

DATA COLLECTION AND PREPROCESSING – GROUP ASSIGNMENT

DOMAIN – CRICKET

GROUP MEMBERS:

- Archit Srivastava (12120052)
- Akhil Reddy (12120022)
- Jatin Sahnani (12120098)
- Pritam Ghosh (12120071)
- Rohan Kumar (12120026)

Executive Summary

Cricket is touted as one of the major world sports in terms of participants, spectators, and media interest. Originating in England, cricket has become hugely popular in South Asian countries, South Africa, Australia, New Zealand, and West Indies, most of them being former British colonies or still under the Crown's influence. (Kumar, 2018)

Since its inception, Cricket has evolved into many formats. The last decade has seen a rise in shorter versions of the game giving rise to international tournaments like the Twenty20 World Cup (2007), the official Indian Premier League (2008) and the Cricket Champion League (2009). (Chelniciuc, 2009) With the increased popularity and tremendous developments, especially in terms of the rise in new talent, Cricket today has become a major attraction, whose performance in all its aspects is an important phenomenon to watch and analyze. As a consequence, there has been an increase in developing of applications and programs that monitor the performance of players in Cricket. (Kumar, 2018)

This project has been undertaken as a part of the Data Cleaning and Pre-processing group assignment. The goal is to scrape through websites pertaining to the data of Cricketers and form a dataset which can be further utilized to perform data analysis and predictive analysis on the players to select them based on their performance and eligibility criteria.

Problem Statement

There has been a huge influx of new and raw talent in Cricket which makes it difficult to keep an eye on all the emerging talent across Countries to watch out for players who are performing well across all formats. The availability of appropriate data to check and benchmark players in terms of formats, consistency, batting order of batsmen, choice of bowler for death overs etc. is always a challenge. Many a time there are players who perform well in the short formats but are inconsistent in longer formats like Test matches and ODIs. There are often times when a player performs well in longer formats of the game but falls short in the growing tournaments like Twenty20. Therefore, the most decisive task is to select the players based on individual performance. Since the data of all the players is very scattered and unorganized, it is difficult to look at the overall picture and compare Batsmen/Bowlers with each other and across formats.

Solution

This project aims to bring all the unstructured and structured data together to check the overall performance and fit for all the players of the current era and compare it with all the Cricketers who have played the game since its inception. This would give us an idea of how to set the benchmarking for current players based on the performance of the veterans and the contemporaries of the game. With the help of our dataset, selectors of any new tournament can look at numerous factors such as past records, current form, batting/bowling average, strike rate, country of the player, etc. Based on this, they would be able to analyze the performance of players of the last decade who have been exposed to all three formats of the game and see how they fare against the masters. To extend the relevance further, crowdsourcing can be done to gather more information about the available data and try predictive analysis.

Challenge

Even though the amount of data available for Cricket is mammoth, adequate data for analysis lacks obtainability. Our major source of gathering unstructured data was espncriinfo.com which has the combined data of all the major players across all formats and tournaments. Since this is the most trusted and reliable website amongst all Cricket enthusiasts, we chose it as our primary web scraping access.

The structured data available to us had a lot of attributes which could not be used but while comparing it with espncriinfo.com, it gave us a broad outline of what our final dataset should look like.

We used Python (BS4 library - BeautifulSoup) to crawl/scrape the data from the website and then used many function techniques to refine the data and clean it to make it comprehensible. Primarily, we needed to first make individual files for Batsmen and Bowlers across 3 formats. Then merge them together, remove duplicates and take care of null values.

Finally, we were able to make a proper structure with unique players and important attributes which can be used for further analysis. Predictive models can be applied to the dataset to find out the 'Best Performers', 'Average players' and 'Needs improvement' so that selection becomes a lot easier.

Let us briefly look at the complications and hurdles we had to overcome so that our final dataset is comprehensible.

- Upon first trying to capture the data, we got some unnecessary table headers which were getting difficult to remove. After further brainstorming, we understood the topology of the website which enabled us to extract table data(td) and table rows(tr) from table body (tbody) and store it in a dictionary format.
- It was imperative to store our data in a dictionary so that the data frame could be imported into an excel sheet.
- Our data was a composition of Batters and Bowlers – for 3 different formats of Cricket. Understandably, there were Cricketers who had played one type of format and not the other format. Their attributes were displayed as '-' so we had to replace them with 0 for consistency.
- In our extracted dataset, the player Name had country and name of Board (if they had played for ICC tournaments) and their numbers of ICC tournaments. We narrowed down the column to only have the player's name. Simultaneously we created another column with only a Country name so that this can also be an added attribute for further analysis.
- The data types of each column were in 'object' format. We changed most of the columns to 'int' format apart from columns where 'float' or 'object' could be better suited.
- There were a few players who had similar names but were from different countries or the same player who had played for a different country. We decided to keep them all together till we merged data as it was required to keep all players intact.
- For certain players where the span of Cricketers was not available (for a certain format of the game like R Powell of WI with Test Cricket Span missing), it was evident that these players had not played those games. For such cases, we have considered 0 in the fields.

