Prediction of Top 10 Batsmen and Bowlers in IPL

Team Member Name	CWID
Anusha Koyilakonda	A20113590
Arjun Achuthan	A20115366
Darshini Priya Konanki	A20108860
Shriraam Murali	A20076092

OKLAHOMA STATE UNIVERSITY
STILLWATER

Contents

EXECUTIVE SUMMARY

The Indian Premier League (IPL) is a professional Twenty20 cricket league in India contested during April and May of every year by teams representing Indian cities. The league was founded by the Board of Control for Cricket in India (BCCI) in 2007 and is regarded as the mother of all cricketing leagues.

The IPL is the most-attended cricket league in the world and in 2014 it was ranked sixth by average attendance among all sports leagues. The brand value of IPL in 2017 was evaluated as US$5.3 billion, according to Duff & Phelps. According to BCCI, the 2015 IPL season contributed ₹11.5 billion (US$182 million) to the GDP of the Indian economy.

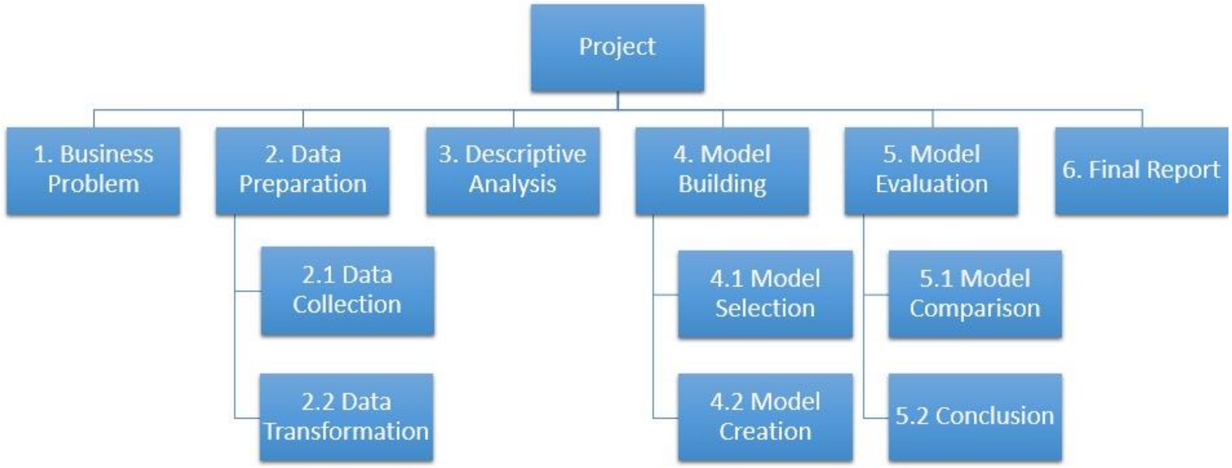
Similar to Baseball, Cricket is a bat-and-ball game between two teams and each contains eleven players who take turns to bat and field. While betting is illegal in India, it is allowed in other countries and predictions could be very useful in such countries. Also, IPL conducts a fantasy league competition via its website which allows the fans to choose XI players based on their interest and awards cash prizes at the end to those who predicted that the players would perform well. This could also be helpful for the owners during the next year auction where a team is allowed to buy or sell a player.

PROJECT SCHEDULE

Date and Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
Project Phase	25-Jan-18	1-Feb-18	8-Feb-18	15-Feb-18	22-Feb-18	1-Mar-18	8-Mar-18	15-Mar-18	22-Mar-18	29-Mar-18	5-Apr-18	12-Apr-18	19-Apr-18	26-Apr-18
Team formation														
Brainstorm ideas for project and dataset identification														
Finalize objective and scope of the project														
Deliverable for Project Phase I														
Data access and consolidation														
Data cleaning														
Data transformation														
Data reduction														
Descriptive Statistics														
Document Preparation and Submission														
Deliverable for Project Phase II														
Revise objective and scope of the project									Spring Break					
Select Modeling Technique														
Data splitting and Sub Sampling														
Build Model and Assess Models														
Modify the models for better model identification														
Document Preparation and Submission														

Legends	
Planned Activity	
Finished Activity as planned	
Activity not finished as per planned	
On going Activity	

WORK BREAKDOWN STRUCTURE



RESOURCE ASSIGNMENT

LEVEL	WBS Code	Task	Task Description	Task Assigned To	Hours spent on the Task
1	1	Business Problem	Identification of the problem	4 (Whole Team)	5
Data Preparation					
2	2.1	Data Collection	Gathering the data	4 (Whole Team)	8
2	2.2	Data Transformation	Imputing the null values with appropriate values and removing the redundancies	2 (Arjun, Anusha)	5
3	3	Descriptive Analysis	Generating Summary Statistics	2 (Darshini, Shriraam)	8
Model Building					
4	4.1	Model Selection	Research done to select the appropriate model	4 (Whole Team)	8
4	4.2	Model Creation	Creating the selected model	2(Darshini, Shriraam)	12
Model Evaluation					
5	5.1	Model Comparison	Comparing and selecting the best model	2 (Anusha, Arjun)	6
5	5.2	Conclusion	Drawing conclusions from the models	4 (Whole Team)	5
6	6	Final Report	Document preparation and Submission	4 (Whole Team)	8

SCOPE OF THE PROJECT

- A cricketing team has 11 players, combination of batsmen, bowlers and a wicketkeeper. Batsmen are those who have expertise in scoring runs. Bowlers are those who propel the ball towards wicket defended by a batsman. The wicket-keeper in the sport of cricket is the player on the fielding side who stands behind the wicket or stumps being watchful of the batsman and be ready to take a catch, stump the batsman out and run out a batsman when occasion arises. A person who possess a good skill of both batting and bowling is called an all-rounder.

