IPL Project

Tashen Naidoo

2023-06-24

The Indian Premiere League

The Indian Premiere League (IPL) is the biggest cricket league in the world and second biggest across all global sports. This competition is also one of the most viewed events, with this year's competition accumulating nearly 500 million views. It is without a doubt one of the most exhilarating competitions globally, as it offers spectators the opportunity to watch the best players in the world face each other.
The IPL has revolutionised the T20 format of the game and has changed the livelihoods of players. This is due to their financial system (auction process), which ensures that the player and the player's nation is compensated. This further helps with cricket development worldwide. Since its inception in 2008, the IPL has provided unparalleled entertainment for all cricket fans and helps young cricket players be exposed to international talent. As a cricket fan, and an IPL fan, I have decided to attempt to model the team's point outcome per season based on their past performances.

Intention

I intend to formulate a model that can predict the final points of a team for each season. There have mainly been 14 games per season, with the exception of 2012 having 16, meaning that each team can score a maximum of 28 points per season (36 in 2012), if they win all of their matches. Losing teams receive no points and games ending in a draw result in each team receiving 1 point. Scoring more points does not mean that a team wins the league, but the top four teams have the opportunity to win the tournament by competing in the play-off rounds. I intend to only formulate a model that predicts the team points for each season. However, the first step is to attain the relevant data.

Data

All data in this project has been sourced from Kaggle. Two data sets were utilised, as I wanted to add more variables to the *points* data set. These data sets first needed to be investigated, to ensure that they are reliable, and then sanitised before performing any supervised learning techniques. The variables included in the model are:

- season: provides the season year of the competition.
- rank: indicates the position that the team finished in each table.
- name: indicates the team's name.
- matchesplayed: the total number of games each team has played per season.
- matcheswon: the total number of games won by a team.

- matcheslost: the total number of games lost by a team.
- noresults: the total number of matches ended in a tie/draw.
- matchpoints: the total number of match points gained by a team.
- nrr: the net run rate per team. This variable indicates the difference in average runs scored per over for a team and the average runs per over scored against a team. It is essentially a performance indicator for the team and is used to differentiate team ranks when teams have the same amount of points.
- toss_prop: the proportion of toss decisions won by the team.

Since the commencement of the IPL, there has been 16 teams throughout the competition in the *points* data set. However, upon further investigation, there have been 18 teams across all IPL competitions - and the *all_match* data set also confirms this. This raises concern, as there are generally only 8 registered teams per season, meaning that the teams are constantly changing, which raises a concern for a predictive model. The data sets are adjusted as follows:

Points data set

This data set indicates the the amount of points, matches won, matches lost, matches drawn, net run rate, and rank of teams for seasons from 2008-2022. This is the main data set used for the model, as it contains most of the appropriate information required to predict a team's point outcome.

All Match data set

This data set contains the information contains games on IPL seasons ranging from 2008-2022. The variable of interest here is *toss_winner*, as I believe that it influences the probability of a team winning, which leads to more points acquired. This will be discussed later.