Chosen domain and Seed sources

We chose the domain of Cricket as it was a common interest amongst all group members.

For the seed sources, we primarily had both unstructured and structured data filed for this domain. This helped us understand the type of data that is already available and gave us a direction to work upon. The current files did not have concise data which is why we decided to crawl and scrape our data from Cricket websites and form a dataset of >5000 Cricketers with >70 important attributes. We looked through many websites which have data about Cricketers and checked what all attributes were present. Websites like espncriinfo.com, howstat.com, icc-cricket.com, and cricmetric.com all had various formats and attribute available but it would have been very difficult to extract data from all these websites separately and then merge them together. We found it interesting that www.espncriinfo.com had all the usable attributes for our data collection process. This compiled data would be perfect for analysis and prediction and could be used by selectors for any new tournament.

The structured and unstructured sources from open domain/ internal sources

Initially, we searched various sources from where we can get data related to cricketers. Our objective was to collect data for various attributes in different formats of cricket. We searched various websites from where data could be fetched and is reliable. We chose ESPNcriinfo as the primary and final source. It is a leading cricket website amongst the top five single-sport websites in the world. Also, it has around 10 million unique users every month, and users spend more time on ESPNcriinfo than almost any other sports site. Thus, reliability and authenticity of the data source was not an issue when we chose ESPNcriinfo.

A preliminary exploration of data found that it had almost every data point we required to collect about cricketers in Test, ODI & T20 for batsmen & bowlers. As most of the data was present in tabular format on the website it was an efficient method to scrap from the source.

We used the below links of ESPNcriinfo to get the data.

T20 Batsmen data:

<https://stats.espncriinfo.com/ci/engine/stats/index.html?class=3;template=results;type=batting>

T20_Bowling data:

<https://stats.espncriinfo.com/ci/engine/stats/index.html?class=3;template=results;type=bowling>

ODI Batsmen data:

<https://stats.espncriinfo.com/ci/engine/stats/index.html?class=2;template=results;type=batting>

ODI Bowling data:

<https://stats.espncriinfo.com/ci/engine/stats/index.html?class=2;template=results;type=bowling>

Test Batsmen data:

<https://stats.espncriinfo.com/ci/engine/stats/index.html?class=1;template=results;type=batting>

Test Bowling data:

<https://stats.espncriinfo.com/ci/engine/stats/index.html?class=1;template=results;type=bowling>

Downloading/ crawling/ collection of data from all the sources: The methods used for this process and the challenges faced.

The tools we have used in our project are – Python, BeautifulSoup, requests, SweetViz and Pandas. And XML, Excel & JSON (all for output).

Our main challenge was to get all the fields & complete data of each page from the website. We started with the ipynb file “Cricinfo data fetch-v1”, to fetch the data from the ESPN Cricinfo website.

We tried fetching the first 2 pages & were able to fetch the data from the webpage. But there was a clear error in how we approached our scraping process as the variables and column headers were not extracted consistently through the pandas' data frame. We tried the iloc/loc function, passing header = “none” while reading the CSV file and finally used `df.drop([1],axis=1,inplace=True)` but none of them had an answer to the formatting error we witnessed.

We were getting “Overall figures” above our dataframe and it was really messing up our table. Below is how it looked after fetching the table from the website.

```
df = pd.read_csv('TestSta.csv',sep='\t')
```

df

Overall figures										
Player	Span	Mat	Inns	NO	Runs	HS	Ave	100.0	50.0	0.0
SR Tendulkar (INDIA)	1989-2013	200	329	33	15921	248*	53.78	51.0	68.0	14.0
BT Bortone (AUS)	1995-2012	168	287	20	12278	257	54.85	44.0	62.0	17.0

```
In [74]: df.columns
```

```
Out[74]: Index([' ', ' Overall figures '], dtype='object')
```

```
error- '[1] not found in axis'
```

```
df.drop(["Overall Figures"],axis=0,inplace=True)
error-Expected tuple, got str
```

```
df.drop(["Overall Figures"],axis=1,inplace=True)
Error- "['Overall Figures'] not found in axis"
```

Few photos of errors and formatting inconsistencies we got back.

Overall figures													
Player	Span	Mat	Inns	NO	Runs	HS	Ave	BF	SR	100.0	50.0	0.0	
SR Tendulkar (INDIA)	1989-2012	463	452	41	18426	200*	44.83	21368	86.23	49.0	96.0	20.0	
KC Sangakkara (Asia/ICC/SL)	2000-2015	404	380	41	14234	169	41.98	18048	78.86	25.0	93.0	15.0	
RT Ponting (AUS/ICC)	1995-2012	375	365	39	13704	164	42.03	17046	80.39	30.0	82.0	20.0	
ST Jayasuriya (Asia/SL)	1989-2011	445	433	18	13430	189	32.36	14725	91.20	28.0	68.0	34.0	
...
MG Bevan (AUS)	1994-2004	232	196	67	6912	108*	53.58	9320	74.16	6.0	46.0	5.0	
G Kirsten (SA)	1993-2003	185	185	19	6798	188*	40.95	9436	72.04	13.0	45.0	11.0	
A Flower (ZIM)	1992-2003	213	208	16	6786	145	35.34	9097	74.59	4.0	55.0	13.0	
Shakib Al Hasan (BAN)	2008-2022	221	209	30	6755	134*	37.73	8212	82.25	9.0	50.0	12.0	
IVA Richards (WI)	1975-1991	187	167	24	6721	189*	47.00	7451	90.20	11.0	45.0	7.0	

