

Data Mining on IPL Dataset

GROUP ID - 26

Mid Semester Report

18 October 2020



GROUP MEMBERS:

Aditya Raghuwanshi	170052	adityarg@iitk.ac.in
Ritesh Naik	170579	riteshn@iitk.ac.in
Shashi Kumar	160646	shashik@iitk.ac.in

Abstract

The Indian Premier League (IPL) is a professional Twenty20 cricket league in India usually contested between March and May of every year by eight teams representing eight different cities or states in India. IPL is very popular among cricket fans in India and is getting more liked by the high intensity & competitive nature of the game. Many official & unofficial sources have been collecting data from IPL matches from 2008. Like other domains, we can use this data to analyse weaknesses/strengths of the players and teams, or an overall performance of the campaign. This type of analysis can be useful for teams and also for predicting viewership. This report includes some of the basic analysis.

Introduction & Motivation

Aims of the project

1. To visualize different stats of players and teams throughout the seasons.
2. Classify players according to their performance into specific groups.
3. Find the scores of each player(or rank them) according to their performance in IPL.
4. Create a best 11 team for IPL using previous scores and other constraints.
5. To study data to find the relation of unconventional factors on each team results like Home ground - Away ground, Toss Winning - Toss Losing, etc.
6. To predict expected number of boundaries (fours & sixes) and targets using previous season stats.

Dataset Needed

We have identified two data sources that suit our need:

1. Official IPL Dataset

It contains following stats:

- Most Runs
- Most Runs (Over)
- Most Fours
- Most Fours (Innings)
- Most Sixes
- Most Sixes (Innings)
- Most Fifties
- Most Centuries
- Fastest Fifties
- Fastest Centuries
- Highest Scores
- Highest Scores (Innings)
- Best Batting Strike Rate
- Best Batting Average
- Biggest Sixes
- Most Wickets
- Most Maidens
- Most Dot Balls
- Most Dot Balls (Innings)
- Best Bowling Average
- Best Bowling Economy
- Best Bowling Economy (Innings)
- Best Bowling Strike Rate
- Best Bowling Strike Rate (Innings)
- Best Bowling Innings
- Most Run Conceded (Innings)
- Fastest Balls
- Most Hat Tricks
- Most Four Wickets
- Player Points
- Team Ranking

Some points about the data:

1. It contains stats / data of all seasons from 2008 to 2019.
2. It does NOT contain match wise data and player wise data.
3. As it does not contain match wise data, extracting IPL 2020 data now is not meaningful

2. Kaggle IPL Dataset

It contains following files:

- deliveries.csv**: Statistics of each ball of each match
- matches.csv**: Overall statistics of each match
- teamwise_home_and_away.csv**: Home and Away win percentages of each team
- Players.xlsx**: Contains details about player name, date of birth, country of origin, batting and bowling preference
- teams.csv**: Names of each match

Some points about the data:

1. It contains stats / data of all seasons from 2008 to 2019.

Sources of Dataset

1. Official IPL Dataset

We have extracted dataset from IPL official website using a PyInquirer Application built from python. We can choose feature(s) and year(s) to get required data.

- Select years from 2008 to 2019 or "all-time" records and select some of or all of the given stats.
- We call the function `get_year_stats()` in our `main()` function. This function then obtains HTML from the url, creates its soup object and passes it to `get_years()` and `get_stats()` methods. URL has following format:

`https://www.iplt20.com/stats/ + <year/> + <stats name>`

- The stats have two variables `stats_url`, `stats_title`. The former store page name of the stats page and the latter stores the title that will be displayed to the user.
- **`get_years(soup)`**, extracts years data from the website. **`re.compile`**, creates a regex to find tags having sub-menu as a substring in its class name. `textbfsoup.find_all`, finds all the a tags with the specified class name. **`year.get_text()`**, creates a list comprehension by applying `get_text()` function on all the elements of the result of the `find_all()` method.

```
1 soup = BeautifulSoup(page.content, 'html.parser')
2 years= year.get_text() for year in soup.find_all('a',class_=re.
    compile(r'sub-menu*'))
```

- We replace All Time Records with all-time. Former being display name and latter being the part of the URL.

```
1 years[-1] = 'all-time'
```

- Similarly, we will extract the stats name and stats page name in **`get_stats(soup)`** function.

```
1 stats = soup.find_all('a', class_=re.compile(r'side*'))
```

- We will use the **`years`** list to create a list of dictionary with **`name`** as key and **`year`** as value.
- We will use the **`stats_title`** list as name to be displayed to the user and **`stats_url`** as its value, which will be returned to the program when a user selects a particular stats.