- At the end of every season, there will be an Orange Cap and a Purple Cap holder who represents the highest run getters and highest wicket takers respectively. The top 10 batsmen and bowlers are always into the limelight to get sold at a higher price in the upcoming seasons.
- We have consolidated the original data file into three data files namely BattingStatistics and BowlingStatistics based on the different skillsets needed to play the game.

OBJECTIVE OF THE PROJECT

- Our main target is to predict the top 10 batsmen and bowlers based on different factors that contributes that makes a player a better batsman and a bowler.
- The target variables for each of the scope is given as follows:
  - To Predict Top 10 Batsmen: Runs
  - To Predict Top 10 Bowlers: Wickets
- The predictor variable(s) will be determined later based on various analysis.

DATA PREPARATION

Data Access

No additional data were considered after the first deliverable. The data for this project was downloaded from the following website as csv file(s):

https://www.kaggle.com/manasgarg/ipl

There are two data files:

S No	File Name	File Size	No of Rows	No of Columns
1	Deliveries.csv	14.73 MB	150461	21
2	Matches.csv	114.35 KB	636	18

We chose this data set because it contains ball by ball record of all the matches that have been played in IPL so far, which will help us in predicting best players accurately. This dataset contains a good mixture of numerical and categorical variables which makes it ideal to perform analysis. We will be performing transformations on dataset to make analysis easier.

Data Transformation

The deliveries.csv is the ball-by-ball data of all the IPL matches including data of the batting team, batsman, bowler, non-striker, runs scored, etc. We are using data from deliveries.csv to create a consolidated data with the Batting statistics details of each player and another entity with bowling statistics of each player by aggregating bowling and batting details. We also include fielding statistics to predict those players who have good skills of fielding in addition to batting and bowling.

For every Batsmen, the following data is derived from the deliveries.csv file:

Batting stats:

- Player_Name
- Total_Innings
- NO

- Runs
- Balls_faced
- Hundreds
- Fifties
- Fours
- Sixes
- Average
- Strike_Rate

For a bowler, the following things were derived from the file deliveries.csv

Bowling Statistics:

- Bowler
- Runs_Given
- Balls_Delivered
- Wickets
- Average
- Strike_Rate
- Economy

We have used Aggregate function with length and sum option to derive the desired statistics.

```
################################################################
#=================BAtting Statistics====================#######
################################################################

#Try to find batsman overall stats, verify in cricbuzz
Runs = aggregate(datafile$batsman_runs, by=list(Category=datafile$batsman), FUN=sum)
names(Runs) <- c("Player_Name","Runs")

#Try to find number of 4s and 6s
da4 = datafile[datafile$batsman_runs==4,]
#aggregate(da4$batsman_runs/4, by=list(Category=da4$batsman), FUN=sum)
Fours = aggregate(da4$batsman_runs/4, by=list(Category=da4$batsman), FUN=sum)
names(Fours) <- c("Player_Name","Fours")

da6 = datafile[datafile$batsman_runs==6,]
Sixes = aggregate(da6$batsman_runs/6, by=list(Category=da6$batsman), FUN=sum)
names(Sixes) <- c("Player_Name","Sixes")

#Try to find scores greater than 50
da50=aggregate(datafile$batsman_runs, by=list(Category=datafile$match_id, Category=datafile$batsman), FUN=sum)
names(da50) <- c("Match_ID","Player_Name","Runs")
da50=da50[(da50$Runs>=50) & (da50$Runs<=99),]
Fifties = aggregate(da50$Runs, by=list(Category=da50$Player_Name), FUN=length)
names(Fifties) <- c("Player_Name","Fifties")


#Try to find scores greater than 100
da100=aggregate(datafile$batsman_runs, by=list(Category=datafile$match_id, Category=datafile$batsman), FUN=sum)
names(da100) <- c("Match_ID","Player_Name","Runs")
da100=da100[da100$Runs>=100,]
Hundreds = aggregate(da100$Runs, by=list(Category=da100$Player_Name), FUN=length)
names(Hundreds) <- c("Player_Name","Hundreds")
```

Data Cleaning

In our Batting and Bowling statistics data that we derived, we have few fields with data as “NA”. These were replaced by Number 0 by using the following command in R.

```
finalruns2[is.na(finalruns2)] <-0
```

No erroneous data was present in the original data set. Once the data transformation was done, the statistics were verified with various websites like cricbuzz which contains the same statistics.

Data Consolidation

The dataset that we selected has two data files as mentioned above.

- Deliveries.csv
- Matches.csv

We need only Deliveries.csv to do the Descriptive Statistics which did not involve any data consolidation. Though we created two files which contains the Batting Statistics and Bowling Statistics, we are not merging for the purpose of predicting players with different skillsets during the next deliverable.

Data Reduction

We removed the records of SuperOver from the original dataset. Super over is the extra over played when the match ends in a tie. This is not considered as legal ball that contributes towards the batsman or bowler's records. Thus, we can remove the records from our analysis as for Bowler's and batsman's statistics we do not need to consider the Super over records. The original data set contains 150000 odd rows which was reduced to 600 odd rows after the data transformation is done. We don't need to apply Factor analysis or Principle Component Analysis to reduce data further. None of the records were removed too.