Altering the data

Team changes over the years. Deccan Chargers represents Hyderabad, a city in Indian, but unfortunately went bankrupt in 2012. The city is now represented by Sunrisers Hyderabad, but they are not a replacement for the Deccan Chargers. The Kochi Tuskers Kerala only existed for one season in the IPL (2011). In 2016/17, the Chennai Super Kings and Rajasthan Royals did not compete, as they were suspended over alleged illegal betting. This gave rise to the Rising Pune Supergiants and Gujarat Lions, who were disbanded once Chennai Super Kings and Rajasthan Royals re-entered the competition. The only IPL team name changes throughout the years have been Dehli Daredevils to Dehli Capitals and more recently Kings XI Punjab to Punjab Kings. It is important to note that these teams are the same, and only differ in name, so they should be treated the same. These changes had a significant impact on trying to create a predictive model with teams that are not present. Therefore, a criteria is needed for which teams are going to be included in the model.

```
source("code/data_load.R")
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0     v purrr   1.0.1
## v tibble  3.2.1     v dplyr   1.1.0
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.4     v forcats 0.5.2
```

```
## Warning: package 'tibble' was built under R version 4.2.3
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

## Warning: package 'knitr' was built under R version 4.2.3

## i Using "','" as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.
## Rows: 126 Columns: 12-- Column specification ------------------------------------------------------------
## Delimiter: ";"
## chr (5): name, short_name, nrr, for, against
## dbl (7): season, rank, matchesplayed, matcheswon, matcheslost, noresult, mat...
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.i Using "','" as de
## Rows: 950 Columns: 20-- Column specification ------------------------------------------------------------
## Delimiter: ";"
## chr  (17): City, Season, MatchNumber, Team1, Team2, Venue, TossWinner, TossD...
## dbl   (2): ID, Margin
## date  (1): Date
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#Points data set adjustment
points <- read_csv2("data/Points.csv") %>%
    janitor::clean_names() %>%
    mutate(rank = as.factor(rank)) %>%
    mutate(season = as.character(season)) %>%
    mutate(nrr = as.numeric(nrr)) %>%
    mutate(name = gsub("Kings XI Punjab", "Punjab Kings", name))
```

```
## i Using "','" as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.
## Rows: 126 Columns: 12-- Column specification ------------------------------------------------------------
## Delimiter: ";"
## chr (5): name, short_name, nrr, for, against
## dbl (7): season, rank, matchesplayed, matcheswon, matcheslost, noresult, mat...
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#All Match data set adjustment
all_match <- read_csv2("data/matches.csv") %>%
    janitor::clean_names() %>%
    mutate(season = case_when(season == "2020/21" ~ "2020",
                              season == "Oct-09" ~ "2010",
                              season == "Aug-07" ~ "2008",
                              TRUE ~ season),
           season = as.character(season)) %>%
    mutate(toss_winner = case_when(toss_winner == "Delhi Daredevils" ~ "Delhi Capitals",
                                   toss_winner == "Kings XI Punjab" ~ "Punjab Kings",
                                   toss_winner == "Rising Pune Supergiants" ~ "Rising Pune Supergiant",
                                   TRUE ~ toss_winner),
           toss_winner = as.character(toss_winner))
```

```
## i Using "','" as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.
## Rows: 950 Columns: 20-- Column specification ----------------------------------------------------------
## Delimiter: ";"
## chr  (17): City, Season, MatchNumber, Team1, Team2, Venue, TossWinner, TossD...
## dbl   (2): ID, Margin
## date  (1): Date
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Now that the data has been loaded and transformed, the coin toss winner needs to be added. The toss decision in a game impacts the chances of the team wining the game. As with most sports, teams evaluate their opponent before they face each other. In cricket, a team might have a higher losing percentage when they don't get to bat first, or when they don't get to bowl first. This decision is made prior to the game with a coin toss. However, this can influence the teams chance of winning a game, especially if their opponent knows their weakness.

However, the impact of the coin toss decision goes beyond the simple analysis that teams have on each other. Weather plays an important role in the game. Weather conditions may make it favourable for a team to bowl first as overcast conditions cause the ball movement to vary greatly and make it more difficult for the batsman to retain their wicket, which essentially leads to more wickets and less runs. If conditions are dry and hot, teams will want to win the toss and elect to bat first, as the ball movement won't vary much, making it easier to protect your wicket, and the ball will move quicker off the bat leading to a higher score for the opposing team to chase. This dynamic makes it advantageous for a team to win the toss, they can elect to either bat/bowl based on several factors that give them the best chance of winning. With this being an important variable, I will need to calculate the coin toss win percentage per team using the *all_matches* data set and add it too the *points* data set.

```
source("code/toss_win_percentage.R")
```

```
## `summarise()` has grouped output by 'season'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'season'. You can override using the
## `.groups` argument.
## Joining with `by = join_by(season, toss_winner, proportion)`
```

```
t <- toss_win(all_match)
```

```
## `summarise()` has grouped output by 'season'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'season'. You can override using the
## `.groups` argument.
## Joining with `by = join_by(season, toss_winner, proportion)`
```

```
#add to the points data set
data <- points %>%
    left_join(t) %>%
 select(-'for', - against)
```

```
## Joining with `by = join_by(season, name)`
```

```
data
```

```
## # A tibble: 126 x 11
##    season rank  name      short~1 match~2 match~3 match~4 nores~5 match~6    nrr
##    <chr>  <fct> <chr>     <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
##  1 2008   1     Rajastha~ RR           14      11       3       0      22  0.632
##  2 2008   2     Punjab K~ KXIP         14      10       4       0      20  0.509
##  3 2008   3     Chennai ~ CSK          14       8       6       0      16 -0.192
##  4 2008   4     Delhi Ca~ DC           14       7       6       1      15  0.342
##  5 2008   5     Mumbai I~ MI           14       7       7       0      14  0.57
##  6 2008   6     Kolkata ~ KKR          14       6       7       1      13 -0.147
##  7 2008   7     Royal Ch~ RCB          14       4      10       0       8 -1.16
##  8 2008   8     Deccan C~ SRH          14       2      12       0       4 -0.467
##  9 2009   1     Delhi Ca~ DC           14      10       4       0      20  0.311
## 10 2009   2     Chennai ~ CSK          14       8       5       1      17  0.951
## # ... with 116 more rows, 1 more variable: proportion <dbl>, and abbreviated
## #   variable names 1: short_name, 2: matchesplayed, 3: matcheswon,
## #   4: matcheslost, 5: noresult, 6: matchpoints
```

The *for* and *against* variables can be ignored, as they are the odds that a team will be on top of the league for each season. Now that the data sets are joined and sanitised, I am able to select the teams for the model.

Data selection

Since the appropriate changes have been made to the data set, I now want to look at the most consistent teams in all seasons of the competition. I intend to split the data 70/30 (training/testing), which means that just under 10 seasons will be used in the training data. This suggests that teams that are present in at least 8 seasons of the IPL are used.