For eg, name: Most Runs and respective value: most-runs.

Value is the name of the page in URL and the name is the title of the page.

- The following code prepares a PyInquirer query and stores their response in `answers` variable. Here we are using a while loop to handle no response from the user, `TypeError` to handle CTRL + C and `EOFError` to handle CTRL + D.

```
1 def user_input(years_q, stats_q, kanpeki):
2     answers = {'years': '', 'stats': ''}
3     try:
4         while len(answers.get('years')) == 0 or len(answers.get('stats')) ==
5             0:
6             questions = [
7                 'type': 'checkbox',
```

```

8     'message': 'Select type',
9     'name': 'years',
10    'choices': years_q,
11  }, {
12    'type': 'checkbox',
13    'message': 'Select statistics',
14    'name': 'stats',
15    'choices': stats_q,
16  }
17  ]
18  answers = prompt(questions)
19  temp = {'years': ''}
20  if answers['years'][0]=='By Season Stats':
21      while len(temp.get('years')) == 0:
22          questions = [
23              {
24                  'type': 'checkbox',
25                  'message': 'Select years',
26                  'name': 'years',
27                  'choices': kanpeki,
28              }
29          ]
30          temp=prompt(questions)
31          answers['years']=temp['years']
32  except TypeError:
33      error_msg()
34  except EOFError:
35      exit_application()
36  return answers['years'], answers['stats']
37

```

- We have obtained year values, stats values (page name and page title) and created a query to obtain user response and store it in a variable called **answers**.
- To extract players data, we define a function called **scrap_data(years, stats)** that will iterate over all the selected value of years and stats page names.

```

1 def scrap_data(years, stats):
2     base = 'https://www.iplt20.com/stats/'
3     base_file_name = 'Result'
4     for year in years:
5         for stat in stats:
6             if stat == 'team-ranking':
7                 url = base + year
8                 data, columns = get_page(url, True)
9                 data_set_name = 'Team-Ranking-' + year
10            else:
11                url = base + year + '/' + stat
12                print(url)
13                data, columns = get_page(url, False)
14                data_set_name = str.title(stat) + '-' + year
15            if len(data) == 0:
16                print(data_set_name + ' : ' + 'Data not Available')
17            else:
18                save_data(data, columns, data_set_name, base_file_name)
19

```

- All the pages of the first type have the same HTML layout and the same goes for the second type. As stated in the first case we dont need any page name and in the second case, the stats name is concatenated to our base URL.

- We then define a function called ***get_page(url, team)*** that will do a preliminary check and create a soup object of the page. The team argument is a boolean value set to True for the first type of page and False for the second type.

We call ***find_col(soup, team)*** function in ***get_page(url, team)*** which will return the header values of all the columns present in the table on the webpage.

```

1  def get_page(url, team):
2      try:
3          page = requests.get(url)
4          if page.status_code == 200:
5              soup = BeautifulSoup(page.content, 'html.parser')
6              data, columns = find_col(soup, team)
7              return data, columns
8          else:
9              print('Website Unreachable')
10             return
11     except requests.exceptions.ConnectionError:
12         print('Please check network connection')
13         return
14