DESCRIPTIVE STATISTICS

For all the numerical columns present in the 2 files, descriptive statistics are done and presented below:

Descriptive Statistics for Batting

```
> describe(finaldata[,-c(1)])
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
Total_Innings	1	465	20.87	30.03	8.0	13.79	8.90	1	157.00	156.00	2.31	5.24	1.39
NO	2	465	4.90	6.84	2.0	3.45	2.97	0	49.00	49.00	2.34	6.85	0.32
Runs	3	465	395.20	781.96	69.0	189.94	96.37	0	4540.00	4540.00	2.97	9.15	36.26
Balls_Faced	4	465	313.56	596.67	68.0	156.94	90.44	0	3403.00	3403.00	2.88	8.52	27.67
Hundreds	5	465	0.10	0.46	0.0	0.00	0.00	0	5.00	5.00	6.20	46.86	0.02
Fifties	6	465	1.95	5.15	0.0	0.57	0.00	0	36.00	36.00	3.84	16.69	0.24
Fours	7	465	36.60	75.58	6.0	16.59	8.90	0	484.00	484.00	3.08	9.92	3.50
Sixes	8	465	14.00	30.85	2.0	6.38	2.97	0	265.00	265.00	3.81	17.74	1.43
Average	9	465	15.51	10.83	13.8	14.72	11.56	0	55.67	55.67	0.64	-0.08	0.50
Strike_Rate	10	465	103.59	38.51	109.9	106.09	32.23	0	233.33	233.33	-0.59	0.69	1.79

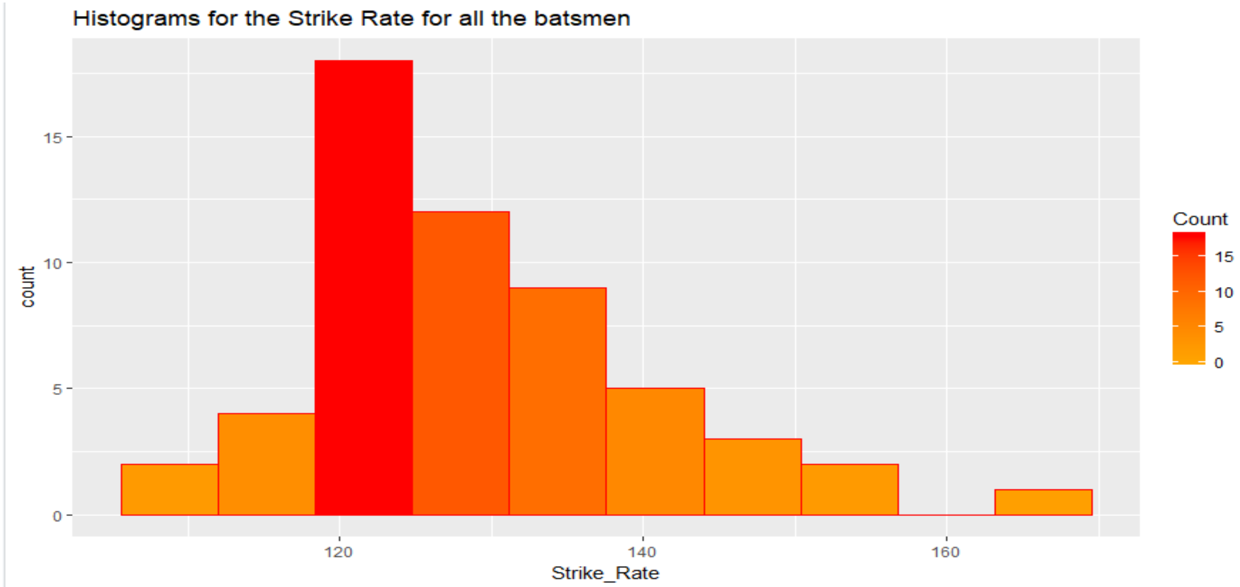
What makes a player a good Batsman?

A player is said to be a good batsman when he has the following characteristics:

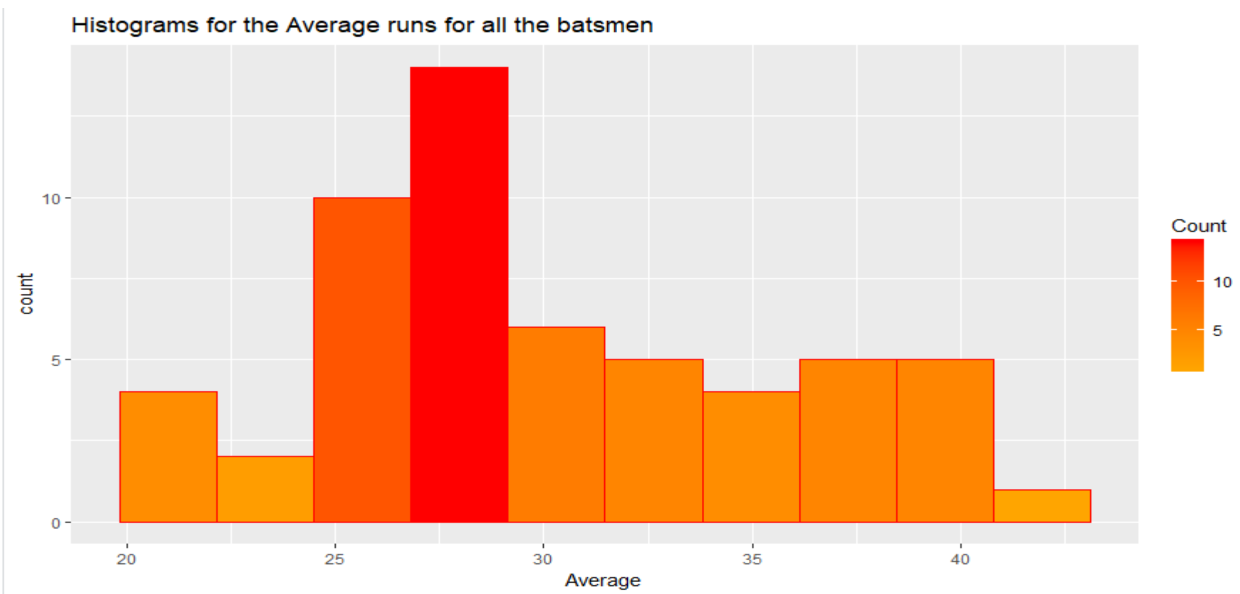
- Higher Average
- Higher Strike Rates
- Higher number of runs scored
- Higher number of Fours and Sixes commonly called as boundaries
- T20 being a shorter format of cricket, scoring a half century or a century is extremely difficult. Hence those players who have fifties and hundreds under their name are good batsman.

Histograms

Strike Rate

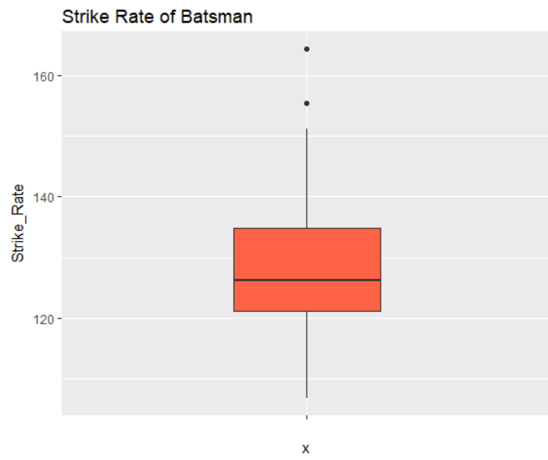

Histograms for the Strike Rate for all the batsmen
count
Strike_Rate
Count
15
10
5
0
120
140
160

Average

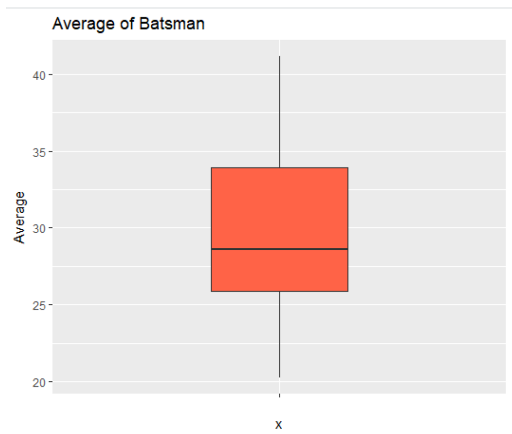

Histograms for the Average runs for all the batsmen
count
Average
Count
10
5
0
20
25
30
35
40

Box Plots

Strike Rate



Average



Top 10 Batsmen with minimum 1000 Runs who has better Strike Rates

