavare le informazioni di interesse in maniera semplice e dinamica. Il suo funzionamento prevede che, come prima cosa, si vada a creare un oggetto BeautifulSoup al quale si passa il contenuto della pagina web HTML. Per far si che i dati ottenuti tramite la risposta HTTP siano leggibili, è inoltre necessario specificare un parsificatore specifico, che si occupi per l'appunto di rendere i dati leggibili ed estraibili.
Esistono varie tipologie di parsificatori, che indichiamo in seguito:

- html.parser: è il parsificatore di default, quello utilizzato dunque nel caso non si specifichi nessun parametro all'interno dell'oggetto BeautifulSoup.
- lxml: parsificatore estremamente veloce ed efficiente, che richiede però un'installazione aggiuntiva.
- html5lib: più lento dei precedenti, punta ad analizzare la pagina web nello stesso modo in cui lo fa il browser web.

La combinazione di Requests e BeautifulSoup risulta dunque perfetta per lo scraping di dati dalle pagine web.

3.3 Selenium

Selenium [5] è un tool di Python usato principalmente per il web scraping e l'automazione del browser web. Il suo funzionamento si basa sostanzialmente sull'emulazione del comportamento di un utente reale all'interno di un browser web. Il suo principale componente è Selenium WebDriver. E' implementato come un driver per browser e non necessita di un server per eseguire i comandi. Ciò gli permette di interagire con i browser più diffusi come Firefox, Chrome, Internet Explorer e Microsoft Edge.

È importante notare che l'uso di Selenium non basta a garantirne il corretto funzionamento, ma è necessario l'utilizzo di un web driver per browser, quali ad esempio Geckodriver per firefox, o ChromeDriver per Chrome. Grazie all'interazione tra queste componenti, sarà possibile avere un riscontro visivo reale, poiché all'esecuzione del codice il browser si aprirà automaticamente, simulando le operazioni del browser.
Come vedremo nei capitoli successivi, Selenium risulta fondamentale quando ci si trova di fronte a pagine web dinamiche, e ne è necessario l'utilizzo per eseguire il web scraping dei dati contenuti in tali pagine.

3.4 Librerie per raccolta e analisi di dati

Una volta estratti i dati e le informazioni tramite il processo di web scraping, risulta fondamentale l'utilizzo di una serie di librerie per la raccolta e l'analisi dei dati. Di seguito vengono elencate alcune delle principali, che saranno poi utilizzate nello sviluppo del nostro scraper, come vedremo nei prossimi capitoli:

- Pandas [6], il cui nome deriva da "panel data", è una libreria Python, utilizzata per il confronto, la manipolazione e l'analisi dei dati. In particolare, offre strutture dati e operazioni per manipolare tabelle numeriche e serie temporali. Pandas DataFrame, ad esempio, è una struttura dati bidimensionale che permette di visualizzare i dati in formato tabulare, e lavorare dunque con righe e colonne.
- Matplotlib [7] è una libreria python, una delle più usate nell'ambito data science, che permette la costruzione di grafici statistici per la visualizzazione di dati.
- Seaborn [8] è una libreria di python che potenzia gli strumenti di data visualization del modulo matplotlib. Seaborn fornisce diverse funzionalità per rappresentare graficamente i dati, in particolare presenta dei metodi che agevolano la costruzione dei grafici statistici.

4 Statistiche NBA e Basketball-Reference

In questo elaborato, come già anticipato, il caso preso in esame è quello relativo alle statistiche dei giocatori di pallacanestro appartenenti alla NBA, sulle quali si andrà ad eseguire un'operazione di web scraping. L'obiettivo è dunque quello di raccogliere le principali voci statistiche di ciascun atleta, così da poter sviluppare un'analisi statistica dei dati. È importante dunque, fornire un minimo di contesto sulla NBA, e in particolare sul sito web dal quale verranno estratte le varie statistiche, Basketball-Reference.

Figura 4.1 - Logo NBA

Figura 4.2- Logo Basketball-Reference

4.1 National Baketball Association

La National Baketball Association, comunemente nota come NBA, è una lega sportiva professionistica statunitense, considerata la principale e più celebre lega di basket a livello planetario. Attualmente è composta da 30 franchigie, (29 negli USA e 1 in Canada), che rappresentano ciascuna una delle principali città nordamericane. Venne fondata nel 1946 a New York, come la fusione delle due principali leghe di basket americane dell'epoca, la BAA e la ABA.

Le 30 squadre sono divise in due "conference", ognuna delle quali ha tre "division", e ogni division ha cinque squadre.

Il campionato sportivo NBA si suddivide in tre fasi principali:

- La "Regular Season", la stagione regolare, composta da 82 partite per squadra suddiviso equamente tra partite in casa e trasferta. Ogni squadra va ad affrontare 4 volte le altre 4 squadre della propria division e altre 6 squadre appartenenti alla stessa conference, 3 volte le restanti squadre della propria conference, e 2 volte le squadre dell'altra conference.

- I “Playoff”, la seconda fase a cui partecipano le migliori 16 squadre della lega, 8 per conference, che si andranno a scontrare in sfide ad eliminazione diretta al meglio delle sette partite (la squadra che vince 4 partite si qualifica al turno successivo).
- Le “Finals”, le finali NBA dove si scontrano le squadre vincitori delle due conference in una serie sempre al meglio delle sette partite.

Nella nostra tesi, andremo ad estrarre ed analizzare solamente le statistiche relative alla prima fase, la regular season, dal momento che rappresenta la lega nella sua interezza, interessando tutte le squadre e giocatori.

4.2 Basketball-Reference

Basketball-Reference è un sito web di proprietà di Sports Reference, azienda leader nel settore delle statistiche sportive, attiva in altri svariati sport oltre al basket, quali baseball, calcio, football americano, hockey. Basketball-Reference è una pagina che raccoglie tutte le informazioni relative al basket professionistico, in particolare alla NBA e agli atleti che ne fanno parte. Al suo interno, è possibile consultare qualsiasi statistica in nostro possesso, a partire dagli anni della fondazione della lega.

Si tratta quindi di un grande database online, che risulta perfetto per il nostro particolare caso d’esame, dal momento che ci è possibile estrarre tutte le informazioni e dati statistici di cui abbiamo bisogno.