1299 rows × 2 columns

```
df.drop(df.columns, axis = 1)
```

Player	Span	Mat	Inns	NO	Runs	HS	Ave	BF	SR	100.0	
SR Tendulkar (INDIA)	1989-2012	463	452	41	18426	200*	44.83	21368	86.23	49.0	

We used `df[df.columns[1]]`, it removed “Overall Figures” along with “50” & “0”

We later dug deeper to understand the topology of a webpage and how tables in an HTML format work. We were able to extract table data(td) and table rows(tr) from table body (tbody) and store it in a dictionary format.

HTML Table Structure

	TD	TD	TD
TR	data	data	data
TR	data	data	data
TR	data	data	data

The Foundational Revised Approach:

Step 1: We chose the URL.

Step 2: We used the most common method of retrieving the webpage URL by using “requests.get”

Step 3: We retrieved the content of the URL using lxml as a parser. We used ‘lxml’ as it can easily help in both XML and HTML in our web scraping task.

Step 4: We used `soup.find_all('table')` to find the tables. It was under class = “engineTable”.

```
Classes of each table:
['engineTable']
['engineTable']
['engineTable']
['engineTable']
['engineTable']
['engineTable']
None
None
```

Step 5: We selected the second engine table as that’s where our data was to be extracted from

```

for name in Player_table.find_all('tbody'):
    rows = name.find_all('tr')
    for row in rows:
        Player = row.find('td',class_ = 'left').text
        Span = row.find_all('td')[1].text
        Mat = row.find_all('td')[2].text
        Inns = row.find_all('td')[3].text
        Overs = row.find_all('td')[4].text
        Mdns = row.find_all('td')[5].text
        Runs = row.find_all('td')[6].text
        Wkts = row.find_all('td')[7].text
        BBI = row.find_all('td')[8].text
        Ave = row.find_all('td')[9].text
        Econ= row.find_all('td')[10].text
        SR=row.find_all('td')[11].text
        fourwickets = row.find_all('td')[12].text
        fivewickets = row.find_all('td')[13].text

```

Our code which extracts every variable

Step 6: `soup.find_all("table",class_ = 'engineTable')..` We created a variable

“Player_table” here where we are getting the table for class ‘enginetable’ complete data.

```

#Print variable,concentrate/
Player_table = soup.find_all("table",class_ = 'engineTable')[2]
#print(soup.prettify()) #checking the data
print(Player_table)

```

```

<thead>
<tr class="headlinks">
<th class="left" nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=player;template=results;type=bowling" title="sort by player name">Player</a></th>
<th class="left" nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=start;template=results;type=bowling" title="sort by start date">Span</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=matches;template=results;type=bowling" title="sort by matches played">Mat</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=innings_bowled;template=results;type=bowling" title="sort by innings bowled in">Inns</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=overs;template=results;type=bowling" title="sort by overs bowled">Overs</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=maidens;template=results;type=bowling" title="sort by maidens earned">Mdns</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=conceded;template=results;type=bowling" title="sort by runs conceded">Runs</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=wickets;orderbyad=reverse;template=results;type=bowling" title="sort by wickets taken">Wkts</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=bbi;template=results;type=bowling" title="sort by best bowling in an innings">BBI</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=average;template=results;type=bowling" title="sort by average">Ave</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=economy;template=results;type=bowling" title="sort by economy">Econ</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=strike_rate;template=results;type=bowling" title="sort by strike rate">SR</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=fourwickets;template=results;type=bowling" title="sort by four wickets">fourwickets</a></th>
<th nowrap=""><a class="black-link" href="/ci/engine/stats/index.html?class=3;filter=advanced;orderby=fivewickets;template=results;type=bowling" title="sort by five wickets">fivewickets</a></th>

```

Conversion of data from original sources to structured data fields

After making the template to extract data from the table we created a dictionary to save the data as it would be easier to convert it to a data frame.

```
Cricketers =  
|  
{ 'Player':Player, "Span":Span, "Matches":Mat, "Innings":Inns, "Ov  
ers":Overs, "Mdns":Mdns, "Runs_given":Runs, "Wkts":Wkts,  
"BBI":BBI, "AVG":Ave, "Econ":Econ, "Strike_rate":SR  
, "4":fourwickets, "5":fivewickets}
```

