Indian Premier League T20 Cricket - Analysis & Predictions

Capstone Project Report

Data Science Professional Certificate Program from HarvardX - Choose Your Own (CYO) Project

Valmeti Srinivas

03/12/2019

Contents

1 Acknowledgements:

I have started the journey of learning R programming language and becoming a Data Scientist in April, 2019 by registering myself in HarvardX Data Science Professional Certificate Program through edX online portal.

I have already submitted my Movielens movie ratings predictions report for which I was scored extremely positively.

My learning journey in this program had been very exhilarating and kept me flushed with enthusiasm all the time. All the 9 courses in the program, culminating in this project are very informative and educational.

I believe personally the program is well designed and apt for data science enthusiasts to start learning. I thank the program instructor Mr.Rafael Irizarry for his superb course delivery and for sharing his data science knowledge and skills.

I also thank the course staff for their excellent technical support through the blog posts and administrative staff for their email assistance.

It is also an opportunity for me to thank the R, R Studio, Stackoverflow, Stackexchange & Kaggle communities for their inputs in my learning quest.

I am also indebted to all programmers, developers and all others for their wonderful and dedicated contribution for the development of various packages and libraries, without which this data science project could have not been undertaken.

I would also like to thank Kaggle for providing so many wonderful and diverse data sets. It was great that I could find a latest data set on my topic of interest - Indian Premier T20 Cricket League (IPL).

Lastly, I express my sincere gratitude to my fellow learners who kept me motivated with their participation in the course, interesting questions, feedback, suggestions and discussions.

Thanks everyone !!

2 Introduction:

Cricket, a bat and a ball game, has a history of over 300 years and is a popular and well followed sport across the world.

Cricket is played between 2 teams, each with a core team of 11 players and some extra players in each match. While one team does the batting, the other team does the fielding (bowling). The team that wins the toss will decide to bat first or field first and the other team takes turns the other way around.

Currently, cricket matches are played primarily in 3 formats, 5 day test matches, 1 day 50 over matches and 20 over (T20) matches.

- Test matches: Initially, teams used to play matches which used to last over 3, 4, 5 or 6 days (currently 5 days is the standard) and each team can play up to 2 innings with a very few restrictions, importantly with no limitation on the time an individual team can bat for. Essentially, the team that ends up with more runs than the opponent in both of their own respective innings put together, and gets the opponent team completely out in both innings, wins the match.

  Teams composition in multi-day matches primarily used to be certain number of batsmen who can bat for long time and certain number of bowlers who can take wickets (get opponent team's batsmen out). This format of cricket was resulting in matches getting ended in no result (boring draw) even after a match was played for full 5 days (for 400+ overs). This had lead to lessening interest for the sport among people.

- One day 50 over matches: In order to prop up the interest for the sport among fans, a new form of cricket, called One Day matches got introduced with the match restricted to one day and each team batting once only. One Day matches are also called limited overs cricket as each team bats for a maximum of 50 overs.

  This form of One Day Cricket gained wide popularity as the matches became more exciting with almost all matches ending in a result. The format of the game made batsmen score quick runs (more runs in less number of balls they face) with some big hitting (scoring 4s & 6s), which is a major attraction for the match watching crowds. In One Day matches, the ability of a player to hit quick runs and restrict opponent from scoring runs became important criteria. This lead to teams having a few all rounders too (who can bat and also bowl), apart from just having pure batsmen and bowlers, which is the case in Test matches.

- T20 matches: With the hectic life style that people are increasingly adopting, they have less time to watch matches even for one full day. They wanted matches that are of shorter duration, yet produce result and excitement. This subsequently lead to another format of Cricket called T20 matches, where in each team is restricted to play once and for a maximum of 20 overs only. Restricting matches to a maximum of 20 overs has become extremely attractive, as this forced batsmen to employ innovative techniques and big hitting to score quick runs (high strike rate) and build big total scores. At the same time, the format put lot of pressure on bowlers to become more innovative and effective in restricting batsmen from scoring (low economy rate) and get them out quickly.

  Because of this limited overs pressure, each ball bowled in a T20 match has become vary valuable, with batsmen trying to score as many runs as possible and bowling team trying to restrict the batsmen from doing so. Today T20 matches are very popular across the playing nations and have become big crowd pullers.

  Sensing the potential for T20 matches, The Board Of Control For Cricket In India (BCCI) had introduced Indian Premier League (IPL T20) in 2008 which is conducted once every year since then.

  IPL is played among teams in a round robin format before the top 4 teams enter the knock out stage. The tournament usually lasts for about 3 months. IPL is probably one of the highest money churning sport events in the world. Given the prize money involved, attractive payment contracts to players,

huge crowds that turn out at the stadium and the huge popularity that Cricket enjoys in India, almost all top class players around the world vie to play in IPL.

I have selected to analyse data related IPL matches, to build models and make some predictions for this project.

More about Cricket in general can be found at https://en.Wikipedia.org/wiki/Cricket.

More information about IPL can be found at https://www.iplt20.com/.

3 Executive Summary:

The traditional thinking of team management while selecting players for test matches is that to have a certain number of pure, good batsmen who can score more runs, certain number of pure, good bowlers who can take wickets and probably to have one or two all rounders. Obviously sticking to this kind of team selection strategy will not work in 50 over one day matches, and completely not the right approach for T20 matches. In T20 matches, every ball and every player counts and the capability of every player to bat and bowl is best exploited given each team bats for 20 overs only. While the objective and number of players remain the same, the number of overs played in T20 matches have come down by more than 10 times (from 400+ to 40) when compared to 5 day test matches. When compared to one day matches, this has come down by 2.5 times.

- 1st Objective:
  This project's main objective is to rate and rank players by their value, by their past performance in IPL T20 matches. We call them top_calibre_players.

  Given the foregoing explanation, team managements can benefit if they switch their player selection approach from the traditional way to a new approach based on a player's calibre as both batsman and bowler. We provide a list of players based on their player value (playing calibre) so that IPL team managements can select best players.

- 2nd Objective:
  We set our second objective of this project is to build a model to predict the winner of a match.
  In T20 matches results are almost inevitable and people will be naturally interested in guessing the match winner in advance. Hence, we will build a few models to predict winner and compare them for the best "F1 score".

This project makes use of the two .csv files provided at https://www.kaggle.com/nowke9/ipldata. The data sets are downloaded and are made available at my github repo https://github.com/valmetisrinivas/CYO_IPL-Cricket_Predictions.git .

The key steps that were undertaken in this project have been summarized as below.

- Data set-up step: Here we install/ load the required libraries, read the data files and create the required data sets.
- Methods & Analysis:

  a. Data Exploration & Results: We do data exploration using various methods including visualization & perform various analysis to understand our data.

  b. Model Building & Results: This step includes a few sub-steps as explained below.

  1st Objective - Building a model to rank players by their playing calibre:

  In order to achieve our first objective of ranking players by their value, we will estimate players ratings based on the below 3 criteria and arrive at the final ranking by summarizing these values into one.

  – Criteria.1: We will rate players by their batting striking rates and bowling economy rates in IPL. We call this list "top_rate_players".
    For any player,
    Batting strike rate = number of runs scored/ number of balls faced
    Bowling economy rate = number of runs given/ number of balls bowled.

  – Criteria.2: We will rate players based on their contribution both in win and loss situations. We rate them based on their frequency of scoring maximum runs and dismissing maximum wickets for their teams in both matches won and lost by their teams. We call this list "top_contri_players".

- Criteria.3: We will also rate players based on how they excel against top_20 bowlers and top_20 batsmen. We call this list "top_excel_players".

Then we summarize all these 3 criteria to arrive at our final list of players by their value. Arriving at this list is our 1st objective. We call this list "top_calibre_players".

2nd Objective - Building a model with maximum "F1 score" to predict the winner of a match:

In order to achieve our second objective of predicting which team will win a match, given a set of conditions, we train and compare a few models for "F1 score".

In machine learning classifications, confusion matrix is an important tool that helps in assessing the model strength.

The "recall" parameter of a confusion matrix determines the strength of a model in terms of what fraction of a team's actual wins are model's true positive predictions.

The "precision" parameter of a confusion matrix determines the strength of a model in terms of what fraction of model's total positive predictions of wins are actually true.

Since, we are interested in both "recall" and "precision", "F1 score", which is based on both of these parameters, is the metric that we choose to determine the strength of the various models that we build.

- Results Discussion: We will briefly discuss the performance of the models that we have built.
- Conclusion: We will highlight how the final model can be used, limitations of the project and what could be the future enhancements.

As one might notice, the project work includes application of various tools and techniques that I have learnt in the program courses including application of the knowledge base and skills in R - data analysis, data visualization, inference, data wrangling, data organization, regression, machine learning and modeling.

Finally, apart from achieving the above stated two goals, the project work includes submitting the project report in Rmd and PDF formats along with the R Script file.

4 Data Set-up:

Let us start with the installation and loading of the required R packages.

```
# Load required packages
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if (!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if (!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")
if (!require(formattable))
  install.packages("formattable", repos = "http://cran.us.r-project.org")
```

Next, we read the data sets that were downloaded from the Kaggle website into "R".

```
# read the .csv datafiles into R
# Read the datafiles into R from my Github repo .
deliveries <- read.csv("https://raw.githubusercontent.com/valmetisrinivas/CYO_IPL-Cricket_Predictions/ma
matches <- read.csv("https://raw.githubusercontent.com/valmetisrinivas/CYO_IPL-Cricket_Predictions/maste
```

THE ABOVE CODE WILL DOWNLOAD THE DATA FILES FROM MY GITHUB REPO. IF THE ABOVE 2 LINES OF CODE DOES NOT WORK FOR YOU, U CAN FIRST CLONE THE PROJECT FROM MY GITHUB REPO AT https://github.com/valmetisrinivas/CYO_IPL-Cricket_Predictions.git AND THEN RUN THE BELOW 3 LINES OF CODE AFTER REMOVING THE COMMENT '#' MARK INFRONT OF THEM.

```
# Read the datafiles into R after cloning
# deliveries <- read.csv("./data/deliveries.csv", header = TRUE, sep = ",", stringsAsFactors = FALSE)
# matches <- read.csv("./data/matches.csv", header = TRUE, sep = ",", stringsAsFactors = FALSE)
```

Before, we proceed further, let us first do some initial inspection and pre-processing of our data files.

```
# inspect variable names in both datasets
names(matches)
```

```
##  [1] "id"              "season"          "city"
##  [4] "date"            "team1"           "team2"
##  [7] "toss_winner"     "toss_decision"   "result"
## [10] "dl_applied"      "winner"          "win_by_runs"
## [13] "win_by_wickets"  "player_of_match" "venue"
## [16] "umpire1"         "umpire2"         "umpire3"
```

```
names(deliveries)
```

```
##  [1] "match_id"         "inning"           "batting_team"
##  [4] "bowling_team"     "over"             "ball"
##  [7] "batsman"          "non_striker"      "bowler"
## [10] "is_super_over"    "wide_runs"        "bye_runs"
## [13] "legbye_runs"      "noball_runs"      "penalty_runs"
## [16] "batsman_runs"     "extra_runs"       "total_runs"
## [19] "player_dismissed" "dismissal_kind"   "fielder"
```

```r
# glimpse of our datasets
glimpse(matches)
```

```
## Observations: 756
## Variables: 18
## $ id              <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,...
## $ season          <int> 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017...
## $ city            <chr> "Hyderabad", "Pune", "Rajkot", "Indore", "Bang...
## $ date            <chr> "2017-04-05", "2017-04-06", "2017-04-07", "201...
## $ team1           <chr> "Sunrisers Hyderabad", "Mumbai Indians", "Guja...
## $ team2           <chr> "Royal Challengers Bangalore", "Rising Pune Su...
## $ toss_winner     <chr> "Royal Challengers Bangalore", "Rising Pune Su...
## $ toss_decision   <chr> "field", "field", "field", "field", "bat", "fi...
## $ result          <chr> "normal", "normal", "normal", "normal", "norma...
## $ dl_applied      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ winner          <chr> "Sunrisers Hyderabad", "Rising Pune Supergiant...
## $ win_by_runs     <int> 35, 0, 0, 0, 15, 0, 0, 0, 97, 0, 0, 0, 0, 17, ...
## $ win_by_wickets  <int> 0, 7, 10, 6, 0, 9, 4, 8, 0, 4, 8, 4, 7, 0, 0, ...
## $ player_of_match <chr> "Yuvraj Singh", "SPD Smith", "CA Lynn", "GJ Ma...
## $ venue           <chr> "Rajiv Gandhi International Stadium, Uppal", "...
## $ umpire1         <chr> "AY Dandekar", "A Nand Kishore", "Nitin Menon"...
## $ umpire2         <chr> "NJ Llong", "S Ravi", "CK Nandan", "C Shamshud...
## $ umpire3         <chr> "", "", "", "", "", "", "", "", "", "", "", ""...
```

```r
glimpse(deliveries)
```

```
## Observations: 179,078
## Variables: 21
## $ match_id         <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ inning           <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ batting_team     <chr> "Sunrisers Hyderabad", "Sunrisers Hyderabad",...
## $ bowling_team     <chr> "Royal Challengers Bangalore", "Royal Challen...
## $ over             <int> 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, ...
## $ ball             <int> 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, ...
## $ batsman          <chr> "DA Warner", "DA Warner", "DA Warner", "DA Wa...
## $ non_striker      <chr> "S Dhawan", "S Dhawan", "S Dhawan", "S Dhawan...
## $ bowler           <chr> "TS Mills", "TS Mills", "TS Mills", "TS Mills...
## $ is_super_over    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ wide_runs        <int> 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ bye_runs         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ legbye_runs      <int> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ noball_runs      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...
## $ penalty_runs     <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ batsman_runs     <int> 0, 0, 4, 0, 0, 0, 0, 1, 4, 0, 6, 0, 0, 4, 1, ...
## $ extra_runs       <int> 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, ...
## $ total_runs       <int> 0, 0, 4, 0, 2, 0, 1, 1, 4, 1, 6, 0, 0, 4, 1, ...
## $ player_dismissed <chr> "", "", "", "", "", "", "", "", "", "", "", "...
## $ dismissal_kind   <chr> "", "", "", "", "", "", "", "", "", "", "", "...
## $ fielder          <chr> "", "", "", "", "", "", "", "", "", "", "", "...
```

The "matches" dataset provides information based on match_id. That is, each observation (row) is information on a different match ever played in IPL.

The "deliveries" dataset provides information on each ball delivered in each innings of each match. That is, each observation (row) provides unique information on a particular ball ever bowled in all matches in IPL until year 2019.

We are interested in checking there are no duplicates or spelling mistakes in the names of teams and venues which reflect across both datasets in many variables. Let us first check that out in "deliveries" dataset.

```
# Inspect variables in "deliveries" datset for spelling mistakes, errors and duplicates
levels(as.factor(deliveries$batting_team))
```

```
##  [1] "Chennai Super Kings"         "Deccan Chargers"
##  [3] "Delhi Capitals"              "Delhi Daredevils"
##  [5] "Gujarat Lions"               "Kings XI Punjab"
##  [7] "Kochi Tuskers Kerala"        "Kolkata Knight Riders"
##  [9] "Mumbai Indians"              "Pune Warriors"
## [11] "Rajasthan Royals"            "Rising Pune Supergiant"
## [13] "Rising Pune Supergiants"     "Royal Challengers Bangalore"
## [15] "Sunrisers Hyderabad"
```

```
levels(as.factor(deliveries$bowling_team))
```

```
##  [1] "Chennai Super Kings"         "Deccan Chargers"
##  [3] "Delhi Capitals"              "Delhi Daredevils"
##  [5] "Gujarat Lions"               "Kings XI Punjab"
##  [7] "Kochi Tuskers Kerala"        "Kolkata Knight Riders"
##  [9] "Mumbai Indians"              "Pune Warriors"
## [11] "Rajasthan Royals"            "Rising Pune Supergiant"
## [13] "Rising Pune Supergiants"     "Royal Challengers Bangalore"
## [15] "Sunrisers Hyderabad"
```

We can notice that there are couple of errors in the team names. "Rising Pune Supergiants" is simply "Rising Pune Supergiant" (without 's' in the end) and "Delhi Daredevils" franchise has renamed itself as "Delhi Capitals" in later seasons of IPL. Let us correct these two details.

```
# preprocess data - correct errors in team names in "deliveries" dataset
deliveries <- deliveries %>%
  mutate(batting_team = str_replace(batting_team, "Delhi Daredevils", "Delhi Capitals")
  )
deliveries <- deliveries %>%
  mutate(batting_team = str_replace(
      batting_team,
      "Rising Pune Supergiants",
      "Rising Pune Supergiant"
    )
  )

deliveries <- deliveries %>%
  mutate(bowling_team = str_replace(bowling_team, "Delhi Daredevils", "Delhi Capitals")
  )
deliveries <- deliveries %>%
  mutate(bowling_team = str_replace(
      bowling_team,
      "Rising Pune Supergiants",
```

```
      "Rising Pune Supergiant"
    )
  )
```

Let us check if the same type of errors exist in "matches" dataset also for team1, team2, winner, toss_winner and venue variables.

```
# Inspect variables in "matches" datset for spelling mistakes, errors and duplicates
levels(as.factor(matches$team1))
```

```
##  [1] "Chennai Super Kings"         "Deccan Chargers"
##  [3] "Delhi Capitals"              "Delhi Daredevils"
##  [5] "Gujarat Lions"               "Kings XI Punjab"
##  [7] "Kochi Tuskers Kerala"        "Kolkata Knight Riders"
##  [9] "Mumbai Indians"              "Pune Warriors"
## [11] "Rajasthan Royals"            "Rising Pune Supergiant"
## [13] "Rising Pune Supergiants"     "Royal Challengers Bangalore"
## [15] "Sunrisers Hyderabad"
```

```
levels(as.factor(matches$team2))
```

```
##  [1] "Chennai Super Kings"         "Deccan Chargers"
##  [3] "Delhi Capitals"              "Delhi Daredevils"
##  [5] "Gujarat Lions"               "Kings XI Punjab"
##  [7] "Kochi Tuskers Kerala"        "Kolkata Knight Riders"
##  [9] "Mumbai Indians"              "Pune Warriors"
## [11] "Rajasthan Royals"            "Rising Pune Supergiant"
## [13] "Rising Pune Supergiants"     "Royal Challengers Bangalore"
## [15] "Sunrisers Hyderabad"
```

```
levels(as.factor(matches$winner))
```

```
##  [1] ""                            "Chennai Super Kings"
##  [3] "Deccan Chargers"             "Delhi Capitals"
##  [5] "Delhi Daredevils"            "Gujarat Lions"
##  [7] "Kings XI Punjab"             "Kochi Tuskers Kerala"
##  [9] "Kolkata Knight Riders"       "Mumbai Indians"
## [11] "Pune Warriors"               "Rajasthan Royals"
## [13] "Rising Pune Supergiant"      "Rising Pune Supergiants"
## [15] "Royal Challengers Bangalore" "Sunrisers Hyderabad"
```

```
levels(as.factor(matches$toss_winner))
```

```
##  [1] "Chennai Super Kings"         "Deccan Chargers"
##  [3] "Delhi Capitals"              "Delhi Daredevils"
##  [5] "Gujarat Lions"               "Kings XI Punjab"
##  [7] "Kochi Tuskers Kerala"        "Kolkata Knight Riders"
##  [9] "Mumbai Indians"              "Pune Warriors"
## [11] "Rajasthan Royals"            "Rising Pune Supergiant"
## [13] "Rising Pune Supergiants"     "Royal Challengers Bangalore"
## [15] "Sunrisers Hyderabad"
```

```
levels(as.factor(matches$venue))
```

```
##  [1] "ACA-VDCA Stadium"
##  [2] "Barabati Stadium"
##  [3] "Brabourne Stadium"
##  [4] "Buffalo Park"
##  [5] "De Beers Diamond Oval"
##  [6] "Dr DY Patil Sports Academy"
##  [7] "Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium"
##  [8] "Dubai International Cricket Stadium"
##  [9] "Eden Gardens"
## [10] "Feroz Shah Kotla"
## [11] "Feroz Shah Kotla Ground"
## [12] "Green Park"
## [13] "Himachal Pradesh Cricket Association Stadium"
## [14] "Holkar Cricket Stadium"
## [15] "IS Bindra Stadium"
## [16] "JSCA International Stadium Complex"
## [17] "Kingsmead"
## [18] "M Chinnaswamy Stadium"
## [19] "M. A. Chidambaram Stadium"
## [20] "M. Chinnaswamy Stadium"
## [21] "MA Chidambaram Stadium, Chepauk"
## [22] "Maharashtra Cricket Association Stadium"
## [23] "Nehru Stadium"
## [24] "New Wanderers Stadium"
## [25] "Newlands"
## [26] "OUTsurance Oval"
## [27] "Punjab Cricket Association IS Bindra Stadium, Mohali"
## [28] "Punjab Cricket Association Stadium, Mohali"
## [29] "Rajiv Gandhi International Stadium, Uppal"
## [30] "Rajiv Gandhi Intl. Cricket Stadium"
## [31] "Sardar Patel Stadium, Motera"
## [32] "Saurashtra Cricket Association Stadium"
## [33] "Sawai Mansingh Stadium"
## [34] "Shaheed Veer Narayan Singh International Stadium"
## [35] "Sharjah Cricket Stadium"
## [36] "Sheikh Zayed Stadium"
## [37] "St George's Park"
## [38] "Subrata Roy Sahara Stadium"
## [39] "SuperSport Park"
## [40] "Vidarbha Cricket Association Stadium, Jamtha"
## [41] "Wankhede Stadium"
```

We see that the same errors in the "deliveries" data set also exist in matches data set under toss_winner","winner","team1" & "team2" variables, plus a few duplicates exist under "venue". Let us correct these details too.

```
# preprocess data - correct errors in team names in "matches" dataset
matches <- matches %>%
  mutate(team1 = str_replace(
    team1, "Delhi Daredevils", "Delhi Capitals"
  ))
```

```
matches <- matches %>%
  mutate(team1 = str_replace(team1, "Rising Pune Supergiants", "Rising Pune Supergiant")
  )
matches <- matches %>%
  mutate(team2 = str_replace(
    team2, "Delhi Daredevils", "Delhi Capitals"
  ))
matches <- matches %>%
  mutate(team2 = str_replace(team2, "Rising Pune Supergiants", "Rising Pune Supergiant")
  )
matches <- matches %>%
  mutate(toss_winner = str_replace(
    toss_winner, "Delhi Daredevils", "Delhi Capitals"
  ))
matches <- matches %>%
  mutate(toss_winner = str_replace(toss_winner, "Rising Pune Supergiants", "Rising Pune Supergiant")
  )

matches <- matches %>%
  mutate(winner = str_replace(
    winner, "Delhi Daredevils", "Delhi Capitals"
  ))
matches <- matches %>%
  mutate(winner = str_replace(winner, "Rising Pune Supergiants", "Rising Pune Supergiant")
  )

matches <- matches %>%
  mutate(venue = str_replace(venue, "Feroz Shah Kotla Ground", "Feroz Shah Kotla")
  )

matches <- matches %>%
  mutate(venue = str_replace(venue, "M Chinnaswamy Stadium", "M. Chinnaswamy Stadium")
  )

matches <- matches %>%
  mutate(venue = str_replace(venue, "MA Chidambaram Stadium, Chepauk", "M. A. Chidambaram Stadium")
  )

matches <- matches %>%
  mutate(venue = str_replace(venue, "Punjab Cricket Association IS Bindra Stadium, Mohali",
        "Punjab Cricket Association Stadium, Mohali")
  )

matches <- matches %>%
  mutate(venue = str_replace(venue, "Rajiv Gandhi Intl. Cricket Stadium",
        "Rajiv Gandhi International Stadium, Uppal")
  )
```

Next, we will check if there are any missing values in our data sets.

```
# Check for NAs and missing values
sum(is.na(deliveries)) == 0
```

```
## [1] TRUE
```

```r
sum(is.na(matches)) == 0
```

```
## [1] TRUE
```

Luckily, we don't have missing values or NA. The data sets are relatively well pre-processed.

Now, we create two data sets "mat_ds" & "del_ds" out of our "matches" & "deliveries" data sets respectively for exploration and analysis.

```r
# Create primary datasets
mat_ds <- matches %>%
  select(
    match_id = id,
    season,
    city,
    team1,
    team2,
    toss_winner,
    toss_dec = toss_decision,
    winner,
    pom = player_of_match,
    venue
  )

del_ds <- deliveries %>%
  select(
    inning,
    match_id,
    over,
    ball,
    batsman,
    bowler,
    runs = batsman_runs,
    bat_team = batting_team,
    bowl_team = bowling_team,
    total_runs,
    dismissal_kind
  ) %>%
  gather(role, player, batsman:bowler) %>%
  mutate(role=as.factor(role))
```

Please note that we have rearranged the variables "batsman" and "bowler" into "role" and "player" variables in our "del_ds" data set using "gather" function for the purpose of our analysis and exploration. We shall take a note and remember in our analysis and use of data that this step ended up in duplicating all the observations (We have double the original observations now).

We have renamed "id", "toss_decision" & "Player_of_match", "batsman_runs", "batting_team" & "bowling_team" to "match_id", "toss_dec", "pom", runs, "bat_team" & "bowl_team" respectively for convenience purpose. We also have ignored a few variables such as the day on which a match was played, umpires who officiated the match, extra runs, fielder, season etc., variables as they clearly do not have any relationship to player value.