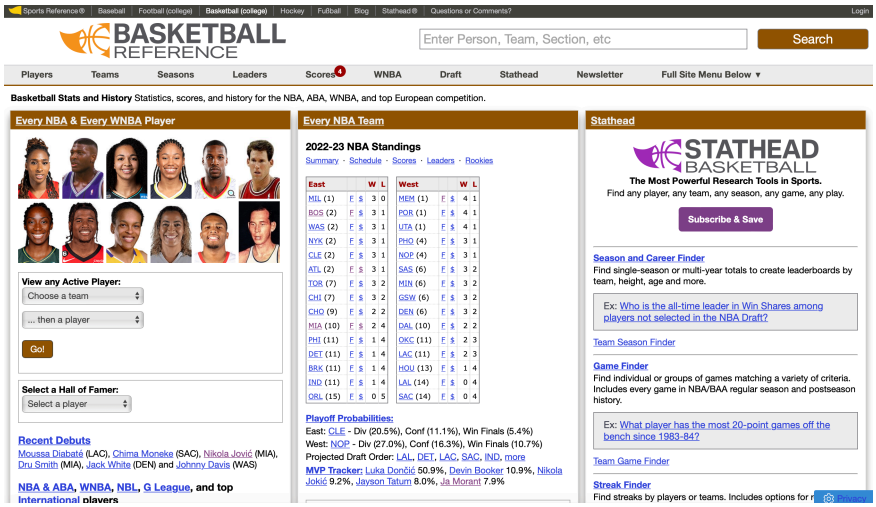

Figura 4.3 - Home page di ww.basketball-reference.com

5 Sviluppo del web scraper

Nel seguente capitolo, si andranno ad illustrare i principali passaggi che hanno portato alla realizzazione del nostro programma in Python, con il quale siamo andati ad estrarre dal web le principali voci statistiche relative agli atleti NBA. A partire dalla presentazione delle varie tecniche e strumenti utilizzati, si mostreranno le varie fasi della progettazione, fino al raggiungimento del risultato finale.
Il linguaggio di programmazione utilizzato è Python, dal momento che, come descritto nel capitolo 3, mette a disposizione una serie di librerie (BeautifulSoup, Requests, Selenium e Pandas) che si dimostrano estremamente funzionali al web scraping, oltre ad una serie di strumenti di analisi che permettono di valutare i dati ottenuti con immediatezza. L'ambiente di sviluppo scelto è Jupyter Notebook, un'applicazione web open source che permette di creare e condividere documenti testuali interattivi, contenenti oggetti quali equazioni, grafici e codice sorgente eseguibile.
Per semplicità, all'interno della tesi non sarà riportato per intero il codice del programma. Il codice sarà però visibile per intero, su Git Hub, nella repository indicata al [1] della bibliografia.

5.1 Presentazione iniziale

Per comprendere ciascuno dei passaggi che andremo a svolgere nella realizzazione del nostro scraper, risulta fondamentale un'analisi e presentazione di ciò che andremo concretamente a progettare. Una fase che ci aiuterà poi nella realizzazione effettiva del programma e nella stesura del codice.
Lo scopo del nostro programma, come già detto più volte, deve essere quello di estrarre dal sito di Basketball-Reference le statistiche principali di tutti i giocatori di uno specifico anno. Per fare questo, è utile quindi andare ad analizzare la specifica sezione del sito da dove si andranno ad estrarre i dati, che nel nostro caso, sarà la pagina di riferimento di ciascuna delle 30 franchigie NBA per l'anno a cui si sta facendo riferimento.
In ciascuno di tali URL (nel caso dell'immagine inserita qui sotto, si fa riferimento alla prima squadra in ordine alfabetico, gli Atlanta Hawks), sono presenti le statistiche dell'anno specificato di tutti i giocatori iscritti a roster, dai punti segnati a partita, al salario guadagnato nel corso dell'anno.

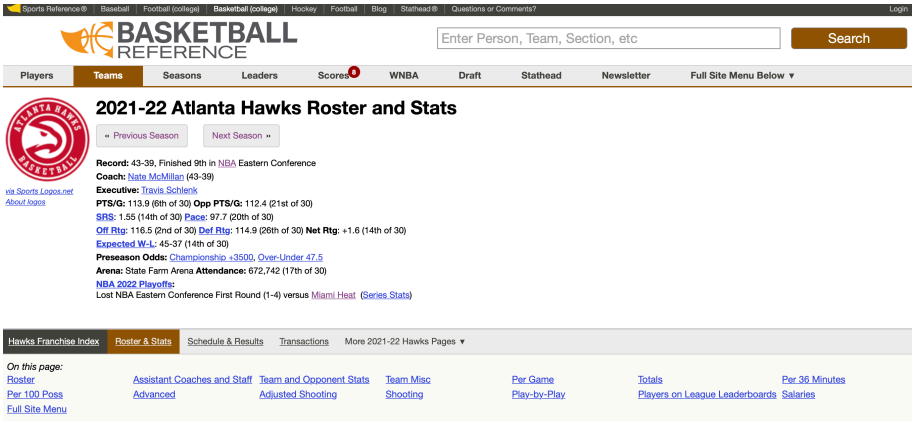

Figura 5.1 - Front end di www.basketball-reference.com/teams/ATL/2022.html

In particolare notiamo subito che il nostro software dovrà essere in grado di navigare in ciascuno degli specifici URL, e sarà dunque necessario, in una prima fase, raccoglierli tutti e 30.

Per fare ciò, sarà creata una funzione ausiliaria che ci permetterà di estrarre in una lista tutte le "URL endings" dalla homepage del sito, così da poter poi scandire ciascun URL mediante un ciclo for.

Nella fase successiva, ci focalizzeremo nell'estrazione effettiva delle statistiche degli atleti. Come si può evincere dall'immagine qui di seguito, le statistiche di nostro interesse sono raccolte in una tabella all'interno della pagina web. Il nostro codice di conseguenza, dovrà essere in grado di selezionare ciascuno dei valori e raccoglierli in un DataFrame contenente tutti i giocatori delle 30 franchigie, con associate le loro personali statistiche.