The dictionary we created

```
{ 'Player': 'Shakib Al Hasan (BAN)', 'Span': '2006-2022', 'Matches': '96', 'Innings': '95', 'Overs': '354.3', 'Mdns': '3', 'Runs  
_given': '2366', 'Wkts': '119', 'BBI': '5/20', 'AVG': '19.88', 'Econ': '6.67', 'Strike_rate': '17.8', '4': '5', '5': '1'}  
{ 'Player': 'TG Southee (NZ)', 'Span': '2008-2021', 'Matches': '92', 'Innings': '90', 'Overs': '332.5', 'Mdns': '2', 'Runs_give  
n': '2729', 'Wkts': '111', 'BBI': '5/18', 'AVG': '24.58', 'Econ': '8.19', 'Strike_rate': '17.9', '4': '1', '5': '1'}  
{ 'Player': 'SL Malinga (SL)', 'Span': '2006-2020', 'Matches': '84', 'Innings': '83', 'Overs': '299.5', 'Mdns': '1', 'Runs_give  
n': '2225', 'Wkts': '107', 'BBI': '5/6', 'AVG': '20.79', 'Econ': '7.42', 'Strike_rate': '16.8', '4': '1', '5': '2'}  
{ 'Player': 'Rashid Khan (AFG/ICC)', 'Span': '2015-2022', 'Matches': '58', 'Innings': '58', 'Overs': '219.2', 'Mdns': '1', 'Runs  
_given': '1357', 'Wkts': '105', 'BBI': '5/3', 'AVG': '12.92', 'Econ': '6.18', 'Strike_rate': '12.5', '4': '4', '5': '2'}  
{ 'Player': 'Shahid Afridi (ICC/PAK)', 'Span': '2006-2018', 'Matches': '99', 'Innings': '97', 'Overs': '361.2', 'Mdns': '4', 'Ru  
ns_given': '2396', 'Wkts': '98', 'BBI': '4/11', 'AVG': '24.44', 'Econ': '6.63', 'Strike_rate': '22.1', '4': '3', '5': '0'}  
{ 'Player': 'Mustafizur Rahman (BAN)', 'Span': '2015-2022', 'Matches': '63', 'Innings': '63', 'Overs': '225.4', 'Mdns': '6', 'Ru  
ns_given': '1710', 'Wkts': '87', 'BBI': '5/22', 'AVG': '19.65', 'Econ': '7.57', 'Strike_rate': '15.5', '4': '3', '5': '1'}  
{ 'Player': 'Saeed Ajmal (PAK)', 'Span': '2009-2015', 'Matches': '64', 'Innings': '63', 'Overs': '238.2', 'Mdns': '2', 'Runs_giv  
en': '1516', 'Wkts': '85', 'BBI': '4/19', 'AVG': '17.83', 'Econ': '6.36', 'Strike_rate': '16.8', '4': '4', '5': '0'}  
{ 'Player': 'Umar Gul (PAK)', 'Span': '2007-2016', 'Matches': '60', 'Innings': '60', 'Overs': '200.3', 'Mdns': '2', 'Runs_give  
n': '1443', 'Wkts': '85', 'BBI': '5/6', 'AVG': '16.97', 'Econ': '7.19', 'Strike_rate': '14.1', '4': '4', '5': '2'}  
{ 'Player': 'IS Sodhi (NZ)', 'Span': '2014-2021', 'Matches': '66', 'Innings': '64', 'Overs': '226.3', 'Mdns': '0', 'Runs_given':  
'1824', 'Wkts': '83', 'BBI': '4/28', 'AVG': '21.97', 'Econ': '8.05', 'Strike_rate': '16.3', '4': '2', '5': '0'}
```

The output

The data to be collected was from 55 pages, therefore we wrote a code involving an f-string with a combination of for loop to retrieve all the pages in the same format. The first page showed below output:

stats.espncricinfo.com/ci/engine/stats/index.html?class=3;filter=advanced;orderby=runs;template=results;type=bowling

icinfo

Live Scores

Series

Teams

News

Features

Videos

Stats

Tests

ODIs

T20Is

All Test/ODI/T20I

- other -

Batting

Bowling

Fielding

All-round

Partnership

Team

Umpire and referee

Aggregate/overall

View overall figures [change view]

Ordered by wickets taken (descending)

Page 1 of 55

Showing 1 - 50 of 2742

First

Previous

Next

Last

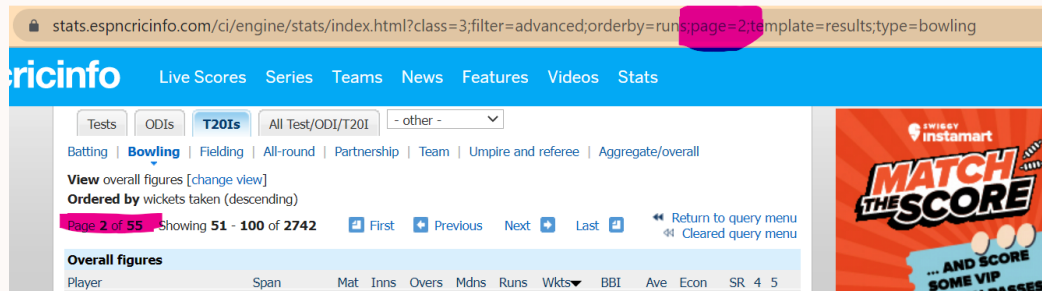
Return to query menu

Cleared query menu

Overall figures

Player	Span	Mat	Inns	Overs	Mdns	Runs	Wkts	BBI	Ave	Econ	SR	4	5
Shakib Al Hasan (BAN)	2006-2022	96	95	354.3	3	2366	119	5/20	19.88	6.67	17.8	5	1