```

- The ***find_col(soup, team)*** will find the column headers, and ***get_team_data(soup, columns)*** and ***get_player_data(soup, columns)*** will scrape the first and second type of data.

```

1  def find_col(soup, team):
2      try:
3          if team:
4              columns = list(filter(None, soup.find('tr', class_='standings-
5                  table__header').get_text().split('\n')))
6              return get_team_data(soup, columns)
7          else:
8              columns = re.sub(r'\n[\s]*', '\n', soup.find('tr', class_=re.
9                  compile(r'top-players__header*')).get_text().strip().split('\n')
10             return get_player_data(soup, columns)
11     except AttributeError:
12         return [], []
13
14 def get_team_data(soup, columns):
15     data = []
16     for i in soup.find_all('td'):
17         data.append(re.sub(r'\n[\s]*', " ", i.get_text().strip()))
18     data = np.array(data).reshape(len(data) // len(columns), len(columns)
19         + 1)
20     data = np.delete(data, 0, axis=1)
21     return data, columns
22
23 def get_player_data(soup, columns):
24     data = []
25     for i in soup.find_all('td', class_=re.compile(r'top-players*')):
26         data.append(re.sub(r'\n[\s]*', " ", i.get_text().strip()))
27     data = np.array(data).reshape(len(data) // len(columns), len(columns))
28     return data, columns

```

- Line #4 command finds all the TR tags with the specified class name, extracts text from it and then splits it to form a list. At last, remove the blank value to get all the column headers.

Line #7 finds TR tags with class that has top_players_header present in the class name, gets text from it substitute all the newline and space characters with just new line characters and then remove the blank space. At last splits it using \n as a delimiter to form a list.

Line #17 extracts all the TD tags and formats the values to get all the team data.

Line #26 extracts player data with tag name TD and class having top-players substring. After which formats it to form a list called data.

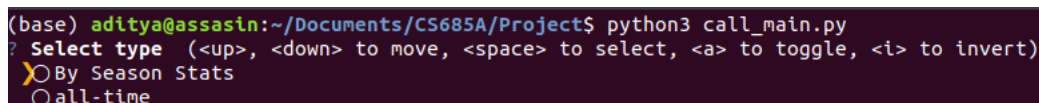
- Line #26 of the function **textitget_player_data(soup, columns)** we convert the data list to a data numpy array and also reshape it from a 1D array to a 2D array. The number of columns is specified by the length of the column list and the number of rows by integer division of length of data list by the length of the columns list. The same goes for the third last line of **get_team_data(soup, columns)**.
- We now extracted the data in a numpy 2d array called **data** and column headers in a list called **columns**.
- We will create a **save_data(data, columns, data_set_name, file_name)** function which will create a dataframe from the data and a file path based on the data in the dataframe. After which it will save dataframe in the form of a CSV and notify the user.

```
1 def save_data(data, columns, data_set_name, file_name):
2     df = pd.DataFrame(data, columns=columns)
3     file_path = './' + file_name + '-' + data_set_name + '.csv'
4     df.to_csv(file_path, index=False)
5     print(data_set_name + ' saved as: ' + os.path.dirname(os.path.abspath(
6         __file__)) + '/' + data_set_name + '.csv')
```

- The **main()** function will call **get_year_stats()** to get year values, stats page names and titles. After which we will call **prepare_question(years, stats_title, stats_url)** to prepare the query for the user. We will get users input from **user_input(years_q, stats_q)** function and then get scrapping by calling **scrap_data(years, stats)** function.

```
1 def main():
2     years, stats_url, stats_title = get_year_stats()
3     years_q, stats_q = prepare_question(years, stats_title, stats_url)
4     years, stats = user_input(years_q, stats_q)
5     print('Collecting and saving data.')
6     scrap_data(years, stats)
7
```