5 Methods & Analysis - Data Exploration & Results:

Let us start our exploration of data with finding out a few unique facts about our data.

```
# Total players
n_distinct(del_ds$player)
```

```
## [1] 559
```

```
# Unique teams
n_distinct(c(unique(mat_ds$team1),unique(mat_ds$team2)))
```

```
## [1] 13
```

```
# Total venues
n_distinct(mat_ds$venue)
```

```
## [1] 36
```

```
# Total matches played
total_played <- mat_ds %>%
  summarize(tot_mat_played = n())
total_played
```

```
##   tot_mat_played
## 1            756
```

```
# Total matches without win/ loss result
total_no_results <- mat_ds %>%
  filter(winner == "") %>%
  summarize(tot_noresults = n())
total_no_results
```

```
##   tot_noresults
## 1             4
```

We can see that only 4 matches (about 0.5%) have ended up with no result (drawn), out of a total of 756 matches played in IPL. This fact vindicates our logic that T20 produces a win/ loss result almost in all matches. Compare this against the over 35% of test matches (multi-day), that ended up in a no-result (draw).

Let us then start analysing the different run types scored. Let us first see which runs have been scored how many times by teams and also by batsmen.

```
# number of times each different run was scored by the team
del_ds %>%
  filter(role == "batsman") %>%
  group_by(run_type = total_runs) %>%
  summarize(count = n()) %>%
  mutate(percent = percent(count / sum(count))) %>%
  arrange(desc(count))
```

```
## # A tibble: 10 x 3
##    run_type count percent
##       <int> <int> <formttbl>
##  1        1 73059 40.80%
##  2        0 63002 35.18%
##  3        4 20599 11.50%
##  4        2 13125 7.33%
##  5        6  8148 4.55%
##  6        3   688 0.38%
##  7        5   339 0.19%
##  8        8    64 0.04%
##  9        7    38 0.02%
## 10       10    16 0.01%
```

```
# number of times a batsman had scored a different run
del_ds %>%
  filter(role == "batsman") %>%
  group_by(run_type = runs) %>%
  summarize(count = n()) %>%
  mutate(percent = percent(count / sum(count))) %>%
  arrange(desc(count))
```

```
## # A tibble: 8 x 3
##   run_type count percent
##      <int> <int> <formttbl>
## 1        0 70845 39.56%
## 2        1 67523 37.71%
## 3        4 20392 11.39%
## 4        2 11471 6.41%
## 5        6  8170 4.56%
## 6        3   587 0.33%
## 7        5    79 0.04%
## 8        7    11 0.01%
```

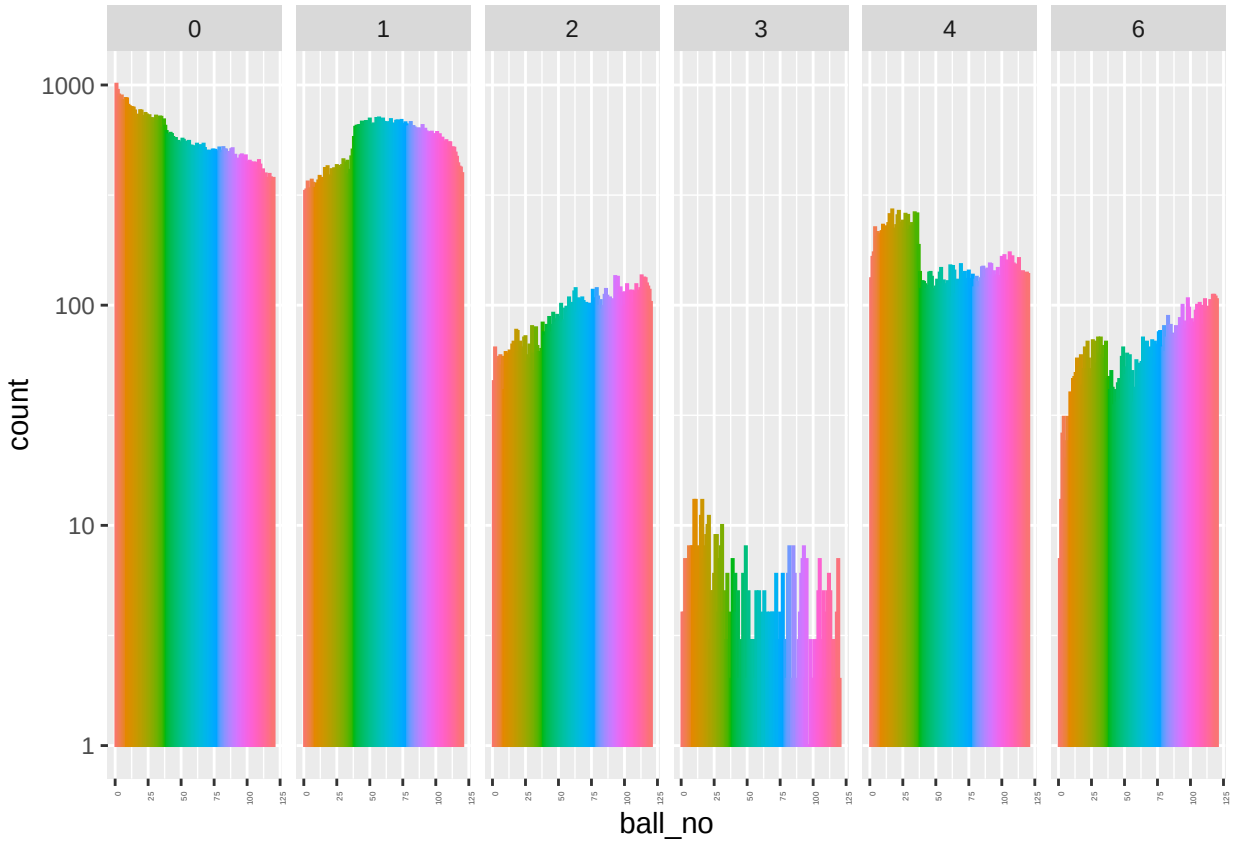
Let us check if there is a relationship between the ball number in an innings and the type of run scored on that ball. For this purpose, we will ignore 5s & 7s which are rarely scored compared to other run types and also we will limit to 120 balls, which is the normal cutoff for an innings.

```
# Major run types scored on each ball and correlation between them
runs_balls <- del_ds %>%
  group_by(match_id, inning) %>%
  mutate(ball_no = 1:n()) %>%
  ungroup() %>%
  filter(role == "batsman") %>%
  filter(runs != "" & runs != "5" & runs != "7" & ball_no %in% 1: 120) %>%
  group_by(ball_no, runs) %>%
  summarize(count=n())
runs_balls
```

```
## # A tibble: 715 x 3
## # Groups:   ball_no [120]
##    ball_no  runs count
##      <int> <int> <int>
```

```
##  1       1     0  1009
##  2       1     1   329
##  3       1     2    45
##  4       1     3     4
##  5       1     4   132
##  6       1     6     7
##  7       2     0   946
##  8       2     1   334
##  9       2     2    64
## 10       2     3     4
## # ... with 705 more rows
```

```
runs_balls %>%
  ggplot(aes(ball_no,count, col=factor(ball_no))) +
  geom_col() +
  scale_y_log10() +
  facet_grid( ~ runs) +
  theme(axis.text.x = element_text(
    angle = 90,
    size = 3,
    hjust = 1
  ),
  legend.position = "none")
```



```r
cor(runs_balls$runs, as.numeric(runs_balls$ball_no))
```

```
## [1] -0.0009551492
```

Clearly there is no relationship between ball number and runs scored of that ball.

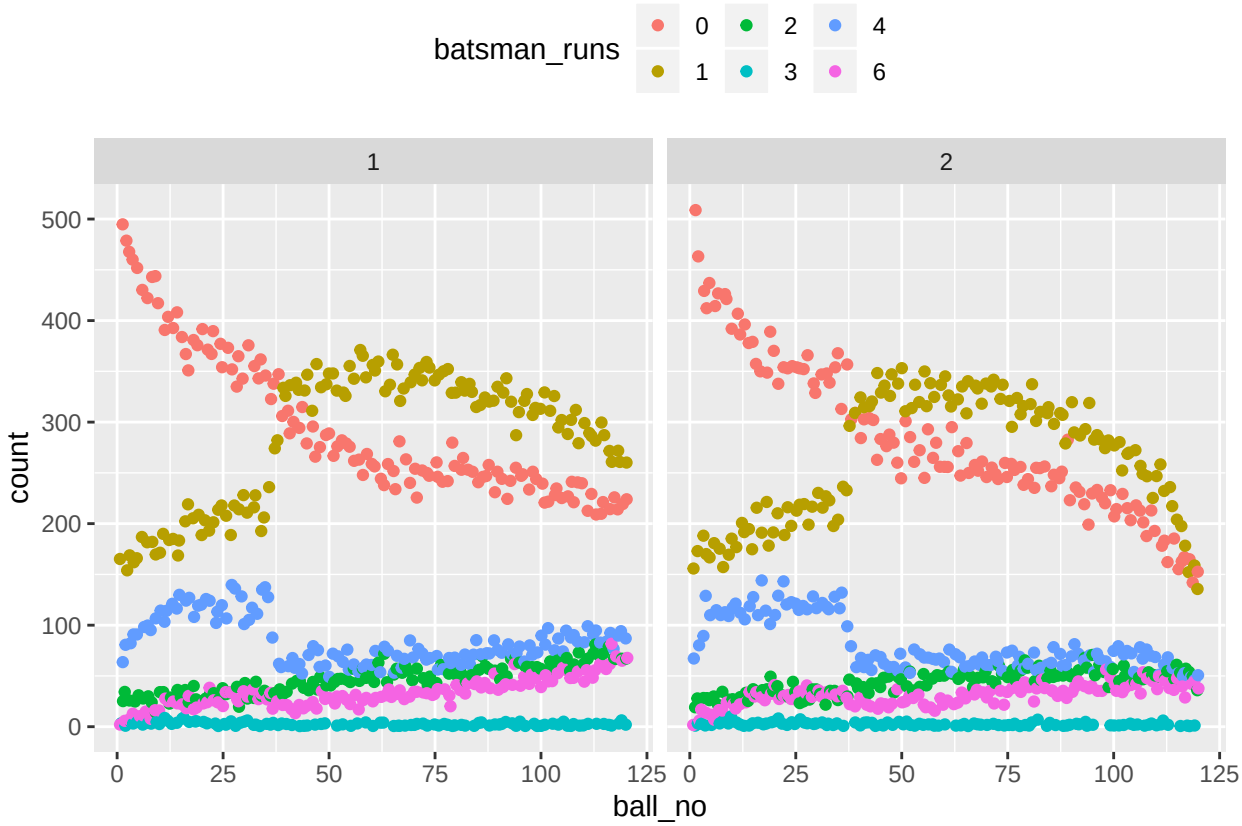
From the plot, we can see the peaks in 4s and 6s at the beginning of the innings is due to the 1st power play overs (6 overs), during which there are fielding restrictions and is a great opportunity for batsmen to score big runs. 4s are scored heavily in the beginning of the innings but dip during the middle of the innings around which time 1s get peaked and 2s continue to rise. This is due to the fact that batting teams try to stabilize their innings during mid overs.

We can see 0s tend to fall as the innings progresses and 2s is the only one which is relatively on the increasing trend throughout the innings. We can also see the 6s, 2s and 4s scored have a slight increasing pattern towards the end of the innings.

These facts indicate that teams try to bat the full 20 overs (through stabilization in mid overs) and try to score quick runs more towards the end of the innings.

As we are looking at runs scored on different balls, let us see how are they scored by batsmen on each ball, given "inning" (1st batting or 2nd batting). For this purpose, let us remove the rarest of rare run types, "5s" and "7s", a batsman ever scores.

```r
# Runs scored on each ball in 1st and 2nd innings
del_ds %>%
  group_by(match_id, inning) %>%
  mutate(ball_no = 1:n()) %>%
  ungroup() %>%
  filter(inning %in% 1:2 & role == "batsman" & runs %in% c(0,1,2,3,4,6) & ball_no %in% 1:120) %>%
  group_by(batting_turn=as.factor(inning), ball_no, batsman_runs=as.factor(runs)) %>%
  summarise (count=n()) %>%
  ggplot(aes(ball_no, count, col=batsman_runs)) +
  geom_point(size=0.5) +
  geom_jitter() +
  theme(legend.position = "top") +
  facet_grid(~ batting_turn)
```



The patterns are similar in both the innings highlighting how batting innings progresses in IPL matches.

From the plot, we can also make out that irrespective of the batting turn, 0s and 1s exchange places with each other around the the initial power play (first 36 balls). The 0s seem to be steadily reducing indicating as innings progresses, batsmen try to score runs of every ball, particularly more boundaries or higher run types. While boundaries drop from high after the initial 36 balls, 1s seem to increase. The first 36 balls are called 1st power play during which fielders are mostly restricted to inner circles, thus hitting boundaries is more easy. After 36 balls, teams try to stabilize innings (not losing quick wickets), which explains increase in 1s and drop in 4s.

0s and 1s, both sharply drop towards the end of 2nd innings. This is once gain due to the fact that teams batting second try to pass the target in the last overs with some big hitting or scoring higher run types on every ball.

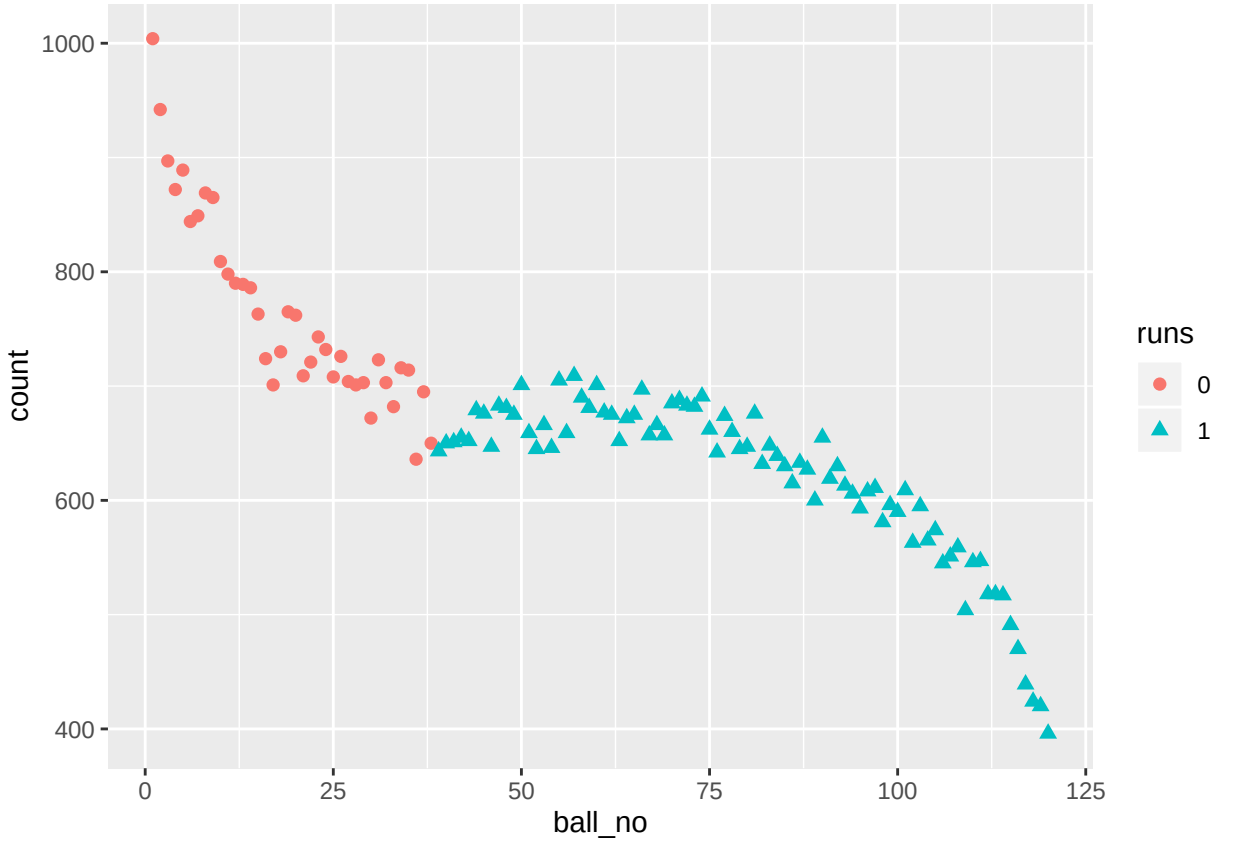
We can also notice that the boundaries (4s & 6s) steadily increase as the innings progresses as batsmen try to finish with big hitting. However, the very small dip in boundaries towards the very end of the innings is due to the fact the teams sometimes get out well before the 120 balls.

Excepting 0s and 1s, all other run types seem to have clear separation almost all the time - 4s seem to be more than 2s, 2s seem to be more than 6s and 6s seem to be more than 3s all most all the time, which is a fact shown in our previous tables.

Next, we look more closely which run type (0s, 1s, 2s, 3s, 4s ,5s, 6s & 7s) was scored highest number of times on a given ball over either of the innings.

```
# Different run types, how they trend during the innings
del_ds %>%
  group_by(match_id, inning) %>%
  mutate(ball_no = 1:n()) %>%
```

```
  ungroup() %>%
  filter (inning %in% 1:2 & role == "batsman" & ball_no %in% 1:120) %>%
  group_by(ball_no=as.numeric(ball_no), runs = as.factor(runs)) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  arrange(ball_no) %>%
  top_n(1) %>%
  ggplot(aes(ball_no,count,col=runs, shape = runs)) +
  geom_point(size = 2)
```



It is amazing to note that only 0s or 1s figure with highest counts for any ball; no other run type has the highest count for any ball. It is even more amazing to note that till about the 1st power play batsmen score more 0s than 1s on all balls, but will score more 1s than 0s, thereafter. This phenomenon is logical because batsmen try to score runs on every ball as the inning progresses and by nature 1s will be scored more often than any other run type. However, it is remarkable to see it actually happening.

At the end of a match, an award for the player of the match will be given for the player who performed best in that match. The number of player of the match awards a player received is a good indication of a player's capability. Below is the list of top 20 player of the match award winners and how many times they have won the same.

```
# top_20 players who got maximum player of the match award
mat_ds %>%
  group_by(pom) %>%
  summarize(player_of_match= n()) %>%
  arrange(desc(player_of_match)) %>%
```

```
  head(n = 20)
```

```
## # A tibble: 20 x 2
##    pom            player_of_match
##    <chr>                    <int>
##  1 CH Gayle                    21
##  2 AB de Villiers              20
##  3 DA Warner                   17
##  4 MS Dhoni                    17
##  5 RG Sharma                   17
##  6 YK Pathan                   16
##  7 SR Watson                   15
##  8 SK Raina                    14
##  9 G Gambhir                   13
## 10 AM Rahane                   12
## 11 MEK Hussey                  12
## 12 V Kohli                     12
## 13 A Mishra                    11
## 14 AD Russell                  11
## 15 DR Smith                    11
## 16 V Sehwag                    11
## 17 JH Kallis                   10
## 18 KA Pollard                  10
## 19 AT Rayudu                    9
## 20 SE Marsh                     9
```

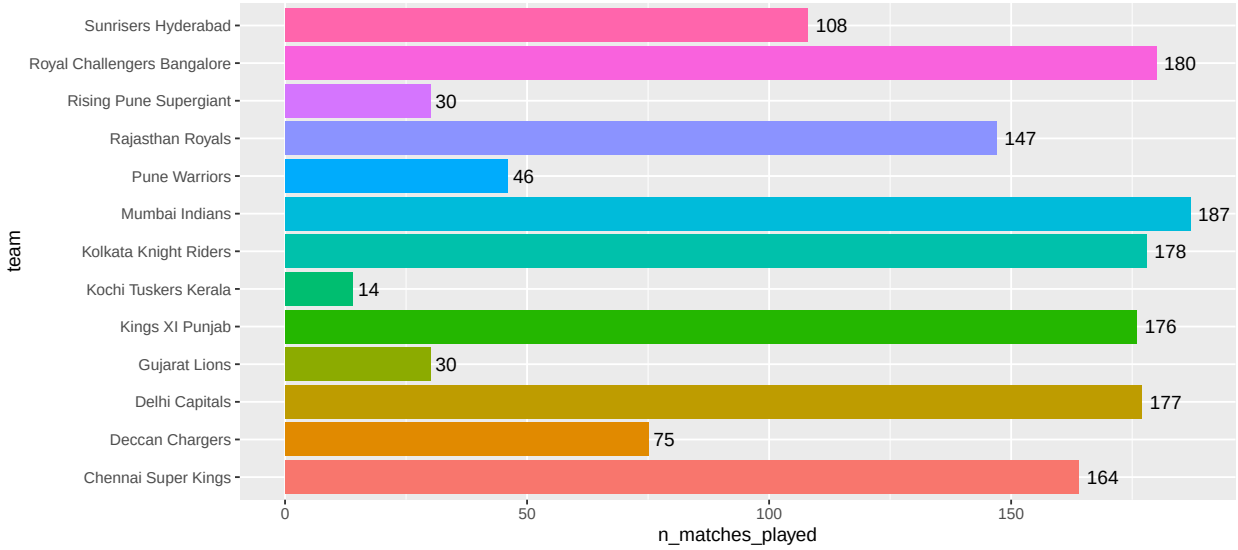
It is also good to check how many matches were played by each team and by each player

```
# no.of matches played by each team
played1 <- mat_ds %>%
  group_by(team1) %>%
  summarize(count1 = n()) %>%
  arrange(team1) %>%
  rename(team = team1)

played2 <- mat_ds %>%
  group_by(team2) %>%
  summarize(count2 = n()) %>%
  arrange(team2) %>%
  rename(team = team2)

matches_team <- played1 %>%
  full_join(played2, by = "team") %>%
  mutate(n_matches_played = count1 + count2) %>%
  select(team, n_matches_played) %>% arrange(desc(n_matches_played))
matches_team
```

```
## # A tibble: 13 x 2
##    team                         n_matches_played
##    <chr>                                   <int>
##  1 Mumbai Indians                            187
##  2 Royal Challengers Bangalore               180
##  3 Kolkata Knight Riders                     178
```

```
##  4 Delhi Capitals                    177
##  5 Kings XI Punjab                   176
##  6 Chennai Super Kings               164
##  7 Rajasthan Royals                  147
##  8 Sunrisers Hyderabad               108
##  9 Deccan Chargers                    75
## 10 Pune Warriors                      46
## 11 Gujarat Lions                      30
## 12 Rising Pune Supergiant             30
## 13 Kochi Tuskers Kerala               14
```

```r
matches_team %>%
  ggplot(aes(team, n_matches_played, fill = team)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  geom_text(aes(label = n_matches_played), hjust = -0.25)
```




```r
# no.of matches played by each player
matches_players <- del_ds %>%
  select(match_id, player) %>%
  group_by(player, match_id) %>%
  slice(1) %>%
  ungroup() %>%
  count(player) %>%
  rename(no_matches_played = n) %>%
  arrange(desc(no_matches_played))
matches_players %>% head(150)
```

```
## # A tibble: 150 x 2
##    player          no_matches_played
##    <chr>                       <int>
##  1 SK Raina                      189
##  2 RG Sharma                     182
##  3 MS Dhoni                      170
```

```
##  4 RV Uthappa                    170
##  5 V Kohli                       170
##  6 KD Karthik                    162
##  7 RA Jadeja                     162
##  8 YK Pathan                     161
##  9 Harbhajan Singh               160
## 10 S Dhawan                      158
## # ... with 140 more rows
```

All these players are top IPL players and having been playing in almost all seasons. Besides these players belong to the top teams (teams that played more matches), indicating they had chance to play more matches because of their team's higher success.

As mentioned, in T20 matches batsman's ability to hit quick runs matters a lot. That means scoring maximum number of 6s and 4s is very important. Besides, big hitting in terms of maximum number of 6s and 4s is a major thriller for crowds and that is a major attraction in the game. The below table lists out the top 20 players who had hit maximum number of 6s and 4s in IPL.

```
# Top_20 batsmen who hit maximum 6s and 4s
del_ds %>%
  filter(role == "batsman" & runs == 6 | runs == 4) %>%
  group_by(player, runs) %>%
  summarize(n = n()) %>%
  spread(runs, n) %>%
  rename(sixes = `6`,fours = `4`) %>%
  select(sixes, fours) %>%
  arrange(desc(sixes)) %>%
  head(20)
```

```
## # A tibble: 20 x 3
## # Groups:   player [20]
##    player         sixes fours
##    <chr>          <int> <int>
##  1 CH Gayle         327   425
##  2 AB de Villiers   214   357
##  3 MS Dhoni         207   297
##  4 SK Raina         195   566
##  5 RG Sharma        194   458
##  6 V Kohli          191   507
##  7 DA Warner        181   459
##  8 SR Watson        177   637
##  9 KA Pollard       175   321
## 10 YK Pathan        161   363
## 11 RV Uthappa       156   436
## 12 Yuvraj Singh     149   277
## 13 BB McCullum      129   293
## 14 AT Rayudu        120   278
## 15 AD Russell       119   252
## 16 DR Smith         117   320
## 17 V Sehwag         106   353
## 18 KD Karthik       101   358
## 19 S Dhawan          96   533
## 20 RR Pant           94   154
```

All these are top batsmen in IPL in all seasons. Naturally they are big hitters and big time idols for fans.