```
source("code/team_selection.R")
tt <- persistent(data)
tt
```

```
## # A tibble: 7 x 2
##   name                        seasons
##   <chr>                         <int>
## 1 Chennai Super Kings              13
## 2 Rajasthan Royals                 13
## 3 Delhi Capitals                   15
## 4 Kolkata Knight Riders            15
## 5 Mumbai Indians                   15
## 6 Punjab Kings                     15
## 7 Royal Challengers Bangalore      15
```

This indicates the 7 teams that are most consistent throughout the IPL. These are the teams that need for the model, so the data set needs to be filtered using their short-name in *data*.

```
data <-  data %>%
    filter(name %in% c("Rajasthan Royals","Chennai Super Kings","Punjab Kings", "Delhi Capitals", "Kolka
dim(points)
```

```
## [1] 126  12
```

```r
dim(data)
```

```
## [1] 101  11
```

The data has been reduced, but not by much and therefore should not affect the models outcome. However, I will find out if this affects the model or not by doing tests on both.

Model set-up

The *matchpoints* of each team at the end of the season will be the the *Target variable* of the model. All of the variables described above will be used in the model, except for the team rank. This variable is excluded as it is determined by the number of points a team accumulates each season.

Splitting and Model Evaluation

The data will be split 70 (training) and 30 (test) utilising the random sampling method. This method is preferred to the stratified sampling, as it does not require each sample to have the same characteristics, since the target variable outcome is based on differences in each team.

```r
source("code/splitting_and_modelling.R")
g <- fit1(data)
```

```
## Warning: package 'caret' was built under R version 4.2.3
```

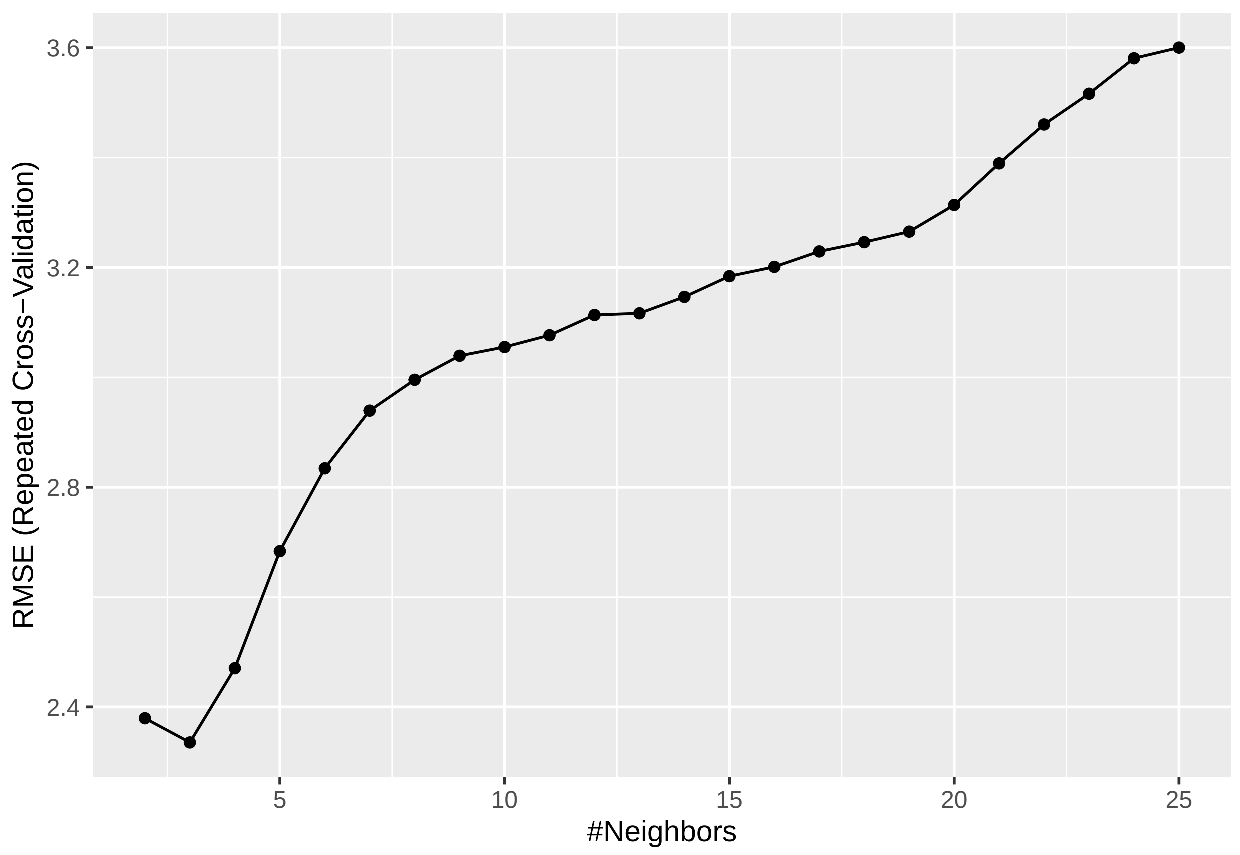
```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
## Warning: package 'rsample' was built under R version 4.2.3
```

```r
g
```



The above is used on the restricted data set, and indicates that the optimal k-fold level is when k=2. Here we have the lowest RMSE present and a great deal of the variation in the outcome is explained by the model (R-squared). However, I would like to compare this result to the unrestricted model. This contains the entire IPL point data set, with no threshold of minimum season requirements.