- Screenshots of the application



```
(base) aditya@assasin:~/Documents/CS685A/Project$ python3 call_main.py
? Select type (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
  ) By Season Stats
  ) all-time
```

```
(base) aditya@assasin:~/Documents/CS685A/Project$ python3 call_main.py
? Select type [By Season Stats]
? Select statistics (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
X Most Runs
  Most Runs (Over)
  Most Fours
  Most Fours (Innings)
  Most Sixes
  Most Sixes (Innings)
  Most Fifties
  Most Centuries
  Fastest Fifties
  Fastest Centuries
  Highest Scores
  Highest Scores (Innings)
  Best Batting Strike Rate
  Best Batting Average
  Biggest Sixes
  Most Wickets
  Most Maidens
  Most Dot Balls
  Most Dot Balls (Innings)
  Best Bowling Average
  Best Bowling Economy
  Best Bowling Economy (Innings)
  Best Bowling Strike Rate
  Best Bowling Strike Rate (Innings)
  Best Bowling Innings
  Most Runs Conceded (Innings)
  Fastest Balls
  Most Hat Tricks
  Most Four Wickets
  Player Points
  Team Ranking
```

```
(base) aditya@assasin:~/Documents/CS685A/Project$ python3 call_main.py
? Select type [By Season Stats]
? Select statistics [most-runs]
? Select years (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
X 2019
  2018
  2017
  2016
  2015
  2014
  2013
  2012
  2011
  2010
  2009
  2008
```

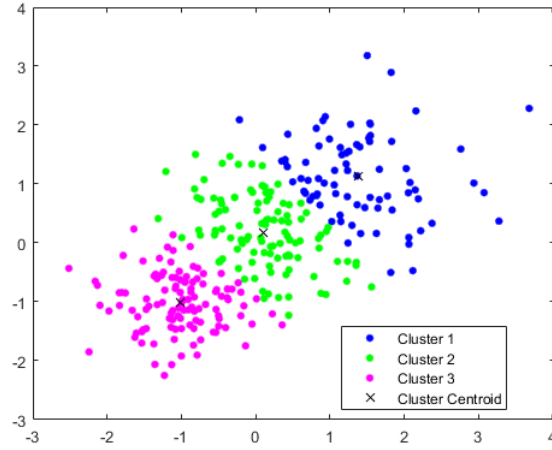
2. Kaggle IPL Dataset

All the files "deliveries.csv", "matches.csv", "teamwise_home_and_away.csv", "Players.csv" and "teams.csv" are directly available from the Kaggle website to download.

Methodology to meet the Aims

Each point will corresponds to previous points in the Aims .

1. For visualization and data presentation , we are going to use different libraries of python and different types of graphs . The main objective is to clean , modify and structure the data accordingly such that it will be easier for us to complete future objectives .
2. We are going to use clustering algorithm to classify the players given their cricket statistics. For example , the given figure is obtained after a clustering algorithm. Similary, we can apply this method in player's data and group them into different categories such that pink might refer to opening batsman, green might refer to middle order and blue might refer to hitter batsman.



3. Score of a player will measure the ability or contribution of a player to the team .We are going to use the different datas of the player's performance in the IPL to find his score .Ranking algorithm can be used to rank the players in the specific criteria . Points of a player is also available in the IPL official website , which we can use to compare our scores with those points.

RANKING ALGORITHM:

$$\text{Opening Batsman Score} = \frac{\frac{\text{totalruns}}{\text{totalmatches}}}{30} + \frac{\text{StrikeRate}}{100}$$

$$\text{Middle Order Batsman Score} = \frac{\frac{\text{NumberOfSixes}}{\text{MaxNumberOfSixes}}}{7} + \frac{\text{StrikeRate}}{100}$$

$$\text{Bowler Scores} = \frac{\frac{\text{Wickets}}{\text{MaxWickets}}}{7} + \frac{\text{Economy}}{100}$$

4. To find the best 11 players of the tournament, I have used results from clustering and ranking algorithm discussed above. From each clusters, I have taken best players according to their scores. I have taken 4 from upper batsmen, 3 from middle order, and 4 from bowlers. We can also manually take indian players for best indian team. We can alter the number of upper batsman, middle order and bowlers as required.
5. Our kaggle dataset contains `teamwise_home_and_away.csv` file. We are performing a simple operation. First, scale total number of matches to 100, because all the teams have played different number of matches. Then, we will find the ratio of "home wins" to "away wins" for each team. If the ratio >>1 for the team, then team performs better on home ground, else if the ratio <<1, then team performs better on away ground.

Our kaggle dataset contains `teams.csv` file. Get the total number of tosses won and tosses lost. Get the total number of matches won when the team loose the toss and also when won the toss. Normalize everything to 100 and take ratio. Ratio is defined as percent of matches won when toss is won by percent of matches won when toss is lost.