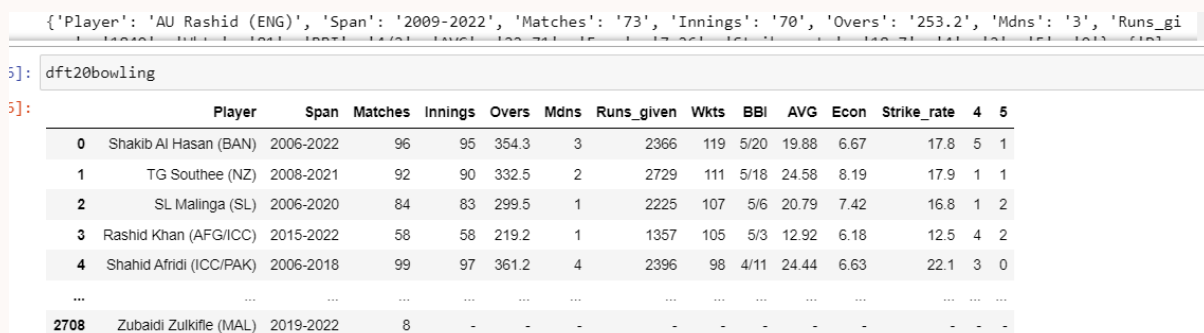
On clicking next page, we could see that page is showing page=2.



Thus created a loop with page={page} in url.

```
for page in range(1,56):
    source =
    requests.get(f'https://stats.espncricinfo.com/ci/engine/stats/index.html?
class=2;filter=advanced;orderby=runs;page=
{page};template=results;type=batting')
    soup = BeautifulSoup(source.content,'lxml')
    Player_table = soup.find_all("table",class_ = 'engineTable')[2]
```

Looping over pages



	Player	Span	Matches	Innings	Overs	Mdns	Runs_given	Wkts	BBI	AVG	Econ	Strike_rate	4	5
0	Shakib Al Hasan (BAN)	2006-2022	96	95	354.3	3	2366	119	5/20	19.88	6.67	17.8	5	1
1	TG Southee (NZ)	2008-2021	92	90	332.5	2	2729	111	5/18	24.58	8.19	17.9	1	1
2	SL Malinga (SL)	2006-2020	84	83	299.5	1	2225	107	5/6	20.79	7.42	16.8	1	2
3	Rashid Khan (AFG/ICC)	2015-2022	58	58	219.2	1	1357	105	5/3	12.92	6.18	12.5	4	2
4	Shahid Afridi (ICC/PAK)	2006-2018	99	97	361.2	4	2396	98	4/11	24.44	6.63	22.1	3	0
...
2708	Zubaidi Zulkifle (MAL)	2019-2022	8	-	-	-	-	-	-	-	-	-	-	-

We converted our list of dictionaries into a data frame

Data cleaning/pre-processing

We had two major tasks to be completed:

- (A) To clean & pre-process the data for each page of Test batsmen, Test bowlers, ODI batsmen, ODI bowlers, T20 batsmen & T20 bowlers. We made individual ipynb files for this.
- (B) To combine all the above 6 data frames created above. We have got 1 ipynb file.

Task A:

- 1) Columns for each data frame are different because we had multiple game formats (T20,Test,ODI). Independent files have been uploaded to the GitHub link for viewing.
- 2) Data fetched had ' - ' values signifying the data was missing because it was not collected or the player hadn't clocked in those game stats in real life. We replaced all these values with '0' to make them eligible for analysis.
- 3) Upon checking the dtypes of each column it was found all were in 'object' format. We changed most of the columns to 'int' format apart from bowling average, economy & SR as the float is better suited for those variables. We also didn't change the dtype of Player, Span, Overs & Best Figure as the same cannot be categorized in either float or int.
- 4) However, as the title/column is different in all 6 data frames (as can be checked in each ipynb file created), our choices of int, float & no actions depend on the data present in the title.
- 5) In the next step, we wanted to see the names of teams/countries present with each player. We created a new variable for data frame to check the same.
- 6) The major objective here is to get the country/team unique as each player can be mapped accordingly in the final sheet.
 - If Ricky Pointing data is shown Ricky Pointing (AUS) in the ODI data frame & Ricky Pointing (ICC/Aus) in another data frame, combining both, would have led to duplicate entries. Thus, we identified the unique Country/team & removed ICC/, /ICC, Asia/, /Asia, Afr/, /Afr, /World from Player columns.

```
array(['BAN', 'NZ', 'SL', 'AFG/ICC', 'ICC/PAK', 'PAK', 'ENG', 'WI', 'IRE',  
      'AFG', 'ICC/NEPAL', 'AUS', 'INDIA', 'SA', 'OMA', 'SA/World',  
      'SCOT', 'UAE', 'NED', 'NEPAL', 'WI/World', 'ENG/IRE', 'NED/SA',  
      'ICC/SL/World', 'KENYA', 'HKG', 'NAM', 'QAT', 'ENG/ITA', 'UGA',  
      'PNG', 'MLT', 'ZIM', 'MAL', 'NAM/SA', 'ICC/NZ', 'JER', 'CAN',  
      'AUS/NED', 'MWI', '3', 'USA/WI', 'KUW', 'GER', 'VAN', 'SGP', 'BOT',  
      'BUL', 'ROM', 'LUX', 'NGA', 'SLE', 'SA/USA', 'Mald', 'Arg',  
      'AFG/GER', 'Fin', 'Belg', '1', 'CZK-R', 'BMUDA', 'DEN',  
      'ENG/World', 'RWN', 'USA', 'ESP', 'Saudi', 'BHR', 'GUE', 'TAN',  
      'Ghana', 'Aut', 'NZ/World', 'GIBR', 'NOR', 'ITA', 'MOZ', 'EST',  
      'GRC', 'SWE', 'ENG/ICC', 'Caym', 'HUN', 'Bhm', 'CYP', 'SUI', 'IOM',  
      'PORT', 'THAI', 'Mex', 'PNM', 'Blz', 'Peru', 'Chile', 'DEN/ENG',  
      '', 'LES', 'CAM', 'Fran', 'SEY', 'PHI', 'Samoa', 'HKG/NZ', 'TKY',  
      'CRC', 'SRB', 'AUS/World', 'BHU', '2', 'Iran', 'ICC/INDIA',  
      'AUS/ICC/NZ', 'BAN/ICC/World'], dtype=object)
```

We used regex to remove these options from column.

```
#2nd replacing ICC/World/Asia/Afr from the country name
```

```
df_t20= df_t20.replace('ICC/', '', regex=True)
df_t20= df_t20.replace('/ICC', '', regex=True)
df_t20= df_t20.replace('/Afr', '', regex=True)
df_t20= df_t20.replace('Afr/', '', regex=True)
df_t20= df_t20.replace('/Asia', '', regex=True)
df_t20= df_t20.replace('Asia/', '', regex=True)
df_t20= df_t20.replace('/World', '', regex=True)
```

One of the formatting inconsistencies we had to deal with was removing the numbers and brackets after the player names as shown below. On the left were the inconsistent rows and on the right is the code to fix them. This was again a major task because when I removed brackets with `str.replace(r"^\([^\)]*\)", "")` & got the values in new column by `combine7['Country/Teams']=combine7['Player'].apply(lambda st: st[st.find("(")+1:st.find(")"]])`, the ['Country/Teams'] was having numerical digits & the country PAK, DEN got removed from the data. We only needed to remove the (1),(2),(3) & keep the country

We removed the same by using below commands.

	Player
13164	Rameez Raja (2) (PAK)
13365	Aftab Ahmed (2) (DEN)
15702	Aftab Ahmed (2) (DEN)
16841	Rameez Raja (2) (PAK)

```
#1st removing (numeric data)
```

```
df_t20['Player']=df_t20['Player'].str.replace(r"^\(.1?\)", "")
df_t20['Player']=df_t20['Player'].str.replace(r"^\(.2?\)", "")
df_t20['Player']=df_t20['Player'].str.replace(r"^\(.3?\)", "")
```

- 7) We didn't split the country data in each ipynb file. The main idea behind this was that if were to remove them and at the time of combining if 2 players were having the same name but played from a different country, accurate data would not get compiled. Thus, to maintain the unique player name, the country is kept in each ipynb file.

Task B:

- 1) We made an excel file with the heading "Player" only, no data. We saved it as "outputdf". We imported all excel as pandas data frames with different variables names.

```
import pandas as pd
```

```
#exporting all the files created from our codes of webscrapping
```

```
df1 = pd.read_excel(r'C:\Users\archi\Test battingv3.xlsx', index_col=0)
df2 = pd.read_excel(r'C:\Users\archi\Test bowlingv3.xlsx', index_col=0)
df3 = pd.read_excel(r'C:\Users\archi\ODI Batstmentv3.xlsx', index_col=0)
df4 = pd.read_excel(r'C:\Users\archi\ODI bowlingv3.xlsx', index_col=0)
df5 = pd.read_excel(r'C:\Users\archi\T20_Battingv3.xlsx', index_col=0)
df6 = pd.read_excel(r'C:\Users\archi\T20_Bowlingv3.xlsx', index_col=0)
```

```
outputdf=pd.read_excel(r'C:\Users\archi\finaloutputv1.xlsx')
```