Next, we see the contribution of the top 20% (approx) of players (as both batsmen & bowlers) to the overall runs scored and dismissals in IPL in all seasons.

```
# Overall runs scored in all IPL matches
oa_IPL_runs <- del_ds %>%
  filter(role == "batsman") %>%
  summarize(overall_runs = sum(total_runs))
oa_IPL_runs
```

```
##   overall_runs
## 1       235290
```

```
# Top_100 (~ 20%) batsmen with max runs
t_bat_runs <- del_ds %>%
  filter(role == "batsman") %>%
  group_by(player) %>%
  summarize(tot_runs = sum(runs))

t100_bat_runs <- t_bat_runs%>%
  arrange(desc(tot_runs)) %>%
  head(n = 100)
t100_bat_runs
```

```
## # A tibble: 100 x 2
##    player         tot_runs
##    <chr>             <int>
##  1 V Kohli            5434
##  2 SK Raina           5415
##  3 RG Sharma          4914
##  4 DA Warner          4741
##  5 S Dhawan           4632
##  6 CH Gayle           4560
##  7 MS Dhoni           4477
##  8 RV Uthappa         4446
##  9 AB de Villiers     4428
## 10 G Gambhir          4223
## # ... with 90 more rows
```

```
# Top_100 (~20%) batsmen contribution to overall runs scored in IPL
percent(sum(t100_bat_runs$tot_runs)/oa_IPL_runs$overall_runs)
```

```
## [1] 78.15%
```

```
# Overall wicket dismissals in all IPL matches
oa_IPL_wickets <- del_ds %>%
  filter(role == "bowler" & dismissal_kind != "") %>%
  summarize(overall_wickets = n())
oa_IPL_wickets
```

```
##   overall_wickets
## 1            8834
```

```r
# Top_100 (~20%) bowlers with max wickets
t_bowl_wickets <- del_ds %>%
  filter(
    role == "bowler" &
      dismissal_kind != "obstructing the field" &
      dismissal_kind != "" &
      dismissal_kind != "retired hurt" & dismissal_kind != "run out"
  ) %>%
  group_by(player) %>%
  summarize(tot_wickets = n())

t100_bowl_wickets <- t_bowl_wickets %>%
  arrange(desc(tot_wickets)) %>%
  head(n = 100)
t100_bowl_wickets
```

```
## # A tibble: 100 x 2
##    player          tot_wickets
##    <chr>                 <int>
##  1 SL Malinga              170
##  2 A Mishra                156
##  3 Harbhajan Singh         150
##  4 PP Chawla               149
##  5 DJ Bravo                147
##  6 B Kumar                 133
##  7 R Ashwin                125
##  8 SP Narine               122
##  9 UT Yadav                119
## 10 RA Jadeja               108
## # ... with 90 more rows
```

```r
# Top_100 (~20%) bowler contribution to overall wicket dismissals in IPL
percent(sum(t100_bowl_wickets$tot_wickets)/oa_IPL_wickets$overall_wickets)
```

```
## [1] 68.52%
```

As we can see the top 20% of players (batsmen) contributed to nearly 80% of runs scored in IPL. similarly the top 20% of players (bowlers) contributed to nearly 70% of all wickets dismissed in IPL. Here, we see Pareto's principal of 80:20 gaining ground.

This is also true given that the top players also are the top players who have batted or bowled maximum balls as shown in the following tables. It is natural that the more balls a batsman faces, he scores more runs on an average, similarly the more a bowler bowls, the more wickets that he gets.

```r
# Top_20 batsmen who faced maximum balls
top_bat_max_balls <- del_ds %>%
  filter(role == "batsman") %>%
  group_by(player) %>%
  summarize(tot_n_balls = n()) %>%
  arrange(desc(tot_n_balls))

top_bat_max_balls %>%
  head(n = 20)
```

```
## # A tibble: 20 x 2
##    player         tot_n_balls
##    <chr>                <int>
##  1 V Kohli               4211
##  2 SK Raina              4044
##  3 RG Sharma             3816
##  4 S Dhawan              3776
##  5 G Gambhir             3524
##  6 RV Uthappa            3492
##  7 DA Warner             3398
##  8 MS Dhoni              3318
##  9 AM Rahane             3215
## 10 CH Gayle              3131
## 11 AB de Villiers        2977
## 12 KD Karthik            2890
## 13 AT Rayudu             2681
## 14 SR Watson             2639
## 15 PA Patel              2444
## 16 MK Pandey             2425
## 17 YK Pathan             2334
## 18 JH Kallis             2291
## 19 BB McCullum           2272
## 20 Yuvraj Singh          2207
```

```
# Top_20 bowlers who bowled maximum no. of balls
top_bowl_max_balls <- del_ds %>%
  filter(role == "bowler") %>%
  group_by(player) %>%
  summarize(tot_n_balls = n()) %>%
  arrange(desc(tot_n_balls))

top_bowl_max_balls %>%
  head(n = 20)
```

```
## # A tibble: 20 x 2
##    player          tot_n_balls
##    <chr>                 <int>
##  1 Harbhajan Singh        3451
##  2 A Mishra               3172
##  3 PP Chawla              3157
##  4 R Ashwin               3016
##  5 SL Malinga             2974
##  6 DJ Bravo               2711
##  7 B Kumar                2707
##  8 P Kumar                2637
##  9 UT Yadav               2605
## 10 SP Narine              2600
## 11 RA Jadeja              2541
## 12 Z Khan                 2276
## 13 DW Steyn               2207
## 14 R Vinay Kumar          2186
## 15 SR Watson              2137
## 16 IK Pathan              2113
## 17 I Sharma               1999
```

```
## 18 A Nehra                1974
## 19 PP Ojha                1945
## 20 RP Singh               1874
```

Next, we will see who are the most expensive bowlers meaning who have conceded maximum runs.

```
# Top_20 bowlers who conceded maximum runs
del_ds %>%
  filter(role == "bowler") %>%
  group_by(player) %>%
  summarize(tot_runs = sum(runs), balls_bowled=n()) %>%
  arrange(desc(tot_runs)) %>%
  head(n = 20)
```

```
## # A tibble: 20 x 3
##    player          tot_runs balls_bowled
##    <chr>              <int>        <int>
##  1 PP Chawla           4022         3157
##  2 Harbhajan Singh     3880         3451
##  3 A Mishra            3727         3172
##  4 DJ Bravo            3532         2711
##  5 UT Yadav            3421         2605
##  6 R Ashwin            3224         3016
##  7 SL Malinga          3218         2974
##  8 RA Jadeja           3117         2541
##  9 P Kumar             3106         2637
## 10 B Kumar             3067         2707
## 11 R Vinay Kumar       2927         2186
## 12 SP Narine           2825         2600
## 13 Z Khan              2691         2276
## 14 SR Watson           2580         2137
## 15 IK Pathan           2569         2113
## 16 I Sharma            2488         1999
## 17 A Nehra             2422         1974
## 18 MM Sharma           2398         1770
## 19 DS Kulkarni         2316         1830
## 20 YS Chahal           2300         1841
```

Though the above bowlers are generally well known and good bowlers, they have conceded many runs because of the fact that they have bowled too many balls over the seasons (note the number of balls they bowled). Naturally the more one bowls the more runs one gives away.

So, in order to find out player value, we shall consider strike rate for batsman (average number of runs scored per ball; the more the better). And, for bowler, we shall consider economy rate (average number of runs given per ball ; the lesser the better).

Here are tables with information on the top 20 batsmen in terms of strike rate, top 20 bowlers in terms of economy rate, top 20 bowlers in terms wicket strike rate (average number of balls bowled to take a wicket) and top 20 bowlers who conceded minimum runs.

```
# Top_20 batsmen with max strike_rate
del_ds %>%
  filter(role == "batsman") %>%
  group_by(player) %>%
```

```
  summarize(runs_scored = sum(runs), balls_batted=n(), strike_rate = sum(runs) / n()) %>%
  arrange(desc(strike_rate)) %>%
  head(n = 20)
```

```
## # A tibble: 20 x 4
##    player          runs_scored balls_batted strike_rate
##    <chr>                 <int>        <int>       <dbl>
##  1 B Stanlake                5            2        2.5
##  2 Umar Gul                 39           19        2.05
##  3 RS Sodhi                  4            2        2
##  4 S Sharma                  8            4        2
##  5 AD Russell             1445          803        1.80
##  6 Shahid Afridi            81           46        1.76
##  7 I Malhotra                7            4        1.75
##  8 S Curran                 95           55        1.73
##  9 K Gowtham               148           86        1.72
## 10 SN Thakur                36           21        1.71
## 11 M Ali                   311          183        1.70
## 12 LJ Wright               106           63        1.68
## 13 SP Narine               803          481        1.67
## 14 KMDN Kulasekara           5            3        1.67
## 15 Shivam Sharma             5            3        1.67
## 16 KK Cooper               116           70        1.66
## 17 BCJ Cutting             240          146        1.64
## 18 Kamran Akmal            128           78        1.64
## 19 BJ Haddin                18           11        1.64
## 20 Rashid Khan              109           67        1.63
```

```
# Top_20 bowlers with best economy rate
del_ds %>%
  filter(role == "bowler") %>%
  group_by(player) %>%
  summarize(runs_given = sum(runs), balls_bowled=n(), economy_rate = sum(runs) / n()) %>%
  arrange(economy_rate) %>%
  head(n = 20)
```

```
## # A tibble: 20 x 4
##    player          runs_given balls_bowled economy_rate
##    <chr>                <int>        <int>        <dbl>
##  1 AC Gilchrist             0            1        0
##  2 NB Singh                14           25        0.56
##  3 LA Carseldine            5            7        0.714
##  4 SS Mundhe                5            7        0.714
##  5 S Dube                   8           11        0.727
##  6 A Roy                   11           15        0.733
##  7 Sachin Baby              8           10        0.8
##  8 AM Rahane                5            6        0.833
##  9 DJ Thornely             38           44        0.864
## 10 SM Harwood              61           67        0.910
## 11 Sohail Tanvir          246          265        0.928
## 12 FH Edwards             144          150        0.96
## 13 LH Ferguson             84           87        0.966
## 14 Mohammad Hafeez         61           63        0.968
```

```
## 15 L Ngidi                162          163        0.994
## 16 GH Vihari               37           37        1
## 17 M Manhas                42           42        1
## 18 MJ Clarke               67           66        1.02
## 19 JW Hastings             63           61        1.03
## 20 A Chandila             242          234        1.03
```

```r
# Top_20 bowlers with best wicket strike rates
top_bowl_max_balls %>%
  full_join(t_bowl_wickets, by = "player") %>%
  mutate(strike_rate = tot_n_balls/ tot_wickets) %>%
  arrange(strike_rate) %>%
  head(20)
```

```
## # A tibble: 20 x 4
##    player        tot_n_balls tot_wickets strike_rate
##    <chr>               <int>       <int>       <dbl>
##  1 AC Gilchrist            1           1        1
##  2 Sachin Baby            10           2        5
##  3 AM Rahane               6           1        6
##  4 LA Carseldine           7           1        7
##  5 SS Mundhe               7           1        7
##  6 DAJ Bracewell          25           3        8.33
##  7 A Joseph               55           6        9.17
##  8 Shoaib Akhtar          46           5        9.2
##  9 T Curran               64           6       10.7
## 10 D du Preez             43           4       10.8
## 11 A Zampa               225          19       11.8
## 12 R Ninan                36           3       12
## 13 Sohail Tanvir         265          22       12.0
## 14 S Dhawan               49           4       12.2
## 15 O Thomas               63           5       12.6
## 16 KM Asif                38           3       12.7
## 17 K Ahmed               241          19       12.7
## 18 CK Langeveldt         165          13       12.7
## 19 Umar Gul              153          12       12.8
## 20 S Vidyut               13           1       13
```

```r
# Top_20 bowlers who conceded minimum runs
del_ds %>%
  filter(role == "bowler") %>%
  group_by(player) %>%
  summarize(tot_runs = sum(runs),balls_bowled=n()) %>%
  arrange(tot_runs) %>%
  head(n = 20)
```

```
## # A tibble: 20 x 3
##    player           tot_runs balls_bowled
##    <chr>               <int>        <int>
##  1 AC Gilchrist            0            1
##  2 AM Rahane               5            6
##  3 LA Carseldine           5            7
##  4 SPD Smith               5            2
```

```
##  5 SS Mundhe               5            7
##  6 SN Khan                 6            2
##  7 KS Williamson           7            6
##  8 Y Gnaneswara Rao        7            6
##  9 RS Gavaskar             8            6
## 10 S Dube                  8           11
## 11 SA Yadav                8            6
## 12 Sachin Baby             8           10
## 13 H Vihari                9            6
## 14 A Roy                  11           15
## 15 BJ Rohrer              11            7
## 16 RA Shaikh              11            6
## 17 RA Tripathi            12            6
## 18 C Ganapathy            13            6
## 19 L Livingstone          13            6
## 20 NB Singh               14           25
```

These are not well known players for their respective skills in batting or bowling. However, they figure in the top 20 just because they batted or bowled too few balls giving significant rates. The few runs scored/ conceded or the few wickets taken really will not matter given the very few balls. We will correct this a bit later using the well known regularizing technique. By the way, though interesting to note, we will not use wicket strike rates and minimum runs given by bowlers per se in our calculations.

In order to start building our models, let us next see which are top scoring teams, winning teams and losing teams.

```
# Total runs scored by each team
deliveries %>%
  select(batting_team, total_runs) %>%
  group_by(team = batting_team) %>%
  summarize(tot_runs = sum(total_runs)) %>%
  arrange(desc(tot_runs))
```

```
## # A tibble: 13 x 2
##    team                         tot_runs
##    <chr>                           <int>
##  1 Mumbai Indians                  29809
##  2 Royal Challengers Bangalore     28126
##  3 Kings XI Punjab                 27893
##  4 Kolkata Knight Riders           27419
##  5 Delhi Capitals                  27018
##  6 Chennai Super Kings             26418
##  7 Rajasthan Royals                22431
##  8 Sunrisers Hyderabad             17059
##  9 Deccan Chargers                 11463
## 10 Pune Warriors                    6358
## 11 Gujarat Lions                    4862
## 12 Rising Pune Supergiant           4533
## 13 Kochi Tuskers Kerala             1901
```

```
# Which team won how many matches
winners <- mat_ds %>%
  group_by(winner) %>%
  filter(winner != "") %>%
```

```
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  rename(team = winner, n_matches_won = count)
winners
```

```
## # A tibble: 13 x 2
##    team                         n_matches_won
##    <chr>                                <int>
##  1 Mumbai Indians                         109
##  2 Chennai Super Kings                    100
##  3 Kolkata Knight Riders                   92
##  4 Royal Challengers Bangalore             84
##  5 Kings XI Punjab                         82
##  6 Delhi Capitals                          77
##  7 Rajasthan Royals                        75
##  8 Sunrisers Hyderabad                     58
##  9 Deccan Chargers                         29
## 10 Rising Pune Supergiant                  15
## 11 Gujarat Lions                           13
## 12 Pune Warriors                           12
## 13 Kochi Tuskers Kerala                     6
```

```
# Which team lost how many matches
lost1 <- mat_ds %>%
  filter(winner != "") %>%
  filter(as.character(winner) != as.character(team1)) %>%
  group_by(team1) %>%
  summarize(count1 = n()) %>%
  arrange(team1) %>%
  rename(team = team1)

lost2 <- mat_ds %>%
  filter(winner != "") %>%
  filter(as.character(winner) != as.character(team2)) %>%
  group_by(team2) %>%
  summarize(count2 = n()) %>%
  arrange(team2) %>%
  rename(team = team2)

losers <- lost1 %>%
  full_join(lost2, by = "team") %>%
  mutate(n_matches_lost = count1 + count2) %>%
  select(-count1,-count2) %>%
  arrange(desc(n_matches_lost))
losers
```

```
## # A tibble: 13 x 2
##    team                         n_matches_lost
##    <chr>                                 <int>
##  1 Delhi Capitals                           98
##  2 Kings XI Punjab                          94
##  3 Royal Challengers Bangalore              93
##  4 Kolkata Knight Riders                    86
```

```
##  5 Mumbai Indians                         78
##  6 Rajasthan Royals                       70
##  7 Chennai Super Kings                    64
##  8 Sunrisers Hyderabad                    50
##  9 Deccan Chargers                        46
## 10 Pune Warriors                          33
## 11 Gujarat Lions                          17
## 12 Rising Pune Supergiant                 15
## 13 Kochi Tuskers Kerala                    8
```

Naturally, the top 3 teams among the winners, Mumbai Indians, Chennai Super Kings & Kolkata Knight Riders are the top 3 teams with maximum IPL championship wins, in that order.

Similarly, Delhi Capitals, Kings XI Punjab & Royal Challengers Bangalore, the top 3 in losers group, are the top 3 unlucky teams, who haven't won the IPL championship even once. But being the next top teams, they have played more matches than other teams.

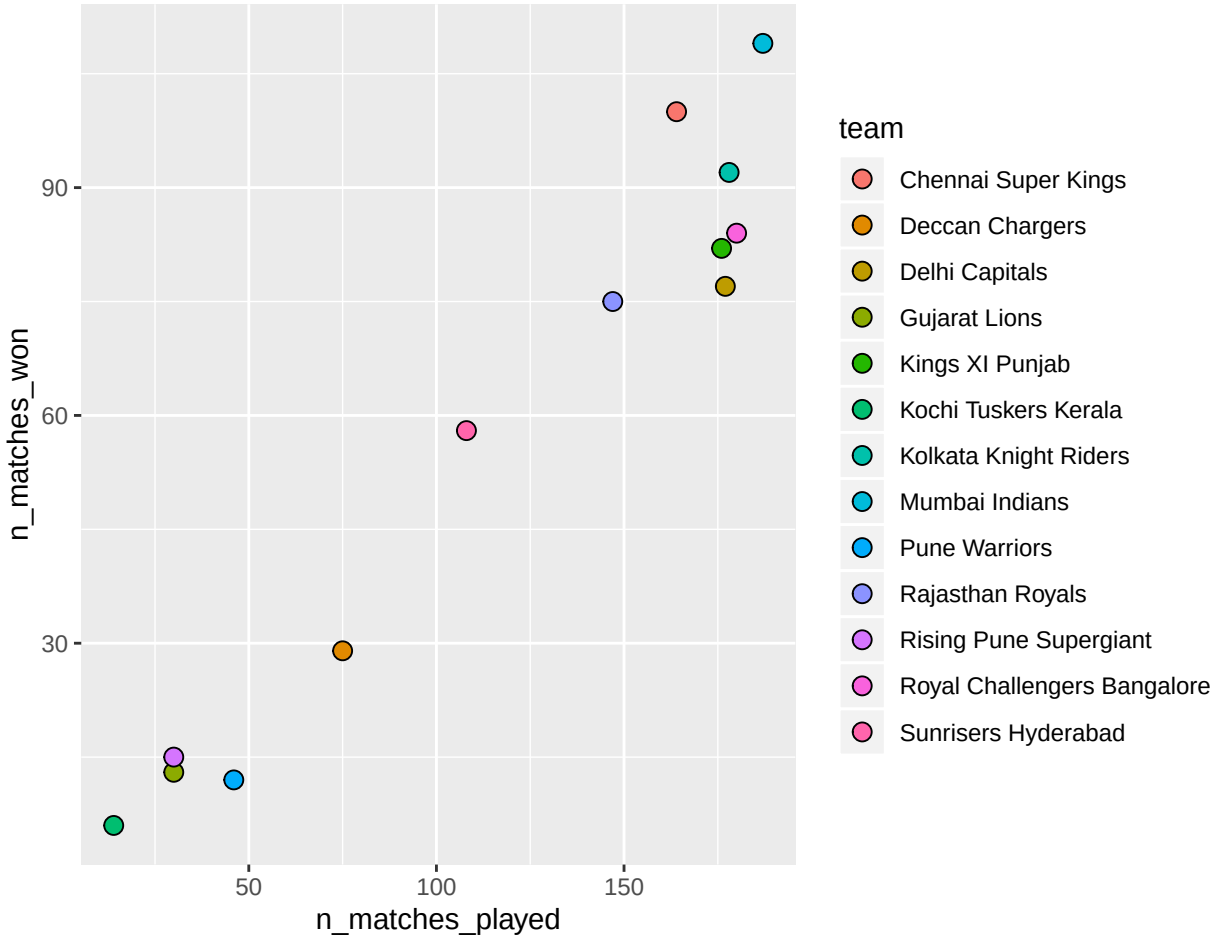
Let us examine the wins and losses of teams by visualizing the data. First, we will check which teams won how many tosses. Then we will check if there is a relationship between number of matches played, number of tosses won and number of matches won.

```
# Won/ lost plots
# Which team won how many tosses
toss_winners <- mat_ds %>%
  group_by(toss_winner) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  rename(team = toss_winner, n_tosses_won = count)
toss_winners
```

```
## # A tibble: 13 x 2
##    team                         n_tosses_won
##    <chr>                               <int>
##  1 Mumbai Indians                         98
##  2 Kolkata Knight Riders                  92
##  3 Delhi Capitals                         90
##  4 Chennai Super Kings                    89
##  5 Kings XI Punjab                        81
##  6 Royal Challengers Bangalore            81
##  7 Rajasthan Royals                       80
##  8 Sunrisers Hyderabad                    46
##  9 Deccan Chargers                        43
## 10 Pune Warriors                          20
## 11 Gujarat Lions                          15
## 12 Rising Pune Supergiant                 13
## 13 Kochi Tuskers Kerala                    8
```

```
teams <- matches_team %>%
  right_join(winners, by = "team") %>%
  right_join(losers, by = "team") %>%
  right_join(toss_winners, by = "team")

# correlation between match-wins and matches played
teams %>%
```

```
  ggplot(aes(n_matches_played, n_matches_won, fill = team)) +
  geom_point(shape=21, size=3)
```



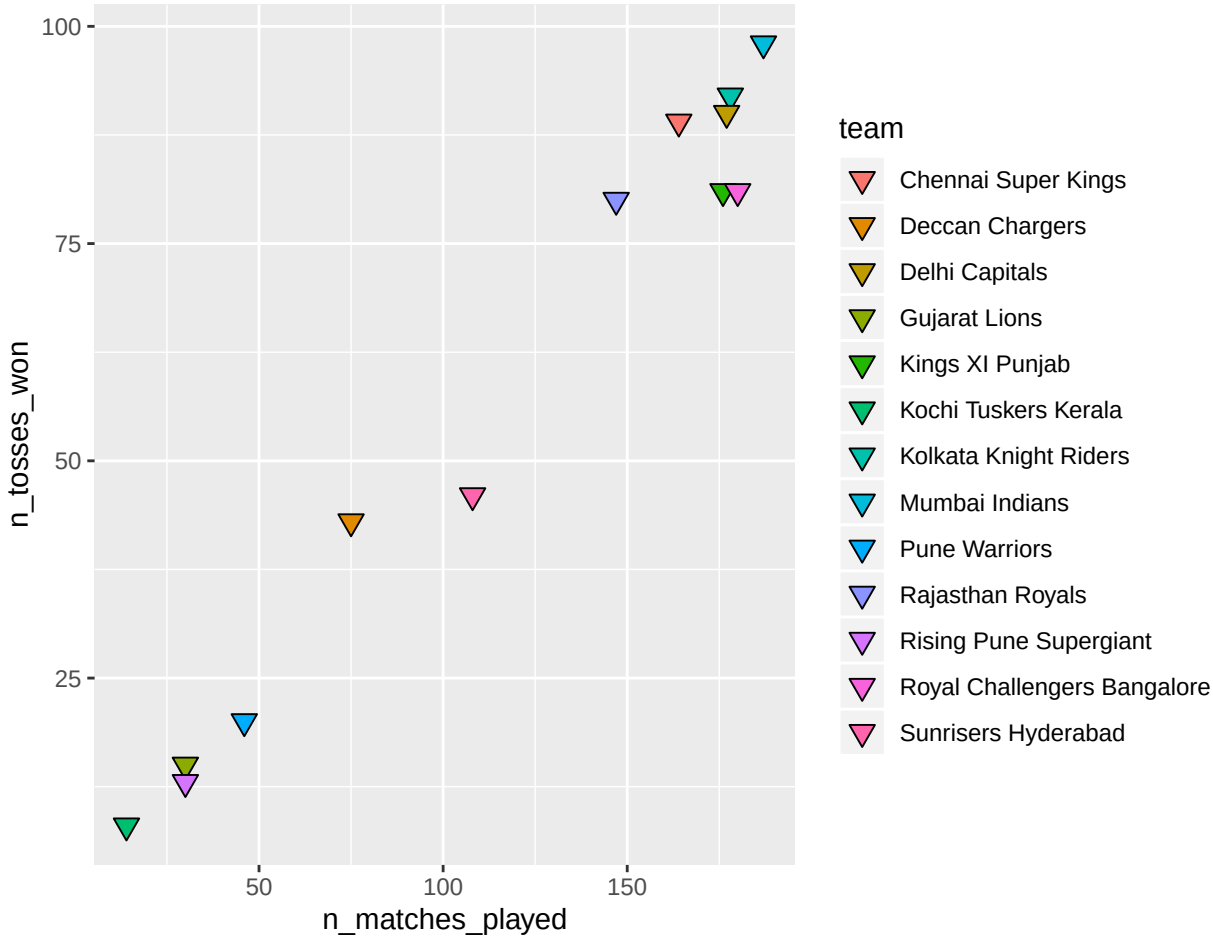

```
cor(teams$n_matches_played, teams$n_matches_won)
```

```
## [1] 0.9727328
```

It is natural that the more matches a team plays that much more chances it has, to win matches. We can see there are a few team which have played a few matches and won a very few. However, we can see a bunch of teams played similar number of matches (about 175 to 180), but the number of matches won by them varies significantly. This is because till knock out stage all team get to play equal number of matches; however, at knock out stage only 4 teams get to play more matches. The more a team wins, it gets more matches to play.