6. From the official IPL dataset, when can fetch top 100 players with most number of 4s(boundaries). Total number of 4s was not available in any data. So, we assumed that total number of 4s in a season is equivalent to total number of most 4s by 100 players. We calculated this data for each season from 2008 to 2019.

We can do similar operation to find total and average number of sixes scored across each season.

Results

CLUSTERING

Opening Batsman

KL Rahul	Quinton de Kock	Shikhar Dhawan	Suryakumar Yadav
David Warner	Ishan Kishan	AB de Villiers	Faf du Plessis
Mayank Agarwal	Devdutt Padikkal	Eoin Morgan	Shreyas Iyer
Manish Pandey	Ben Stokes	Shubman Gill	Nitish Rana
Rohit Sharma	Jonny Bairstow	Rishabh Pant	Jos Buttler
Shane Watson	Ambati Rayudu	Chris Gayle	Steve Smith
Kane Williamson	Aaron Finch	Wriddhiman Saha	Ruturaj Gaikwad

Middle Order Batsman

Rahul Tewatia	Marcus Stoinis	Sam Curran	Ravindra Jadeja
Kieron Pollard	Sanju Samson	Nicholas Pooran	Sunil Narine
Hardik Pandya	Andre Russell	Prithvi Shaw	MS Dhoni
Virat Kohli	Rahul Tripathi	Shimron Hetmyer	Dinesh Karthik
Glenn Maxwell	Robin Uthappa	Shivam Dube	Vijay Shankar
Abdul Samad	Tom Curran	Priyam Garg	Ajinkya Rahane
Abhishek Sharma	Mandeep Singh	Deepak Hooda	Saurabh Tiwary
Gurkeerat Mann Singh	Riyan Parag	Krishnappa Gowtham	Josh Philippe
Kedar Jadhav	Sarfaraz Khan	Mahipal Lomror	Yashasvi Jaiswal
Narayan Jagadeesan	Murali Vijay	Mohammad Nabi	Simran Singh
Tom Banton	Rinku Singh		

Bowlers

Jofra Archer	Kagiso Rabada	Jasprit Bumrah	Rashid Khan
Trent Boult	Anrich Nortje	Pat Cummins	Mohammad Shami
T Natarajan	Yuzvendra Chahal	Axar Patel	Varun Chakravarthi
Washington Sundar	Ravi Bishnoi	Krunal Pandya	Sandeep Sharma
Deepak Chahar	Rahul Chahar	Ravichandran Ashwin	Chris Morris
Navdeep Saini	Shreyas Gopal	Mohammed Siraj	Jason Holder
James Pattinson	Kartik Tyagi	Murugan Ashwin	Chris Jordan
Shardul Thakur	Shivam Mavi	Nathan Coulter-Nile	Arshdeep Singh
Khaleel Ahmed	Isuru Udana	Lockie Ferguson	Sheldon Cottrell
Kamlesh Nagarkoti	Lungi Ngidi	Dwayne Bravo	Piyush Chawla
Shahbaz Nadeem	Prasidh Krishna	Jaydev Unadkat	Tushar Deshpande
Bhuvneshwar Kumar	Karn Sharma	Ankit Rajpoot	Harshal Patel
Jimmy Neesham	Josh Hazlewood	Amit Mishra	Daniel Sams
Adam Zampa	Imran Tahir	Dale Steyn	Kuldeep Yadav
Mitchell Santner	Varun Aaron	Shahbaz Ahmed	Jayant Yadav
Moeen Ali	Pravin Dubey	Umesh Yadav	Siddarth Kaul
Mujeeb Ur Rahman	Andrew Tye	Mohit Sharma	Alex Carey
Basil Thampi	Sandeep Warrier	Karun Nair	Avesh Khan
Harpreet Brar	Chris Green	Dhawal Kulkarni	Monu Kumar
Ishant Sharma	Nikhil Naik	Shreevats Goswami	Mitchell Marsh
David Miller			