```r
#I need to work with the sanitised data sets and join them.
source("code/toss_win_percentage.R")
```

```
## ‘summarise()‘ has grouped output by ’season’. You can override using the
## ‘.groups‘ argument.
## ‘summarise()‘ has grouped output by ’season’. You can override using the
## ‘.groups‘ argument.
## Joining with ‘by = join_by(season, toss_winner, proportion)‘
```

```r
t <- toss_win(all_match)
```

```
## ‘summarise()‘ has grouped output by ’season’. You can override using the
## ‘.groups‘ argument.
## ‘summarise()‘ has grouped output by ’season’. You can override using the
## ‘.groups‘ argument.
## Joining with ‘by = join_by(season, toss_winner, proportion)‘
```

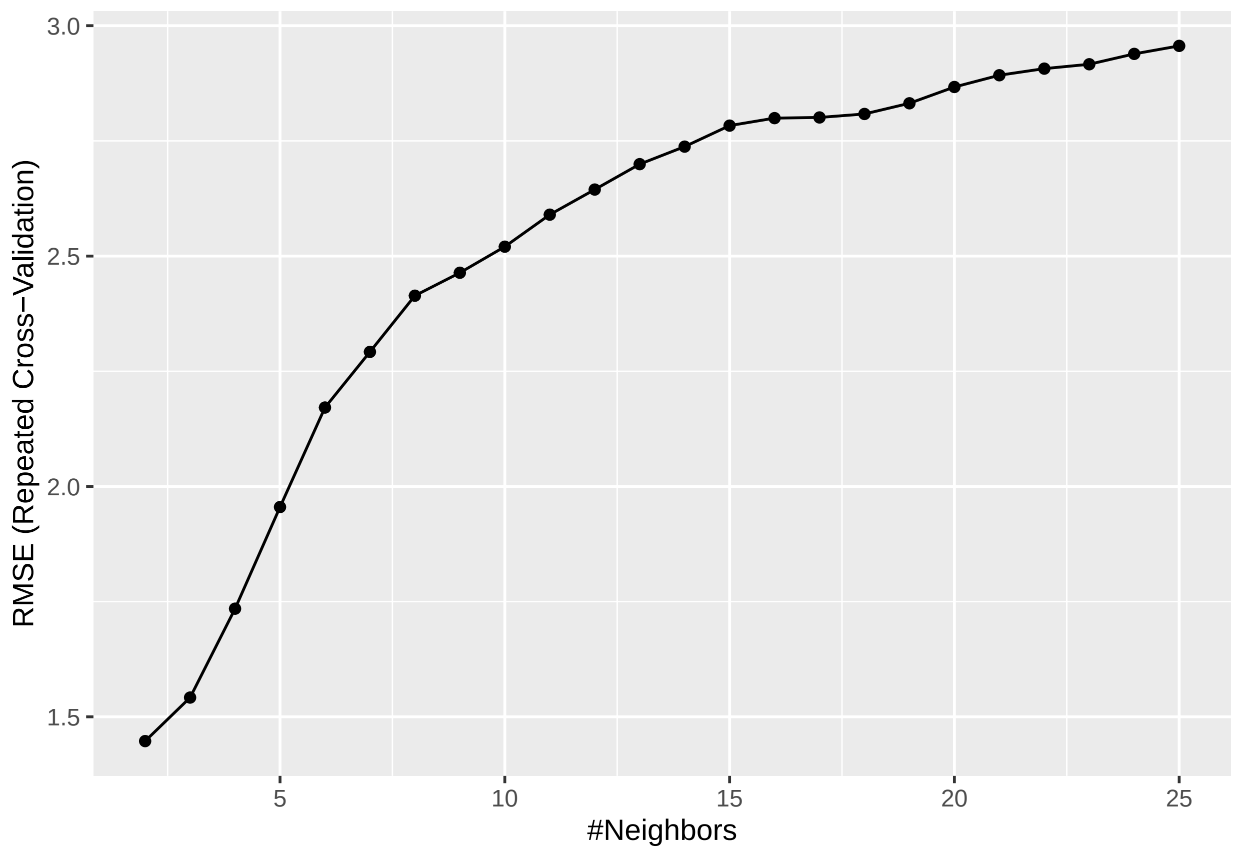
```r
points
```