```
> minruns = finaldata[finaldata$Runs>=1000,]
> topstr = head(minruns[order(minruns$Strike_Rate, decreasing= T),], n = 10)
> topstr
       Player_Name Total_Innings NO Runs Balls_Faced Hundreds Fifties Fours Sixes Average Strike_Rate
144     GJ Maxwell            56  7 1228         747        0       6    96    82   25.06      164.39
434      V Sehwag            104  5 2728        1755        2      16   334   106   27.56      155.44
86       CH Gayle            100 12 3626        2398        5      21   294   265   41.20      151.21
21  AB de Villiers           118 27 3473        2344        3      22   287   156   38.16      148.17
191     KA Pollard           113 32 2344        1599        0      12   157   147   28.94      146.59
456      YK Pathan           133 35 2904        1996        1      13   239   147   29.63      145.49
104      DA Warner           114 15 4014        2824        3      36   401   160   40.55      142.14
103      DA Miller            64 20 1563        1105        1       8   104    78   35.52      141.45
384       SK Raina           157 24 4540        3264        1      31   402   173   34.14      139.09
401      SR Watson            98 14 2622        1891        2      14   257   122   31.21      138.66
```

Top 10 Batsmen with minimum 1000 Runs who has better Average

```
> topavg = head(minruns[order(minruns$Average, decreasing= T),], n = 10)
> topavg
       Player_Name Total_Innings NO Runs Balls_Faced Hundreds Fifties Fours Sixes Average Strike_Rate
86        CH Gayle           100 12 3626        2398        5      21   294   265   41.20      151.21
104       DA Warner          114 15 4014        2824        3      36   401   160   40.55      142.14
220     LMP Simmons           29  2 1079         852        1      11   109    44   39.96      126.64
376        SE Marsh           69  7 2477        1866        1      20   266    78   39.95      132.74
185       JP Duminy           73 23 1993        1596        0      14   123    78   39.86      124.87
245      MEK Hussey           58  7 1977        1612        1      15   198    52   38.76      122.64