RANKING

Opening Batsman

Players	Scores
KL Rahul	2.888638095238095
Mayank Agarwal	2.849348484848485
Ishan Kishan	2.6861714285714284
Shikhar Dhawan	2.6590647058823524
AB de Villiers	2.596288888888889
Faf du Plessis	2.5587820512820514
Nicholas Pooran	2.5375761904761904
David Warner	2.4880666666666666
Sanju Samson	2.4817571428571426
Kieron Pollard	2.4725333333333332

Middle Order Batsman

Players	Scores
Kieron Pollard	2.6475333333333333
Hardik Pandya	2.6231333333333333
Nicholas Pooran	2.5304333333333333
Ishan Kishan	2.4576000000000002
Sanju Samson	2.4555666666666665
AB de Villiers	2.3540666666666668
Eoin Morgan	2.1841
Quinton de Kock	2.1383333333333333
Chris Gayle	2.1380666666666666
Jofra Archer	2.1269333333333336

Bowlers

Players	Scores
Rashid Khan	1.9702048417132216
Jasprit Bumrah	1.940118870728083
Kagiso Rabada	1.839328537170264
Jofra Archer	1.7353689567430024
Trent Boult	1.7116269343370976 =
Yuzvendra Chahal	1.6887005649717515
Varun Chakravarthy	1.5900584795321637
Anrich Nortje	1.5676599125943582
Mohammad Shami	1.483469467133411
Washington Sundar	1.4411633109619686

Best 11 Players

Players	Scores
KL Rahul	2.888638095238095
Mayank Agarwal	2.849348484848485
Ishan Kishan	2.6861714285714284
Shikhar Dhawan	2.6590647058823524

Opening Batsman with their score

Players	Scores
Kieron Pollard	2.6475333333333333
Hardik Pandya	2.6231333333333333
Nicholas Pooran	2.5304333333333333

Middle Order Batsman with their score

Players	Scores
Rashid Khan	1.9702048417132216
Jasprit Bumrah	1.940118870728083
Kagiso Rabada	1.839328537170264
Jofra Archer	1.7353689567430024

Bowlers with their score

HOME-AWAY MATCHES

Ratio \rightarrow *LEAST* $:\leq 0.1$; $0.1 < \textit{MODERATE} \leq 0.4$; $0.4 < \textit{MOST}$

LEAST effect on home-away games: Rising Pune Supergiant & Mumbai Indians

MOST effect on home-away games : Kochi Tuskers Kerala & Gujarat Lions have most impact on home-away games

Rest have **MODERATE** effects.

Rising Pune Supergiant, Kochi Tuskers Kerala & Gujarat Lions have played very few

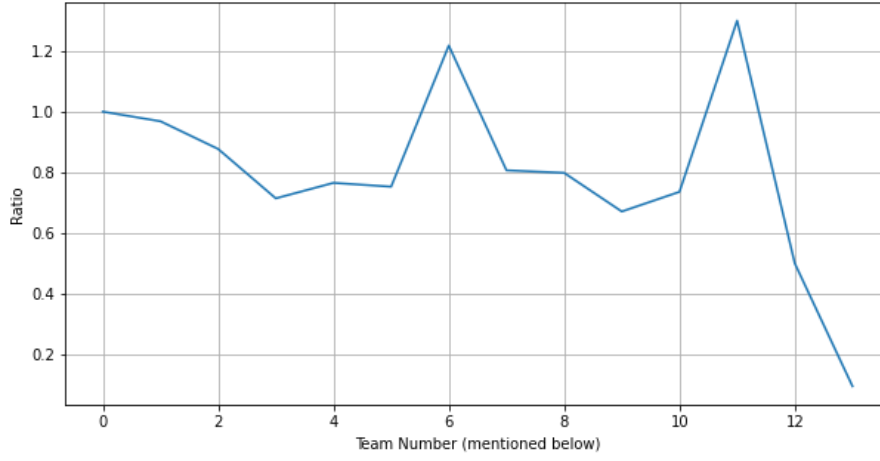


Figure 1: Ratio of home-away wins of each team

matches relative to other teams, so it is expected that they will move to MODERATE section after several matches. Only Mumbai Indians show very less effect after playing significant matches.