Per Game Share & Export ▾ Glossary

Regular Season | Playoffs

Rk		Age	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS/G
1	Trae Young	23	76	76	34.9	9.4	20.3	.460	3.1	8.0	.382	6.3	12.3	.512	.536	6.6	7.3	.904	0.7	3.1	3.7	9.7	0.9	0.1	4.0	1.7	28.4
2	John Collins	24	54	53	30.8	6.3	11.9	.526	1.2	3.3	.364	5.1	8.7	.588	.576	2.5	3.1	.793	1.7	6.1	7.8	1.8	0.6	1.0	1.1	3.0	16.2
3	De'Andre Hunter	24	53	52	29.8	4.8	10.8	.442	1.4	3.7	.379	3.4	7.2	.474	.506	2.4	3.1	.765	0.5	2.8	3.3	1.3	0.7	0.4	1.3	2.9	13.4
4	Kevin Huerter	23	74	60	29.6	4.7	10.3	.454	2.2	5.6	.389	2.5	4.7	.530	.559	0.6	0.7	.808	0.4	3.0	3.4	2.7	0.7	0.4	1.2	2.5	12.1
5	Bogdan Bogdanović	29	63	27	29.3	5.4	12.6	.431	2.7	7.3	.368	2.7	5.3	.516	.537	1.5	1.8	.843	0.5	3.5	4.0	3.1	1.1	0.2	1.1	2.1	15.1
6	Chaundee Brown Jr.	23	3	2	27.7	3.0	8.3	.360	2.0	5.0	.400	1.0	3.3	.300	.480	1.7	2.0	.833	0.3	4.3	4.7	1.3	0.7	0.0	0.0	2.3	9.7
7	Clint Capela	27	74	73	27.6	5.0	8.2	.613	0.0	0.0	.000	5.0	8.1	.614	.613	1.1	2.3	.473	3.8	8.1	11.9	1.2	0.7	1.3	0.6	2.2	11.1
8	Wes Iwundu	27	3	1	27.3	2.7	6.0	.444	1.0	1.7	.600	1.7	4.3	.385	.528	1.0	1.3	.750	1.0	3.3	4.3	0.0	0.3	0.0	0.3	2.7	7.3
9	Danilo Gallinari	33	66	18	25.3	3.9	9.0	.434	1.7	4.5	.381	2.2	4.6	.485	.528	2.1	2.4	.904	0.5	4.1	4.7	1.5	0.4	0.2	0.6	1.4	11.7
10	Cam Reddish	22	34	7	23.4	4.1	10.1	.402	1.7	4.5	.379	2.4	5.6	.421	.487	2.1	2.4	.900	0.6	1.9	2.5	1.1	1.0	0.3	1.3	1.3	11.9
11	Cameron Oliver	25	2	0	21.5	5.0	7.5	.667	0.5	1.5	.333	4.5	6.0	.750	.700	1.0	1.5	.667	1.0	2.0	3.0	1.5	0.5	0.5	0.0	2.5	11.5
12	Onyeka Okongwu	21	48	6	20.7	3.3	4.7	.690	0.0	0.0		3.3	4.7	.690	.690	1.7	2.3	.727	2.4	3.5	5.9	1.1	0.6	1.3	0.9	3.1	8.2
13	Delon Wright	29	77	8	18.9	1.6	3.5	.454	0.6	1.5	.379	1.0	2.0	.510	.535	0.7	0.8	.857	0.6	2.2	2.9	2.4	1.2	0.2	0.6	0.7	4.4
14	Malcolm Hill	26	3	0	15.3	1.7	2.7	.625	1.0	1.7	.600	0.7	1.0	.667	.813	1.3	1.3	1.000	0.7	1.3	2.0	0.3	1.3	0.3	0.0	2.3	5.7
15	Lou Williams	35	56	0	14.3	2.2	5.7	.391	0.7	1.8	.363	1.6	3.9	.404	.448	1.2	1.4	.859	0.3	1.3	1.6	1.9	0.5	0.1	0.8	0.9	6.3
16	Timothé Luwawu-Cabarrot	26	52	18	13.2	1.4	3.6	.398	0.8	2.3	.361	0.6	1.2	.469	.516	0.7	0.8	.854	0.3	1.3	1.6	0.8	0.3	0.1	0.4	1.5	4.4
17	Lance Stephenson	31	6	0	11.7	0.8	2.2	.385	0.0	0.5	.000	0.8	1.7	.500	.385	0.2	0.3	.500	0.2	2.3	2.5	1.8	0.0	0.0	0.7	1.3	1.8
18	Solomon Hill	30	13	1	10.7	0.2	1.5	.150	0.2	1.0	.154	0.1	0.5	.143	.200	0.0	0.0		0.5	1.3	1.8	0.9	0.3	0.2	0.1	1.2	0.6
19	Gorgui Dieng	32	44	3	8.4	1.2	2.5	.473	0.7	1.5	.426	0.5	1.0	.545	.603	0.4	0.6	.731	0.7	2.0	2.8	0.8	0.3	0.3	0.5	1.2	3.5
20	Skylar Mays	24	28	5	7.9	1.1	2.3	.500	0.3	0.9	.320	0.9	1.4	.615	.563	0.3	0.3	.889	0.2	0.7	0.9	0.6	0.3	0.0	0.4	0.3	2.9
21	Kevin Knox	22	17	0	6.5	0.9	2.6	.356	0.3	1.5	.192	0.6	1.1	.579	.411	0.5	0.7	.750	0.4	0.9	1.3	0.4	0.1	0.1	0.1	0.8	2.7
22	Jalen Johnson	20	22	0	5.5	1.0	1.9	.537	0.1	0.6	.231	0.9	1.3	.679	.573	0.2	0.3	.714	0.1	1.1	1.2	0.1	0.1	0.1	0.4	0.4	2.4
23	Cat Barber	27	3	0	4.3	0.0	1.3	.000	0.0	0.0		0.0	1.3	.000	.000	0.0	1.3	.000	0.3	0.7	1.0	1.0	0.0	0.0	0.3	0.3	0.0
24	Sharife Cooper	20	13	0	3.0	0.2	1.1	.214	0.1	0.5	.167	0.2	0.6	.250	.250	0.0	0.0		0.0	0.4	0.4	0.4	0.0	0.0	0.4	0.1	0.5

Figura 5.2 - tabella contenente le statistiche di interesse

Anche in questo caso, sarà utile fornire una funzione ausiliaria che ci permetta di creare una lista contente tutte le denominazioni delle singole statistiche (Age, Games, Minutes Played etc.), che sarà poi estremamente funzionale nella creazione del DataFrame. Andremo poi a completare il nostro DataFrame aggiungendo un'ulteriore voce statistica, il salario di ogni giocatore guadagnato durante la stagione, contenuto in una tabella posizionata sul fondo della pagina. Per fare questo, come vedremo al punto 5.3, dovremo far fronte al problema delle pagine web dinamiche.