21  AB de Villiers           118 27 3473        2344        3      22   287   156   38.16      148.17
268        MS Dhoni          143 49 3560        2604        0      17   251   156   37.87      136.71
432         V Kohli          141 23 4418        3403        4      30   383   160   37.44      129.83
399       SPD Smith           62 16 1703        1293        1       5   150    45   37.02      131.71
```

Top 10 Batsmen with minimum 1000 Runs who has more Runs

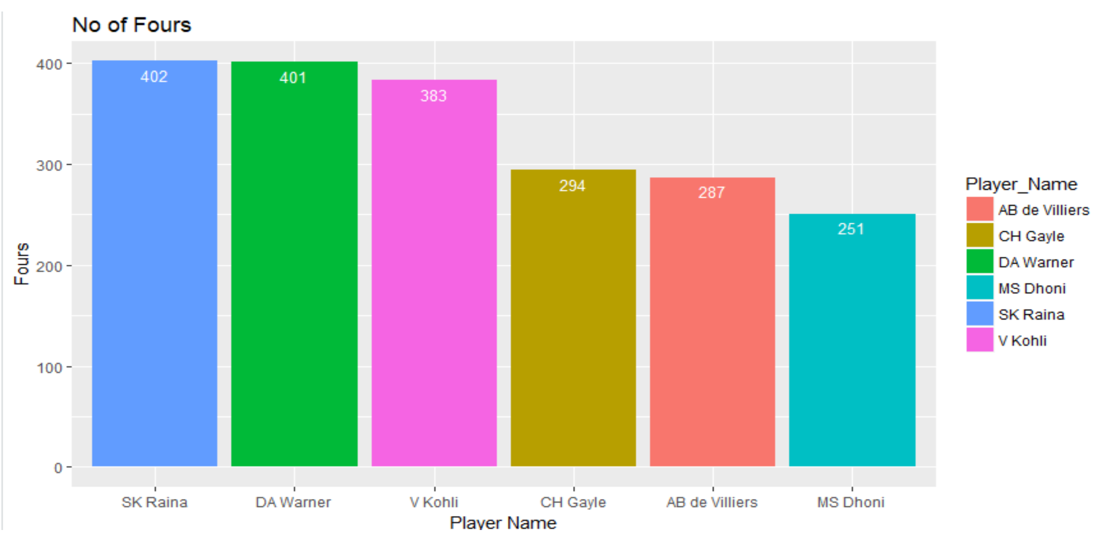
```
> topruns = head(finaldata[order(finaldata$Runs, decreasing= T),], n = 10)
> topruns
       Player_Name Total_Innings NO Runs Balls_Faced Hundreds Fifties Fours Sixes Average Strike_Rate
384       SK Raina           157 24 4540        3264        1      31   402   173   34.14      139.09
432        V Kohli           141 23 4418        3403        4      30   383   160   37.44      129.83
325      RG Sharma           154 25 4207        3214        1      32   354   173   32.61      130.90
138     G Gambhir           147 16 4132        3316        0      35   484    58   31.54      124.61
104      DA Warner           114 15 4014        2824        3      36   401   160   40.55      142.14
342    RV Uthappa           143 15 3778        2870        0      22   377   125   29.52      131.64
86        CH Gayle           100 12 3626        2398        5      21   294   265   41.20      151.21
348       S Dhawan           126 17 3561        2922        0      28   401    71   32.67      121.87
268       MS Dhoni           143 49 3560        2604        0      17   251   156   37.87      136.71
21  AB de Villiers           118 27 3473        2344        3      22   287   156   38.16      148.17
```

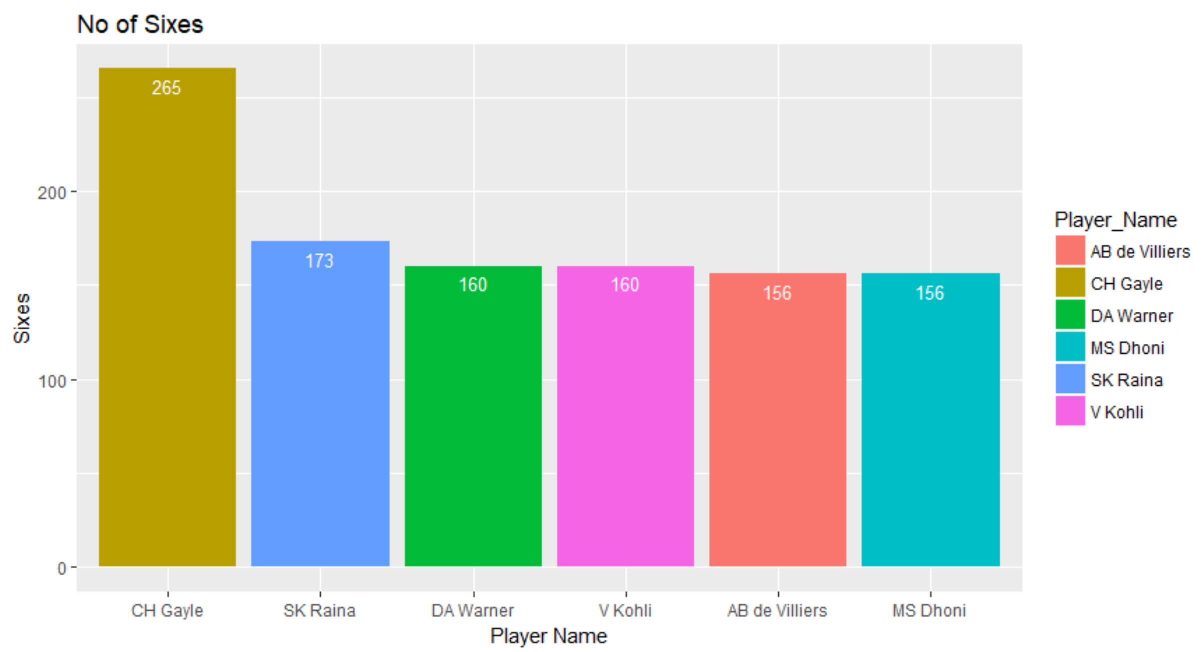
There are 6 players who are found common in at least 2 of the above 3 statistics. They are:

- SK Raina
- V Kohli
- CH Gayle
- MS Dhoni
- AB de Villiers
- DA Warner

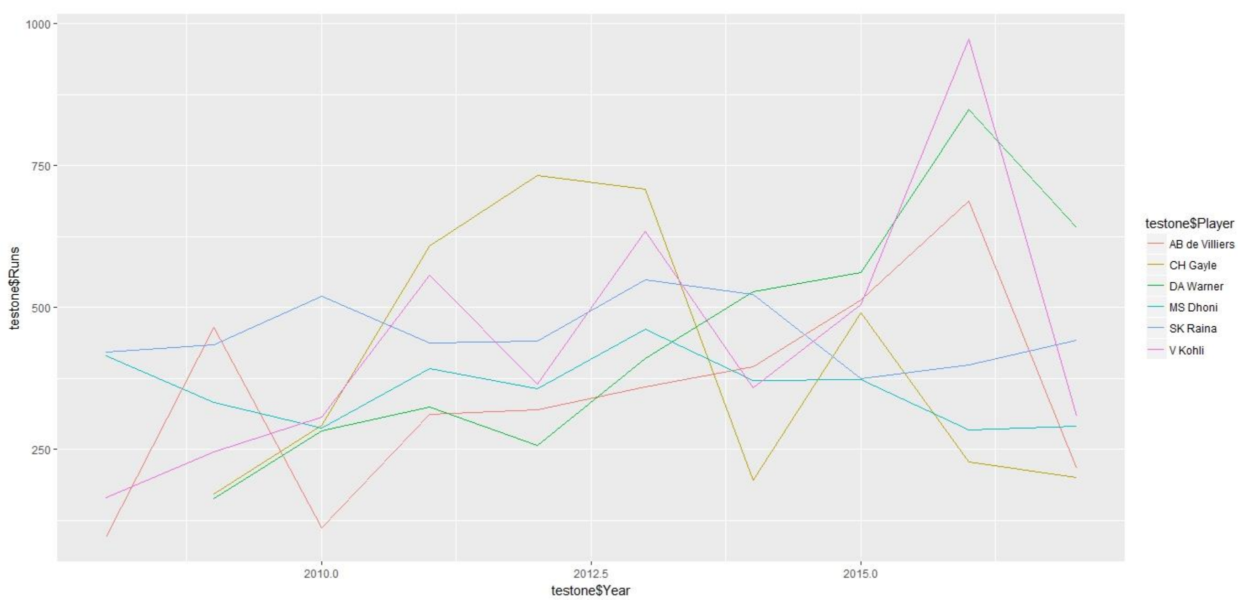
Bar Graphs

Boundaries (Fours and Sixes)





Timeline graph of runs scored by Batsmen across different seasons:



Descriptive Statistics for Bowling

