Terada Asavapakuna 1012869

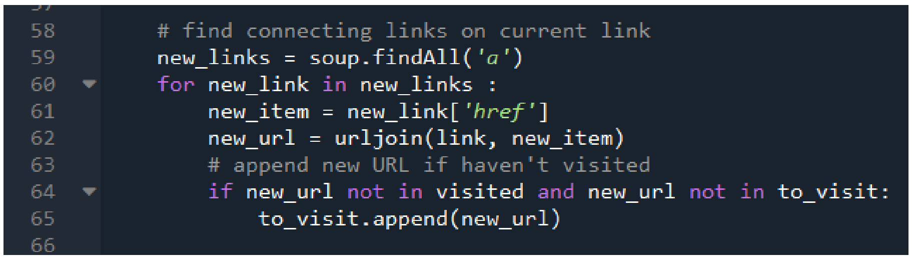
COMP20008 Project 1 Task 6 Report

Task 1 - Crawling Method

The data on each page is crawled by first starting on the main page: http://comp20008-jh.eng.unimelb.edu.au:9889/main/, which is our base url. After scanning the main page, the code looks for links on the page. If one or more links exist on the page, the crawler adds to a list of urls that needs to be visited (*to_visit*).

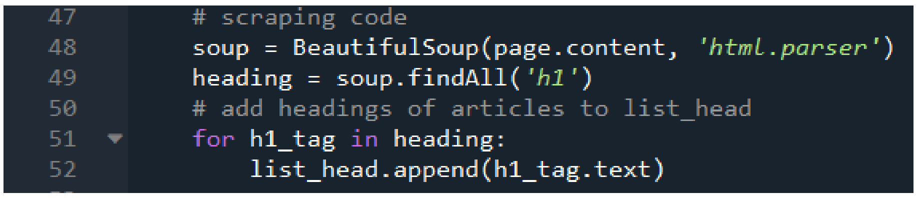
This is repeated in a while loop. As we crawl each page, we remove the link that we just visited in *to_visit* using pop, adding the popped link into our list of visited URLs and marking it as True (visited).

If more links are found within that link, it is checked if it is in our *visited* or *to_visit* list. If it already exists, we do not append it (we discard). Links are found on each page by using the function findAll from the Beautiful Soup library as shown:



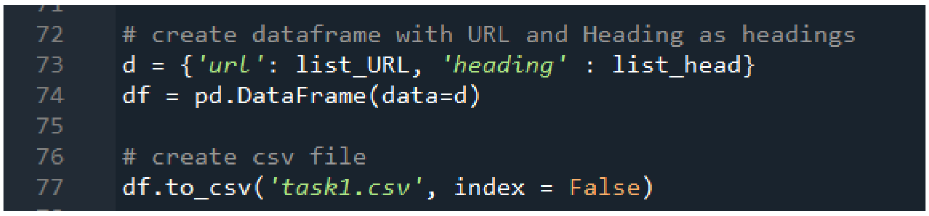
```
    # find connecting links on current link
    new_links = soup.findAll('a')
    for new_link in new_links :
        new_item = new_link['href']
        new_url = urljoin(link, new_item)
        # append new URL if haven't visited
        if new_url not in visited and new_url not in to_visit:
            to_visit.append(new_url)
```

When each page is visited, Beautiful Soup is used to extract the heading as shown below marked by the h1 tag:



```
    # scraping code
    soup = BeautifulSoup(page.content, 'html.parser')
    heading = soup.findAll('h1')
    # add headings of articles to list_head
    for h1_tag in heading:
        list_head.append(h1_tag.text)
```

When a list of URLs and headings is generated, a dataframe is created using the pandas library with appropriate headings and is output to a csv file.



```
# create dataframe with URL and Heading as headings
d = {'url': list_URL, 'heading' : list_head}
df = pd.DataFrame(data=d)

# create csv file
df.to_csv('task1.csv', index = False)
```