```
## # A tibble: 126 x 12
##    season rank  name      short~1 match~2 match~3 match~4 nores~5 match~6    nrr
##    <chr>  <fct> <chr>     <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
##  1 2008   1     Rajastha~ RR           14      11       3       0      22  0.632
##  2 2008   2     Punjab K~ KXIP         14      10       4       0      20  0.509
##  3 2008   3     Chennai ~ CSK          14       8       6       0      16 -0.192
##  4 2008   4     Delhi Ca~ DC           14       7       6       1      15  0.342
##  5 2008   5     Mumbai I~ MI           14       7       7       0      14  0.57
##  6 2008   6     Kolkata ~ KKR          14       6       7       1      13 -0.147
##  7 2008   7     Royal Ch~ RCB          14       4      10       0       8 -1.16
##  8 2008   8     Deccan C~ SRH          14       2      12       0       4 -0.467
##  9 2009   1     Delhi Ca~ DC           14      10       4       0      20  0.311
## 10 2009   2     Chennai ~ CSK          14       8       5       1      17  0.951
## # ... with 116 more rows, 2 more variables: ‘for‘ <chr>, against <chr>, and
## #   abbreviated variable names 1: short_name, 2: matchesplayed, 3: matcheswon,
## #   4: matcheslost, 5: noresult, 6: matchpoints
```

```
dat <- points %>%
    left_join(t) %>%
     select(-'for', - against)
```

```
## Joining with ‘by = join_by(season, name)‘
```

```
#split and fit the model
source("code/alt_fit.R")
fit2(dat)
```



The unrestricted model appears to have a lower RMSE than the restricted data set. Therefore, this data set must be carried forward.

Engineering

I need to assess the model and make the necessary changes to improve the RMSE of the model. Each of the steps that follow is to improve the model, by investigating the variables in the model.

Missing data?

```
colSums(is.na(dat))
```

```
##        season          rank          name    short_name matchesplayed 
##             0             0             0             0             0 
##    matcheswon   matcheslost      noresult   matchpoints           nrr 
##             0             0             0             0             0 
##    proportion 
##             0
```

No data is missing from this data set. Now the variance of each variable needs to be assessed.

```
#variance test
    library(caret)
    library(rsample)
    (set.seed(123))
```

```
## NULL
```

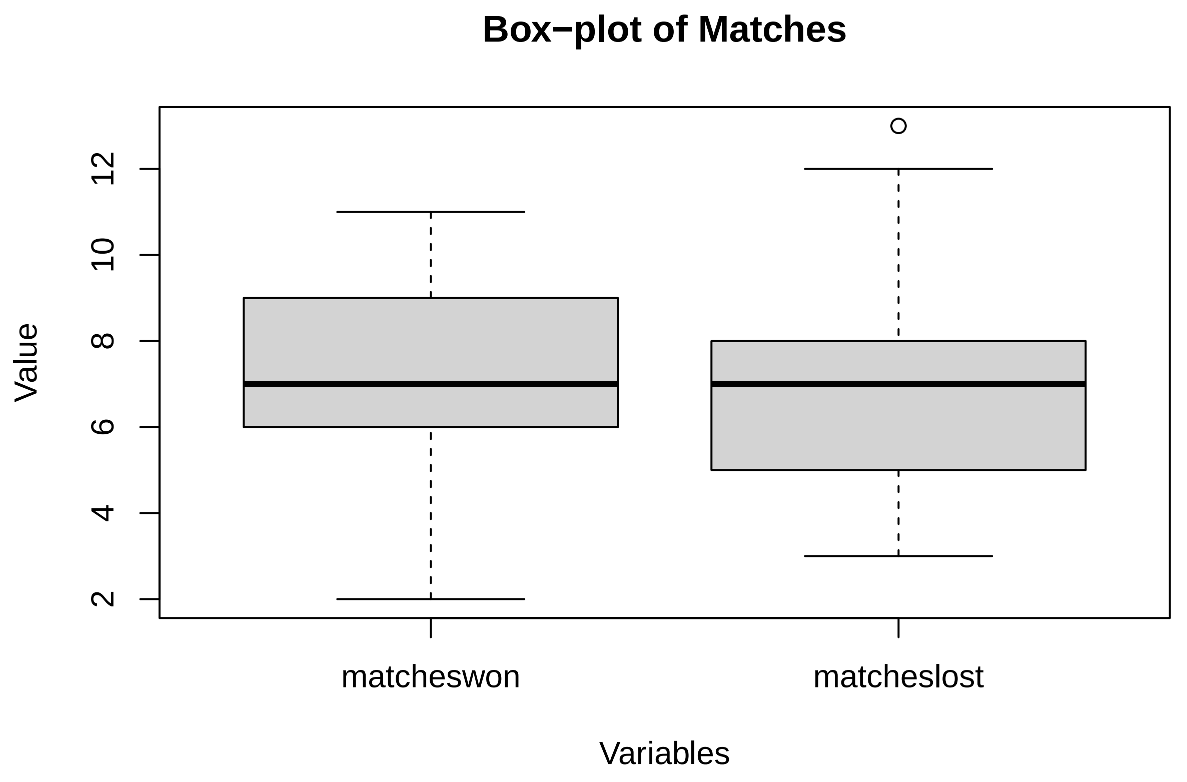
```
    split1 <- initial_split(dat, prop = 0.7, strata = "matchpoints")
    train1 <- training(split1)
    test1 <- testing(split1)

caret::nearZeroVar(train1, saveMetrics = TRUE) %>%
    tibble::rownames_to_column() %>%
    filter(nzv)
```