Salaries

Rk		Salary
1	John Collins	$23,000,000
2	Danilo Gallinari	$20,475,000
3	Bogdan Bogdanović	$18,000,000
4	Clint Capela	$17,103,448
5	Delon Wright	$8,526,316
6	Trae Young	$8,326,471
7	De'Andre Hunter	$7,775,400
8	Onyeka Okongwu	$6,104,280
9	Kevin Knox	$5,845,978
10	Lou Williams	$5,000,000
11	Kevin Huerter	$4,253,357
12	Gorgui Dieng	$4,000,000
13	Jalen Johnson	$2,659,560
14	Timothé Luwawu-Cabarrot	$1,939,350
15	Lance Stephenson	$138,019
16	Wes Iwundu	$102,831
17	Chris Clemons	$95,930
18	Cameron Oliver	$85,578
19	Malcolm Hill	$53,176
20	Malik Ellison	$53,176
21	Justin Tillman	$53,176
22	Chaundee Brown	$53,176
23	Cat Barber	$53,176
24	Skylar Mays	$34,231
25	Sharife Cooper	$0

Figura 5.3 - tabella contenente i salari

Una volta creato il nostro DataFrame di riferimento, potremo procedere con un'analisi dei dati ricavati, come vedremo nel capitolo conclusivo.

5.2 Funzioni ausiliarie

Nel nostro programma utilizzeremo due funzioni ausiliarie, chiamate **get_teams_url()** e **get_header**(). La prima, ci permetterà di salvare in una lista tutte le terminazioni degli URL delle 30 squadre NBA, la seconda di salvare in una lista tutte le voci statistiche che utilizzeremo nella creazione del nostro DataFrame.

Per **get_teams_url()**, ci si rende immediatamente conto che l'URL della pagina specifica di ogni squadra, è formato da "www.basketball-reference.com/teams/sigla della squadra/anno.html", e ci basterà dunque ricavare la sigla di ogni franchigia per raggiungere

l'obiettivo. Quest'ultime, sono fortunatamente contenute all'interno della home page di basketball-reference, in un menu a tendina chiamato "select team pages" mostrato nell'immagine di seguito. Si può notare facilmente inoltre, osservando il codice HTML della pagina stessa, che le terminazioni si trovano all'interno di un tag con "id=teams".

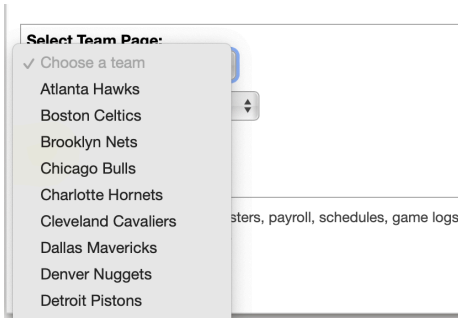

Figura 5.4 - Front end della sezione Select Team Page

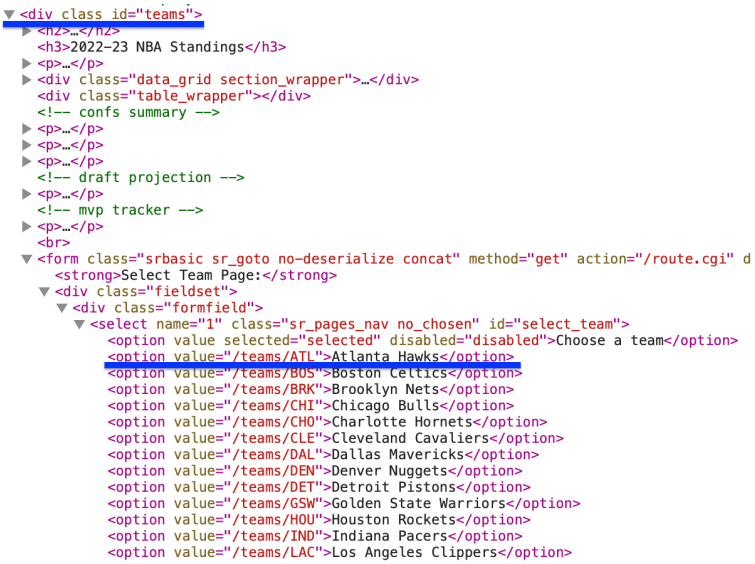

Figura 5.5 - codice HTML della sezione Select Team Page

Per isolare il codice HTML nel quale ci vogliamo soffermare, si utilizzeranno requests e BeautifulSoup, introdotte nel terzo capitolo. Procederemo dunque con una richiesta HTTP, fornendo l'URL della home page del sito, e con BeautifulSoup eseguiremo il parsing del codice ottenuto.

In particolare, nel creare un BeautifulSoup Object, forniamo come parametro il parsificatore lxml, il più veloce ed efficiente.