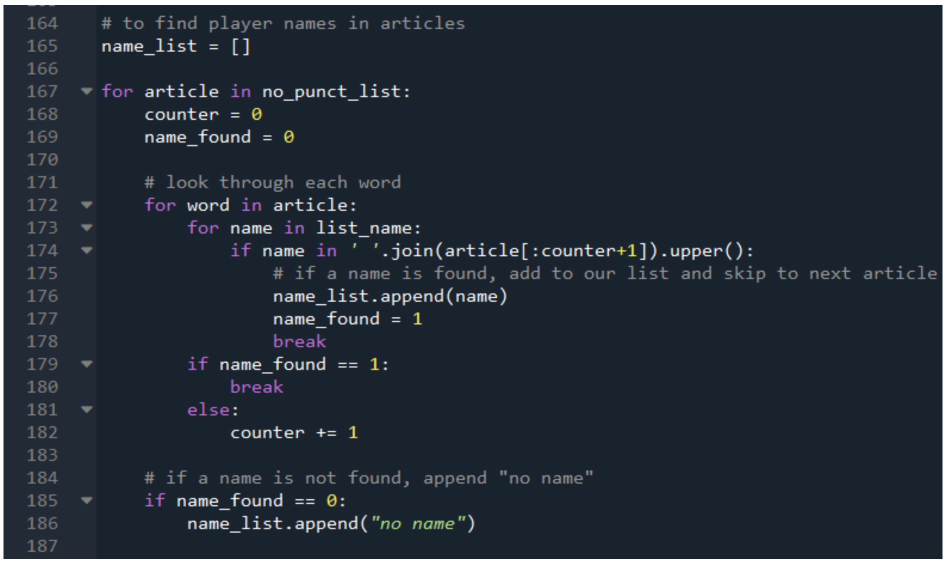
Task 2 - Scraping Data

Extracting player name:

Firstly, to know which names are valid, the JSON file is read and passed through a loop to find player names which is then appended to a list of full names.

Secondly, the findAll function from Beautiful Soup is again used to extract the paragraphs of each URL instead of article headings. The text “Previous Article” and “Next Article” from the bottom is removed (the last 4 indexes). After each text is extracted, the punctuation is also stripped as it can be included at the beginning or the end of our names as it will not match with the names given in the JSON file.

In addition, to detect a full valid player name, two for loops are used to loop through each word in each article. If a name is not found, the loop looks at the first up to the second word and so on (first to the third word) until a name is detected in that article. If it is detected it is appended into the list and break is used to terminate the loop to move on to the next article. If there is no name, ‘no name’ is appended to the list to conserve its article index position in the list.



```python
# to find player names in articles
name_list = []

for article in no_punct_list:
    counter = 0
    name_found = 0

    # look through each word
    for word in article:
        for name in list_name:
            if name in ' '.join(article[:counter+1]).upper():
                # if a name is found, add to our list and skip to next article
                name_list.append(name)
                name_found = 1
                break
        if name_found == 1:
            break
        else:
            counter += 1

    # if a name is not found, append "no name"
    if name_found == 0:
        name_list.append("no name")

```

Extracting first valid score:

The regular expression used:
(((6-[0-4]).)|((7-(6|5)).)|(([0-4]-6).)|(((6|5)-7).)|(\((\d{1,})(-|/)(\d{1,})\).)|((8|9|(\d\d))-([6-9]|(\d\d)))|(([6-9]|(\d\d))-(8|9|(\d\d)))){2,9}

Each set score is only valid if a side wins:
If the first player wins a set, the only valid scoring system would be: 6 - (0 to 4) or 7- (5 or 6)
If the second player wins a set, the valid scoring system would be the opposite: (0 to 4)-6 or (5 or 6)-7

There is a “.” at the end of each scoring in case there is punctuation next to it e.g. a full stop.
Tie-break scoring is recognised by using “\(“ and “\)” to determine starting and ending brackets, \d{1,} is used as tie-break scoring can go up as high as possible with a minimum of 1 digit. (-|/) is used as tie-break scoring sometimes uses slashes instead of hyphens.