```
## [1] rowname       freqRatio     percentUnique zeroVar       nzv          
## <0 rows> (or 0-length row.names)
```

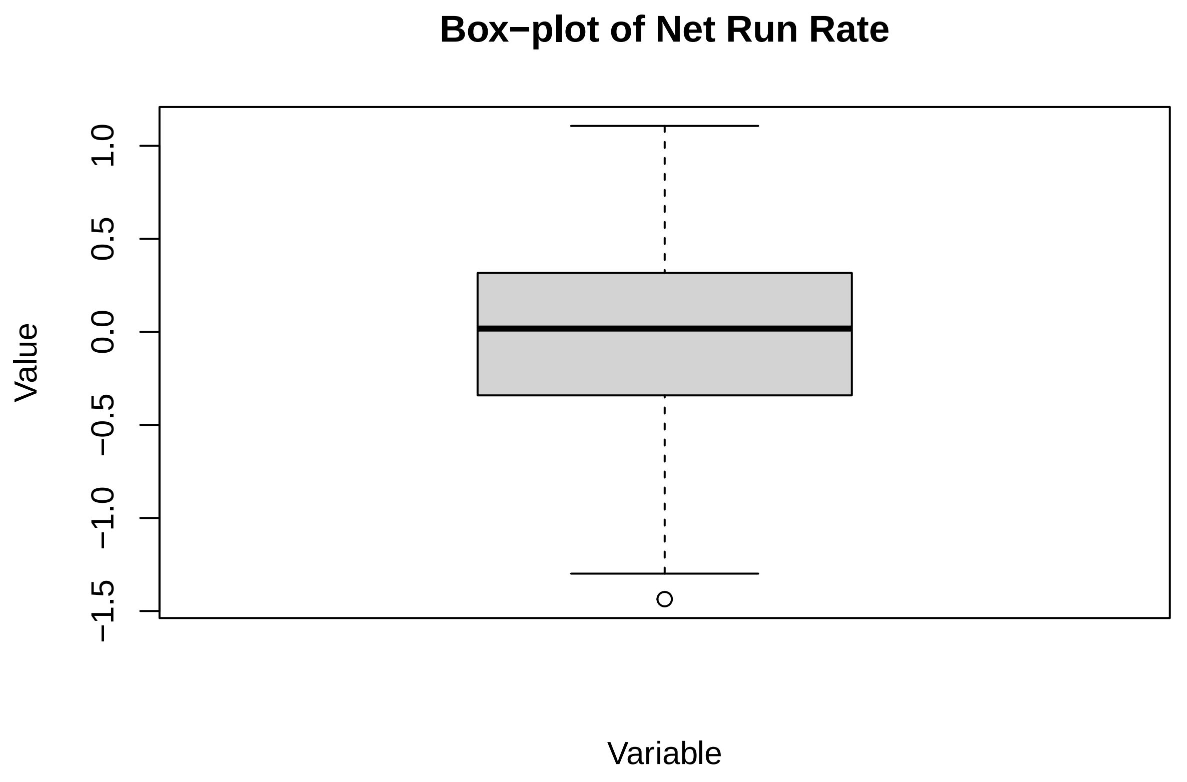
This result indicates that all the variables should remain in the model, as the variance in each is what is influencing the target. This is no surprise, as the variables in the model are all included due to their impact on the position of the team at the end of each season.

All of the predictor variables are numeric, so it would be wise to normalise the distribution of the data. Now the distribution of each variable should be examined.

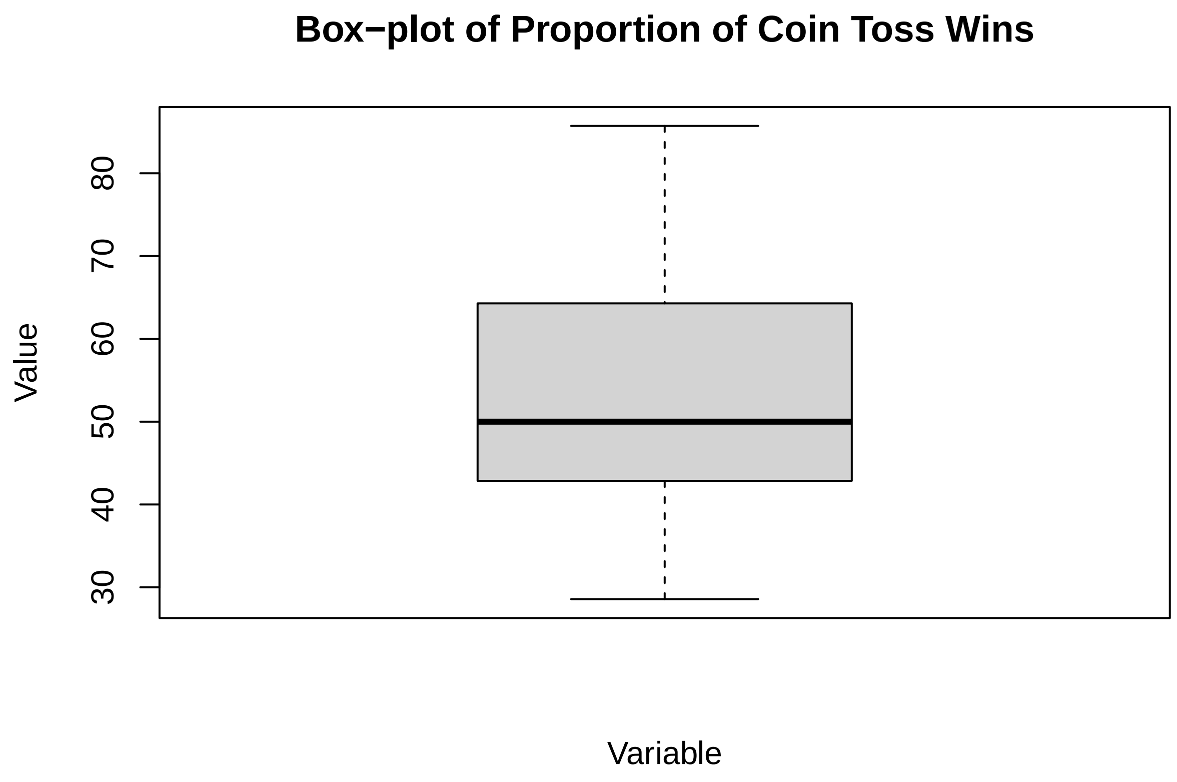
```
#Matches won and lost
source("code/box.R")
box(dat)
```