```
> describe(bowl2[,c(2:6)])
                vars   n   mean     sd median trimmed    mad min  max range skew kurtosis    se
Runs_Given         1 356 533.89 708.83 242.00  375.21 286.14   0 3385  3385 1.99     3.48 37.57
Balls_Delivered    2 356 407.84 562.67 187.00  280.16 221.65   1 2919  2918 2.09     4.02 29.82
Wickets            3 356  18.72  27.19   8.00   12.45  10.38   0  154   154 2.23     4.94  1.44
Average            4 356  30.08  20.27  28.69   28.33  10.22   0  136   136 1.38     4.50  1.07
Strike_Rate        5 356  21.50  12.82  21.37   21.00   7.37   0   85    85 0.74     2.69  0.68
```

What makes a player a good Bowler?

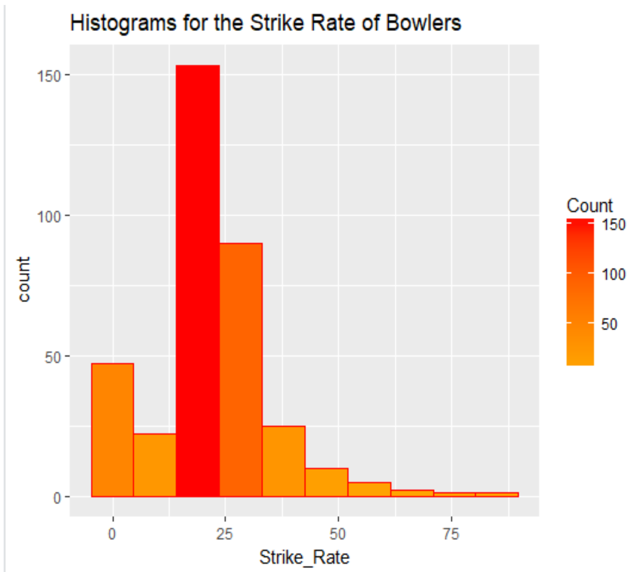
A player is said to be a good bowler when he has the following characteristics:

- Higher Wickets taken
- Lower Average

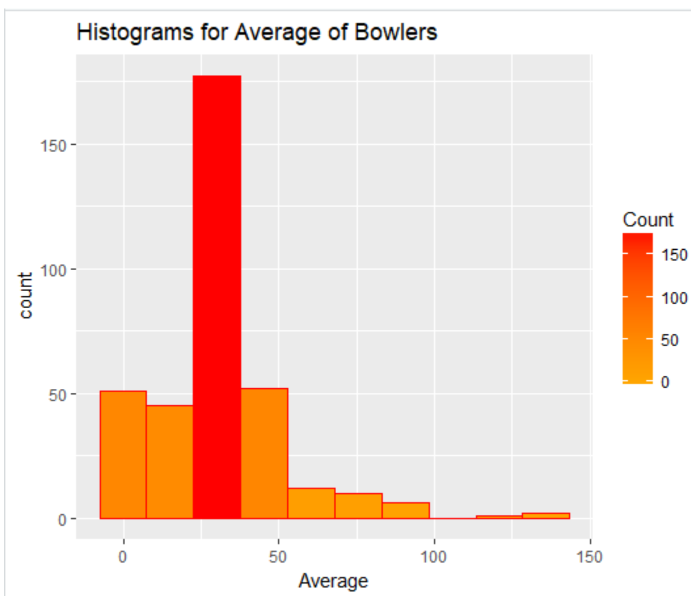
- Lower Economy
- Lower Runs given

Histograms

Strike Rate

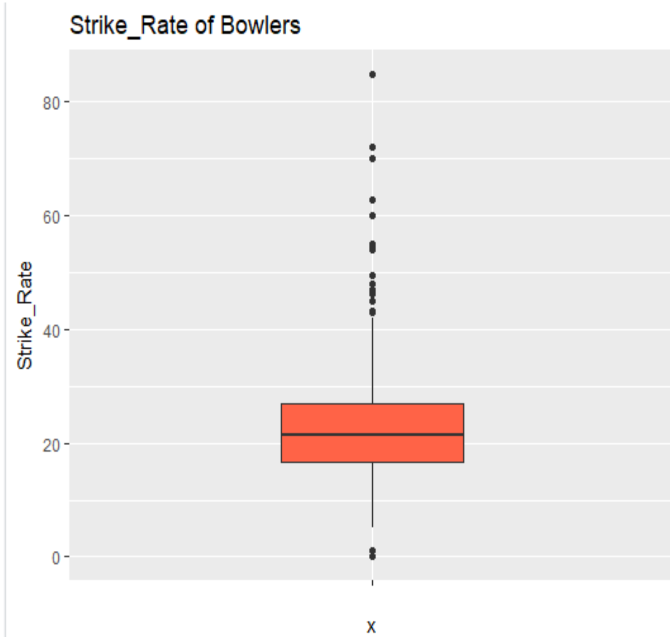


Average

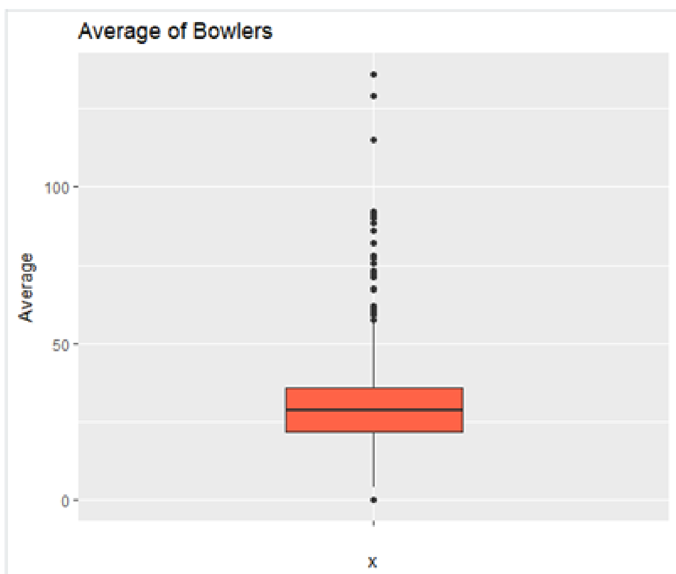


Box Plots

Strike Rate



Average



Top 10 bowlers with minimum 300 balls who has least Average:

