Table des matières

Table des figures

Introduction

1 Part 1 - webscraping

1.1 Question 1

We want to retrieve data for the players and the teams. The starting URL to do this are `http://www.basketball-reference.com/players/` and `http://www.basketball-reference.com/teams/` respectively.

1.2 Question 2

We decided to scrap the pages of all the players and not only the active ones, as the data could prove useful to assess the salaries of the players. However we also created a function `active_players_BS` which return which players are active and which are not.

The players are classified according to the first letter of their last name. We first retrieve all the letters which have players whose name starts with it, from `http://www.basketball-reference.com/players`. For example, there is no player whose name starts with X. For a letter c, we then download the page `http://www.basketball-reference.com/players/c` and extract the links to the players' page.

1.2.1 Question 2.a

The regular expression we use to retrieve the letters is :

```
<a href="/players/([a-z]+)/">[A-Z]+</a></td>
```

The regular expression we use to identify the links to the players' page is the following :

```
[^p]><a href="(/players/./.+)"
```

1.2.2 Question 2.b

The code we use to retrieve the letters with BeautifulSoup is :

```
for row in soup('td', {'class': 'align_center bold_text valign_bottom xx_large_text'}):
        letter = str(row.a.get('href').split('/')[2])
```

The code we use to identify the links with BeautifulSoup is the following :

```
for player in soup.tbody.find_all('tr'):
        link = str(player.td.a.get('href'))
```

1.2.3 Question 2.c

We choosed to use the regex code because it is faster and use less memory. In fact, we ran 100 instances of both versions and obtained the following results :

Method	Time	Memory size
regex	1.167 s	2.6 MB
BS	3.911 s	3.7 MB

Those results were obtained using pre-downloaded pages to avoid measuring the download times.

Furthermore, the regex code is more readable since it contains less functions that need to called.

1.3 Question 3

1.3.1 Question 3.a

Using the links parser from 1.2 , the entire player database was parsed using BeautifulSoup and basic profile information was scraped from the website. A standard template was designed to hold variables such as PlayerID (derived from the player's HTML link was used as an unique identifier), name, positions, shooting hand, height, weight, birth date, city, state/country of birth, experience and death date. Where there are no values listed n the profile page, null values are assigned the variables.

1.3.2 Question 3.b

The method used to scrap the basic statistics and salaries data, is as follows :

- **Scraping phase** by analyzing the structure of the tables (HTML tags) into .csv files
- Adding an unique **Player ID** at each table constructed as follows : `/players/b/bryanko01.html` becomes bbryanko01. Each player then can be accessed with its unique key
- **Cleaning phase** where some column's formats are modified so that they can be loaded and manipulated in the SQLite database

Among all statistics of the basketball players, scraping some tables are sufficient to do the analysis requested in Part 2 (see 2). The tables selected as player's statistics are below :

- Totals
- Per Game

These tables deals with statistics by season (Totals) and by game (Per Game) which is enough to conduct the requested analysis. However, since tables are not exactly identical (HTML tags), scraping them wasn't as easy as expected. Numerous specialties have had to be taken in consideration.

As an example, here is referenced a special case encountered in Totals and Per Game tables when the player qualified to All-Star games. Next to the Season value, a star appears. Below the HTML code corresponding to the Totals table of Kobe Bryant.

```
<td align="left" >
        <a href="/players/b/bryanko01/gamelog/1998/">1997-98</a>
        <span class="bold_text" style="color:#c0c0c0"> &#x2605;</span>
</td>
```

Scraping these web pages had been tricky in which the code should catch all these specialties. For further information about the code providing the player's statistics scraper, see the `player_statistics_BS.py` file.

The output tables could be find in Data/Part2/3b folder.

1.3.3 Question 3.c

Scraping the player's salaries tables require to scrap two main tables :

- Salaries (previous salaries)
- Contract (current and futures contracts if any)

Thus, the method is to merge these two tables. Here again, some specialties on tables structure render the scraping tricky. Indeed, Contract table doesn't have a thead, tbody and tfoot tag. So its scraping is done differently. Furthermore, salary's format is $15,000,000 for instance. Unchanged, salary cannot be loaded to the SQLite database. Dollar symbol and commas have been removed so that the new salary's format is 15000000 (Cleaning phase).

1.3.4 Question 3.d

1.3.1 can be repeated using Regex. I have written a sample skeleton code (`profile_parser_regex.py`) to parse the player profile using Regex. Using BS to parse seems to be easier because we can make use of existing methods to call various functions like `soup.find`, `soup.findNext` and `soup.get_text`. Using these functions make the code more readable and easily understandable. It allows for easy debugging should the program break due to an uncaught exception. Regex tends to be more rigid and inflexible. BeautifulSoup allows us to traverse the parse tree to look for specific objects at specific part of the HTML document. With regex, we have to search the entire document for each search item. We prefer to use BS to parse HTML and obtain the strings we are interested in. Then we can use regex to retrieve the information in those strings. We feel then that the best way is to use a combination of the two.

1.4 Question 4

For the same reasons as in 1.2, we use Beautifulsoup to scrap the franchise pages. More precisely, we use this package to navigate into the html tree, but we still need the regex package to make simple operations on string variables that can be found on the leaves of the tree.