Importing all data frames to combine

- 2) Then we took 'Player' columns from each data frame to get unique players from 6 complete data frames. We combined data frames based on unique player names. There were approximately 17000 players and upon removing duplicates we got 5733 'Player' unique values.
- 3) Merging each file was a major trouble. We used 'join', but it didn't work. The problem occurred was that we got 5797 rows. However, our unique rows were 5733.

```
In [81]: #Combining the 5th file
combine5=combine4.merge(df6,on='Player',how='left')
combine5
```

Out[81]:

	Player	TEST_Span	TEST_Matches_x	TEST_Innings_x	TEST_Not-Outs	TEST_RunsMade	TESTHighest-Score	TEST_Batting_Average	TEST_Hundereds	TEST_Fift
0	SR Tendulkar (INDIA)	1989-2013	200.0	329.0	33.0	15921.0	248*	53.78	51.0	6
1	RT Ponting (AUS)	1995-2012	168.0	287.0	29.0	13378.0	257	51.85	41.0	6
2	JH Kallis (SA)	1995-2013	166.0	280.0	40.0	13289.0	224	55.37	45.0	5
3	R Dravid (INDIA)	1996-2012	164.0	286.0	32.0	13288.0	270	52.31	36.0	6
4	AN Cook (ENG)	2006-2018	161.0	291.0	16.0	12472.0	294	45.35	33.0	5
...
5792	Zamir Khan (AFG)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
5793	S Zargar (Mex)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
5794	Zeshan Arif (Aut)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
5795	Ziaur Rahman (AFG)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
5796	Zoheb Malek (MLT)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

5797 rows x 11 columns

- 4) We used a combination of pd.merge and drop duplicates to combine all the data frames.

```
#### merging all the files
res1 = pd.merge(workdf8, df1.drop_duplicates(subset='Player'), how='left', left_on='Player', right_on='Player')
res2 = pd.merge(res1, df2.drop_duplicates(subset='Player'), how='left', left_on='Player', right_on='Player')
res3 = pd.merge(res2, df3.drop_duplicates(subset='Player'), how='left', left_on='Player', right_on='Player')
res4 = pd.merge(res3, df4.drop_duplicates(subset='Player'), how='left', left_on='Player', right_on='Player')
res5 = pd.merge(res4, df5.drop_duplicates(subset='Player'), how='left', left_on='Player', right_on='Player')
combine7 = pd.merge(res5, df6.drop_duplicates(subset='Player'), how='left', left_on='Player', right_on='Player')
```

combine7

	Player	TEST_Span	TEST_Matches_x	TEST_Innings_x	TEST_Not-Outs	TEST_RunsMade	TESTHighest-Score	TEST_Batting_Average	TEST_Hundereds	TEST_Fift
0	SR Tendulkar (INDIA)	1989-2013	200.0	329.0	33.0	15921.0	248*	53.78	51.0	6
1	RT Ponting (AUS)	1995-2012	168.0	287.0	29.0	13378.0	257	51.85	41.0	6
2	JH Kallis (SA)	1995-2013	166.0	280.0	40.0	13289.0	224	55.37	45.0	5
3	R Dravid (INDIA)	1996-2012	164.0	286.0	32.0	13288.0	270	52.31	36.0	6
4	AN Cook (ENG)	2006-2018	161.0	291.0	16.0	12472.0	294	45.35	33.0	5
...

- 5) We renamed the columns after combining as the merge tags of x and y had to be removed.

- 6) We replaced the NaN values with 0. Majorly our idea behind removing NaN with '0' was that only those players will be having NaN values who have not played in that particular format of Cricket data collected.
- 7) Changed the data type of fields with int & float as required.
- 8) Getting a new column "Country/Team" and moving it next to the player column while removing the brackets within with data is present.

```
#Adding Teams column
combine7['Country/Teams']=combine7['Player'].apply(lambda st: st[st.find("(")+1:st.find(")"]])
#Moving it next to Player column
column_to_move = combine7.pop('Country/Teams')
combine7.insert(1, 'Country/Teams', column_to_move)
#Remove the parenthesis & data inside it in Player column
combine7['Player'] = combine7['Player'].str.replace(r"\([()]*\)", "")

<ipython-input-13-5bf022e6c9b>:7: FutureWarning: The default value of regex will change from True to False in a future version.
combine7['Player'] = combine7['Player'].str.replace(r"\([()]*\)", "")
```

	Player	Country/Teams	TESTbatting_Span	TESTbatting_Matches	TESTbatting_Innings	TESTbatting_Not Out	TESTbatting_RunsMade	TESTbattingHighest Score
0	SR Tendulkar	INDIA	1989-2013	200	329	33	15921	248*
1	RT Ponting	AUS	1995-2012	168	287	29	13378	257
2	JH Kallis	SA	1995-2013	166	280	40	13289	224
3	R Dravid	INDIA	1996-2012	164	286	32	13288	270
4	AN Cook	ENG	2006-2018	161	291	16	12472	294

Inserting a new Country/Team column using lambda & replace

- 9) Exported the same to the final excel file & JSON file.