```r
#Net run rate
source("code/box.R")
box1(dat)
```



```r
#Proportion of coin tosses won
source("code/box.R")
box2(dat)
```



The spread of the data does not vary too much, but the proportion does. However, all variables will be normalised using the recipes package. This will all be accounted for in the next step.

Model evaluation

Since there is no missing data and variance is present, all the variables will help the model. The distribution of each variable differs, as seen above. This means that it is probably best to standardise the variables, which will improve the model overall.

Now I will apply all of these steps in one chunk and evaluate the model.

```
library(recipes)
```

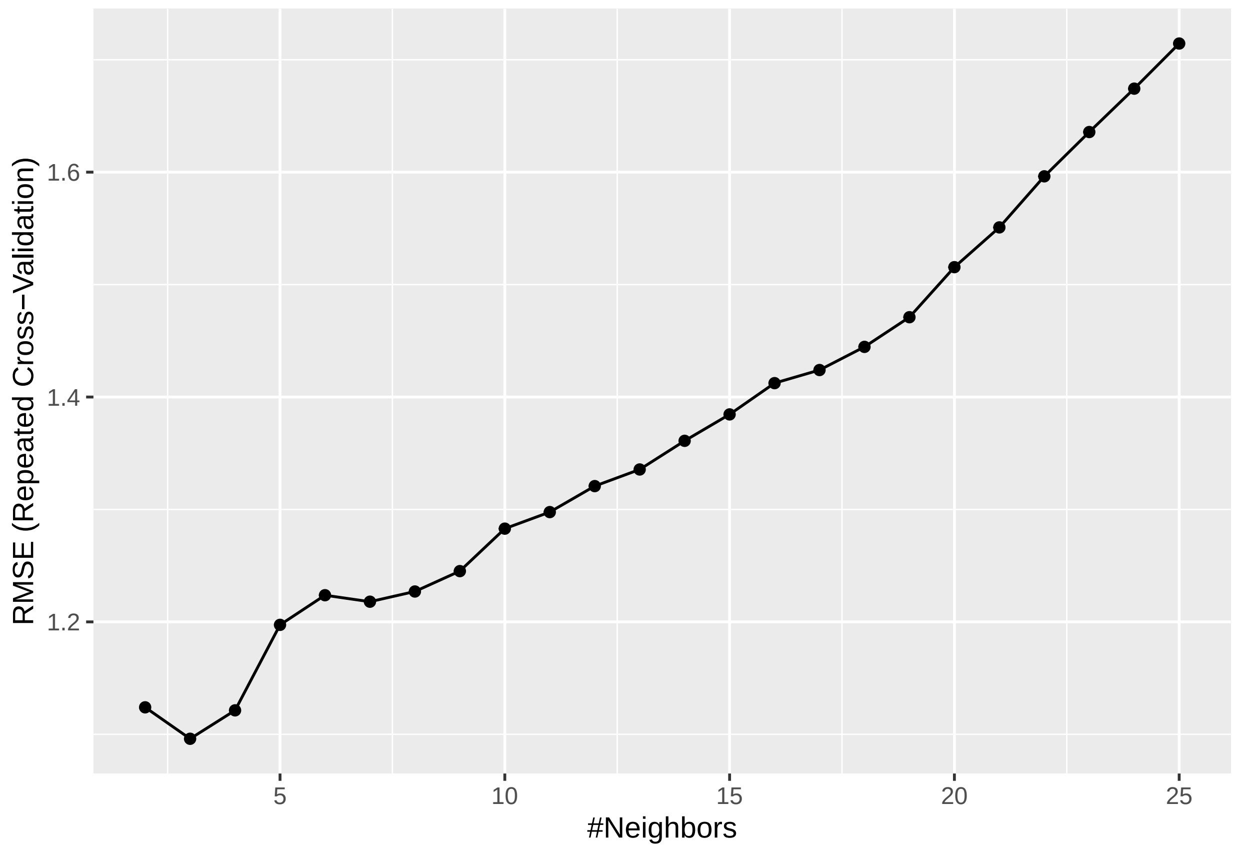
```
## Warning: package 'recipes' was built under R version 4.2.3
```