Now, let us see if there is a correlation between match wins and toss wins and also between number of matches played and toss wins.

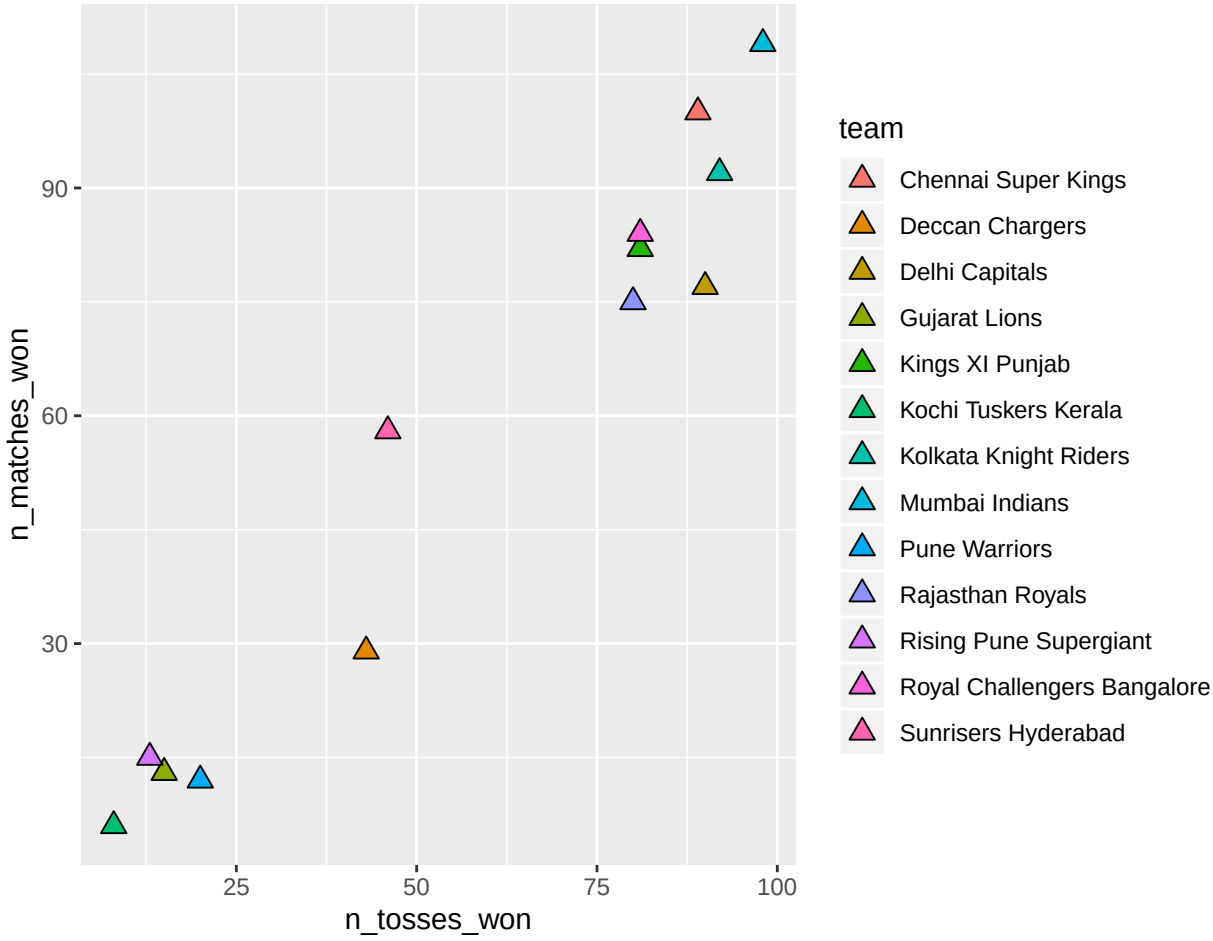
```
# correlation between matches played and toss-wins
teams %>%
  ggplot(aes(n_matches_played, n_tosses_won, fill = team)) +
  geom_point(shape=25, size = 3)
```




```
cor(teams$n_matches_played, teams$n_tosses_won)
```

```
## [1] 0.9874886
```

```
# correlation between match-wins and toss-wins
teams %>%
  ggplot(aes(n_tosses_won, n_matches_won, fill = team)) +
  geom_point(shape=24, size = 3)
```




```
cor(teams$n_matches_won, teams$n_tosses_won)
```

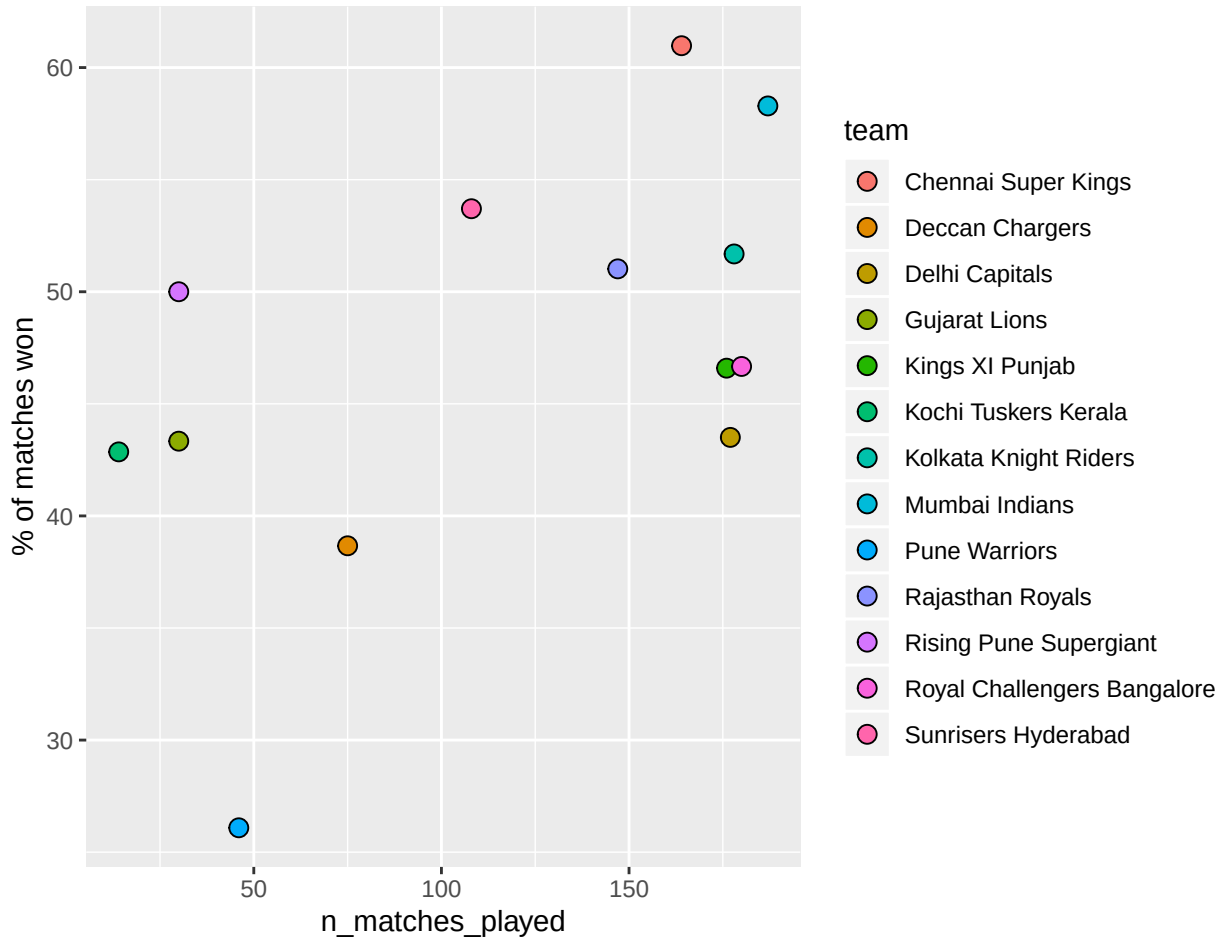
```
## [1] 0.9756731
```

It is understandable that the more matches that a team plays, the more chances it has to win the toss, hence the high positive correlation. In the 1st plot, we can also see all the teams won the toss about 50% of time as toss is a pure chance event and is always 50%:50%.

The 2nd plot, indicates an interesting fact. The team that had won more tosses also won more matches with a high positive correlation. There shall be an obvious reason which we will demonstrate a bit later. However, to reveal the reason, it is that teams which win toss, usually opt for fielding (batting 2nd) as that generally leads to the toss winning team winning the match.

Now, let us check the win % of the teams.

```
# correlation between win% & matches played
teams %>%
  mutate(matches_won_percent = n_matches_won * 100 / n_matches_played) %>%
  ggplot(aes(matches_won_percent, n_matches_played, fill =
               team)) +
  geom_point(shape=21, size = 3) +
  xlab("% of matches won") +
  coord_flip()
```




```
cor(teams$n_matches_won * 100 / teams$n_matches_played,
    teams$n_matches_played)
```

```
## [1] 0.527997
```

Playing more number of matches might give teams more wins compared to team that have played lesser number of matches. But the correlation indicates that win % eventually regresses towards 50% mark. From the plot, we can make out that for established teams (which have played large number of matches), the win % varies between 40% and 60%.

Now, getting back to the toss subject, team that wins the toss will select either to bat or to field first, the other team accordingly ends up taking the other role. The roles will be reversed after the first team finishes its batting innings. Let us see which team won how many tosses and how the decision of batting first or fielding first impacted wins.

```
# Toss wins by different teams
toss_wins <- mat_ds %>%
  filter(winner != "") %>%
  group_by(toss_winner) %>%
  summarize(t_wins = n())
toss_wins
```

```
## # A tibble: 13 x 2
```

```
##    toss_winner                t_wins
##    <chr>                       <int>
##  1 Chennai Super Kings            89
##  2 Deccan Chargers                43
##  3 Delhi Capitals                 89
##  4 Gujarat Lions                  15
##  5 Kings XI Punjab                81
##  6 Kochi Tuskers Kerala            8
##  7 Kolkata Knight Riders          92
##  8 Mumbai Indians                 98
##  9 Pune Warriors                  20
## 10 Rajasthan Royals               78
## 11 Rising Pune Supergiant         13
## 12 Royal Challengers Bangalore    80
## 13 Sunrisers Hyderabad            46
```

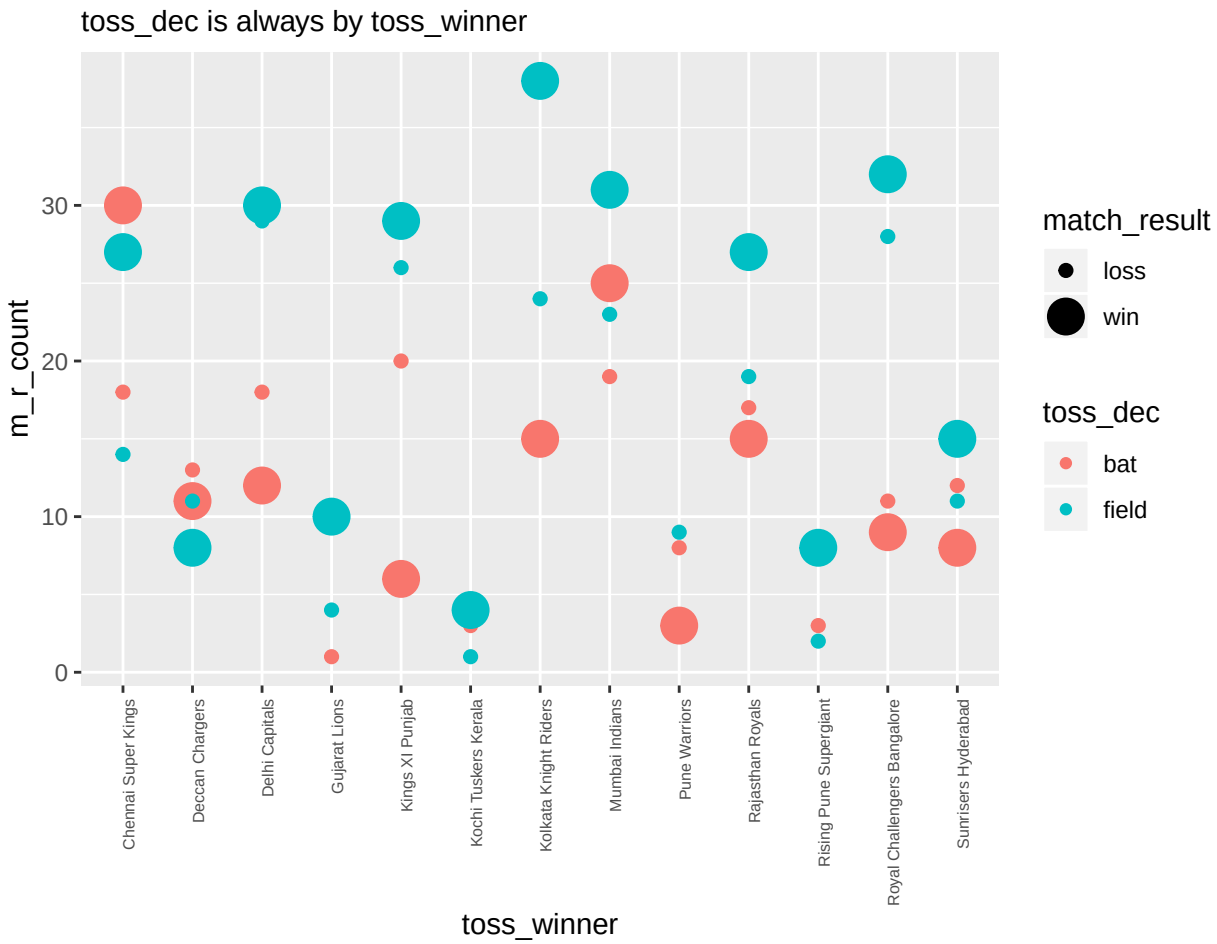
```
# Toss wins Vs match wins or losses based on batted/ fielded first
toss_wins_results <- mat_ds %>%
  filter(winner != "") %>%
  mutate(match_result = ifelse(toss_winner==winner, "win", "loss")) %>%
  group_by(toss_winner, toss_dec, match_result) %>%
  summarize(m_r_count = n())

toss_wins_results_prcnts <- toss_wins_results %>%
  spread(match_result, m_r_count) %>%
  full_join(toss_wins, by = "toss_winner") %>%
  mutate(win_prcnt=win*100/(t_wins),
         loss_prcnt= loss*100/(t_wins)) %>%
  select(toss_winner, t_wins, toss_dec, win, win_prcnt, loss, loss_prcnt)

toss_wins_results_prcnts
```

```
## # A tibble: 26 x 7
## # Groups:   toss_winner, toss_dec [26]
##    toss_winner         t_wins toss_dec   win win_prcnt  loss loss_prcnt
##    <chr>                <int> <chr>    <int>     <dbl> <int>      <dbl>
##  1 Chennai Super Kings     89 bat         30     33.7     18      20.2
##  2 Chennai Super Kings     89 field       27     30.3     14      15.7
##  3 Deccan Chargers         43 bat         11     25.6     13      30.2
##  4 Deccan Chargers         43 field        8     18.6     11      25.6
##  5 Delhi Capitals          89 bat         12     13.5     18      20.2
##  6 Delhi Capitals          89 field       30     33.7     29      32.6
##  7 Gujarat Lions           15 bat         NA     NA        1       6.67
##  8 Gujarat Lions           15 field       10     66.7      4      26.7
##  9 Kings XI Punjab         81 bat          6      7.41    20      24.7
## 10 Kings XI Punjab         81 field       29     35.8     26      32.1
## # ... with 16 more rows
```

```
toss_wins_results %>%
  ggplot(aes(toss_winner, m_r_count, col = toss_dec, size = match_result)) +
  geom_point()  +
  theme(axis.text.x = element_text(
    angle = 90,
```

```
    size = 6,
    hjust = 1
  ),
  legend.position = "right") +
  labs(subtitle="toss_dec is always by toss_winner")
```

```
## Warning: Using size for a discrete variable is not advised.
```



From the above plot, we can observe the following.

Looking at Blue dots, we can say that when teams after winning the toss, have selected to field first, most of the teams seem to have won (big dots) more matches.

Similarly, looking at Red dots, we can say that when teams after winning the toss, have selected to bat first, most of the teams seem to have lost (small dots) more matches.

Let us see what happened when teams have lost tosses.

```
# Toss losses by different teams
toss_losses <- mat_ds %>%
  filter(winner != "") %>%
  mutate(toss_loser = ifelse(toss_winner == team1, team2, team1)) %>%
  group_by(toss_loser) %>%
```

```
  summarize(t_losses = n())
toss_losses
```

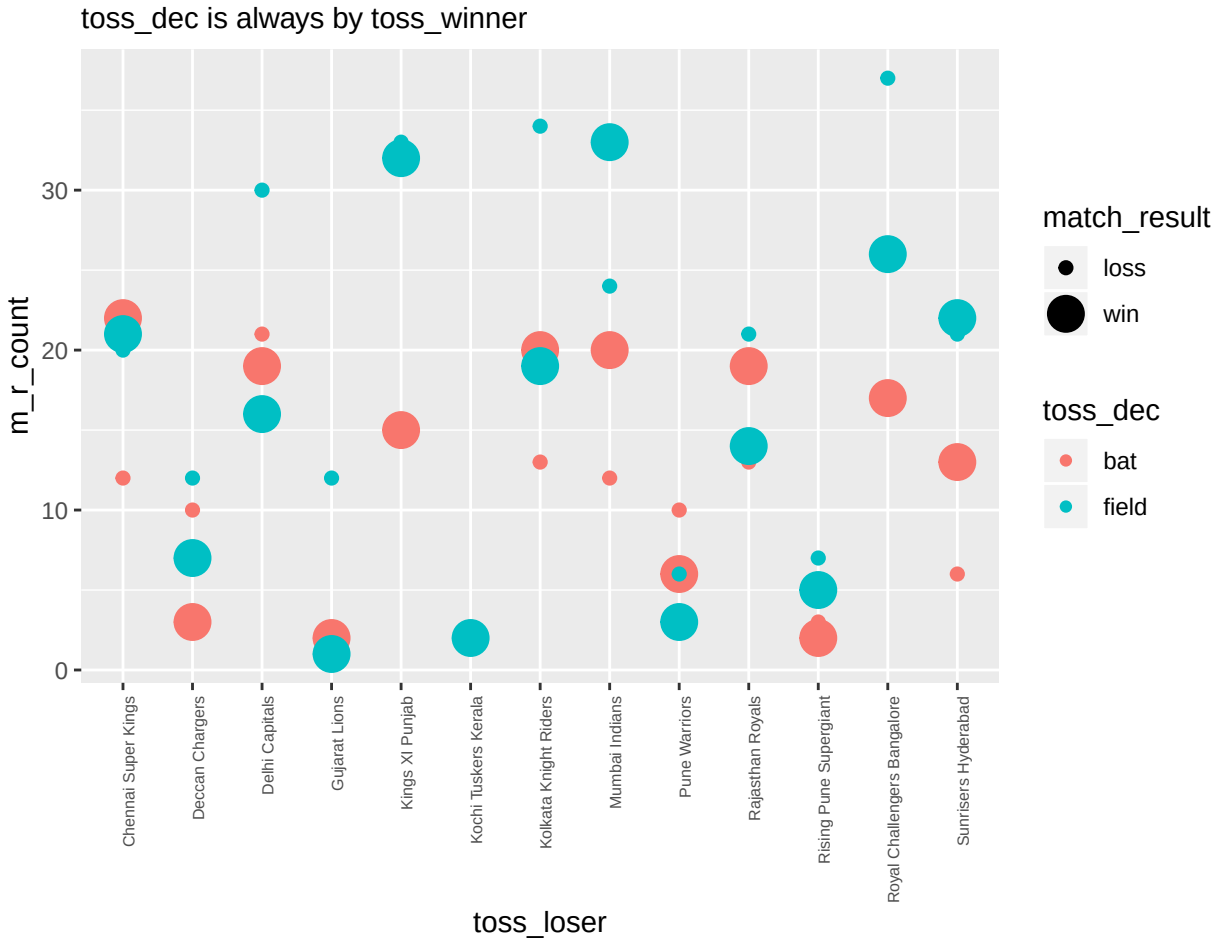
```
## # A tibble: 13 x 2
##    toss_loser                  t_losses
##    <chr>                          <int>
##  1 Chennai Super Kings               75
##  2 Deccan Chargers                   32
##  3 Delhi Capitals                    86
##  4 Gujarat Lions                     15
##  5 Kings XI Punjab                   95
##  6 Kochi Tuskers Kerala               6
##  7 Kolkata Knight Riders             86
##  8 Mumbai Indians                    89
##  9 Pune Warriors                     25
## 10 Rajasthan Royals                  67
## 11 Rising Pune Supergiant            17
## 12 Royal Challengers Bangalore       97
## 13 Sunrisers Hyderabad               62
```

```
# Toss losses Vs match wins or losses based on batted/ fielded first
toss_losses_results <- mat_ds %>%
  filter(winner != "") %>%
  mutate(toss_loser = ifelse(toss_winner==team1, team2, team1)) %>%
  mutate(match_result = ifelse(toss_loser==winner, "win", "loss")) %>%
  group_by(toss_loser, toss_dec, match_result) %>%
  summarize(m_r_count = n())

toss_losses_results_prcnts <- toss_losses_results %>%
  spread (match_result, m_r_count) %>%
  full_join(toss_losses, by = "toss_loser") %>%
  mutate(win_prcnt=win*100/(t_losses),
         loss_prcnt= loss*100/(t_losses))%>%
  select(toss_loser, t_losses, toss_dec, win, win_prcnt, loss, loss_prcnt)

toss_losses_results_prcnts
```

```
## # A tibble: 26 x 7
## # Groups:   toss_loser, toss_dec [26]
##    toss_loser          t_losses toss_dec   win win_prcnt  loss loss_prcnt
##    <chr>                  <int> <chr>    <int>     <dbl> <int>      <dbl>
##  1 Chennai Super Kings       75 bat         22     29.3     12       16
##  2 Chennai Super Kings       75 field       21     28       20       26.7
##  3 Deccan Chargers           32 bat          3      9.38    10       31.2
##  4 Deccan Chargers           32 field        7     21.9     12       37.5
##  5 Delhi Capitals            86 bat         19     22.1     21       24.4
##  6 Delhi Capitals            86 field       16     18.6     30       34.9
##  7 Gujarat Lions             15 bat          2     13.3     NA       NA
##  8 Gujarat Lions             15 field        1      6.67    12       80
##  9 Kings XI Punjab           95 bat         15     15.8     15       15.8
## 10 Kings XI Punjab           95 field       32     33.7     33       34.7
## # ... with 16 more rows
```

```
toss_losses_results %>%
  ggplot(aes(toss_loser, m_r_count, col = toss_dec, size = match_result)) +
  geom_point() +
  theme(axis.text.x = element_text(
    angle = 90,
    size = 6,
    hjust = 1
  ),
  legend.position = "right") +
  labs(subtitle="toss_dec is always by toss_winner")
```

```
## Warning: Using size for a discrete variable is not advised.
```



As earlier, this time too, from the above plot, we can observe the following.

First, we shall note:
Blue dots = toss_winner fielding first = toss_loser batting first and
Red dots = toss_winner batting first = toss_loser fielding first

Looking at Blue dots, we can say that when teams after losing the toss, have been asked to bat first (Blue dot indicates toss winner to field first), most of the teams seem to have lost (big dots) more matches.

Similarly, looking at Red dots, we can say that when teams after losing the toss, have been asked to field first (Red dot indicates toss winner to bat first), most of the teams seem to have won (small dots) more

matches.