```
> leastAvg = head(bowl3[order(bowl3$Average, decreasing= F),], n = 10)
> leastAvg
              Bowler Runs_Given Balls_Delivered Wickets Average Strike_Rate economy
88     DE Bollinger        693             576      37   18.73       15.57    7.22
25   AD Mascarenhas        356             308      19   18.74       16.21    6.94
303       SL Malinga       2932            2558     154   19.04       16.61    6.88
191      MF Maharoof        520             420      27   19.26       15.56    7.43
216 NM Coulter-Nile        719             563      36   19.97       15.64    7.66
187         MA Starc        693             580      34   20.38       17.06    7.17
207         MR Marsh        414             315      20   20.70       15.75    7.89
249      Rashid Khan        358             324      17   21.06       19.06    6.63
51           B Kumar       2339            1981     111   21.07       17.85    7.08
125      Imran Tahir        992             716      47   21.11       15.23    8.31
```

Top 10 bowlers with minimum 300 balls and have least Strike Rates:

```
> leastStrikeRate = head(bowl3[order(bowl3$Strike_Rate, decreasing= F),], n = 10)
> leastStrikeRate
              Bowler Runs_Given Balls_Delivered Wickets Average Strike_Rate economy
125      Imran Tahir        992             716      47   21.11       15.23    8.31
191      MF Maharoof        520             420      27   19.26       15.56    7.43
88      DE Bollinger        693             576      37   18.73       15.57    7.22
216 NM Coulter-Nile        719             563      36   19.97       15.64    7.66
207         MR Marsh        414             315      20   20.70       15.75    7.89
344        VY Mahesh        499             339      21   23.76       16.14    8.83
25    AD Mascarenhas        356             308      19   18.74       16.21    6.94
89          DJ Bravo       2755            2018     122   22.58       16.54    8.19
303       SL Malinga       2932            2558     154   19.04       16.61    6.88
10           A Singh        620             473      28   22.14       16.89    7.86
```

Top 10 bowlers with who bowled a minimum 300 balls and have least economy:

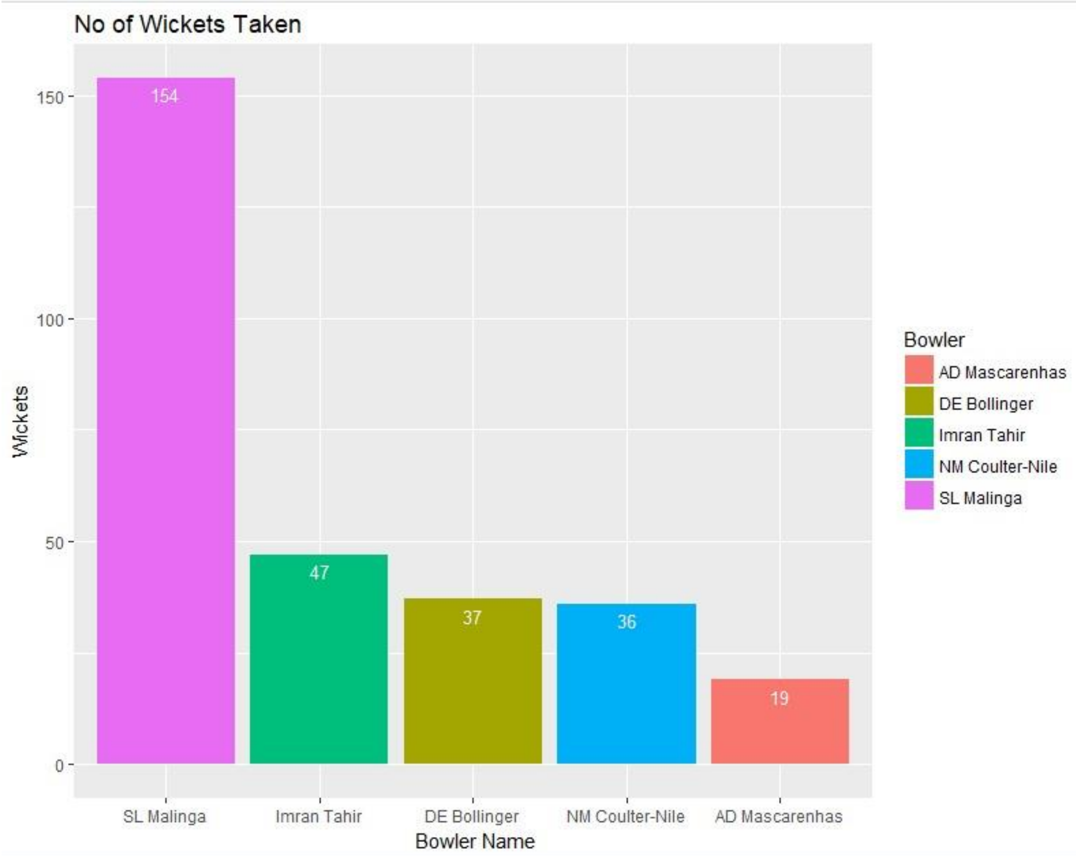
```
> leastEconomy = head(bowl3[order(bowl3$economy, decreasing= F),], n = 10)
> leastEconomy
               Bowler Runs_Given Balls_Delivered Wickets Average Strike_Rate economy