A questo punto, segue la fase di estrazione dell'informazione. Per fare ciò, utilizzeremo i metodi di ricerca **find()** e **find_all()**, due dei metodi principali della libreria, che permettono di trovare frazioni della pagina HTML indicando il tag nel quale sono contenute (come intuibile dal nome, find_all() si differenzia da find() perché a differenza del secondo, trova tutti i tag presenti all'interno del codice, e non solo il primo). Si crea dunque una variabile "teams_option" in cui si raccolgono tutte le righe di codice col tag <option>. A questo punto, mediante un ciclo for, si inseriscono tutti gli attributi <value> (che sono concretamente le terminazioni degli URL che si vuole estrarre) all'interno di una lista, dall'opzione 2 alla 31, poiché la prima opzione, come si vede chiaramente in figura 9, è solamente l'header "Choose a team" del menu a tendina.

```
def get_teams_url():
    home_url = "https://www.basketball-reference.com"
    data = requests.get(home_url)
    soup = BeautifulSoup(data.content, "lxml")
    teams_html = soup.find('div',id='teams')
    teams_option = teams_html.find_all('option')[1:31]
    teams = []
    for each_option in teams_option:
        teams.append(each_option.get('value'))

    return teams
```

Figura 5.6 - Funzione get_teams_url()

L'output di get_teams_url() è il seguente (per semplicità mostriamo solo i primi 3 elementi).

```
['/teams/ATL', '/teams/BOS', '/teams/BRK']
```

Figura 5.7 - Output di get_teams_url()

Per quanto riguarda invece **get_header()**, ci si rifà prima di tutto all'URL della pagina specifica di una delle 30 squadre, che forniremo nella request. Come si vede in figura 7, i dati che si vuole estrarre sono i nomi delle voci statistiche presenti nell'header della tabella, a partire da "Age" fino a "PTS/G". Anche in questo caso, osservando il codice HTML della pagina, si capisce che tali informazioni si trovino all'interno del tag <table> con "id=per_game".

```
▼ <table class="stats_table sortable now_sortable" id="per_game" data-cols-to-freeze=",2">
    <caption>Per Game Table</caption>
  ▶ <colgroup>…</colgroup>
  ▼ <thead>
    ▼ <tr>
        <th aria-label="Rk" data-stat="ranker" scope="col" class="ranker poptip
        sort_default_asc show_partial_when_sorting center" data-tip="Rank">Rk</th>
        <th aria-label=" " data-stat="player" scope="col" class=" poptip sort_default_asc
        center"> </th>
        <th aria-label="Age" data-stat="age" scope="col" class=" poptip sort_default_asc
        center" data-tip="Player's age on February 1 of the season">Age</th>
        <th aria-label="Games" data-stat="g" scope="col" class=" poptip center" data-tip=
        "Games">G</th>
        <th aria-label="Games Started" data-stat="gs" scope="col" class=" poptip center" data-
        tip="Games Started">GS</th>
        <th aria-label="Minutes Played Per Game" data-stat="mp_per_g" scope="col" class="
        poptip center" data-tip="Minutes Played Per Game">MP</th>
        <th aria-label="Field Goals Per Game" data-stat="fg_per_g" scope="col" class=" poptip
        center" data-tip="Field Goals Per Game">FG</th>
```

Figura 5.8 - Codice HTML dell'header della tabella in figura 7

Ripetiamo dunque l'operazione di ricerca, vista in get_teams_url(),utilizzando il metodo find(). Ricerchiamo poi ognuno dei tag <th>, ciascuno dei quali contiene una della voci

statistiche, e mediante un ciclo for andiamo ad inserire gli attributi “text” in una lista, a partire dal terzo tag <th>, dal momento che i primi due non contengono informazioni di nostro interesse.

```
def get_header():

    first_team_url = get_teams_url()
    team_url= "https://www.basketball-reference.com" + first_team_url[0] + "/2022.html"
    data = requests.get(team_url)
    soup = BeautifulSoup(data.content, "lxml")
    h = soup.find('table', id="per_game")
    h1= h.find('tr')
    h2 = h1.find_all('th')[2:28]
    header = []
    for each_th in h2:
        each_th.text.replace(' ', '')
        header.append(each_th.text)

    return header
```

Figura 5.9 - Funzione get_header

L’output di get_header() è il seguente (per semplicità mostriamo solo i primi 3 elementi).

```
['Age', 'G', 'GS']
```

Figura 5.10 - output di get_header()

5.3 Dynamic Websites

Uno dei principali problemi che si presentano nel processo di web scraping, è quello delle pagine web dinamiche. Tale problema è tipico dei siti web più moderni, che fanno un pesante uso di Javascript al loro interno per permettere il “rendering” di dati interattivi. La conseguenza di ciò, si traduce nell’impossibilità di estrarre concretamente i dati dal codice HTML della pagina, che risultano nascosti e inutilizzabili.

Nel nostro specifico caso, questa eventualità si verifica nel momento in cui si vuole estrarre il dato relativo al salario di ciascuno dei giocatori di una squadra, essendo tale informazione contenuta in una tabella nel fondo della pagina, e quindi ne risulta impossibile il caricamento. Utilizzando BeautifulSoup come parsificatore difatti, non è possibile andare a lavorare con il codice HTML di cui si avrebbe bisogno.

Per ovviare a questo tipo di imprevisto esistono varie possibilità, la più comune delle quali è quella dell’utilizzo di Selenium WebDriver.

Selenium, come detto in 3.3, permette di emulare il comportamento di un utente reale all'interno di un browser, e ci permette di ottenere del codice HTML "utilizzabile" per il web scraping.

Andiamo dunque a creare una funzione, chiamata **get_all_teams_html(year)**, che riceve come argomento l'anno a cui vogliamo fare riferimento per estrarre le statistiche, e va a scaricare le pagine web di ciascuna delle squadre utilizzando Selenium. Quando si utilizza Selenium difatti, risulta più efficiente scaricare nel proprio dispositivo, in una fase iniziale, la pagina web, per ridurre così il tempo di esecuzione del programma.

A questo punto, mediante un ciclo for eseguiamo uno scorrimento tra tutti gli URL delle 30 franchigie. Per ciascuno di essi poi, andiamo a creare un oggetto webdriver, ricordandoci di inizializzare, come specificato al 3.3, il web driver per il browser che si vuole utilizzare (nel nostro caso Chromedriver), fornendo come argomento il suo "executable path".