Grand slam scores are recognised differently from sets, the last (5th) set could look like a set or be a lot higher. Therefore, if it is different the scoring would be higher than a 7, this means that it could be an 8, 9 or double digits (assuming it does not go up to 100) to [6 to 9] or double digits or the opposite (second player wins). The highest number of sets it can go up to is 5 including tie-breaks only for the first 4 sets, the last has no tie-break. Therefore {2,9} is used as the minimum number of sets played in a game is 2, and maximum 9 (5 sets + 4 tie-break)

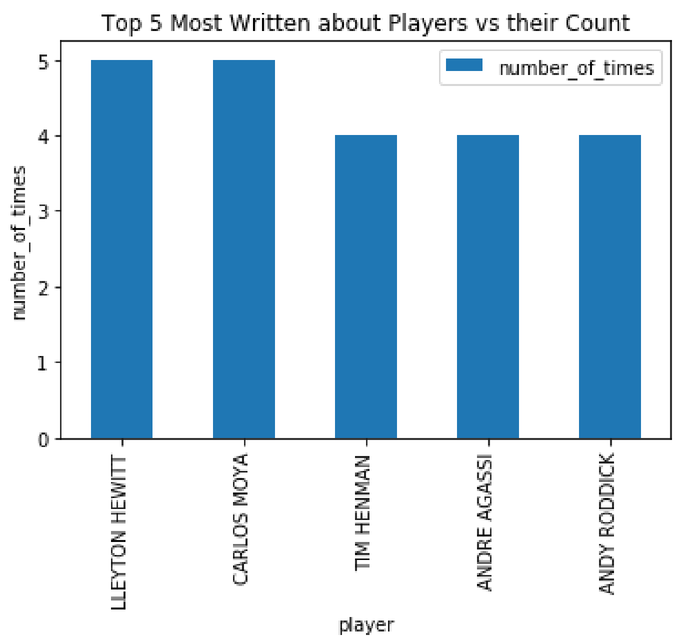
Output:

Once player names and first completed scores are extracted separately, to remove non-existing names and scores in articles. The index of the articles with non-existing names and scores are combined and removed from the list of URLs, headings, names and scores and then output to a new csv file.

Terada Asavapakuna 1012869

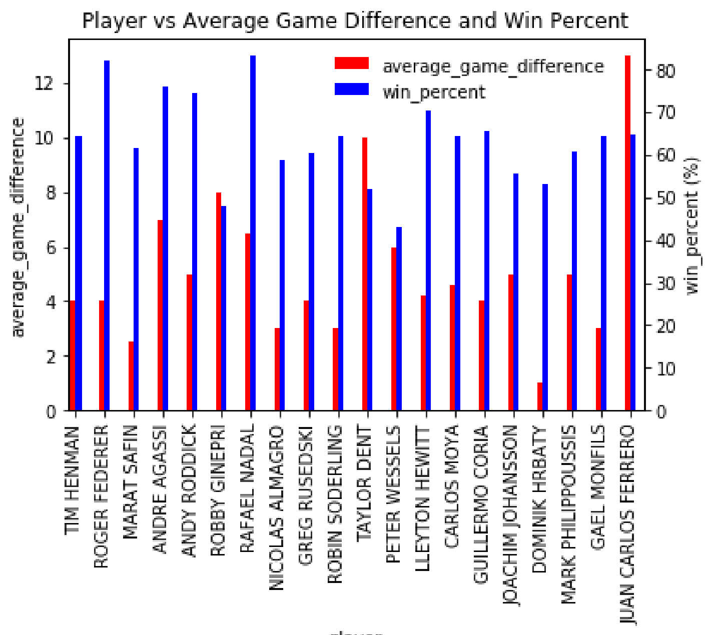
Task 4 & 5 - Analysis:

Task 4 bar graph:



The five most written about players (x) against the number of times the article is written about these five players (y).