1.4.1 Question 4.a

First of all, we want to scrap the basic team information. These are contained in the header of the franchise page. The html of this header can be found in the first `div` tag whose class is `mobile_text`. This tag is very badly organized because it consists in a succession of tags included in each other. Instead of having a list of sibling tags, all its successors are included in each other. Therefore, the Beautifulsoup code can be very tedious to write as each tag can only be reached by going down the hierarchy. Fortunately, we can avoid this problem with the function `get_text()` that automatically searches the text element in the html code and concatenate them. When we apply it to the header of the franchise page, we obtain a string containing all the basic information of the team. If we take the example of Atlanta Hawks franchise page, we get the following text :

Location Atlanta, Georgia

Team Names Atlanta Hawks, St. Louis Hawks, Milwaukee Hawks, Tri-Cities Blackhawks

Seasons 66 ; 1949-50 to 2014-15

Record 2584-2609, .498 W-L %

Playoff Appearances 43

Championships 1

Once we have this text, we just have to use the appropriate regex expressions to retrieve all the information we are interested in. In our project, we scrap them all, except the team names (we will scrap them in another table). You can notice that some pieces of information are not atomic. The `Location` field for example, contains two information : city and state. Thus, we split all information that are not atomic into several pieces. `Location` is split into city and state, `Seasons` into number of seasons, first season and last season and `Record` into number of wins, number of losses and win-loss percentage. We choose to identify a season with the civil year of the beginning of the season that is to say that the season 2014-15 is denoted as 2014. Besides, we have to choose a unique identifier for each franchise, since each one can have several names throughout the years. This id is a three letters abbreviation of the franchise name as it is used in the url of the pages of the franchise. For example, the abbreviation of Atlanta Hawks is `ATL` because the url of its page is `http://www.basketball-reference.com/teams/ATL/`.

All the basic information of all teams are scraped by the script named `team_scraper_BS` and they are stored in the same table named `teams_basic_info.csv`.

1.4.2 Question 4.b

Now we want to retrieve all the team statistics by season. These data are available on the same page as the team basic information and are already well-organized as we can find them in a table below the header. As this scraping job is very similar to what we did in 1.2, we will not describe it again. However, we can point out the fact that the name of the team can change throughout the seasons and that we don't use it as it is. Instead, we use a three letters abbreviation that can be found in the url pointing to the team roster of the corresponding season. The table containing the scraping result thus has both `Franchise_id` and `Team_id` columns. All statistics of all teams are scraped buy the script gathered in a csv file named `teams_statistics`. These data are not sufficient to answer all questions of Part 2. In order to get players experience, we also have to scrap rosters of all teams for each season. These data are available in `roster_statistics.csv`.

1.5 Question 5

We thought that some of the highest salaries may be explained by a phenomenon of exaggerated enthusiasm for specific players who are playing well during the regular season but either cannot get their team to the playoffs or perform poorly once in the playoffs. To investigate this hypothesis, we decided to scrap the playoff statistics, that is to say statistics of teams that reached the final part of the competition. We thought that, by focusing on the best teams for each season, we would be able to assess the players who perform well when it really matters (in the playoffs). These data were scraped on the page `http://www.basketball-reference.com/playoffs/` with the same technique as in 1.2 and 1.4, since they are also stored in a well-structured table on the website.

Our recommendation to team owners is to consider players that appear in this page differently than other players. Performance during the playoffs should be more valued than performance during the regular season. Also the owners should pay attention to discrepancies between regular season production and playoffs production. It would help them not to pay salaries that are not deserved (to player performing worse during the playoffs) and use this money for more efficient players (who perform well in the playoffs).

2 Part 2

2.1 Question 1

Below is the database structure of the differents tables. In total, there are 7 tables. The principal key for tables in figure 1, 2, 3 and 4 is `playerid`. For others tables, figure 5, 6 and 7, the principal key is `teamid`.

playerid	varchar(12)	`playerid` varchar(12) NOT NULL
name	varchar(56)	`name` varchar(56) NOT NULL
position1	varchar(2)	`position1` varchar(2)
position2	varchar(2)	`position2` varchar(2)
position3	varchar(2)	`position3` varchar(2)
shoots	char(1)	`shoots` char(1)
height	integer	`height` integer
weight	integer	`weight` integer
dob	date	`dob` date
city	varchar(56)	`city` varchar(56)
state	varchar(56)	`state` varchar(56)
experience	integer	`experience` integer
dod	date	`dod` date