Eseguiamo infine un comando javascript utilizzando il metodo **execute_script()**, seguito dal metodo **sleep()**, per fare in modo che possa essere visualizzato l'intero contenuto della pagina.

```
def get_all_teams_html(year):

    teams_url = get_teams_url()
    for each_url in teams_url:
        team_url = "https://www.basketball-reference.com" + each_url + "/" + str(year) + ".html"
        d_url = each_url.replace('/teams', '')
        driver = webdriver.Chrome(executable_path = '/Users/giacomosanguin/Documents/Chromedriver/chromedriver')
        driver.get(team_url)
        driver.execute_script("window.scrollTo(1,10000)")
        time.sleep(3)

        with open("teams{}.html".format(d_url+str(year)), "w+") as f:
            f.write(driver.page_source)
```

Figura 5.11 - Funzione get_all_teams_html(year)

5.4 Completamento del programma

Una volta definite le funzioni ausiliarie, e risolto il problema delle pagine web dinamiche, è possibile mettere assieme tutti i vari elementi e procedere con l'estrazione effettiva dei dati all'interno di un DataFrame.

Per prima cosa, si utilizzerà un ciclo for per aprire le 30 pagine web di ciascuna squadra, scaricate precedentemente tramite Selenium. Per ciascuna di esse poi, andremo quindi a ricercare mediante l'utilizzo dei metodi find() e find_all(), come in parte già visto nei paragrafi precedenti, le informazioni da estrarre. Per quanto riguarda le statistiche individuali di ogni giocatore, contenute nella tabella in figura 5.2, queste si trovano all'interno del tag

<table> con “id=per_game” del codice HTML della pagina. Ciascun tag <tr> poi (a partire dal secondo, dal momento che il primo si riferisce all’header della tabella), come visibile nella figura 5.12 qui di seguito, fa riferimento ad una riga della tabella, e quindi ad un singolo giocatore.

```
▼ <table class="stats_table sortable now_sortable" id="per_game" data-cols-to-freeze=",2">
    <caption>Per Game Table</caption>
  ▶ <colgroup>…</colgroup>
  ▶ <thead>…</thead>
  ▼ <tbody>
    ▼ <tr data-row="0">
        <th scope="row" class="center " data-stat="ranker">1</th>
      ▶ <td class="left " data-stat="player" csk="Young,Trae">…</td>
        <td class="center " data-stat="age">23</td>
      ▶ <td class="right " data-stat="g">…</td>
        <td class="right " data-stat="gs">76</td>
        <td class="right " data-stat="mp_per_g">34.9</td>
        <td class="right " data-stat="fg_per_g">9.4</td>
        <td class="right " data-stat="fga_per_g">20.3</td>
        <td class="right " data-stat="fg_pct">.460</td>
        <td class="right " data-stat="fg3_per_g">3.1</td>
        <td class="right " data-stat="fg3a_per_g">8.0</td>
        <td class="right " data-stat="fg3_pct">.382</td>
        <td class="right " data-stat="fg2_per_g">6.3</td>
        <td class="right " data-stat="fg2a_per_g">12.3</td>
        <td class="right " data-stat="fg2_pct">.512</td>
        <td class="right " data-stat="efg_pct">.536</td>
        <td class="right " data-stat="ft_per_g">6.6</td>
        <td class="right " data-stat="fta_per_g">7.3</td>
        <td class="right " data-stat="ft_pct">.904</td>
        <td class="right " data-stat="orb_per_g">0.7</td>
        <td class="right " data-stat="drb_per_g">3.1</td>
        <td class="right " data-stat="trb_per_g">3.7</td>
        <td class="right " data-stat="ast_per_g">9.7</td>
        <td class="right " data-stat="stl_per_g">0.9</td>
        <td class="right " data-stat="blk_per_g">0.1</td>
        <td class="right " data-stat="tov_per_g">4.0</td>
        <td class="right " data-stat="pf_per_g">1.7</td>
        <td class="right " data-stat="pts_per_g">28.4</td>
      </tr>
    ▶ <tr data-row="1">…</tr>
    ▶ <tr data-row="2">…</tr>
```

Figura 5.12 - Codice HTML della tabella in figura 5.2

Si dovranno dunque andare ad associare all’interno di un dizionario, il nome del giocatore e tutte le sue statistiche, contenute proprio all’interno del tag <tr> a lui riferito.
Per differenziare poi le singole statistiche contenute all’interno dei tag <td>, sfrutteremo una lista “header”, inizializzata precedentemente grazie alla funzione get_header(), per associare a ciascuna voce statistica il corretto valore. I passaggi appena illustrati, sono visibili alla figura 5.13 qui di seguito.

```
for each_tr in per_game:
    #creating a dictionary that will contain players and their respective stats
    players = {}
    players['Name'] = each_tr.find('a').text
    players['Team'] = team_name
    name = each_tr.find('a').text
    per_game2 = each_tr.find_all('td')[1:]
    for each_td in per_game2:
        stats_player.append(each_td.text)
    for n in range(1,26):
        players[header[n]] = stats_player[n]
```

Figura 5.13 - Codice relativo all’estrazione delle statistiche dei giocatori

Una volta raccolte tutte le statistiche, non resta che estrarre dalla tabella in figura 5.3 l'informazione relativa al salario di ogni giocatore, e aggiungerla al dizionario precedentemente creato. Si andrà dunque a selezionare in una variabile "salary", la parte di codice relativa a tale tabella, contenuta all'interno del tag <table> con "id=salaries2", e per ciascuna riga <tr>, relativa ad uno dei giocatori, andremo ad associare il dato ricercato, al giocatore a cui fa riferimento.

Nel fare ciò, come visibile all'immagine 5.14 qui di seguito, ci si rende conto che per alcune delle pagine HTML delle 30 franchigie, la tabella dei salari contiene delle righe intermedie prive di alcun tipo di valore, e questo risulterebbe un problema in fase di estrazione, poiché lo scraper andrebbe ad estrarre delle informazioni inesistenti, provocando un errore in fase di esecuzione.

Figura 5.14 - riga "anomala"

È essenziale perciò andare ad "eliminare" dal codice da noi parsificato, tali righe, contenute all'interno di un tag <tr> con "class=thead", mediante il metodo **decompose()** appartenente alla libreria BeautifulSoup, come vediamo nel pezzo di codice in figura 5.15.

```
problem = salary.select("tr.thead")
if problem:
    salary.find('tr', class_='thead').decompose()
```

Figura 5.15 - Codice relativo all'eliminazione delle righe anomale

A questo punto, non resta che unire le statistiche e il salario di ogni giocatore in un'unica lista, che andremo poi a visualizzare in formato DataFrame, sfruttando le funzionalità della libreria Pandas.