From the above two plots, we can clearly conclude that when teams have batted second, most teams seems to have won more matches than lost. This fact influences teams to select fileding first after winning the toss. We can see the evidence for the same from the below tables.

```
# No. of times teams selected to bat or field first after winning toss
mat_ds %>%
  group_by(toss_dec) %>%
  summarize(count = n())
```

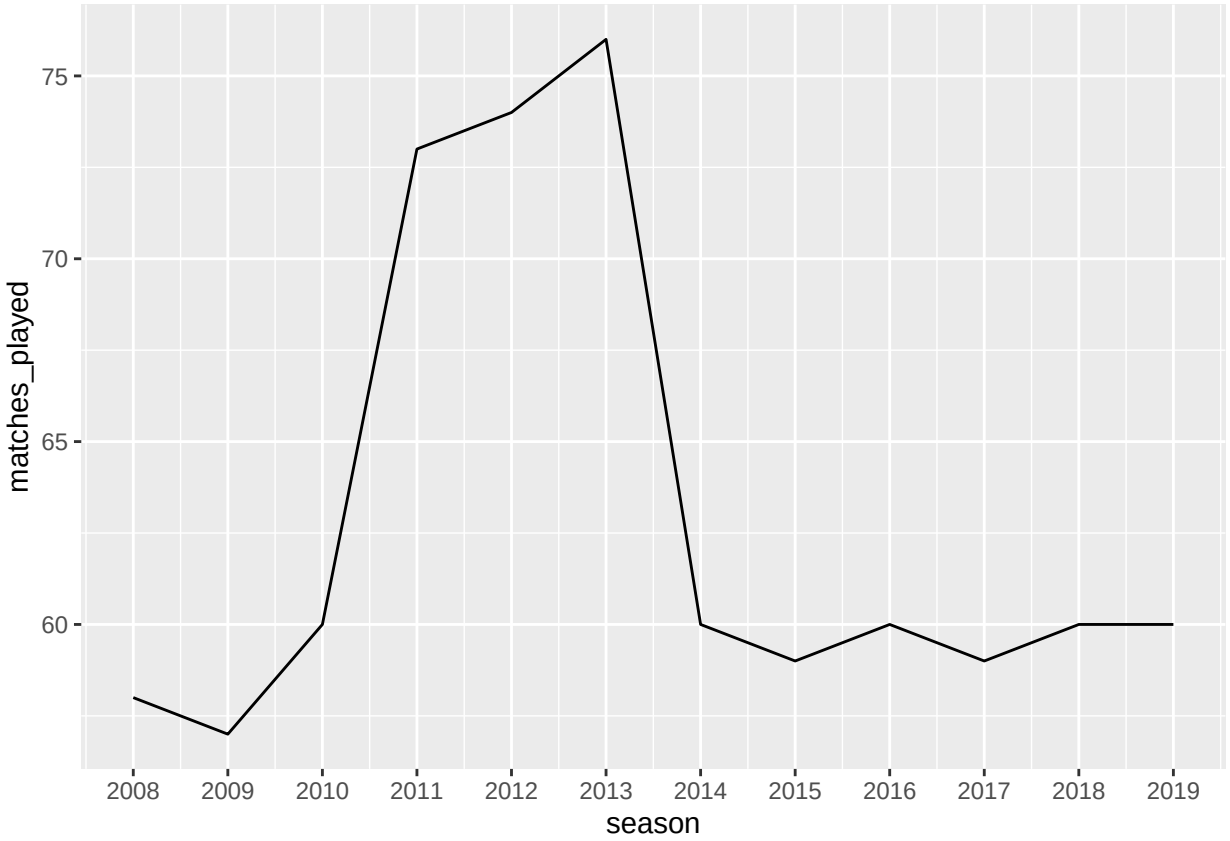
```
## # A tibble: 2 x 2
##   toss_dec count
##   <chr>    <int>
## 1 bat        293
## 2 field      463
```

```
# No. of times individual teams selected to bat or field first after winning toss
mat_ds %>%
  group_by(toss_winner, toss_dec) %>%
  summarize(count = n()) %>%
  spread(toss_dec, count) %>%
  arrange(desc(field))
```

```
## # A tibble: 13 x 3
## # Groups:   toss_winner [13]
##    toss_winner                  bat field
##    <chr>                      <int> <int>
##  1 Kolkata Knight Riders         30    62
##  2 Royal Challengers Bangalore   20    61
##  3 Delhi Capitals                31    59
##  4 Kings XI Punjab               26    55
##  5 Mumbai Indians                44    54
##  6 Rajasthan Royals              32    48
##  7 Chennai Super Kings           48    41
##  8 Sunrisers Hyderabad           20    26
##  9 Deccan Chargers               24    19
## 10 Gujarat Lions                  1    14
## 11 Rising Pune Supergiant         3    10
## 12 Pune Warriors                 11     9
## 13 Kochi Tuskers Kerala           3     5
```

Next, let us see how many matches were played over the seasons. We will also see how many matches teams have won batting first and batting second.

```
# Number of matches played over seasons
matches %>%
  group_by(season) %>%
  summarize(matches_played=n()) %>%
  ggplot(aes(season, matches_played)) +
  geom_line() +
  scale_x_continuous(breaks = seq(2007, 2019, by = 1))
```




```
# How many matches teams won batting first and batting second
# Total no. of matches won batting first across seasons
mat_ds %>%
  filter(team1==winner) %>%
  summarize(count=n())
```

```
##   count
## 1   335
```

```
# Total no. of matches won batting second across seasons
mat_ds %>%
  filter(team2==winner) %>%
  summarize(count=n())
```

```
##   count
## 1   417
```

```
# No. of matches won batting first season-wise
wins_bat_1st <- mat_ds %>%
  filter(team1==winner) %>%
  group_by(season) %>%
  summarize(wins_bat_1st=n())
wins_bat_1st
```

```
## # A tibble: 12 x 2
```

```
##    season wins_bat_1st
##     <int>        <int>
##  1   2008           22
##  2   2009           27
##  3   2010           32
##  4   2011           32
##  5   2012           34
##  6   2013           37
##  7   2014           23
##  8   2015           32
##  9   2016           19
## 10   2017           26
## 11   2018           28
## 12   2019           23
```

```
sum(wins_bat_1st$wins_bat_1st)
```

```
## [1] 335
```

```
# No. of matches won batting second  season-wise
wins_bat_2nd <- mat_ds %>%
  filter(team2==winner) %>%
  group_by(season) %>%
  summarize(wins_bat_2nd=n())
wins_bat_2nd
```

```
## # A tibble: 12 x 2
##    season wins_bat_2nd
##     <int>        <int>
##  1   2008           36
##  2   2009           30
##  3   2010           28
##  4   2011           40
##  5   2012           40
##  6   2013           39
##  7   2014           37
##  8   2015           25
##  9   2016           41
## 10   2017           33
## 11   2018           32
## 12   2019           36
```

```
sum(wins_bat_2nd$wins_bat_2nd)
```

```
## [1] 417
```

```
wins_bat_1st %>%
  inner_join(wins_bat_2nd, by="season") %>%
  gather(batting_turn, wins, wins_bat_1st:wins_bat_2nd) %>%
  ggplot(aes(season, wins, col=batting_turn)) +
  geom_line() +
  scale_x_continuous(breaks = seq(2007, 2019, by = 1))
```

The numbers and the 2nd plot clearly shows that teams batting 2nd have won more matches than teams that have batted first in all but two seasons.

At this point, let us check how many runs were scored batting first and how many were scored batting second in totals and run type wise.

```
# Winning margin
del_ds %>%
  filter(role=="batsman" & inning %in% 1:2) %>%
  group_by(inning) %>%
  summarize(total_runs_scored=sum(runs))
```

```
## # A tibble: 2 x 2
##   inning total_runs_scored
##    <int>             <int>
## 1      1            116525
## 2      2            106565
```

```
del_ds %>%
  group_by(match_id, inning) %>%
  mutate(ball_no = 1:n()) %>%
  ungroup() %>%
  filter(inning %in% 1:2 & runs %in% c(0,1,2,3,4,5,6,7) & ball_no %in% 1:120) %>%
  group_by(inning, batsman_runs=factor(runs)) %>%
  summarise(count=n()) %>%
  spread(inning,count)
```

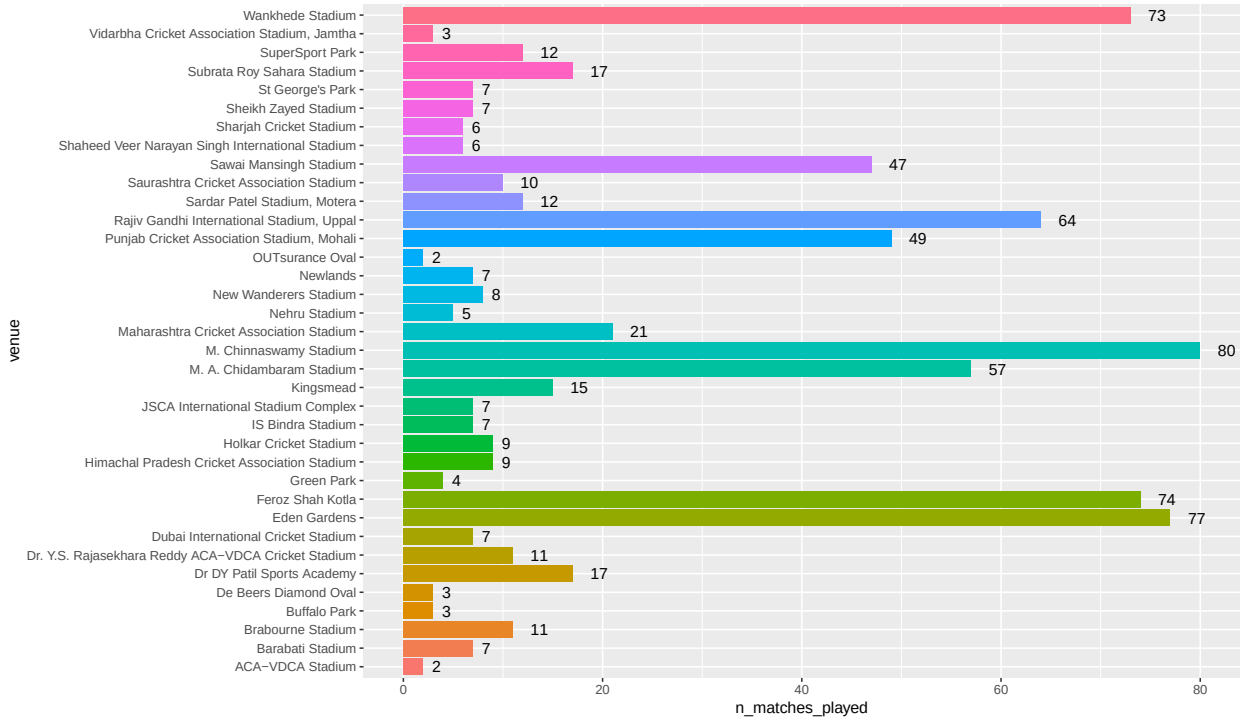
```
## # A tibble: 8 x 3
##   batsman_runs   `1`   `2`
##   <fct>        <int> <int>
## 1 0            35825 36579
## 2 1            34451 33124
## 3 2             5737  5481
## 4 3              290   293
## 5 4            10252 10374
## 6 5               36    47
## 7 6             4048  3914
## 8 7                3     4
```

Here, we see an astonishing fact. Though, we have found out that teams batting 2nd have won more matches than teams batting 1st, the above figures show teams batting 1st have scored much more total runs than teams batting 2nd. In fact, all the totals for run types pretty much indicate teams batting 1st have scored more. These two facts are contrary to each other.

However, a deeper thought gives away the reason. Teams batting 1st may win matches with huge margins if they get opponents out quickly, but teams batting 2nd always win with very thin margins, often just with a run, due to the fact that they need to just cross the target set by the 1st team.

Let us check the number of matches played at each venue and how many matches were played at the top 10 venues.

```
# no.of matches played at each venue
matches_venue <- mat_ds %>%
  group_by(venue) %>%
  summarize(n_matches_played = n()) %>%
  arrange(desc(n_matches_played))

matches_venue %>%
  ggplot(aes(venue, n_matches_played, fill = venue)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  geom_text(aes(label = n_matches_played), hjust = -1)
```




```
# % of matches played at top_10 venues
t10_n_matches_played <- matches_venue %>%
  head(10)
t10_n_matches_played
```

```
## # A tibble: 10 x 2
##    venue                                      n_matches_played
##    <chr>                                                 <int>
##  1 M. Chinnaswamy Stadium                                   80
##  2 Eden Gardens                                             77
##  3 Feroz Shah Kotla                                         74
##  4 Wankhede Stadium                                         73
##  5 Rajiv Gandhi International Stadium, Uppal                64
##  6 M. A. Chidambaram Stadium                                57
##  7 Punjab Cricket Association Stadium, Mohali               49
##  8 Sawai Mansingh Stadium                                   47
##  9 Maharashtra Cricket Association Stadium                  21
## 10 Dr DY Patil Sports Academy                               17
```

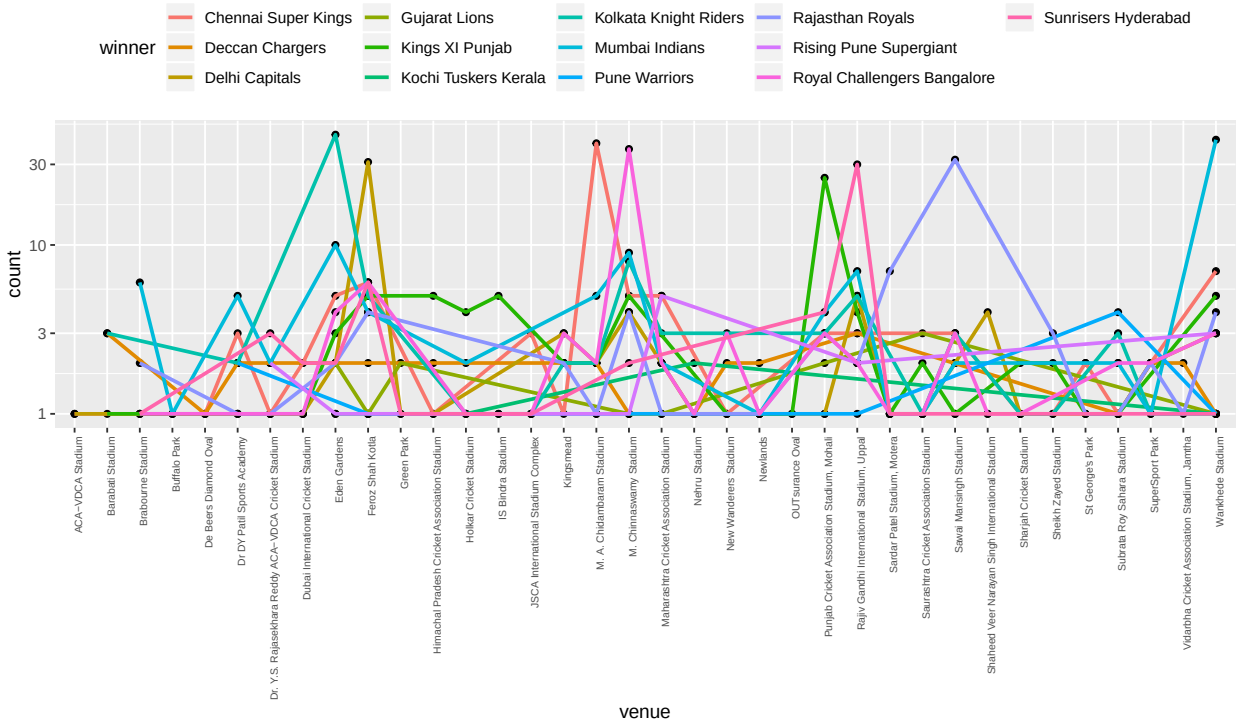
```
percent(sum(t10_n_matches_played$n_matches_played)/total_played)
```

```
## [1] 73.94%
```

We can see that most of the matches were played at India's well known, big stadiums. In fact, the top 10 venues account for nearly 75% of the matches. All these top stadiums are home venues for respective teams naturally suggesting why more matches are played in them. The other reason is that as teams move into knockout stage, most of the matches are limited to these stadiums.

Now, let us check if venue has any effect on team winning matches.

```
# Venue effect on team wins
mat_ds %>%
  filter(winner != "") %>%
  group_by(venue, winner) %>%
  summarize(count = n()) %>%
  ggplot(aes(venue, count, group = winner)) +
  geom_point() +
  geom_line(aes(col = winner), size = 1) +
  scale_y_log10() +
  theme(axis.text.x = element_text(
    angle = 90,
    size = 6,
    hjust = 1
  ),
  legend.position = "top", legend.title.align=0)
```

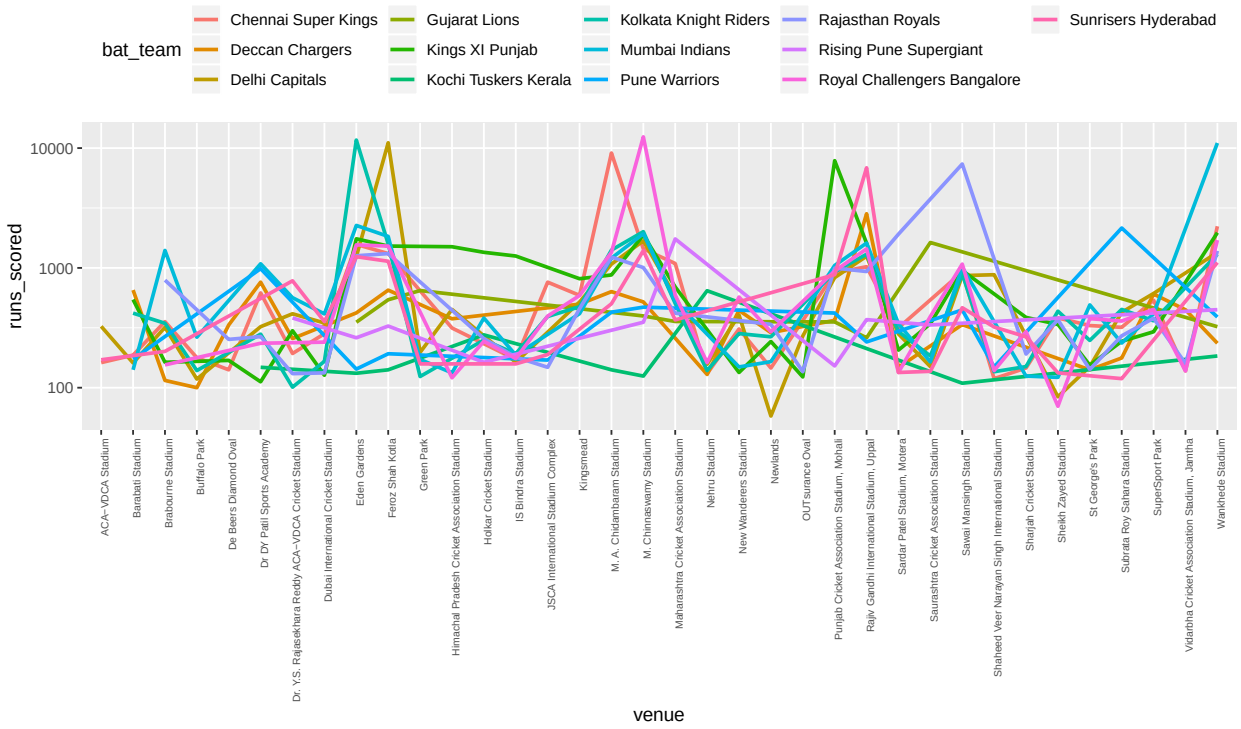


Though the plot lines at first don't show any significant relationship between match wins and venue, a closer look at the peaks for each line reveals an interesting fact. As a cricket fan I know that the highest of the peaks for each line are the home grounds for respective teams. This means teams win a lot more at home grounds than at other grounds clearly suggesting a strong relationship between home ground and match-wins for any team. The reasons are primarily teams get a lot more crowd support and also teams will be better accustomed to play at home venues.

To confirm the effect of venue, let us also see if venue had played any role on teams scoring runs.

```
# Venue effect on team runs
del_ds %>%
  full_join(mat_ds, by ="match_id") %>%
  select(match_id, bat_team, role, total_runs, venue) %>%
  filter(role == "batsman") %>%
```

```
  group_by(venue, bat_team) %>%
  summarize(runs_scored = sum(total_runs)) %>%
  ggplot(aes(venue, runs_scored, group = bat_team)) +
  geom_line(aes(col = bat_team), size = 1) +
  scale_y_log10() +
  theme(axis.text.x = element_text(
    angle = 90,
    size = 6,
    hjust = 1
  ),
  legend.position = "top", legend.title.align=0)
```



The peaks for each line again represent home grounds for respective teams. In effect, we can say that teams playing at home not only won more matches but also scored more runs.

Now that we have done enough data exploration, let us get into the task of building our model to determine player values.

6 Methods & Analysis - Model Building & Results:

6.1 1st Objective - Building a model to rank players by their playing calibre:

A player value depends upon

- his ability to score quick runs (highest strike rates) and bowl economically (lowest economy rates)
- his contribution made to the runs scored by the team and the wickets dismissed by the team in matches that have been both won and lost by his team
- his ability to score quick runs against top bowlers (we will consider top 20 bowlers by their economy rate) and to bowl economically against top batsmen (we will consider top 20 batsmen by their strike rates).
- all players are rated as a btasman and as a bowler irrespective of their actual or primary domain. Hence, the nomenclature, “batsman” or “bowler” in the model building refers to all players.

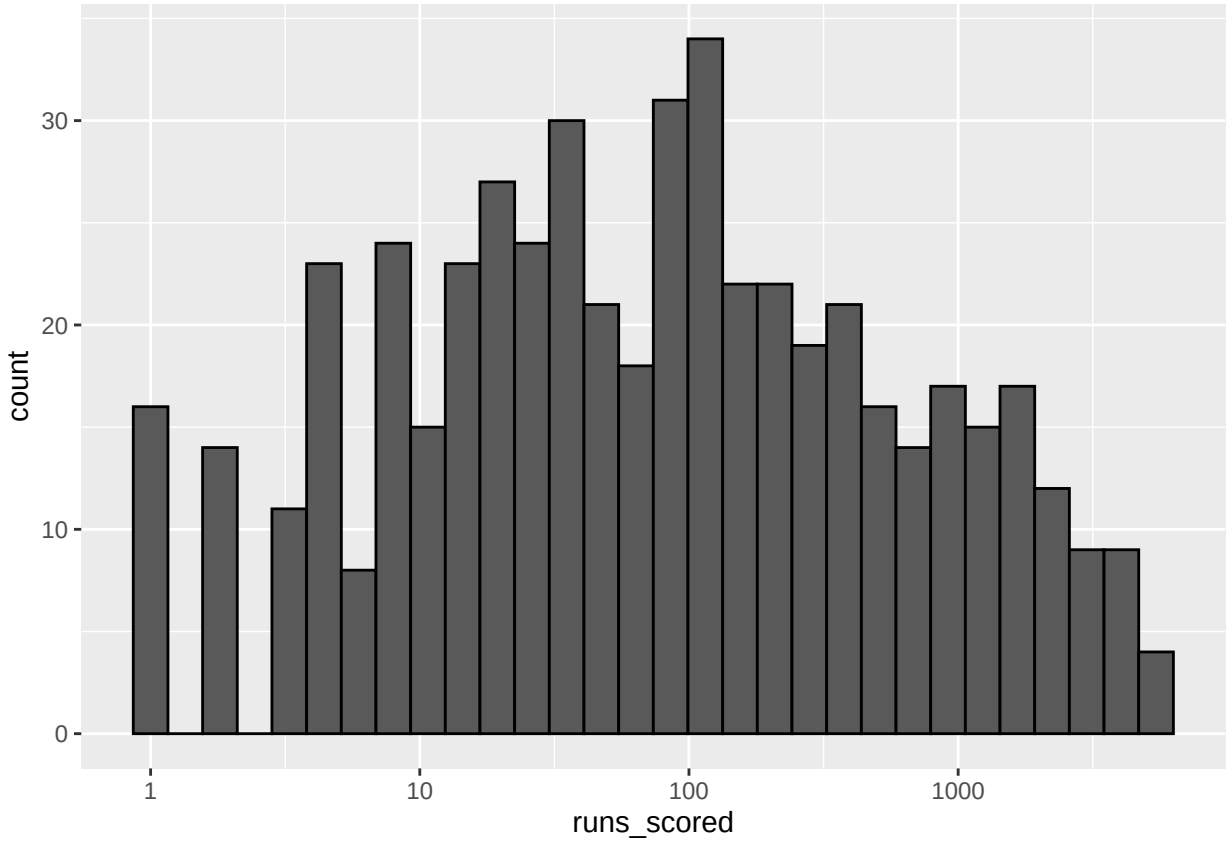
6.1.1 TOP_RATE_PLAYERS:

6.1.1.1 Order of players with best batting striking rates & bowling economy rates

The first step in our model building to develop players values is to determine each player's batting strike rate and bowling economy rate.

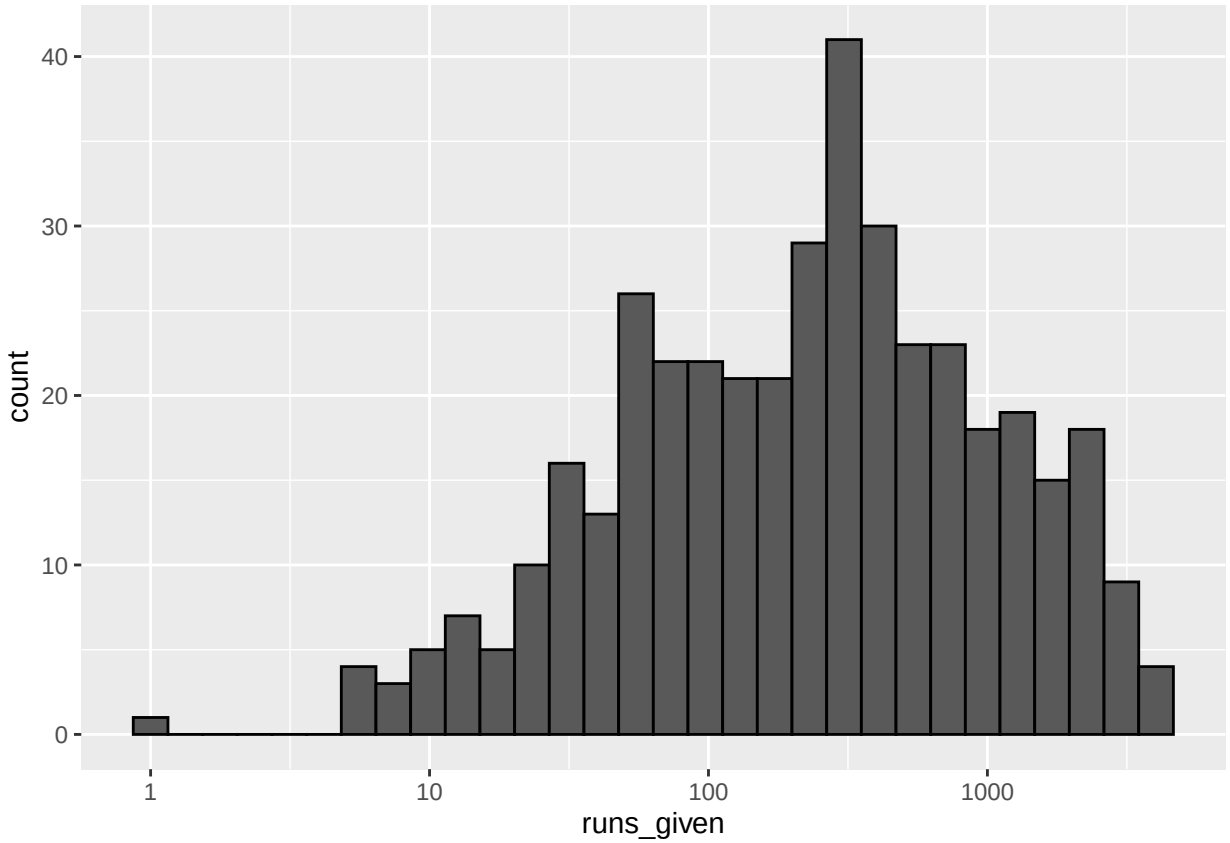
Before going ahead, let us take a look at the distribution of runs scored by batsmen (determines strike rate) and runs given away by bowlers (determines economy rate) for normality.

```
# Distribution of runs scored by batsmen
del_ds %>%
  filter(role == "batsman") %>%
  group_by(player) %>%
  summarize(runs_scored = sum(runs)) %>%
  mutate(runs_scored = runs_scored + 1) %>%
  ggplot(aes(runs_scored)) +
  geom_histogram(aes(), bins=30, colour="black") +
  scale_x_log10()
```




```
# Distribution of runs given by bowlers
del_ds %>%
  filter(role == "bowler") %>%
  group_by(player) %>%
  summarize(runs_given = sum(runs)) %>%
```

```
  mutate(runs_given = runs_given + 1) %>%
  ggplot(aes(runs_given)) +
  geom_histogram(aes(), bins=30, colour="black") +
  scale_x_log10()
```



The histogram distribution follows normal curve shape, hence we can assume the runs data distribution is normal.