```
##
## Attaching package: 'recipes'
```

```
## The following object is masked from 'package:stringr':
##
##     fixed
```

```
## The following object is masked from 'package:stats':
##
##     step
```

```r
blue <- recipe(matchpoints ~ matcheswon+matcheslost+noresult+
                      nrr+proportion, data = train1) %>%
    step_nzv(all_nominal()) %>%
    step_center(all_numeric(), - all_outcomes()) %>%
    step_scale(all_numeric(), -all_outcomes())
#resample
c <- trainControl(method = "repeatedcv",
                  number = 10,
                  repeats = 5)
hgrid <- expand.grid(k = seq(2, 25, by = 1))
knn <- train(blue,
             data = train1,
             method = "knn",
             trControl = c,
             tuneGrid = hgrid,
             metric = "RMSE")
ggplot(knn)
```



This shows that the the RMSE has been lowered, which means that the model has been improved. The steps implemented have resulted in a better model. Lets test this on the test data.

```r
prepr <- prep(blue, training = train1)
prepr
```

```
##
```

```
## -- Recipe ----------------------------------------------------------------------

##

## -- Inputs

## Number of variables by role

## outcome:   1
## predictor: 5

##

## -- Training information

## Training data contained 87 data points and no incomplete rows.

##

## -- Operations

## * Sparse, unbalanced variable filter removed: <none> | Trained

## * Centering for: matcheswon, matcheslost, noresult, nrr, proportion | Trained

## * Scaling for: matcheswon, matcheslost, noresult, nrr, proportion | Trained
```

Now I am able to apply this to the test data.

```
baked <- bake(prepr, new_data = train1)
baked_t <- bake(prepr, new_data = test1)
baked
```

```
## # A tibble: 87 x 6
##    matcheswon matcheslost noresult    nrr proportion matchpoints
##         <dbl>       <dbl>    <dbl>  <dbl>      <dbl>       <dbl>
##  1     -1.04        0.527    1.53   0.563     -0.771          11
##  2     -2.04        1.51     1.53  -1.74      -0.234           7
##  3     -0.543       0.527   -0.519 -1.15      -0.771          12
##  4     -0.543       0.527   -0.519  0.404     -0.771          12
##  5     -0.543       0.527   -0.519 -0.519      0.303          12
##  6     -1.54        1.02     1.53  -0.349     -1.31            9
##  7     -1.54        1.02     1.53  -1.01       0.841           9
##  8     -1.54        2.01     1.53  -1.14       0.236           9
##  9     -1.54        2.50    -0.519 -1.23      -1.17            8
## 10     -1.54        2.50    -0.519 -2.20       0.841           8
## # ... with 77 more rows
```

```
baked_t
```

```
## # A tibble: 39 x 6
##    matcheswon matcheslost noresult    nrr proportion matchpoints
##         <dbl>       <dbl>    <dbl>  <dbl>      <dbl>       <dbl>
##  1     1.94       -1.94     -0.519  1.27       1.91           22
##  2     0.452      -0.459    -0.519 -0.472     -1.31           16
##  3    -1.54        1.51     -0.519 -2.52      -1.31            8
##  4    -2.53        2.50     -0.519 -1.05       0.841           4
##  5    -0.0458      0.0340   -0.519  0.364      1.38           14
##  6    -0.0458      0.0340   -0.519 -1.09      -0.771          14
##  7     1.45       -1.45     -0.519  2.23       0.841          20
##  8    -1.54        1.51     -0.519 -1.08      -1.31            8
##  9     1.45       -0.459    -0.519 -0.277      1.18           20
## 10     0.452       0.0340    1.53  -0.112     -1.17           17
## # ... with 29 more rows
```

This shows that the model has improved. A concern is that I am not able view the team name and season, and not sure how to incorporate this into the model.

The addition of different variables may have a strong impact on the models ability to predict the team points per season, such as average wickets per season. This model is limited by the data available. However, I believe that the methodology followed provides a good foundation for predicting the team points per season in the IPL.