5.5 Output del programma

L'output del programma, sarà il DataFrame creato in precedenza, che risulterà essere una tabella con n righe, e m colonne, dove n è numero dei giocatori complessivo per l'anno in questione + 1, e m il numero delle voci statistiche selezionate in fase di web scraping. Come vediamo in figura 5.16 inoltre, grazie alle funzionalità di Jupyter Notebook, l'output viene visualizzato in maniera ordinata e completa.

	Name	Team	G	GS	MP	FG	FGA	FG%	3P	3PA	...	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS/G	Salary
0	Trae Young	ATL	76	76	34.9	9.4	20.3	.460	3.1	8.0	...	0.7	3.1	3.7	9.7	0.9	0.1	4.0	1.7	28.4	8326471
1	John Collins	ATL	54	53	30.8	6.3	11.9	.526	1.2	3.3	...	1.7	6.1	7.8	1.8	0.6	1.0	1.1	3.0	16.2	23000000
2	De'Andre Hunter	ATL	53	52	29.8	4.8	10.8	.442	1.4	3.7	...	0.5	2.8	3.3	1.3	0.7	0.4	1.3	2.9	13.4	7775400
3	Kevin Huerter	ATL	74	60	29.6	4.7	10.3	.454	2.2	5.6	...	0.4	3.0	3.4	2.7	0.7	0.4	1.2	2.5	12.1	4253357
4	Bogdan Bogdanović	ATL	63	27	29.3	5.4	12.6	.431	2.7	7.3	...	0.5	3.5	4.0	3.1	1.1	0.2	1.1	2.1	15.1	18000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
710	Alize Johnson	WAS	3	0	6.0	0.7	2.0	.333	0.0	0.7	...	1.7	2.3	4.0	0.0	0.0	0.0	1.0	0.0	1.3	99380
711	Cassius Winston	WAS	7	0	5.6	0.6	1.6	.364	0.3	0.9	...	0.0	0.1	0.1	1.0	0.0	0.0	0.4	1.3	2.0	0
712	Jordan Goodwin	WAS	2	0	3.0	0.0	1.5	.000	0.0	0.5	...	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.5	0.0	53176
713	Jaime Echenique	WAS	1	0	3.0	0.0	0.0		0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	53176
714	Joel Ayayi	WAS	7	0	2.9	0.1	0.9	.167	0.0	0.1	...	0.1	0.3	0.4	0.6	0.0	0.0	0.0	0.0	0.3	0

715 rows × 28 columns

Figura 5.16 - output del programma

6 Analisi dei dati

Nel seguente capitolo, verrà svolta un'analisi dei dati ottenuti mediante il processo di web scraping, sfruttando le funzionalità delle librerie Matplotlib e Seaborn. Esistono difatti moltissimi spunti e valutazioni interessanti che possono essere fatti osservando il DataFrame ottenuto, contenente tutte le voci statistiche principali di ogni giocatore.
In particolare, in questa tesi andremo ad analizzare la correlazione tra alcuni di questi dati, che ci porteranno a dei risultati non sempre così scontati.

6.1 Correlazione tra minuti giocati e punti per partita

Una prima analisi possibile, è quella della correlazione tra i minuti giocati da ciascun giocatore in media per partita, il cui dato è contenuto nella colonna "MP" del nostro DataFrame, e quello dei punti segnati per partita, il cui dato è contenuto nella colonna "PTS/G". Si tratta di una correlazione piuttosto intuitiva, dal momento che è piuttosto logico e naturale che più un giocatore giochi, più ha la possibilità di segnare dei punti. Al contrario, con meno minuti a disposizione, è scontato che tralasciando qualche eccezione, verranno segnati meno punti. Ciononostante, è utile sfruttare gli strumenti forniti dalle librerie Matplotlib e Seaborn, per ottenere una prova concreta di tali valutazioni.
Utilizzando dunque il metodo scatterplot(), possiamo andare a visualizzare un grafico di dispersione che risulta molto utile per valutare tale correlazione, come si vede in figura 6.1.
Da tale grafico, si evince infatti che, come pronosticato, al crescere dei minuti, salgono automaticamente anche i punti segnati. Si nota inoltre, la presenza di alcuni "outliers", dei valori anomali che ci mostrano come, in alcuni casi, il risultato della relazione non sia così scontato.

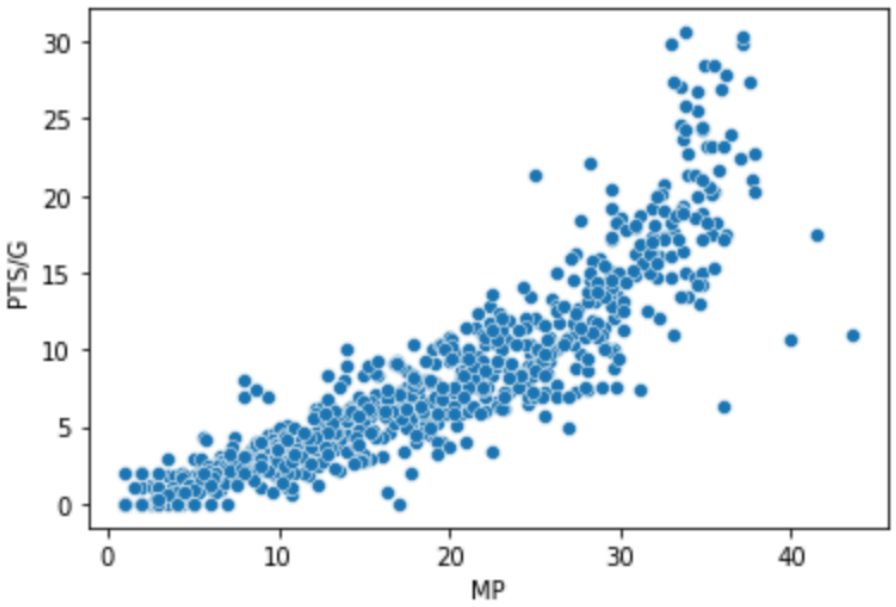

Figura 6.1 - Correlazione tra minuti giocati e punti segnati

6.2 Correlazione tra Approximate Value e salario percepito

Una correlazione molto più stimolante da analizzare rispetto alla precedente, è quella tra la performance effettiva di un giocatore sul campo da gioco, e il salario che percepisce. Normalmente, verrebbe naturale immaginare che i giocatori più pagati siano anche i più determinanti, in sostanza, i migliori. Ma questa è una valutazione che risulta in molti casi errata, poiché spesso ci si trova di fronte a casi di giocatori che performano in maniera diversa rispetto al contratto firmato prima dell'inizio della stagione.