Next, we calculate some basic statistics on balls faced and runs scored by batsmen and balls bowled and runs given by bowlers.

```
# batsmen average & median number of balls and runs
batsmen_avgs <- del_ds %>%
  filter(role == "batsman") %>%
  group_by(player) %>%
  summarize(tot_balls = n(), tot_runs = sum(runs)) %>%
  summarize (
    avg_balls = mean(tot_balls),
    median_balls = median(tot_balls),
    avg_runs = mean(tot_runs),
    median_runs = median(tot_runs),
    max(tot_runs),
    min(tot_runs),
    max(tot_balls),
    min(tot_balls)
  )
```

```
t(as.matrix(batsmen_avgs))
```

```
##                     [,1]
## avg_balls       347.0504
## median_balls     70.5000
## avg_runs        432.7248
## median_runs      74.0000
## max(tot_runs)  5434.0000
## min(tot_runs)     0.0000
## max(tot_balls) 4211.0000
## min(tot_balls)    1.0000
```

```
# bowler average & median number of balls and runs
bowler_avgs <- del_ds %>%
  filter(role == "bowler") %>%
  group_by(player) %>%
  summarize(tot_balls = n(), tot_runs = sum(runs)) %>%
  summarize (
    avg_balls = mean(tot_balls),
    median_balls = median(tot_balls),
    avg_runs = mean(tot_runs),
    median_runs = median(tot_runs),
    max(tot_runs),
    min(tot_runs),
    max(tot_balls),
    min(tot_balls)
  )

t(as.matrix(bowler_avgs))
```

```
##                     [,1]
## avg_balls       442.1679
## median_balls    196.0000
## avg_runs        551.3235
## median_runs     254.0000
## max(tot_runs)  4022.0000
## min(tot_runs)     0.0000
## max(tot_balls) 3451.0000
## min(tot_balls)    1.0000
```

Now, we will calculate the strike rates for batsmen and economy rates for bowlers using regularization technique. We have earlier seen that the players with highest batting strike rates and best economy rates are not well known for their skills in respective domains (batting or bowling). However, they ended best because of the fact that they played very few balls, resulting in best rates. In order to neutralize this effect, we use penalties to calculate revised batting strike rates or bowling economy rates.

From the batsmen and bowler statistics generated above we see the median values are much smaller than the average values. Hence, we use median values as the penalty terms to regularize as this will not effect much the rates of regular, known players in respective domains but will reduce the effects for the players who had batted/ bowled a very few balls.

Then we take a look again at the players with highest batting strike rates and lowest economy rates.

```r
# Top players with strike rates & economy rates after regularisation using median runs
# and median balls
str_rates <- del_ds %>%
  filter(role == "batsman") %>%
  group_by(player) %>%
  summarize(reg_str_rate = (sum(runs) + batsmen_avgs$median_runs) / (n() +
      batsmen_avgs$median_balls)) %>%
  arrange(desc(reg_str_rate))

str_rates %>%
  head(20)
```

```
## # A tibble: 20 x 2
##    player         reg_str_rate
##    <chr>                 <dbl>
##  1 AD Russell             1.74
##  2 SP Narine              1.59
##  3 RR Pant                1.59
##  4 GJ Maxwell             1.52
##  5 M Ali                  1.52
##  6 J Bairstow             1.49
##  7 HH Pandya              1.48
##  8 AB de Villiers         1.48
##  9 V Sehwag               1.47
## 10 JC Buttler             1.47
## 11 CH Morris              1.45
## 12 BCJ Cutting            1.45
## 13 CH Gayle               1.45
## 14 K Gowtham              1.42
## 15 KA Pollard             1.40
## 16 KH Pandya              1.40
## 17 N Pooran               1.39
## 18 DA Warner              1.39
## 19 YK Pathan              1.38
## 20 CR Brathwaite          1.38
```

```r
eco_rates <- del_ds %>%
  filter(role == "bowler") %>%
  group_by(player) %>%
  summarize(reg_eco_rate = (sum(runs) + bowler_avgs$median_runs) /
(n() + bowler_avgs$median_balls)) %>%
  arrange(reg_eco_rate)

eco_rates %>%
  head(20)
```

```
## # A tibble: 20 x 2
##    player           reg_eco_rate
##    <chr>                   <dbl>
##  1 DW Steyn                 1.06
##  2 M Muralitharan           1.07
##  3 R Ashwin                 1.08
##  4 Sohail Tanvir            1.08
```

```
##  5 A Kumble                 1.09
##  6 SL Malinga               1.10
##  7 SP Narine                1.10
##  8 SW Tait                  1.11
##  9 DP Nannes                1.12
## 10 MA Starc                 1.13
## 11 Rashid Khan              1.13
## 12 Harbhajan Singh          1.13
## 13 WD Parnell               1.14
## 14 RE van der Merwe         1.14
## 15 J Botha                  1.14
## 16 B Kumar                  1.14
## 17 DL Vettori               1.15
## 18 FH Edwards               1.15
## 19 DE Bollinger             1.15
## 20 A Chandila               1.15
```

Now as expected we can see that the top players for batting strike rates and bowling economy rates are all top, regular players in the respective domains of batting and bowling.

Next, using the regularized strike rates and economy rates, we construct top rated players, Naturally, we can expect all players who are in batsmen list may not figure in bowler list, and vice versa. This will reintroduce NAs when we try to combine strike rates and economy rates to arrive at player values. We use a similar technique as regularization to replace these NAs. For those players who have never batted, we will assume them to score minimum runs in maximum balls. Hence, we will use median runs and average balls for replacing NAs. Similarly, for players who have never bowled, we will assume them to give away more runs in less balls. Hence, we will use average runs and median balls for replacing NAs.

With the above approach, let us see who are our top rated players.

```
# Top rate players based on strike rates & economy rates
top_rate_players <- str_rates %>%
  full_join(eco_rates, by = "player") %>%
  mutate(reg_str_rate = replace_na(
    reg_str_rate,
    batsmen_avgs$median_runs / batsmen_avgs$avg_balls
  )) %>%
  mutate(reg_eco_rate = replace_na(reg_eco_rate, bowler_avgs$avg_runs /
                                   bowler_avgs$median_balls)) %>%
  mutate(player_value = 100 * (reg_str_rate + 1 / reg_eco_rate))

top_rate_players %>%
  arrange(desc(player_value)) %>%
  select(player, player_value) %>%
  mutate(rank = row_number()) %>%
  head(50) %>%
  knitr::kable()
```

player	player_value	rank
SP Narine	249.8296	1
AD Russell	245.4557	2
M Ali	234.0577	3
CH Gayle	228.1677	4
GJ Maxwell	225.5855	5

player	player_value	rank
KH Pandya	225.0858	6
CH Morris	223.7291	7
YK Pathan	222.7373	8
Rashid Khan	221.7830	9
HH Pandya	218.6351	10
SR Watson	218.4355	11
Harbhajan Singh	217.4723	12
K Gowtham	217.2212	13
SK Raina	216.8137	14
KK Cooper	216.5388	15
Mohammad Nabi	216.3234	16
KA Pollard	216.2749	17
BCJ Cutting	216.2111	18
V Sehwag	215.7857	19
MF Maharoof	214.2093	20
JA Morkel	213.8583	21
Shahid Afridi	213.4270	22
Bipul Sharma	212.1187	23
SN Khan	211.7002	24
KP Pietersen	211.6698	25
CR Brathwaite	211.4528	26
RN ten Doeschate	211.2386	27
Ankit Sharma	211.1351	28
KS Williamson	209.8720	29
ST Jayasuriya	209.8612	30
AC Gilchrist	209.3954	31
Umar Gul	209.3522	32
DL Chahar	207.8779	33
RA Tripathi	207.6624	34
N Rana	207.6340	35
RG Sharma	207.5759	36
LJ Wright	207.2914	37
Yuvraj Singh	206.3828	38
M Morkel	206.2108	39
JP Duminy	205.7658	40
JD Ryder	205.6850	41
A Ashish Reddy	205.6455	42
STR Binny	204.9562	43
SM Pollock	204.8870	44
V Kohli	204.3044	45
S Curran	204.2641	46
Shakib Al Hasan	204.2299	47
BA Stokes	204.0291	48
A Symonds	203.9560	49
C de Grandhomme	203.9522	50

As we could see the list includes some match winning top all round players who are big hitters with high strike rates and bowl tight overs.

6.1.2 TOP_CONTRI_PLAYERS:

6.1.2.1 Order of players with best number of highest contributions in won & lost matches

Having rated players by their striking and economy rates, the next step is to determine the contribution of each player as a batsman and as a bowler for his team in terms of runs scored and wickets taken in both matches that the team had won and lost.

For that, let us find out which team had won and which team had lost which of the 752 matches.

```
# Which teams have won which matches and lost which matches
# Which matches which teams have won
won_t1 <- mat_ds %>%
  filter(winner != "") %>%
  filter(as.character(winner) == as.character(team1)) %>%
  select(match_id, team = team1)

won_t2 <- mat_ds %>%
  filter(winner != "") %>%
  filter(as.character(winner) == as.character(team2)) %>%
  select(match_id, team = team2)

won_matches <- won_t1 %>%
  bind_rows(won_t2)

# Which matches which teams have lost
lost_t1 <- mat_ds %>%
  filter(winner != "") %>%
  filter(as.character(winner) != as.character(team1)) %>%
  select(match_id, team = team1, winner)

lost_t2 <- mat_ds %>%
  filter(winner != "") %>%
  filter(as.character(winner) != as.character(team2)) %>%
  select(match_id, team = team2, winner)

lost_matches <- lost_t1 %>%
  bind_rows(lost_t2)
```

Then we find out who are the batsmen who have scored maximum runs in the matches that their teams have won and similarly the bowlers who have dismissed maximum of opponent's wickets in the matches that their teams have won.

```
# Batsmen score contribution in won matches
# Top scorer for winning sides
batsman_contr_w <- del_ds %>%
  full_join(won_matches, by = "match_id") %>%
  filter(role == "batsman" & bat_team == team) %>%
  group_by(match_id, player) %>%
  summarize(batsman_score = sum(runs)) %>%
  top_n(1, batsman_score) %>%
  full_join(won_matches, by = "match_id")
batsman_contr_w
```

```
## # A tibble: 767 x 4
## # Groups:   match_id [752]
##    match_id player       batsman_score team
##       <int> <chr>                <int> <chr>
##  1        1 Yuvraj Singh            62 Sunrisers Hyderabad
##  2        2 SPD Smith               84 Rising Pune Supergiant
##  3        3 CA Lynn                 93 Kolkata Knight Riders
##  4        4 GJ Maxwell              44 Kings XI Punjab
##  5        5 KM Jadhav               69 Royal Challengers Bangalore
##  6        6 DA Warner               76 Sunrisers Hyderabad
##  7        7 N Rana                  50 Mumbai Indians
##  8        8 HM Amla                 58 Kings XI Punjab
##  9        9 SV Samson              102 Delhi Capitals
## 10       10 N Rana                  45 Mumbai Indians
## # ... with 757 more rows
```

```
# Bowler wicket taking contribution in won matches
# Top wicket taker for winning sides
bowler_contr_w <- del_ds %>%
  full_join(won_matches, by = "match_id") %>%
  filter(role=="bowler" & bowl_team == team) %>%
  filter (dismissal_kind %in% c("bowled", "caught", "caught and bowled", "hit wicket",
"lbw", "stumped")) %>%
  select(match_id, team, bowl_team, player, dismissal_kind) %>%
  group_by(match_id, player) %>%
  summarize(bowler_wckts = n()) %>%
  top_n(1, bowler_wckts) %>%
  full_join(won_matches, by = "match_id")
bowler_contr_w
```

```
## # A tibble: 1,246 x 4
## # Groups:   match_id [752]
##    match_id player         bowler_wckts team
##       <int> <chr>                 <int> <chr>
##  1        1 A Nehra                   2 Sunrisers Hyderabad
##  2        1 B Kumar                   2 Sunrisers Hyderabad
##  3        1 Rashid Khan               2 Sunrisers Hyderabad
##  4        2 Imran Tahir               3 Rising Pune Supergiant
##  5        3 Kuldeep Yadav             2 Kolkata Knight Riders
##  6        4 Sandeep Sharma            2 Kings XI Punjab
##  7        5 B Stanlake                2 Royal Challengers Bangalore
##  8        5 Iqbal Abdulla             2 Royal Challengers Bangalore
##  9        5 P Negi                    2 Royal Challengers Bangalore
## 10        6 Rashid Khan               3 Sunrisers Hyderabad
## # ... with 1,236 more rows
```

Note that it is common for more than one bowler to dismiss the same number of opponent batsmen in the same match, while two batsmen scoring the same top score for their teams in the same match is rare.

Let us reorder our above table for batsmen contribution in terms highest individual contributions and also count the number of times a batsman had made highest contribution to his team.

```r
# Top_batsmen on winning sides in the order of highest individual scores
winning_t_scores <- del_ds %>%
  full_join(won_matches, by = "match_id") %>%
  filter(role == "batsman" & bat_team == team) %>%
  group_by(match_id) %>%
  summarize(team_score = sum(total_runs)) %>%
  full_join(batsman_contr_w, by = "match_id") %>%
  arrange(desc(batsman_score))
winning_t_scores
```

```
## # A tibble: 767 x 5
##    match_id team_score player        batsman_score team
##       <int>      <int> <chr>                 <int> <chr>
##  1      411        263 CH Gayle                175 Royal Challengers Banga~
##  2       60        222 BB McCullum             158 Kolkata Knight Riders
##  3      562        235 AB de Villie~           133 Royal Challengers Banga~
##  4      620        248 AB de Villie~           129 Royal Challengers Banga~
##  5      372        215 CH Gayle                128 Royal Challengers Banga~
##  6      206        246 M Vijay                 127 Chennai Super Kings
##  7       36        209 DA Warner               126 Sunrisers Hyderabad
##  8      516        226 V Sehwag                122 Kings XI Punjab
##  9     7953        187 SR Watson               121 Chennai Super Kings
## 10      243        193 PC Valthaty             120 Kings XI Punjab
## # ... with 757 more rows
```

```r
# Top_batsmen on winning sides in terms no.of top_scores
win_scores <- winning_t_scores %>%
  group_by(player) %>%
  summarize(batsman_count = n()) %>%
  arrange(desc(batsman_count))
win_scores
```

```
## # A tibble: 143 x 2
##    player         batsman_count
##    <chr>                  <int>
##  1 DA Warner                 28
##  2 G Gambhir                 27
##  3 RG Sharma                 26
##  4 CH Gayle                  25
##  5 SK Raina                  21
##  6 AB de Villiers            20
##  7 AM Rahane                 20
##  8 AT Rayudu                 18
##  9 SR Watson                 18
## 10 V Sehwag                  18
## # ... with 133 more rows
```

The players appear at the top of the list seem to be consistent with their reputation.

We will also check bowlers' contribution too in terms of maximum wicket taking contribution in the matches that their teams have won.

```
# Top_bowlers on winning sides in terms no.of maximum wickets
win_wickets <- bowler_contr_w %>%
  group_by(player) %>%
  summarize(bowler_count = n()) %>%
  arrange(desc(bowler_count))
win_wickets
```

```
## # A tibble: 238 x 2
##    player          bowler_count
##    <chr>                  <int>
##  1 SL Malinga                30
##  2 Harbhajan Singh           27
##  3 A Mishra                  26
##  4 DJ Bravo                  25
##  5 R Vinay Kumar             22
##  6 UT Yadav                  22
##  7 PP Chawla                 20
##  8 B Kumar                   19
##  9 SP Narine                 19
## 10 R Ashwin                  18
## # ... with 228 more rows
```

Now, we will do the whole exercise of finding batsmen's contribution and bowlers' contribution to their teams in the matches that the teams have lost.

```
# Batsmen score contribution in lost matches
# Top scorer for losing sides
batsman_contr_l <- del_ds %>%
  full_join(lost_matches, by = "match_id") %>%
  filter(role == "batsman" & bat_team == team) %>%
  group_by(match_id, player) %>%
  summarize(batsman_score = sum(runs)) %>%
  top_n(1, batsman_score) %>%
  full_join(lost_matches, by = "match_id") %>%
  rename(losing_team=team)
batsman_contr_l
```

```
## # A tibble: 771 x 5
## # Groups:   match_id [752]
##    match_id player      batsman_score losing_team         winner
##       <int> <chr>               <int> <chr>               <chr>
##  1        1 CH Gayle               32 Royal Challengers ~ Sunrisers Hydera~
##  2        2 JC Buttler             38 Mumbai Indians      Rising Pune Supe~
##  3        3 SK Raina               68 Gujarat Lions       Kolkata Knight R~
##  4        4 BA Stokes              50 Rising Pune Superg~ Kings XI Punjab
##  5        5 RR Pant                57 Delhi Capitals      Royal Challenger~
##  6        6 DR Smith               37 Gujarat Lions       Sunrisers Hydera~
##  7        7 MK Pandey              81 Kolkata Knight Rid~ Mumbai Indians
##  8        8 AB de Vill~            89 Royal Challengers ~ Kings XI Punjab
##  9        9 MA Agarwal             20 Rising Pune Superg~ Delhi Capitals
## 10       10 DA Warner              49 Sunrisers Hyderabad Mumbai Indians
## # ... with 761 more rows
```

```r
# Bowler wicket taking contribution in lost matches
# Top wicket taker for losing sides
bowler_contr_l <- del_ds %>%
  full_join(lost_matches, by = "match_id") %>%
  filter(role=="bowler" & bowl_team == team) %>%
  filter(dismissal_kind %in% c("bowled", "caught", "caught and bowled", "hit wicket",
                               "lbw", "stumped")) %>%
  select(match_id, team, bowl_team, player) %>%
  group_by(match_id, player) %>%
  summarize(bowler_wckts = n()) %>%
  top_n(1, bowler_wckts) %>%
  full_join(lost_matches, by = "match_id") %>%
  rename(losing_team=team)
bowler_contr_l
```

```
## # A tibble: 1,226 x 5
## # Groups:   match_id [752]
##    match_id player      bowler_wckts losing_team          winner
##       <int> <chr>              <int> <chr>                <chr>
##  1        1 A Choudhary            1 Royal Challengers B~ Sunrisers Hydera~
##  2        1 STR Binny              1 Royal Challengers B~ Sunrisers Hydera~
##  3        1 TS Mills               1 Royal Challengers B~ Sunrisers Hydera~
##  4        1 YS Chahal              1 Royal Challengers B~ Sunrisers Hydera~
##  5        2 HH Pandya              1 Mumbai Indians       Rising Pune Supe~
##  6        2 MJ McClena~            1 Mumbai Indians       Rising Pune Supe~
##  7        2 TG Southee             1 Mumbai Indians       Rising Pune Supe~
##  8        4 Imran Tahir            2 Rising Pune Supergi~ Kings XI Punjab
##  9        5 CH Morris              3 Delhi Capitals       Royal Challenger~
## 10        6 P Kumar                1 Gujarat Lions        Sunrisers Hydera~
## # ... with 1,216 more rows
```

```r
# Top_batsmen on losing sides in the order of highest individual scores
losing_t_scores <- del_ds %>%
  full_join(lost_matches, by = "match_id") %>%
  filter(role == "batsman" & bat_team == team) %>%
  group_by(match_id) %>%
  summarize(team_score = sum(total_runs)) %>%
  full_join(batsman_contr_l, by = "match_id") %>%
  arrange(desc(batsman_score))
losing_t_scores
```

```
## # A tibble: 771 x 6
##    match_id team_score player   batsman_score losing_team   winner
##       <int>      <int> <chr>            <int> <chr>         <chr>
##  1     7935        190 RR Pant            130 Delhi Capita~ Sunrisers Hyde~
##  2       68        214 A Symon~           117 Deccan Charg~ Rajasthan Roya~
##  3      517        199 WP Saha            115 Kings XI Pun~ Kolkata Knight~
##  4    11331        196 AM Raha~           108 Rajasthan Ro~ Delhi Capitals
##  5    11144        203 SV Sams~           106 Rajasthan Ro~ Sunrisers Hyde~
##  6    11319        180 CH Gayle           105 Kings XI Pun~ Royal Challeng~
##  7       22        198 HM Amla            104 Kings XI Pun~ Mumbai Indians
##  8       46        189 HM Amla            104 Kings XI Pun~ Gujarat Lions
##  9    11315        204 KL Rahul           104 Kings XI Pun~ Mumbai Indians
```

```
## 10      410        185 SR Wats~           101 Rajasthan Ro~ Chennai Super ~
## # ... with 761 more rows
```

```
# Top_batsmen on losing sides in terms no.of top_scores
loss_scores <- losing_t_scores %>%
  group_by(player) %>%
  summarize(batsman_count = n()) %>%
  arrange(desc(batsman_count))
loss_scores
```

```
## # A tibble: 166 x 2
##    player       batsman_count
##    <chr>                <int>
##  1 RV Uthappa              22
##  2 V Kohli                 22
##  3 S Dhawan                20
##  4 DA Warner               19
##  5 CH Gayle                18
##  6 RG Sharma               18
##  7 SK Raina                17
##  8 Yuvraj Singh            17
##  9 JP Duminy               15
## 10 KD Karthik              14
## # ... with 156 more rows
```

```
# Top_bowlers on losing sides in terms no.of maximum wickets
loss_wickets <- bowler_contr_l %>%
  group_by(player) %>%
  summarize(bowler_count = n()) %>%
  arrange(desc(bowler_count))
loss_wickets
```

```
## # A tibble: 262 x 2
##    player          bowler_count
##    <chr>                  <int>
##  1 B Kumar                   27
##  2 A Mishra                  22
##  3 PP Chawla                 22
##  4 DJ Bravo                  21
##  5 YS Chahal                 21
##  6 DW Steyn                  19
##  7 Harbhajan Singh           19
##  8 R Ashwin                  19
##  9 AB Dinda                  18
## 10 IK Pathan                 18
## # ... with 252 more rows
```

Next, we summarize the batsmen's contribution in won and lost matches to arrive at his contribution for their teams in won and lost matches.

```
# Top batsmen contribution in won matches & lost matches - arranged by contribution in WON matches
top_contri_batsmen <- win_scores %>%
```

```
  rename(contribution_in_WON_matches = batsman_count) %>%
  full_join(loss_scores, by = "player") %>%
  rename(contribution_in_LOST_matches = batsman_count) %>%
  arrange(desc(contribution_in_WON_matches))
top_contri_batsmen
```

```
## # A tibble: 195 x 3
##    player         contribution_in_WON_matches contribution_in_LOST_matches
##    <chr>                                <int>                         <int>
##  1 DA Warner                               28                            19
##  2 G Gambhir                               27                            13
##  3 RG Sharma                               26                            18
##  4 CH Gayle                                25                            18
##  5 SK Raina                                21                            17
##  6 AB de Villiers                          20                            13
##  7 AM Rahane                               20                            10
##  8 AT Rayudu                               18                             9
##  9 SR Watson                               18                             6
## 10 V Sehwag                                18                             8
## # ... with 185 more rows
```

```
# Top batsmen contribution in won matches & lost matches - arranged by contribution in LOST matches
top_contri_batsmen <- win_scores %>%
  rename(contribution_in_WON_matches = batsman_count) %>%
  full_join(loss_scores, by = "player") %>%
  rename(contribution_in_LOST_matches = batsman_count) %>%
  arrange(desc(contribution_in_LOST_matches))
top_contri_batsmen
```

```
## # A tibble: 195 x 3
##    player       contribution_in_WON_matches contribution_in_LOST_matches
##    <chr>                              <int>                         <int>
##  1 V Kohli                               17                            22
##  2 RV Uthappa                            14                            22
##  3 S Dhawan                              17                            20
##  4 DA Warner                             28                            19
##  5 RG Sharma                             26                            18
##  6 CH Gayle                              25                            18
##  7 SK Raina                              21                            17
##  8 Yuvraj Singh                           6                            17
##  9 JP Duminy                              2                            15
## 10 KD Karthik                             9                            14
## # ... with 185 more rows
```

```
# Top batsmen overall contribution in won matches & lost matches
top_contri_batsmen <- top_contri_batsmen %>%
  mutate(batsman_contribution = contribution_in_LOST_matches +
           contribution_in_WON_matches) %>%
  select(
    player,
    batsman_contribution,
    contribution_in_LOST_matches,
```

```
    contribution_in_WON_matches
  ) %>%
  arrange(desc(batsman_contribution))
top_contri_batsmen
```

```
## # A tibble: 195 x 4
##    player     batsman_contribut~ contribution_in_LOST~ contribution_in_WON~
##    <chr>                   <int>                 <int>                 <int>
##  1 DA Warner                  47                    19                    28
##  2 RG Sharma                  44                    18                    26
##  3 CH Gayle                   43                    18                    25
##  4 G Gambhir                  40                    13                    27
##  5 V Kohli                    39                    22                    17
##  6 SK Raina                   38                    17                    21
##  7 S Dhawan                   37                    20                    17
##  8 RV Uthappa                 36                    22                    14
##  9 AB de Vil~                 33                    13                    20
## 10 AM Rahane                  30                    10                    20
## # ... with 185 more rows
```

Note that we have arranged the batsmen's contribution in the order of won matches and also lost matches.