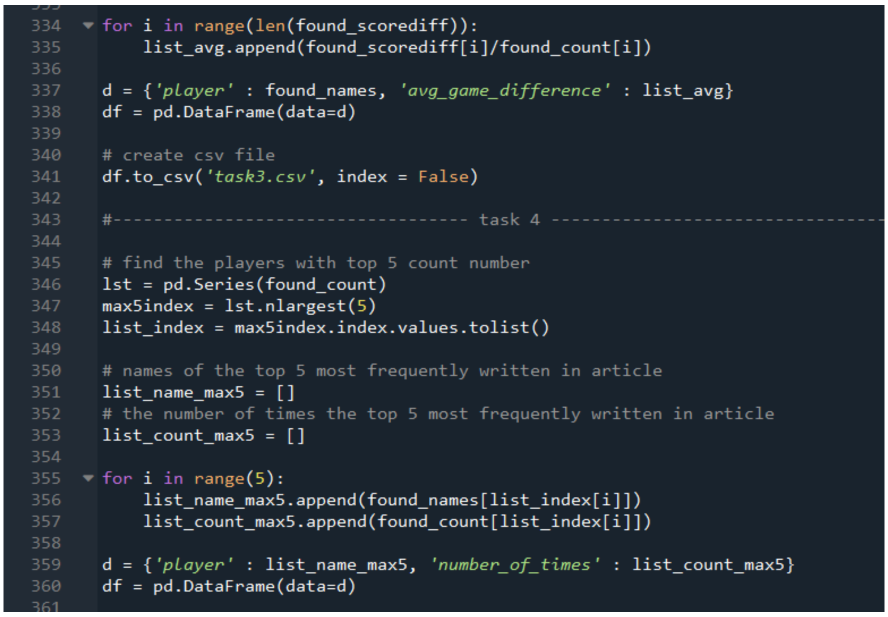
Task 5 double bar graph:



Players the article has mentioned (x) against their average game difference and win percentage on opposite y axes.

Task 4 - Data:

The bar graph uses the top five players mentioned and the number of times each, this is obtained in task 3 when keeping track of the total number of articles for each player (when calculating average game difference), the nlargest() function from pandas DataFrame is used to obtain the first n rows in descending order (when n = 5) of players (*found_names*) and the number of times (*found_count*). If the number of players is smaller than 5 then the code written will produce an error.



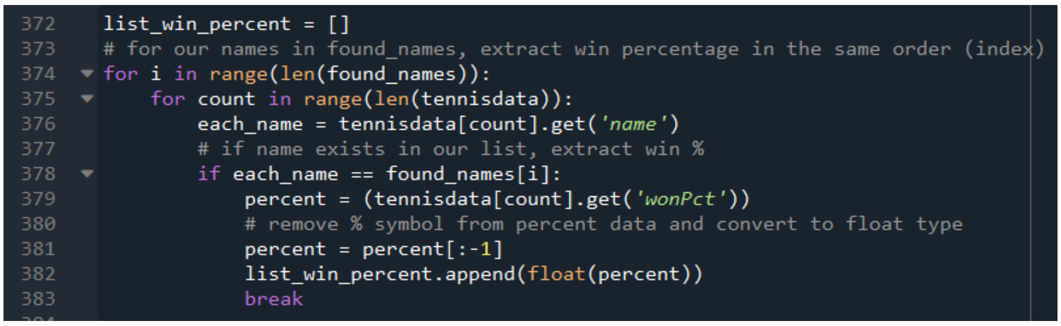
```
for i in range(len(found_scorediff)):
    list_avg.append(found_scorediff[i]/found_count[i])

d = {'player' : found_names, 'avg_game_difference' : list_avg}
df = pd.DataFrame(data=d)

# create csv file
df.to_csv('task3.csv', index = False)

#---------------------------------- task 4 -------------------------------

# find the players with top 5 count number
lst = pd.Series(found_count)
max5index = lst.nlargest(5)
list_index = max5index.index.values.tolist()

# names of the top 5 most frequently written in article
list_name_max5 = []
# the number of times the top 5 most frequently written in article
list_count_max5 = []

for i in range(5):
    list_name_max5.append(found_names[list_index[i]])
    list_count_max5.append(found_count[list_index[i]])

d = {'player' : list_name_max5, 'number_of_times' : list_count_max5}
df = pd.DataFrame(data=d)
```