Per fare questo, sfrutteremo la metrica Approximate Value [9], introdotta da Dean Oliver [10], analista americano ricordato come uno dei pionieri dell'introduzione dell'analisi dei dati all'interno del mondo del basket. Si tratta di una metrica piuttosto semplice, che ci permette però di dare già una valutazione piuttosto esaustiva delle performance di ciascun giocatore. La formula dell'Approximate Value è la seguente:

$$Credits = (Points) + (Rebounds) + (Assists) + (Steals) + (Blocks) - (FieldGoalMissed) - (FreeThrowsMissed) - (Turnovers)$$

$$ApproximateValue = \frac{(Credits)^{\frac{3}{4}}}{21}$$

Visualizzando quindi il grafico di dispersione relativo alla correlazione tra Approximate Value e salario percepito, visibile in figura 6.2, osserviamo un risultato estremamente interessante.

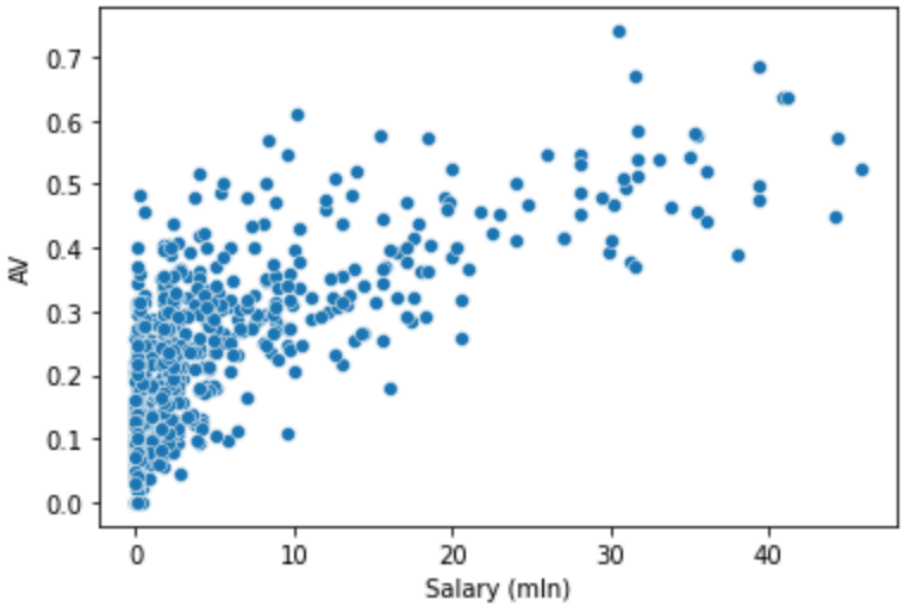

Figura 6.2 - Correlazione tra Approximate Value e salario percepito

In generale, dal grafico si evince chiaramente che, i giocatori più pagati, siano tendenzialmente anche i giocatori più determinanti, ma allo stesso modo, si nota che i giocatori dall'Approximate Value più alto, siano distribuiti in maniera piuttosto uniforme in tutte le varie fasce di salario. In particolare, il giocatore dall'AV più alto, non raggiunge nemmeno le prime 30 posizioni in ordine di salario e al contrario, il giocatore più pagato, ha un AV inferiore rispetto a giocatori che percepiscono meno del 25% del suo stipendio. Un risultato per certi versi inaspettato, ma che dipende sostanzialmente dal fatto che la retribuzione di un atleta NBA professionista, non dipende dai risultati sportivi che otterrà, bensì da quelli che ha ottenuto precedentemente.

7 Conclusioni

Nei precedenti capitoli, abbiamo illustrato il presente progetto di tesi, a partire da un'introduzione sulle tecniche e le tematiche affrontate, passando poi per l'applicazione effettiva delle conoscenze acquisite tramite lo sviluppo di un web scraper in grado di raccogliere tutte le statistiche dei giocatori NBA per un determinato anno, fino ad arrivare ad un'analisi statistica dei dati ottenuti.

Le criticità principali riscontrate, sono indubbiamente la risoluzione del problema delle pagine web dinamiche, che ha reso necessario l'utilizzo di un procedimento differente per l'estrazione dei dati, e l'anomala formattazione di alcune tabelle dalle quali si volevano estrarre dei dati, affrontate al punto 5.3 e 5.4.

Per quanto riguarda la fase di analisi finale, si sono viste solo alcune delle innumerevoli possibilità di valutazione dei dati, che potrebbero essere sfruttati, in prospettiva futura, in diversi altri casi e applicazioni.

In conclusione, possiamo certamente affermare che il lavoro svolto per raggiungere il risultato finale, ci ha permesso di comprendere l'utilità che uno strumento come il web scraping, può avere all'interno di svariati campi, che vanno al di là dell'ambito affrontato nel presente elaborato.

Bibliografia

[1] *GitHub repository contenente codice del programma*
https://github.com/jiecksanguin/Web-Scraping-of-NBA-statistical-data.git

[2] V. Krotov, L. Johnson, L. Silva, *Tutorial: Legality and Ethics of Web Scraping*, 2020

[3] *Requests Documentation*
https://requests.readthedocs.io/en/latest/

[4] *BeautifulSoup Documentation*
https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[5] *Selenium Documentation*
https://www.selenium.dev/selenium/docs/api/py/api.html

[6] *Pandas Documentation*
https://pandas.pydata.org/docs/

[7] *Matplotlib Documentation*
https://matplotlib.org/stable/index.html

[8] *Seaborn Documentation*
https://seaborn.pydata.org

[9] *Approximate Value metric Documentation*
https://www.nbastuffer.com/analytics101/approximate-value/

[10] D. Oliver, *Basketball on Paper: Rules and Tools for Performance Analysis*, 2004