Similarly, we summarize the bowlers' contribution in won and lost matches to arrive at his contribution for their teams in won and lost matches.

```
# Top bowlers contribution in won matches and lost matches - arranged by contribution in WON matches
top_contri_bowlers <- win_wickets %>%
  rename(contribution_in_WON_matches = bowler_count) %>%
  full_join(loss_wickets, by = "player") %>%
  rename(contribution_in_LOST_matches = bowler_count) %>%
  arrange(desc(contribution_in_WON_matches))
top_contri_bowlers
```

```
## # A tibble: 301 x 3
##    player          contribution_in_WON_matches contribution_in_LOST_matches
##    <chr>                                 <int>                        <int>
##  1 SL Malinga                               30                           18
##  2 Harbhajan Singh                          27                           19
##  3 A Mishra                                 26                           22
##  4 DJ Bravo                                 25                           21
##  5 R Vinay Kumar                            22                           18
##  6 UT Yadav                                 22                           13
##  7 PP Chawla                                20                           22
##  8 B Kumar                                  19                           27
##  9 SP Narine                                19                           18
## 10 R Ashwin                                 18                           19
## # ... with 291 more rows
```

```
# Top bowlers contribution in won matches and lost matches - arranged by contribution in LOST matches
top_contri_bowlers <- win_wickets %>%
  rename(contribution_in_WON_matches = bowler_count) %>%
  full_join(loss_wickets, by = "player") %>%
  rename(contribution_in_LOST_matches = bowler_count) %>%
```

```
  arrange(desc(contribution_in_LOST_matches))
top_contri_bowlers
```

```
## # A tibble: 301 x 3
##    player          contribution_in_WON_matches contribution_in_LOST_matches
##    <chr>                                 <int>                        <int>
##  1 B Kumar                                  19                           27
##  2 A Mishra                                 26                           22
##  3 PP Chawla                                20                           22
##  4 DJ Bravo                                 25                           21
##  5 YS Chahal                                14                           21
##  6 Harbhajan Singh                          27                           19
##  7 R Ashwin                                 18                           19
##  8 DW Steyn                                 16                           19
##  9 SL Malinga                               30                           18
## 10 R Vinay Kumar                            22                           18
## # ... with 291 more rows
```

```
# Top bowlers overall contribution in won matches and lost matches
top_contri_bowlers <- top_contri_bowlers %>%
  mutate(bowler_contribution = contribution_in_LOST_matches +
           contribution_in_WON_matches) %>%
  select(
    player,
    bowler_contribution,
    contribution_in_LOST_matches,
    contribution_in_WON_matches
  ) %>%
  arrange(desc(bowler_contribution))
top_contri_bowlers
```

```
## # A tibble: 301 x 4
##    player     bowler_contributi~ contribution_in_LOST~ contribution_in_WON~
##    <chr>                   <int>                 <int>                 <int>
##  1 A Mishra                   48                    22                    26
##  2 SL Malinga                 48                    18                    30
##  3 B Kumar                    46                    27                    19
##  4 DJ Bravo                   46                    21                    25
##  5 Harbhajan~                 46                    19                    27
##  6 PP Chawla                  42                    22                    20
##  7 R Vinay K~                 40                    18                    22
##  8 R Ashwin                   37                    19                    18
##  9 SP Narine                  37                    18                    19
## 10 YS Chahal                  35                    21                    14
## # ... with 291 more rows
```

For bowlers also, we have arranged the contribution in the order of won matches and also lost matches.

Now that we know top contribution batsmen and bowlers, let us summarize the information into top contribution players.

Here, first, we replace NAs with 0s as NAs are introduced for players who did not contribute.

The more number of matches a player plays, that many more chances he has to make highest contributions for his team. To regularize the contribution effect of players who played more matches than others, we assume they would have made median contribution after playing average number of matches, and this is true for all players. Hence, we use median contribution and average matches in arriving at player contributions. We have earlier explained how regularization works as a penalty term.

```
# Top_contribution players in won/ lost matches
top_contri_players <- top_contri_batsmen %>%
  full_join(top_contri_bowlers, by="player")

top_contri_players[is.na(top_contri_players)] <- 0

top_contri_players <- top_contri_players %>%
  mutate(player_contribution = batsman_contribution + bowler_contribution) %>%
  select(player,player_contribution, batsman_contribution, bowler_contribution)
top_contri_players %>%
  arrange(desc(player_contribution))
```

```
## # A tibble: 410 x 4
##    player         player_contributi~ batsman_contributi~ bowler_contributi~
##    <chr>                       <dbl>               <dbl>              <dbl>
##  1 DJ Bravo                       56                  10                 46
##  2 SR Watson                      52                  24                 28
##  3 CH Gayle                       49                  43                  6
##  4 JH Kallis                      48                  23                 25
##  5 A Mishra                       48                   0                 48
##  6 SL Malinga                     48                   0                 48
##  7 DA Warner                      47                  47                  0
##  8 RG Sharma                      47                  44                  3
##  9 SK Raina                       46                  38                  8
## 10 Harbhajan Sin~                 46                   0                 46
## # ... with 400 more rows
```

```
# Average and median of top contribution by players
stats_contri <- top_contri_players %>%
  summarize(avg_contri_pp = mean(player_contribution),
            med_contri_pp = median(player_contribution))

# Average and median of matches played by players
stats_matches <- matches_players %>%
  summarize(avg_mat_played = mean(no_matches_played),
            med_mat_played = median(no_matches_played))

top_contri_players <- top_contri_players %>%
  full_join(matches_players, by ="player") %>%
  mutate(player_contri_rate=(player_contribution+stats_contri$med_contri_pp)/
           (no_matches_played+stats_matches$avg_mat_played))

top_contri_players %>%
  select(player, player_contribution, player_contri_rate) %>%
  arrange(desc(player_contri_rate)) %>%
  head(50) %>%
  knitr::kable()
```

player	player_contribution	player_contri_rate
JH Kallis	48	0.4113901
DJ Bravo	56	0.3717456
Imran Tahir	26	0.3597091
YS Chahal	35	0.3500875
CH Gayle	49	0.3477674
SL Malinga	48	0.3457430
SR Watson	52	0.3448261
K Rabada	12	0.3448223
B Kumar	46	0.3438773
MJ McClenaghan	25	0.3435990
AD Russell	27	0.3429176
R Vinay Kumar	40	0.3323245
DA Warner	47	0.3303094
DW Steyn	35	0.3239183
S Kaul	19	0.3133484
VR Aaron	19	0.3133484
AJ Tye	13	0.3124959
Sandeep Sharma	26	0.3112010
SE Marsh	26	0.3080060
JP Faulkner	23	0.3054274
BA Stokes	15	0.3044837
R Sharma	18	0.3038644
NM Coulter-Nile	12	0.2996215
A Mishra	48	0.2964640
SP Narine	37	0.2962412
M Morkel	25	0.2947133
MA Starc	12	0.2941138
IK Pathan	34	0.2914094
S Gopal	13	0.2910923
AB Dinda	26	0.2873544
K Ahmed	7	0.2864530
A Kumble	16	0.2840880
J Archer	10	0.2833967
Z Khan	32	0.2825730
Rashid Khan	17	0.2822553
KK Cooper	11	0.2808951
AR Patel	27	0.2807953
RP Singh	27	0.2807953
LMP Simmons	12	0.2787422
S Sreesanth	16	0.2762404
WD Parnell	11	0.2757317
BJ Hodge	21	0.2735208
Shakib Al Hasan	21	0.2735208
Azhar Mahmood	10	0.2723697
Mohammed Shami	17	0.2713153
DE Bollinger	11	0.2707546
MF Maharoof	9	0.2685911
SN Thakur	13	0.2681358
RR Pant	18	0.2669880
UT Yadav	35	0.2663921

As we can see, the contribution rates have essentially regulirised the effect by reducing the contribution levels

of the players who have played more matches.

Now that we have identified top-rate-players (players with best batting strike rates and bowling economy rates) and top-contributing-players (players who have contributed maximum number of times both when their teams have won or lost in terms of scoring runs and taking wickets), let us address one more performance area that will be used to calculate final player values.

We call this player-excellence-rate factor which primarily measures batsmen strike rate against top 20 economy bowlers and also bowlers economy rate against top 20 strike batsmen. Thus we ensure that our list provides player rates based on their performance against the best in business.

6.1.3 TOP_EXCEL_PLAYERS:

6.1.3.1 Order of players with best performance against best bowlers and best batsmen

We start with listing out top 20 batsmen in terms of their strike rate and top 20 bowlers in terms of their economy rate.

We have earlier rated players by their batting strike rates and bowling economy rates after regularisation and called these lists "str_rates" and "eco_rates" respectively. Let us use these lists to generate the top 20 strike batsmen and top 20 economy bolwers in order to calculate excel_rates for all players.

```
# Top 20 strike batsmen and top 20 economy bowlers
top_20_batsmen <- str_rates %>%
  head(20)
top_20_batsmen
```

```
## # A tibble: 20 x 2
##    player         reg_str_rate
##    <chr>                 <dbl>
##  1 AD Russell             1.74
##  2 SP Narine              1.59
##  3 RR Pant                1.59
##  4 GJ Maxwell             1.52
##  5 M Ali                  1.52
##  6 J Bairstow             1.49
##  7 HH Pandya              1.48
##  8 AB de Villiers         1.48
##  9 V Sehwag               1.47
## 10 JC Buttler             1.47
## 11 CH Morris              1.45
## 12 BCJ Cutting            1.45
## 13 CH Gayle               1.45
## 14 K Gowtham              1.42
## 15 KA Pollard             1.40
## 16 KH Pandya              1.40
## 17 N Pooran               1.39
## 18 DA Warner              1.39
## 19 YK Pathan              1.38
## 20 CR Brathwaite          1.38
```

```
top_20_bowlers <- eco_rates %>%
  head(20)
top_20_bowlers
```

```
## # A tibble: 20 x 2
##    player           reg_eco_rate
##    <chr>                   <dbl>
##  1 DW Steyn                 1.06
##  2 M Muralitharan           1.07
##  3 R Ashwin                 1.08
##  4 Sohail Tanvir            1.08
##  5 A Kumble                 1.09
##  6 SL Malinga               1.10
```

```
##  7 SP Narine                1.10
##  8 SW Tait                  1.11
##  9 DP Nannes                1.12
## 10 MA Starc                 1.13
## 11 Rashid Khan              1.13
## 12 Harbhajan Singh          1.13
## 13 WD Parnell               1.14
## 14 RE van der Merwe         1.14
## 15 J Botha                  1.14
## 16 B Kumar                  1.14
## 17 DL Vettori               1.15
## 18 FH Edwards               1.15
## 19 DE Bollinger             1.15
## 20 A Chandila               1.15
```

Then we calculate the strike rates of all batsmen against the best 20 bowlers and economy rates of all bowlers against the best 20 batsmen.

All batsmen might have not played against top 20 bowlers and all bowlers might have not played against top 20 batsmen. Hence, to normalize batting strike rates and bowling strike rates for all batsmen, we assume they would have scored less runs in average number of balls. Similarly, we can assume all bowlers would have given away more runs in less number of balls.

```
# batsmen strike rate against top_20 bowlers
sr_vs_t20_bowlers <- deliveries %>%
  filter(bowler %in% top_20_bowlers$player) %>%
  group_by(player = batsman) %>%
  summarize(sr_t20 = (sum(batsman_runs) + batsmen_avgs$median_runs) / (n() +
         batsmen_avgs$avg_balls)) %>%
  arrange(desc(sr_t20))
sr_vs_t20_bowlers %>%
  head(20) %>%
  mutate(rank = row_number())
```

```
## # A tibble: 20 x 3
##    player         sr_t20  rank
##    <chr>           <dbl> <int>
##  1 AB de Villiers  0.870     1
##  2 SR Watson       0.846     2
##  3 DA Warner       0.822     3
##  4 SK Raina        0.806     4
##  5 RV Uthappa      0.790     5
##  6 V Kohli         0.786     6
##  7 YK Pathan       0.783     7
##  8 MS Dhoni        0.765     8
##  9 CH Gayle        0.741     9
## 10 S Dhawan        0.720    10
## 11 RG Sharma       0.691    11
## 12 G Gambhir       0.689    12
## 13 BB McCullum     0.680    13
## 14 AM Rahane       0.659    14
## 15 AC Gilchrist    0.658    15
## 16 KD Karthik      0.658    16
## 17 JP Duminy       0.647    17
```

```
## 18 V Sehwag         0.645    18
## 19 AT Rayudu        0.641    19
## 20 DR Smith         0.640    20
```

```
# bowlers economy rate against top_20 batsmen
er_vs_t20_batsmen <- deliveries %>%
  filter(batsman %in% top_20_batsmen$player) %>%
  group_by(player = bowler) %>%
  summarize(er_t20 = (sum(batsman_runs) + bowler_avgs$avg_runs) / (n() +
        bowler_avgs$median_balls)) %>%
  arrange(er_t20)
er_vs_t20_batsmen %>%
  head(20) %>%
  mutate(rank = row_number())
```

```
## # A tibble: 20 x 3
##    player          er_t20  rank
##    <chr>            <dbl> <int>
##  1 Harbhajan Singh   1.73     1
##  2 R Ashwin          1.74     2
##  3 SP Narine         1.78     3
##  4 DJ Bravo          1.82     4
##  5 JJ Bumrah         1.82     5
##  6 DS Kulkarni       1.85     6
##  7 SL Malinga        1.85     7
##  8 YS Chahal         1.92     8
##  9 P Kumar           1.92     9
## 10 B Kumar           1.92    10
## 11 DW Steyn          1.94    11
## 12 AR Patel          1.94    12
## 13 RA Jadeja         1.94    13
## 14 A Mishra          1.97    14
## 15 UT Yadav          1.97    15
## 16 PP Chawla         1.97    16
## 17 Rashid Khan       1.99    17
## 18 SR Watson         2.01    18
## 19 Sandeep Sharma    2.01    19
## 20 Z Khan            2.01    20
```

Now we generate the table for excellence players by joining the above two lists, replacing the NAs as earlier without affecting others and arranging them by the strike rate in descending order.

```
# Top excellence players
top_excel_players <- er_vs_t20_batsmen %>%
  full_join(sr_vs_t20_bowlers, by = "player") %>%
  mutate(sr_t20 = replace_na(sr_t20, batsmen_avgs$median_runs / batsmen_avgs$avg_balls)) %>%
  mutate(er_t20 = replace_na(er_t20, bowler_avgs$avg_runs / bowler_avgs$median_balls))
```

```
# Top 50 Excellence Players
top_excel_players %>%
  select(player, sr_t20, er_t20) %>%
  arrange(desc(sr_t20)) %>%
  head(50) %>%
  knitr::kable()
```

player	sr_t20	er_t20
AB de Villiers	0.8701406	2.812875
SR Watson	0.8464007	2.006728
DA Warner	0.8216833	2.812875
SK Raina	0.8059269	2.330107
RV Uthappa	0.7896125	2.812875
V Kohli	0.7864723	2.810671
YK Pathan	0.7833729	2.207911
MS Dhoni	0.7646577	2.812875
CH Gayle	0.7407404	2.665148
S Dhawan	0.7202782	2.731031
RG Sharma	0.6907171	2.724523
G Gambhir	0.6889001	2.812875
BB McCullum	0.6798980	2.812875
AM Rahane	0.6590200	2.812875
AC Gilchrist	0.6580043	2.812875
KD Karthik	0.6579230	2.812875
JP Duminy	0.6471892	2.411876
V Sehwag	0.6451609	2.812875
AT Rayudu	0.6406005	2.812875
DR Smith	0.6395753	2.458842
SV Samson	0.6348433	2.812875
SE Marsh	0.6279306	2.812875
MK Tiwary	0.6236933	2.814401
DPMD Jayawardene	0.6187604	2.812875
KA Pollard	0.6121924	2.324072
RR Pant	0.6087224	2.812875
KL Rahul	0.6063455	2.812875
JH Kallis	0.6054579	2.030277
WP Saha	0.6048418	2.812875
Yuvraj Singh	0.6044655	2.596451
RA Jadeja	0.6022790	1.943305
MA Agarwal	0.6006772	2.812875
SPD Smith	0.5961683	2.812875
GJ Maxwell	0.5912107	2.576077
MK Pandey	0.5877410	2.812875
PA Patel	0.5752299	2.812875
M Vijay	0.5614641	2.716538
SR Tendulkar	0.5586312	2.823383
BJ Hodge	0.5545283	2.781617
F du Plessis	0.5512560	2.812875
SS Iyer	0.5471879	2.812875
R Dravid	0.5407564	2.812875
KC Sangakkara	0.5323191	2.812875
DA Miller	0.5273600	2.812875
MEK Hussey	0.5253249	2.812875
KM Jadhav	0.5153471	2.812875
IK Pathan	0.5148590	2.259891
AJ Finch	0.5104671	2.882206
TM Dilshan	0.5097581	2.787679
STR Binny	0.5042926	2.385260

6.1.4 TOP_CALIBER_PLAYERS:

6.1.4.1 Order of players based on their summarized player values

Now, we have done all the ground work, it is time to rank players by their caliber. For this we calculate the final player values using the already calculated values for top-rated-players, player-contribution-rate and player-excellence-rates. Here, once again, we replace the NAs in batsmen strike rates using median runs in average balls and economy rates using average runs in median balls.

```
# Players by their calibre - strike rate + economy rate,
# contribution in win/ loss situation and
# player's excellence against the best in business

top_calibre_players <- top_rate_players %>%
  select(-player_value) %>%
  full_join(top_excel_players, by = "player") %>%
  mutate(sr_t20 = replace_na(sr_t20, batsmen_avgs$median_runs / batsmen_avgs$avg_balls)) %>%
  mutate(er_t20 = replace_na(er_t20, bowler_avgs$avg_runs / bowler_avgs$median_balls)) %>%
  full_join(top_contri_players, by="player") %>%

  mutate(player_value = 100 * ((reg_str_rate + sr_t20) + 1 /
                                 (reg_eco_rate + er_t20) +
                                 player_contri_rate)) %>%
  select(player, player_value) %>%
  arrange(desc(player_value)) %>%
  mutate(rank = row_number())

top_calibre_players %>%
  head(20)
```

```
## # A tibble: 20 x 3
##    player         player_value  rank
##    <chr>                 <dbl> <int>
##  1 SR Watson              286.     1
##  2 CH Gayle               279.     2
##  3 AD Russell             278.     3
##  4 AB de Villiers         274.     4
##  5 DA Warner              272.     5
##  6 YK Pathan              268.     6
##  7 SK Raina               265.     7
##  8 RR Pant                264.     8
##  9 SP Narine              264.     9
## 10 V Sehwag               258.    10
## 11 GJ Maxwell             254.    11
## 12 V Kohli                253.    12
## 13 KA Pollard             251.    13
## 14 RG Sharma              247.    14
## 15 DR Smith               246.    15
## 16 RV Uthappa             244.    16
## 17 DJ Bravo               243.    17
## 18 S Dhawan               242.    18
## 19 MS Dhoni               241.    19
## 20 SE Marsh               241.    20
```

6.1.5 TOP_150_CALIBRE_PLAYERS:

6.1.5.1 Top 150 players in terms of player value

Finally, since there are 8 teams in IPL and if we assume 18 players per team including extra players, we can have about 150 players in our list. Below is the list of top 150 players by their player value.

```
# Top 150 calibre players by player value
top_150_calibre_players <- top_calibre_players %>%
  head(150)

top_150_calibre_players%>%
  knitr::kable()
```

player	player_value	rank
SR Watson	286.2781	1
CH Gayle	279.4763	2
AD Russell	278.0665	3
AB de Villiers	274.2306	4
DA Warner	271.7955	5
YK Pathan	267.9109	6
SK Raina	265.3342	7
RR Pant	264.2226	8
SP Narine	264.0961	9
V Sehwag	257.7910	10
GJ Maxwell	253.8788	11
V Kohli	253.1660	12
KA Pollard	250.5207	13
RG Sharma	246.7369	14
DR Smith	246.3674	15
RV Uthappa	243.7751	16
DJ Bravo	243.0456	17
S Dhawan	242.2364	18
MS Dhoni	241.1492	19
SE Marsh	240.5851	20
KL Rahul	239.7893	21
AC Gilchrist	239.5414	22
JH Kallis	238.1602	23
JP Duminy	235.7496	24
Yuvraj Singh	235.4209	25
HH Pandya	235.3596	26
JA Morkel	234.4170	27
RA Jadeja	233.7764	28
CH Morris	232.1027	29
BB McCullum	232.0753	30
SPD Smith	231.5587	31
AM Rahane	230.9525	32
G Gambhir	230.7353	33
BJ Hodge	228.0052	34
Harbhajan Singh	227.9050	35
F du Plessis	227.1501	36
JC Buttler	226.5156	37
CA Lynn	225.9773	38

player	player_value	rank
SV Samson	225.9674	39
IK Pathan	225.3059	40
M Ali	224.8685	41
N Rana	224.8264	42
KD Karthik	224.8215	43
AT Rayudu	224.0341	44
AR Patel	222.3960	45
KH Pandya	221.9977	46
AJ Finch	221.5096	47
JP Faulkner	220.4721	48
ML Hayden	219.9535	49
WP Saha	219.9048	50
DA Miller	219.4818	51
M Vijay	219.4604	52
BA Stokes	219.1206	53
Shakib Al Hasan	217.7197	54
Q de Kock	217.4018	55
SR Tendulkar	217.2887	56
KK Cooper	217.1938	57
LMP Simmons	217.1079	58
KP Pietersen	217.0258	59
SS Iyer	216.7518	60
A Symonds	216.7315	61
AD Mathews	216.2824	62
Rashid Khan	214.9284	63
ST Jayasuriya	214.5834	64
MC Henriques	214.2633	65
MK Tiwary	213.8250	66
DPMD Jayawardene	213.8078	67
Azhar Mahmood	212.7629	68
RA Tripathi	211.9048	69
MF Maharoof	211.4654	70
MEK Hussey	210.4893	71
DJ Hussey	210.3496	72
MK Pandey	210.2057	73
STR Binny	209.9834	74
NLTC Perera	209.8843	75
MA Agarwal	209.8140	76
M Vohra	209.6447	77
HM Amla	208.8056	78
CR Brathwaite	208.6980	79
KS Williamson	208.4386	80
PA Patel	207.7451	81
KC Sangakkara	207.5114	82
LRPL Taylor	207.3271	83
K Gowtham	207.2508	84
DL Chahar	205.5721	85
M Morkel	204.9599	86
R Dravid	204.8285	87
JD Ryder	204.3287	88
KM Jadhav	204.2013	89
A Ashish Reddy	203.7592	90

player	player_value	rank
BCJ Cutting	203.6903	91
Mandeep Singh	203.6576	92
KK Nair	203.2654	93
TM Dilshan	203.0610	94
J Bairstow	202.5690	95
MJ McClenaghan	201.4567	96
P Shaw	201.2977	97
J Archer	200.7274	98
HV Patel	200.6798	99
AS Yadav	199.8031	100
CL White	199.6614	101
J Botha	199.3507	102
SA Yadav	199.3059	103
JR Hopes	199.0792	104
AJ Tye	198.7340	105
PP Chawla	198.1781	106
Bipul Sharma	198.1225	107
S Gill	197.9656	108
Umar Gul	197.8405	109
R Vinay Kumar	197.3019	110
DT Christian	196.9072	111
R Ashwin	196.7360	112
DW Steyn	196.5716	113
RN ten Doeschate	196.5103	114
Ishan Kishan	196.4574	115
SN Khan	196.0900	116
RS Bopara	196.0602	117
V Shankar	195.9876	118
C de Grandhomme	195.8497	119
SN Thakur	195.7367	120
Mohammad Nabi	195.5601	121
KV Sharma	195.5498	122
Ankit Sharma	195.1889	123
Shahid Afridi	194.9222	124
S Curran	194.5918	125
Gurkeerat Singh	193.7496	126
MS Gony	193.4382	127
S Gopal	193.0095	128
S Badrinath	192.9870	129
P Negi	192.4278	130
R Bhatia	192.2423	131
N Pooran	192.2009	132
LJ Wright	191.9673	133
SS Tiwary	191.7566	134
GJ Bailey	191.6742	135
PJ Cummins	191.4881	136
EJG Morgan	190.7330	137
TL Suman	190.2680	138
MP Stoinis	190.1025	139
AM Nayar	189.6730	140
MV Boucher	189.5303	141
OA Shah	189.1976	142

player	player_value	rank
HH Gibbs	189.1440	143
B Kumar	189.1200	144
TM Head	188.6646	145
MJ Guptill	188.6074	146
PD Collingwood	188.3372	147
NV Ojha	188.2495	148
D Wiese	188.0056	149
R Tewatia	187.6414	150

6.2 2nd Objective - Building a model with maximum F1 Score to predict the winner of a match:

Now, let us start addressing our second objective of predicting which team wins a particular match given a specific set of predictor variables. For this purpose, we use our matches dataset.