0:Rising Pune Supergiant	1:Mumbai Indians	2:Chennai Super Kings	3:Delhi Capitals
4:Sunrisers Hyderabad	5:Rajasthan Royals	6:Deccan Chargers	7:Kings XI Punjab
8:R. Challengers Bangalore	9:Kolkata Knight Riders	10:Delhi Daredevils	11:Pune Warriors
	12:Kochi Tuskers Kerala	13:Gujarat Lions	

Except **Deccan Chargers** & **Pune Warriors**, all other teams have played better in away matches than in home matches.

Conclusion: So, on average teams have slight effect of home-away match and most of the teams perform better in away matches.

TOSS WINNING

Ratio → *LEAST* : ≤ 0.1 ; $0.1 < \text{MODERATE} \leq 0.6$; $0.6 < \text{MOST}$ **LEAST** affected: ['Rajasthan Royals', 'Mumbai Indians']

MODERATE affected: ['Pune Warriors', 'Kolkata Knight Riders', 'Kochi Tuskers Kerala', 'Chennai Super Kings', 'Rising Pune Supergiants', 'Delhi Daredevils', 'Deccan Chargers', 'Delhi Capitals', 'Sunrisers Hyderabad', 'Royal Challengers Bangalore', 'Kings XI Punjab']

MOST affected: ['Gujarat Lions', 'Rising Pune Supergiant']

Almost all teams won their matches when they won the match toss, except Mumbai Indians, Sunrisers Hyderabad, Pune Warriors, Kings XI Punjab. Out of which Mumbai Indians have LEAST affect and Sunrisers Hyderabad, Pune Warriors, Kings XI Punjab have MODERATE affect.

Conclusion: Therefore, on average teams win their match when they win the toss

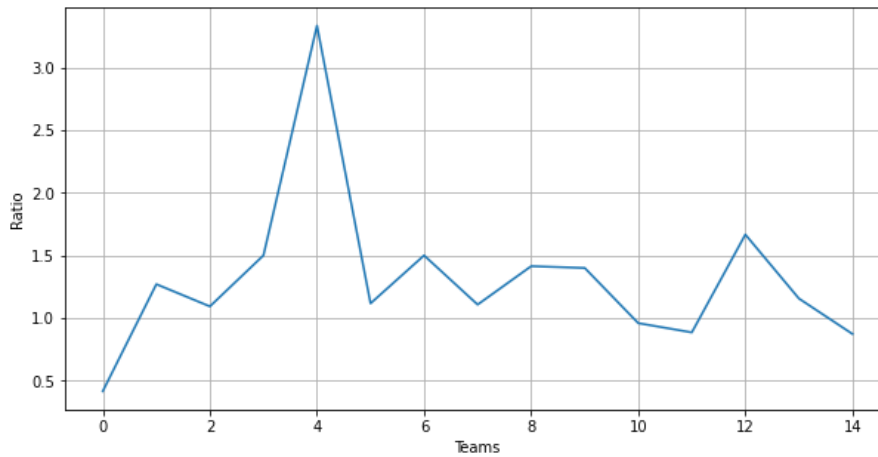
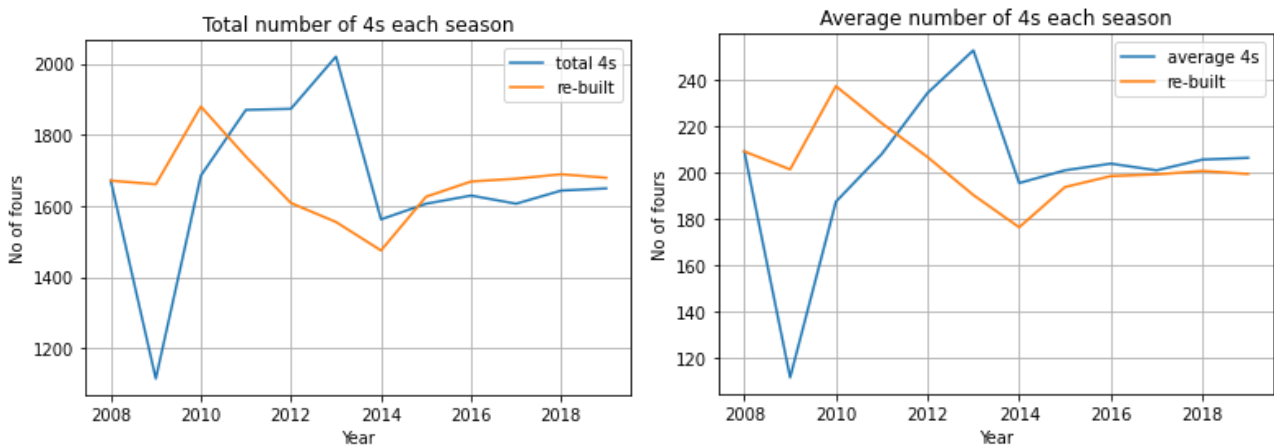