FIGURE 1 – Profile table schema. *Source : SQLite Browser*

playerid	varchar(12)	`playerid` varchar(12) NOT NULL
season	integer	`season` integer
age	integer	`age` integer
team	char(3)	`team` char(3)
league	char(3)	`league` char(3)
position	varchar(6)	`position` varchar(6)
g	integer	`g` integer
gs	integer	`gs` integer
mp	integer	`mp` integer
fg	integer	`fg` integer
fga	integer	`fga` integer
fg_percent	double	`fg_percent` double
three_point	integer	`three_point` integer
three_point_a	integer	`three_point_a` integer
three_point_percent	double	`three_point_percent` double
two_point	integer	`two_point` integer
two_point_a	integer	`two_point_a` integer
two_point_percent	double	`two_point_percent` double
efg_percent	double	`efg_percent` double
ft	integer	`ft` integer
fta	integer	`fta` integer
ft_percent	double	`ft_percent` double
orb	integer	`orb` integer
drb	integer	`drb` integer
trb	integer	`trb` integer
ast	integer	`ast` integer
stl	integer	`stl` integer
blk	integer	`blk` integer
tov	integer	`tov` integer
pf	integer	`pf` integer
pts	integer	`pts` integer

FIGURE 2 – Player's Totals table schema *Source : SQLite Browser*

2.2 Question 2

Active players during 2011-2012 season According to the SQLite script `question_2.sql` in SQL directory, there are 452 actives players in 2011-2012 season among the 4 288 players in total.

Distribution in each position —- 20 C 91 C-PF 1 PF 90 PF-SF 1 PG 78 SF 86 SF-PF 1 SG 84

playerid	varchar(12)	`playerid` varchar(12) NOT NULL
season	integer	`season` integer
age	integer	`age` integer
team	char(3)	`team` char(3)
league	char(3)	`league` char(3)
position	varchar(6)	`position` varchar(6)
g	integer	`g` integer
gs	integer	`gs` integer
mp	double	`mp` double
fg	double	`fg` double
fga	double	`fga` double
fg_percent	double	`fg_percent` double
three_point	integer	`three_point` integer
three_point_a	double	`three_point_a` double
three_point_percent	double	`three_point_percent` double
two_point	integer	`two_point` integer
two_point_a	double	`two_point_a` double
two_point_percent	double	`two_point_percent` double
ft	double	`ft` double
fta	double	`fta` double
ft_percent	double	`ft_percent` double
orb	double	`orb` double
drb	double	`drb` double
trb	double	`trb` double
ast	double	`ast` double
stl	double	`stl` double
blk	double	`blk` double
tov	double	`tov` double
pf	double	`pf` double
pts	double	`pts` double

FIGURE 3 – Player's Per Game table schema *Source : SQLite Browser*

playerid	varchar(12)	`playerid` varchar(12) NOT NULL
season	integer	`season` integer
team	varchar(24)	`team` varchar(24)
league	char(3)	`league` char(3)
salary	integer	`salary` integer

FIGURE 4 – Salary table schema *Source : SQLite Browser*

playerID	TEXT	`playerID` TEXT
active	TEXT	`active` TEXT

FIGURE 5 – Active Players table schema *Source : SQLite Browser*

teamid	varchar(12)	`teamid` varchar(12) NOT NULL
city	varchar(24)	`city` varchar(24)
state	varchar(24)	`state` varchar(24)
season	integer	`season` integer
first_season	integer	`first_season` integer
last_season	integer	`last_season` integer
wins	integer	`wins` integer
losses	integer	`losses` integer
winlose_percent	double	`winlose_percent` double
playoff_app	integer	`playoff_app` integer
championships	integer	`championships` integer

FIGURE 6 – Team Information table schema *Source : SQLite Browser*

Average age avg age = 30.01

Average weight avg weight = 219.947

Average experience avg experience = 7.35

teamid	varchar(12)	`teamid` varchar(12) NOT NULL
season	integer	`season` integer
league	char(3)	`league` char(3)
team	varchar(24)	`team` varchar(24) NOT NULL
win	integer	`win` integer
loss	integer	`loss` integer
wl_percent	double	`wl_percent` double
finish	integer	`finish` integer
srs	double	`srs` double
pace	double	`pace` double
rel_pace	double	`rel_pace` double
ortg	double	`ortg` double
rel_ortg	double	`rel_ortg` double
drtg	double	`drtg` double
rel_drtg	double	`rel_drtg` double
playoffs	varchar(128)	`playoffs` varchar(128)

FIGURE 7 – Team Statistics table schema *Source : SQLite Browser*

Average salary in the season avg salary = 4699756.0

Average salary in the career avg career sal = 36147056.02

2.3 Question 3

2.3.1 Question 3.a

We interpreted this question as we needed to select a list of current active players who had a salary in 2011-2012 season, and list the top 10% from the reduced sample size. The query would have been this.

2.3.2 Question 3.b

Bottom 10% worst paid players who are currently active and had a salary from 2011-2012 season

2.3.3 Question 3.c

Middle 50% by pay of players who are currently active and had a salary from 2011-2012 season

2.3.4 Question 3.d

The metrics above can be represented in a cross-tabulation format through the use of the case statement. The query below is repeated for all the teams in 2011-2012 season and each result is concatenated using `UNION ALL`. The query below shows the average age where team name is the first column for each row and year is the column. The query is then repeated and an `UNION ALL` is performed to join each query. The process is repeated for the following teams (`ATL BOS CHA CHI CLE DAL DEN DET GSW HOU IND LAC LAL MEM MIA MIL`

2.4 Question 4

2.4.1 Question 4.a

2.4.2 Question 4.b

2.4.3 Question 4.c

2.5 Question 5

Conclusion