Task 4 - Analysis:

According to the graph, the most common player mentioned amongst all articles is tied between Lleyton Hewitt and Carlos Moya at 5 times each in the list of articles given. The second most written about player is tied between Henman, Agassi and Roddick. However, the difference between the number of times a player is written about is not as significant, it does not show clearly who really is the most written about player as there is a lot of room for error (if there is in scraping data in the code). Therefore, a larger sample size would be preferred to make data clearer between each player.

Task 5 - Data:

The double y-axis bar graph uses the cleansed list of existing players (*found_names*) in the article from task 2 (where there is also a completed score in that article) and the list of average game difference (AGD) from task 3. Both lists are in the same article order (as index in each list). Lastly, the win percentage is extracted from the JSON file by comparing if the name in *found_names* matches each looped name in the file, if it does we can extract and append the win percentage to *list_win_percent* (removing the "%" symbol and converting it to a float) of that player.



```python
list_win_percent = []
# for our names in found_names, extract win percentage in the same order (index)
for i in range(len(found_names)):
    for count in range(len(tennisdata)):
        each_name = tennisdata[count].get('name')
        # if name exists in our list, extract win %
        if each_name == found_names[i]:
            percent = (tennisdata[count].get('wonPct'))
            # remove % symbol from percent data and convert to float type
            percent = percent[:-1]
            list_win_percent.append(float(percent))
            break
```

From the data in the graph, and the graph from task 4. There is no correlation between the most common written-about player and win percentage (or AGD). A player can be written about a high number of times, but it does not mean that they win many games. Also, we cannot infer much from the AGD in the graph as it does not tell us if the player has won or lost (by a lot or almost lost many games itself).

Appropriateness of selecting the first-named player:

Selecting the first-named player does not guarantee that the article is written about that player. For example, a tennis article could write about scores of all players in the first round of tennis, not specifically focusing on a single player. This affects the AGD as the wrong count (*found_count*) is used for that player.

Also, the first completed match score does not mean that it refers to that player. For example, an article that has the sentence: "Roger Federer was injured leaving the match last week at 6-1 4-5 which secured Rafael Nadal to advance to next round claiming victory over Novak Djokovic at 6-4 7-5". In this example, Roger Federer would be extracted along with the score "6-4 7-5" in which he did not play. This also affects the AGD as Federer did not win with a score of 6-4 7-5 in my example.

Method of figuring out whether the first-named player won or lost:

Instead of finding the absolute value of the AGD, remove the abs() function. If the AGD is positive the first-named player won, if it is a negative they lost. However, this does not guarantee the first completed match score corresponds to the player as well as the loss (if negative AGD) or win (if positive AGD).

Other useful information to extract from articles:

The date or year of which the article was written could also be extracted to know which article is relevant in our data collection, that is if we would like to look at a specific year. An article too old could result in inaccurate and possibly invalid scores. Also, if an article is too old, there could be articles written about a retired player or no articles written about a new player (this affects the popularity of players written in articles in task 4).

This could be extracted by looking through articles with words that consist of four integers that start with "20.." or "19.." or something more specific eg. the year "2004". This could be achieved by using regular expressions like "20\d\d".

By extracting articles relevant to a specific year, we can better understand player performance as we can look at how many games the player has won or lost in that year. So that we can keep track if a player needs to undergo more training if their AGD is negative. Also, we can even predict who will win the next tournament if their AGD is positive and high.