Figure 2: Ratio of no of toss won & toss lost for a won match

NUMBER OF FOURS (BOUNDARIES)

We have tried to find total number of 4s & average number of 4s. We have used **moving average** method to model the trend followed throughout the 11 seasons.

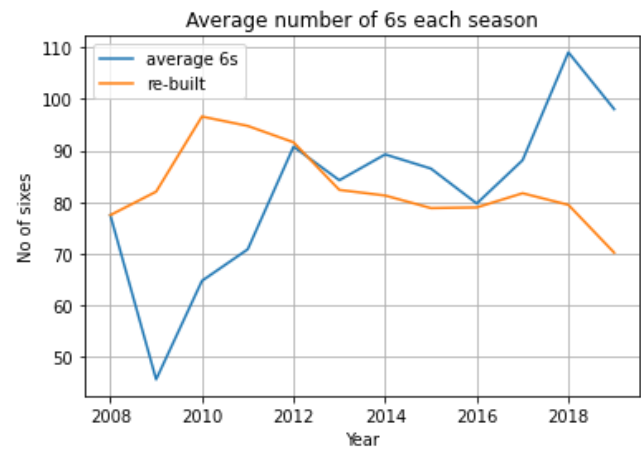
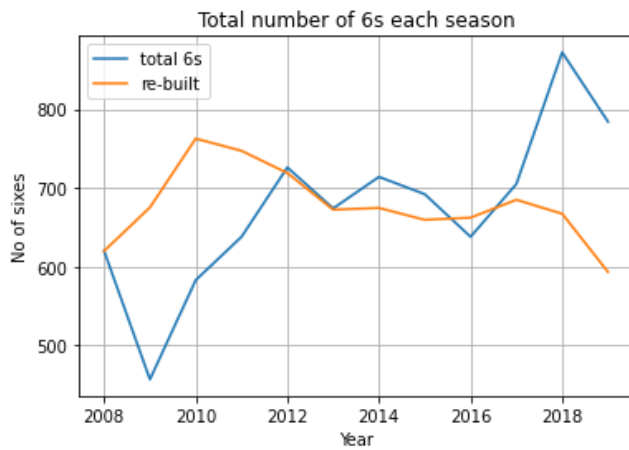


Prediction: Total number of 4s = 1646
Average number of 4s = 205

NUMBER OF SIXES (BOUNDARIES)

From the official IPL dataset, when can fetch top 100 players with most number of sixes(boundaries). Total number of sixes was not available in any data. So, we assumed that total number of sixes in a season is equivalent to total number of most sixes by 100 players. We calculated this data for each season from 2008 to 2019.

We have tried to find total number of sixes & average number of sixes. We have used **moving average method** to model the trend followed throughout the 11 seasons.



Prediction: Total number of sixes = 771
Average number of sixes = 96

Future Direction

1. Including data from Ranji Trophy as it reflects the latest performance of every player, and provides more data for new players or players with insignificant data.
2. To predict the result of each match.
3. To perform online learning, i.e., updating the data as season goes.
4. To also include data of viewership, so as to analyse expected revenue generated from the tournament.
5. To analyse weaknesses for each team.