Exploratory Data Analysis

For carrying out the exploratory data analysis, we used the Sweetviz library package. For carrying out EDA we faced a challenge when using the Sweetviz package, as it was considering some variables as mixed variables so we had to convert them to either categorical variables using `astype(str)` or convert them to numeric variables using `pd.numeric()`.

Once, we converted the data types we were able to carry out a detailed analysis of each variable, showing the variable type, the Min and Max values for numeric variables, the frequency for each variable etc. A correlation graph was also produced, showing correlations between different variables. The EDA file is attached as an HTML output.

The data collected by us is the most appropriate structured data for carrying out Historical analytics and judging the performance of a current playing player across formats, be it Test, ODI or T-20. The dataset can be processed into an ML prediction model or an analytical framework spark where one can perform a Linear regression model to carry out various predictive analytics.

Crowdsourcing strategy to enhance the data

Over the past decade, crowdsourcing has emerged as a favorable method to address some of the rising challenges associated with data collection. In the world of cricket, the importance of data has been increasing day by day, with a lot of new metrics coming into the picture which may not be available readily when scrapping for data online. All teams today have analysts, who analyze players on varied metrics like 'How would a batsman respond to a certain delivery?' or 'At what strike rate does a batsmen bat for the first 15 overs' or 'Which bowler is effective in the death overs', 'Field mapping for a certain batsman' etc. Such kinds of variables are not readily available as the usual stats of batting strike rate or bowling average.

To make our dataset more valuable for the prediction analysis, a whole new set of metric variables must be developed and for this enhancement of data, crowdsourcing is one of the methods we would like to employ. Crowdsourcing involves obtaining data or information with help of a large number of people, through various mediums, the most common being the Internet. Crowdsourcing is a prevailing strategy and problem-solving process that can enhance any data to the desired result but is only effective when followed with the right strategy and relevant goals.

We will follow the below steps as part of our crowdsourcing strategy:

- 1) **Setting up a distinct process** – We need to set up distinct goals and ask relevant questions required for our data. Irrelevant questions or ambiguity in the final goal may result in varied data that may not be helpful. Our focus here would be on defining the metrics which would be used for predictive analysis.
- 2) **Relevant Experts** – The crowd should comprise experts that are relevant to our dataset. In our case, we will be crowdsourcing from a diverse crowd of cricketing experts, which would comprise Coaches, Players, Commentators, Cricket statisticians, and sports journalists.
- 3) **Significant data** – The data provided to experts for their inputs and suggestions, should be properly structured and should be aligned to our goal. In our case, we compiled all

batting and bowling stats, and for all 3 formats into one dataset, making it simple, so that the crowd doesn't have to refer to multiple datasets. Using the dataset, inferences could be made regarding the metrics defined earlier.

- 4) **Significant feedback** – We must make sure that we receive significant feedback from each member of the crowd and also, we have to filter out any feedback that is not relevant to our mission. This feedback would be essential in drawing our focus to the larger picture – the selection of players for a team.
- 5) **A Safe space** – We must ensure that people are willing and feel comfortable to share their ideas with us and must provide a medium that is accessible to all. The crowd of experts should be given a free hand and authority to use their wisdom and skill set to provide valuable inputs and insights.
- 6) **Open to feedback** – We should not outrightly dismiss people's ideas as this may create an environment of distrust amongst the crowd.

References

1. Thiago Santos Figueira (May 28, 2021) *Web-scraping tables in Python using beautiful soup*. Retrieved from <https://medium.com/geekculture/web-scraping-tables-in-python-using-beautiful-soup-8bbc31c5803e>
2. GeeksforGeeks (May 13, 2022) *Implementing Web Scraping in Python with BeautifulSoup*. Retrieved from <https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/>
3. Eugenia Anello (August 13, 2021) *A Simple Introduction to Web Scraping with BeautifulSoup*. Retrieved from <https://www.analyticsvidhya.com/blog/2021/08/a-simple-introduction-to-web-scraping-with-beautiful-soup/>
4. John Watson Rooney (May 22, 2020) *Web Scraping with Python: Ecommerce Product Pages*. Retrieved from <https://www.youtube.com/watch?v=nCuPv3tf2Hg>
5. Jovian (April 15, 2021) *Let's Build a Python Web Scraping Project from Scratch*. Retrieved from <https://www.youtube.com/watch?v=RKsLLG-bzEY>
6. Grepper (2020) *All Python Answers*. Retrieved from <https://www.codegrepper.com/code-examples/python/>
7. Stackoverflow (2021) *Duplicated rows when merging dataframe in Python*. Retrieved from <https://stackoverflow.com/questions/39019591/duplicated-rows-when-merging-dataframes-in-python>
8. Sai Kumar (February 9, 2018) *Data Mining for Performance Analysis in Cricket*. Retrieved from <https://analyticsindiamag.com/data-mining-performance-analysis-cricket/>
9. Adelina Chelniciuc (December 29, 2009) *A new approach to measuring performance in cricket: the Castrol Index*. Retrieved from <https://www.performancemagazine.org/a-new-approach-to-measuring-performance-in-cricket-%E2%80%93-the-castrol-index/>