Given the nature of our dataset, which is small and extremely categorical, in fact all variables are factors, we can expect the prediction metrics to be not very high. This is also due to the fact that we have many classifiers.

In order to set about to achieve our objective, we will adopt the below explained conditions.

1. Since we have seen venue, batting_turn, toss wins have significant effect on the runs scored and matches won by teams, we will consider "venue", "toss_winner" & "toss_decision" variables along with "team1 (first team to bat)" & "team2 (second team to bat)" as predictor variables to predict the "winner (response variable)" of the match.
2. We will only consider the top 8 teams, which have played the most matches and also the current teams for our model building and predictions.
3. We have 8 different classifiers; 8 different teams that can win matches. Since this is a classification problem and our data is all factors, we use a few machine learning methods ("Naive Bayes", "Regression Tree", "Random Forest", "Multinomial Regression", "Linear Discriminant Analysis" and "K Nearest Neighbours") to train models and predict results.
4. Accuracy will not be the measure of our model prediction strength. This is because, we are not really interested in predicting negatives; rather we are more interested in positive predictions. Besides we have all factor data and many classifiers. Thus our metric of model evaluation will be "F1 Score", which is based on "recall" and "precision".
5. What the models do is to predict the winner of a match given the opponent, venue, toss winner and toss decision.
6. We will create two data sets, "train_set" and "test_set" for training and testing our models.

Let us start with some wrangling of our "matches" dataset to create the data sets for training and testing of models.

First, let us pick our top 8 teams from "matches_team" object that we have created earlier, which lists out teams by the number of matches played. We call this new list of 8 teams as "teams".

```
# Create the list of top 8 teams from matches_team (teams Vs matches played)
teams <- matches_team %>%
  top_n(8) %>%
  select (team)

teams
```

```
## # A tibble: 8 x 1
##   team
##   <chr>
## 1 Mumbai Indians
## 2 Royal Challengers Bangalore
## 3 Kolkata Knight Riders
## 4 Delhi Capitals
```

```
## 5 Kings XI Punjab
## 6 Chennai Super Kings
## 7 Rajasthan Royals
## 8 Sunrisers Hyderabad
```

Now, let us get into creating our master dataset, training and test datasets, check their dimensions and names, and train and test a few models. This involves some amount of data wrangling and pre-processing.

```
# Do required pre-processing and data wrangling
dat_set <- matches %>%
  select(first_bat_team = team1,
         second_bat_team = team2, winner, venue, toss_winner, toss_decision) %>%
  filter(winner != "" & first_bat_team %in% teams$team & second_bat_team %in% teams$team) %>%
  mutate_all(funs(str_replace_all(., "Chennai Super Kings", "CSK"))) %>%
  mutate_all(funs(str_replace_all(., "Delhi Capitals", "DC"))) %>%
  mutate_all(funs(str_replace_all(., "Kings XI Punjab", "KP"))) %>%
  mutate_all(funs(str_replace_all(., "Kolkata Knight Riders", "KKR"))) %>%
  mutate_all(funs(str_replace_all(., "Mumbai Indians", "MI"))) %>%
  mutate_all(funs(str_replace_all(., "Rajasthan Royals", "RR"))) %>%
  mutate_all(funs(str_replace_all(., "Royal Challengers Bangalore", "RCB"))) %>%
  mutate_all(funs(str_replace_all(., "Sunrisers Hyderabad", "SRH"))) %>%
  mutate(first_bat_team = as.factor(first_bat_team), second_bat_team = as.factor(second_bat_team),
         winner = as.factor(winner), venue = as.factor(venue), toss_decision = as.factor(toss_decision)
         toss_winner = as.factor(toss_winner))

any(is.na(dat_set))
```

```
## [1] FALSE
```

```
summary(dat_set)
```

```
##  first_bat_team second_bat_team     winner
##  MI     : 87    KKR    : 85     MI     : 94
##  KP     : 80    DC     : 83     CSK    : 89
##  CSK    : 75    RCB    : 79     KKR    : 77
##  RCB    : 71    MI     : 73     KP     : 67
##  KKR    : 69    CSK    : 71     RCB    : 67
##  DC     : 66    KP     : 71     RR     : 63
##  (Other):120    (Other):106     (Other):111
##                                        venue      toss_winner
##  Eden Gardens                             : 68   MI     : 87
##  M. Chinnaswamy Stadium                   : 65   CSK    : 81
##  Feroz Shah Kotla                         : 62   DC     : 79
##  Wankhede Stadium                         : 62   KKR    : 77
##  M. A. Chidambaram Stadium                : 49   RR     : 69
##  Punjab Cricket Association Stadium, Mohali: 41   KP     : 68
##  (Other)                                  :221   (Other):107
##  toss_decision
##  bat  :217
##  field:351
##
##
##
```

```
##
##
```

```
dim(dat_set)
```

```
## [1] 568   6
```

Note that we have renamed "team1" and "team2" variables as "first_bat_team" and "second_bat_team", replaced the names of teams by abbreviations and coerced all variables into factors. We do not have any NAs in our master dataset created and have 568 observations corresponding to different matches played between teams.

Now, let us create our train and test datasets and ensure all variable value labels in test dataset also exist in train dataset. We will also check the dimensions and ensure we have same variables in both train and test datasets.

```
# Limit the number of deciamal places to 4
options(digits=4)
# if using R 3.5 or earlier, use `set.seed(1)` instead - to get same results every time
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# test set will be approx 10% of our dat set
test_index <- createDataPartition(dat_set$winner, times = 1, p = 0.1, list = FALSE)

train_set <- dat_set[-test_index,]
temp_set <- dat_set[test_index,]

# Make sure all variable values in test set are also in train set
test_set <- temp_set %>%
  semi_join(train_set, by = "first_bat_team") %>%
  semi_join(train_set, by = "second_bat_team") %>%
  semi_join(train_set, by = "venue") %>%
  semi_join(train_set, by = "toss_decision") %>%
  semi_join(train_set, by = "toss_winner")

# Add rows removed from temp set back into train set
removed <- anti_join(temp_set, test_set)
train_set <- rbind(train_set, removed)

# Check dimensions & variable names of train_set and test_set
dim(train_set)
```

```
## [1] 508   6
```

```
dim(test_set)
```

```
## [1] 60  6
```

```
names(train_set)
```

```
## [1] "first_bat_team"  "second_bat_team" "winner"          "venue"
## [5] "toss_winner"     "toss_decision"
```

```
names(test_set)
```

```
## [1] "first_bat_team"  "second_bat_team" "winner"          "venue"
## [5] "toss_winner"     "toss_decision"
```

Let us start our model building to predict winner of matches subject to previously explained variables.

Since our data is non-numeric, particularly the response variable, linear regression will not work. Hence, let us try to build our models based on "Naive Bayes", "Regression Tree", "Random Forest", "Multinomial Regression", "Linear Discriminant Analysis" and "K Nearest Neighbours" methods, one after the other.

Let us start first with "Naive Bayes" method.

6.2.1 Model based on "Naive Bayes" method:

"Naive Bayes" model is expected to converge fast and is a good training method for a few yet independent predictor variables.

```
# Fit the Model based on "Naive Bayes" method, predict, test, calculate F1 score for all classes
fit_nb <- train(winner ~ ., method = "naive_bayes", data = train_set)
pre_nb <- predict(fit_nb, test_set)
F1_nb <- confusionMatrix(pre_nb, test_set$winner)$byClass[,"F1"]
F1_nb <- as.data.frame(t(F1_nb)) %>% mutate(avg_F1_score = rowMeans(.))
F1_nb
```

```
##   Class: CSK Class: DC Class: KKR Class: KP Class: MI Class: RCB Class: RR
## 1         NA        NA     0.2222        NA    0.4706         NA    0.6154
##   Class: SRH avg_F1_score
## 1     0.1961           NA
```

As we can see, though Naive Bayes quickly converged in training the model, it did not predict for certain classes, thereby did not produce F1 scores for those classes.

Please note that we are also calculating average F1 score, "avg_F1_score", for all the classes, to compare different models for the prediction strength. In this case, it did not produce an average score (resulted in NA).

At this point, let us also build a table to keep a note of all our results for different models as we go on improving and create our final model. For better readability sake, we will also replace the column names by getting rid of "Class:" portion from all columns (classes).

```
# Make column names more readable
colnames(F1_nb) = gsub("Class: ", "", colnames(F1_nb))
# F1 table for different models
F1_table <- data.frame(Model = "Naive Bayes") %>% bind_cols(F1_nb)

F1_table %>% knitr::kable()
```

Model	CSK	DC	KKR	KP	MI	RCB	RR	SRH	avg_F1_score
Naive Bayes	NA	NA	0.2222	NA	0.4706	NA	0.6154	0.1961	NA

6.2.2 Model based on "rpart" method:

Next, we try "rpart", a classification and regression tree algorithm, which will build a single tree to construct our model.

```
# Fit the Model based on "rpart (CART)" method, predict, test, calculate F1 score for all classes
fit_rp <- train(winner ~ ., method = "rpart", data = train_set)
pre_rp <- predict(fit_rp, test_set)
F1_rp <- confusionMatrix(pre_rp, test_set$winner)$byClass[,"F1"]
F1_rp <- as.data.frame(t(F1_rp)) %>% mutate(avg_F1_score = rowMeans(.))
F1_rp
```

```
##   Class: CSK Class: DC Class: KKR Class: KP Class: MI Class: RCB Class: RR
## 1     0.1667        NA     0.5882        NA    0.3333     0.5714        NA
##   Class: SRH avg_F1_score
## 1       0.25           NA
```

Though "rpart" did a better job predicting different classes, it too has produced NAs for a few classes. It too has the same deficiencies of our previous model in terms of not predicting certain classes.

Let us update our F1 table.

```
# Make column names more readable
colnames(F1_rp) = gsub("Class: ", "", colnames(F1_rp))
# Update F1 table - continued.2
F1_table <- bind_rows(F1_table,
                      data.frame(Model = "CART (rpart)") %>% bind_cols(F1_rp))

F1_table %>% knitr::kable()
```

Model	CSK	DC	KKR	KP	MI	RCB	RR	SRH	avg_F1_score
Naive Bayes	NA	NA	0.2222	NA	0.4706	NA	0.6154	0.1961	NA
CART (rpart)	0.1667	NA	0.5882	NA	0.3333	0.5714	NA	0.2500	NA

6.2.3 Model based on "Multinom" method:

Next, we will try multinomial logistic regression to train our model using "multinom" function. Since our response variable includes 8 classes, this model might provide better results predicting all classes.

```
# Fit the Model based on "multinom" method, predict, test, calculate F1 score for all classes
fit_mn <- train(winner ~ ., method = "multinom", data = train_set, trace = FALSE)
pre_mn <- predict(fit_mn, test_set)
F1_mn <- confusionMatrix(pre_mn, test_set$winner)$byClass[,"F1"]
F1_mn <- as.data.frame(t(F1_mn)) %>% mutate(avg_F1_score = rowMeans(.))
F1_mn
```

```
##   Class: CSK Class: DC Class: KKR Class: KP Class: MI Class: RCB Class: RR
## 1     0.2857    0.1818     0.5882    0.3636      0.72        0.4    0.3077
##   Class: SRH avg_F1_score
## 1     0.5714       0.4273
```

As expected, this model predicted for all classes and for the first time, we have the average F1 score. We can see using this model, our predictions will be the best for "Mumbai Indians" (MI) to be the winner with 0.72 F1 score. Not bad.

So, let us update our F1 table and move forward building our next model using Linear Discriminant Analysis (LDA) which works based on dimension reduction. Let us recall that toss winner and toss decision are highly correlated, hence, LDA might very well be able reduce the predictor dimensions.

```
# Make column names more readable
colnames(F1_mn) = gsub("Class: ", "", colnames(F1_mn))
# Update F1 table - continued.4
F1_table <- bind_rows(F1_table,
                      data.frame(Model = "Multinom") %>% bind_cols(F1_mn))

F1_table %>% knitr::kable()
```

Model	CSK	DC	KKR	KP	MI	RCB	RR	SRH	avg_F1_score
Naive Bayes	NA	NA	0.2222	NA	0.4706	NA	0.6154	0.1961	NA
CART (rpart)	0.1667	NA	0.5882	NA	0.3333	0.5714	NA	0.2500	NA
Multinom	0.2857	0.1818	0.5882	0.3636	0.7200	0.4000	0.3077	0.5714	0.4273

6.2.4 Model based on "LDA" method:

```
# Fit the Model based on "LDA" method, predict, test, calculate F1 score for all classes
fit_lda <- train(winner ~ ., method = "lda", data = train_set)
pre_lda <- predict(fit_lda, test_set)
F1_lda <- confusionMatrix(pre_lda, test_set$winner)$byClass[,"F1"]
F1_lda <- as.data.frame(t(F1_lda)) %>% mutate(avg_F1_score = rowMeans(.))
F1_lda
```

```
##   Class: CSK Class: DC Class: KKR Class: KP Class: MI Class: RCB Class: RR
## 1     0.6667    0.3636     0.6154    0.5455    0.6667     0.5882     0.625
##   Class: SRH avg_F1_score
## 1     0.6154       0.5858
```

This is great. We have now a much higher average F1 score of 0.585 (58.5%) and the LDA model had predicted much better than all the previous models. We can see that for most classes the F1 score is well above the average 0.50 (a random guess value for one among two teams to be the match winner).

Let us update our F1 table.

```
# Make column names more readable
colnames(F1_lda) = gsub("Class: ", "", colnames(F1_lda))
# Update F1 table - continued.5
F1_table <- bind_rows(F1_table,
                      data.frame(Model = "LDA") %>% bind_cols(F1_lda))
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

```
F1_table %>% knitr::kable()
```

Model	CSK	DC	KKR	KP	MI	RCB	RR	SRH	avg_F1_score
Naive Bayes	NA	NA	0.2222	NA	0.4706	NA	0.6154	0.1961	NA
CART (rpart)	0.1667	NA	0.5882	NA	0.3333	0.5714	NA	0.2500	NA
Multinom	0.2857	0.1818	0.5882	0.3636	0.7200	0.4000	0.3077	0.5714	0.4273
LDA	0.6667	0.3636	0.6154	0.5455	0.6667	0.5882	0.6250	0.6154	0.5858

6.2.5 Model based on "Random Forest" method:

Next, we will see how the "Random Forest" method works, where we build more trees and use cross validation unlike "rpart", which is based on only one tree.

```
# Fit the Model based on "rf (Random Forest)" method, predict, test,
# calculate F1 score for all classes
trainctrl <- trainControl(method="cv")
fit_rf <- train(winner ~ ., method = "rf", data = train_set,  trControl=trainctrl)
pre_rf <- predict(fit_rf, test_set)
F1_rf <- confusionMatrix(pre_rf, test_set$winner)$byClass[,"F1"]
F1_rf <- as.data.frame(t(F1_rf)) %>% mutate(avg_F1_score = rowMeans(.))
F1_rf
```

```
##   Class: CSK Class: DC Class: KKR Class: KP Class: MI Class: RCB Class: RR
## 1      0.375    0.3333     0.4444       0.2    0.5385     0.4615    0.5714
##   Class: SRH avg_F1_score
## 1     0.1818       0.3883
```

Though "Random Forest" did a decent job and predicted for all classes, its performance is not as good as the "Multinom" or "LDA" methods..

Let us update our F1 table and move to building and testing our final model, based on "KNN" method.

```
# Make column names more readable
colnames(F1_rf) = gsub("Class: ", "", colnames(F1_rf))
# Update F1 table - continued.3
F1_table <- bind_rows(F1_table,
                      data.frame(Model = "Random Forest (rf)") %>% bind_cols(F1_rf))
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

```
F1_table %>% knitr::kable()
```

Model	CSK	DC	KKR	KP	MI	RCB	RR	SRH	avg_F1_score
Naive Bayes	NA	NA	0.2222	NA	0.4706	NA	0.6154	0.1961	NA
CART (rpart)	0.1667	NA	0.5882	NA	0.3333	0.5714	NA	0.2500	NA
Multinom	0.2857	0.1818	0.5882	0.3636	0.7200	0.4000	0.3077	0.5714	0.4273
LDA	0.6667	0.3636	0.6154	0.5455	0.6667	0.5882	0.6250	0.6154	0.5858
Random Forest (rf)	0.3750	0.3333	0.4444	0.2000	0.5385	0.4615	0.5714	0.1818	0.3883

6.2.6 Model based on "KNN" method:

Now, for the last time, we build our final Model based on a slightly different technique, K Nearest Neighbors (KNN) method. Let us hope it improves our results.

```
# Fit the Model based on "KNN" method, predict, test, calculate F1 score for all classes
fit_knn <- train(winner ~ ., method = "knn", data = train_set)
pre_knn <- predict(fit_knn, test_set)
F1_knn <- confusionMatrix(pre_knn, test_set$winner)$byClass[,"F1"]
F1_knn <- as.data.frame(t(F1_knn)) %>% mutate(avg_F1_score = rowMeans(.))
F1_knn
```

```
##   Class: CSK Class: DC Class: KKR Class: KP Class: MI Class: RCB Class: RR
## 1     0.3333       0.2     0.4444       0.4    0.5833        0.5    0.4615
##   Class: SRH avg_F1_score
## 1     0.2222       0.3931
```

KNN also predicted for all classes, however, the predictions and F1 score did not improve more than what LDA did.

Below is our final updated F1 table for different models that we used for training and testing.

```
# Make column names more readable
colnames(F1_knn) = gsub("Class: ", "", colnames(F1_knn))
# Update F1 table - Final
F1_table <- bind_rows(F1_table,
                      data.frame(Model = "KNN") %>% bind_cols(F1_knn))
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

```
F1_table %>% knitr::kable()
```

Model	CSK	DC	KKR	KP	MI	RCB	RR	SRH	avg_F1_score
Naive Bayes	NA	NA	0.2222	NA	0.4706	NA	0.6154	0.1961	NA
CART (rpart)	0.1667	NA	0.5882	NA	0.3333	0.5714	NA	0.2500	NA
Multinom	0.2857	0.1818	0.5882	0.3636	0.7200	0.4000	0.3077	0.5714	0.4273
LDA	0.6667	0.3636	0.6154	0.5455	0.6667	0.5882	0.6250	0.6154	0.5858
Random Forest (rf)	0.3750	0.3333	0.4444	0.2000	0.5385	0.4615	0.5714	0.1818	0.3883
KNN	0.3333	0.2000	0.4444	0.4000	0.5833	0.5000	0.4615	0.2222	0.3931

7 Results Discussion:

We have set 2 objectives for this project, the first objective was to come out with a list of players based on their playing calibre (player_value) and the second objective was to develop a model to predict the winner of a match.

For the 1st objective, we have selected the following 3 criteria as the basis:

1. Player's ability as a batsman in terms of strike rate and as a bowler in terms of economy rate. We called these players as top_rate_players.
2. Player's contribution as both batsman and bowler in the matches that their teams have won and lost. We called these players as top_contri_players.
3. Player's ability as a batsman to have high strike rates against top economy bowlers and as a bowler to have low economy rates against top striking batsmen. We called these players as top_excel_players.

The resulting lists that we have got in all the above 3 cases are consistent with IPL's official statistics and facts and also that of other sports research entities.

We have combined these 3 lists into one to arrive at the final list estimating the player value (calibre). We called these players as top_calibre_players. Again, this list is consistent with many other findings on IPL official statistics.

The main difference here to note is that this project is different from other researches from the fact that this project uniquely determines player calibre based on 3 very important criteria, which other findings/ research analysis did not do, hence is tremendously useful.

The traditional approach to player selection was purely as a batsman or bowler or an all-rounder. What is required are players with the ability to perform CONSISTANTLY good as as a STRIKE batsman and ECONOMIC bowler, a drastically different approach. Hopefully, this project might change the thought process of team managements in selecting players for their franchises in IPL.

Below is the top 150 players' list.

```
# Top 150 players by player value (Calibre)
top_150_calibre_players
```

```
## # A tibble: 150 x 3
##    player         player_value  rank
##    <chr>                 <dbl> <int>
##  1 SR Watson              286.     1
##  2 CH Gayle               279.     2
##  3 AD Russell             278.     3
##  4 AB de Villiers         274.     4
##  5 DA Warner              272.     5
##  6 YK Pathan              268.     6
##  7 SK Raina               265.     7
##  8 RR Pant                264.     8
##  9 SP Narine              264.     9
## 10 V Sehwag               258.    10
## # ... with 140 more rows
```

For the 2nd objective, we have selected a few variables from "matches" dataset based on our data analysis and intuition to predict the winner for a given match between 2 teams at a specific venue. We also have set certain data constraints and accordingly wrangled and pre-processed our data.

Since we want to have a model that correctly, positively predicts a winner of a match, subject to other variables, our interest is specifically in "F1 score", which is a combination of "precision" and "recall (sensitivity)" metrics, rather than accuracy, sensitivity, specificity, precision or any other metric alone. We have decided to use average F1 score to compare the strength of different models that we have built.

Building models, testing them for the accuracy or sensitivity or F1 score of predicted values is an iterative process, based on our understanding of the data, problem at hand, exploration and intuition.

In fact, all the stages of data science project are iterative and include data visualization, analysis, model building & testing repeated, several times. In the process, we use various techniques like data wrangling, plotting, data organization, regression, machine learning and modeling.

Accordingly, the approach we took was step-by-step, iterative and employing various analysis and model building techniques.

Excepting "Naive Bayes" and "rpart", all our models predicted for all classes and produced an average F1 score. "LDA" seems to be the best model for this project due to its high average F1 score compared to the other models. However, if our interest is to just predict for "Mumbai Indians" as winners, "Multinom" method does a better job.

The samples we have for different teams (number of observations) is not similar. Some teams have played much more matches than others. This introduces bias in our data and thus our models may not do effective job in predicting the winners. For example, "Chennai Super Kings", which is top consistant performing team did not play for 2 seasons. This fact could have probably reduced its prediction as the winner by some of the models in favour of other teams.

Below table shows the F1 scores calculated by different models for different classes of "winner" in our test dataset.

```
# Final F1 table
F1_table %>% knitr::kable()
```

Model	CSK	DC	KKR	KP	MI	RCB	RR	SRH	avg_F1_score
Naive Bayes	NA	NA	0.2222	NA	0.4706	NA	0.6154	0.1961	NA
CART (rpart)	0.1667	NA	0.5882	NA	0.3333	0.5714	NA	0.2500	NA
Multinom	0.2857	0.1818	0.5882	0.3636	0.7200	0.4000	0.3077	0.5714	0.4273
LDA	0.6667	0.3636	0.6154	0.5455	0.6667	0.5882	0.6250	0.6154	0.5858
Random Forest (rf)	0.3750	0.3333	0.4444	0.2000	0.5385	0.4615	0.5714	0.1818	0.3883
KNN	0.3333	0.2000	0.4444	0.4000	0.5833	0.5000	0.4615	0.2222	0.3931

The advantage of these models developed in this project is that if the datasets on international matches/ teams are used, the models can be adopted to rank players by their playing calibre and predict winners of matches for international T20 matches/ teams too (not just to IPL). In fact, these models can be adopted for similar purposes for domestic matches at any level as long as the corresponding datasets are available

8 Conclusion:

It is time to conclude a few points about the project.

- The main objective of this project to develop a model to list players by their calibre based on their performances in IPL T20 tournaments. Acoordingly, we have generated a list of players based on their playing calibre and ranked them. In the process we have also explored and unearthed some very interesting facts through statistics and visualisations.
- The other objective is to predict the winner of a match between 2 specific teams, subject to who wins the toss, what the toss decision is and which is the venue. We have achieved some good measure of success with "LDA", "Multinom" and "KNN" methods.

  The models that we have developed can help

  - IPL team managements to select best teams,

  - enthrall cricket lovers with some interesting statistics and

  - interested audience with some rewarding predictions on which team would win a match. One could particularly predict "Mumbai Indians" as the winner with relatively high certainity using "Multinom" model {or} other teams as the winners using "LDA" & a few other models, given a F1 score of more than 0.50, which is better than a guess.

  - The models developed can be adopted for international or domestic or any level of T20 matches/ teams.
- There are several other minor or rare criteria that contributes a player value such as fielding, catching, captaining, involvement, wicket keeping, etc., Since our dataset did not contain much of the related data, we were not employ the same at this point of time.
- The next logical step for this project is to gather additional data as mentioned in the above point and further work on the models and develop better and precise models.

  One area for improvement is the list of players include several retired players. We can update the player list with players who are currently playing.

  One other area is, in IPL T20 teams are restricted to recruit a certain maximum number of foreign players. Based on additional data, we can make 2 lists, one for Indian (domestic) players and one for foreign players, based on their value.

————————-END————————-