310         SP Narine       2031            1925      95   21.38       20.26    6.33
235          R Ashwin       2499            2290     100   24.99       22.90    6.55
5            A Kumble       1058             965      45   23.51       21.44    6.58
110        GD McGrath        357             324      12   29.75       27.00    6.61
249       Rashid Khan        358             324      17   21.06       19.06    6.63
183    M Muralitharan       1696            1524      63   26.92       24.19    6.68
104          DW Steyn       2306            2058      92   25.07       22.37    6.72
251 RE van der Merwe        498             443      21   23.71       21.10    6.74
97         DL Vettori        878             777      28   31.36       27.75    6.78
303        SL Malinga       2932            2558     154   19.04       16.61    6.88
```

There are 8 players who are found common in at least 2 of the above 3 statistics. They are:

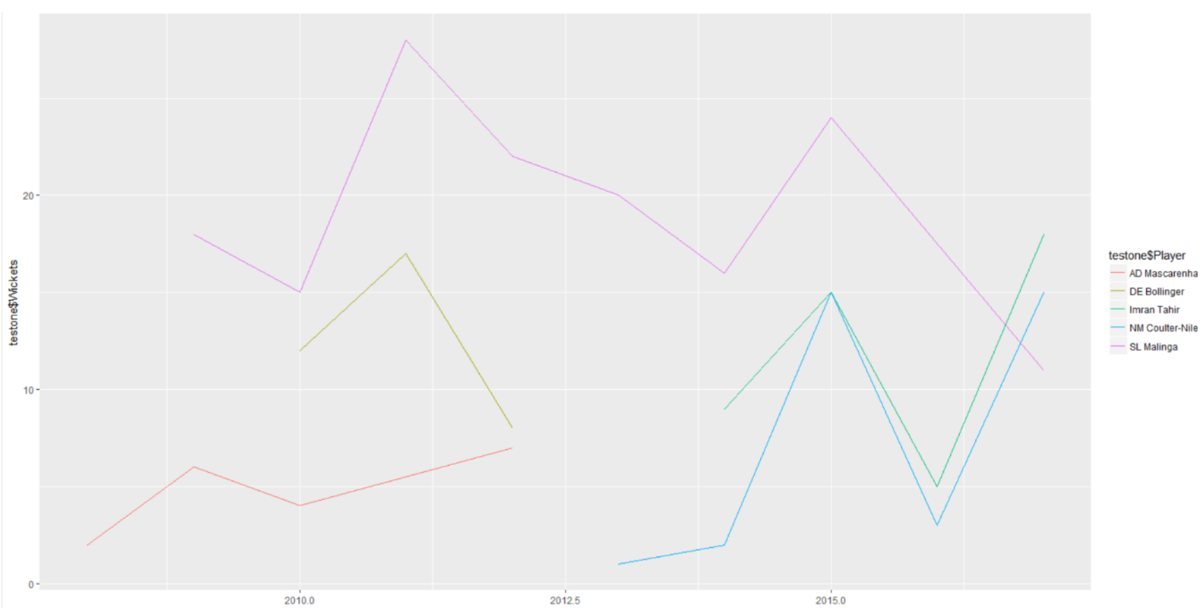
- AD Mascarenhas
- DE Bollinger
- Imran Tahir
- MF Maharoof
- MR Marsh
- NM Coulter-Nile
- Rashid Khan
- SL Malinga

Bar Graphs

Number of Wickets Taken by each Bowler in all the seasons is presented below:



Timeline graph of wickets taken by Bowlers across different seasons:



DATA DICTIONARY

Deliveries.csv

Column	Description	Datatype	Source
match_id	The unique identifier for each match played in the IPL	Numeric	https://www.kaggle.com/manasgarg/ipl
inning	Each team plays for single innings in a match. Each inning has 20 Overs	Numeric	https://www.kaggle.com/manasgarg/ipl
batting_team	Team that is batting in that particular inning.	Numeric	https://www.kaggle.com/manasgarg/ipl
bowling_team	Team that is bowling in that particular inning.	Numeric	https://www.kaggle.com/manasgarg/ipl
over	The Over number currently played.	Numeric	https://www.kaggle.com/manasgarg/ipl
ball	The ball number currently played in the current Over.	Numeric	https://www.kaggle.com/manasgarg/ipl
batsman	The player facing the ball.	String	https://www.kaggle.com/manasgarg/ipl
non_striker	The non-striker batsmen who is not currently facing the ball	String	https://www.kaggle.com/manasgarg/ipl
bowler	The person who is bowling for the current over.	String	https://www.kaggle.com/manasgarg/ipl
is_super_over	Extra over played if the match ends in tie. It's a tie breaker over.	Numeric	https://www.kaggle.com/manasgarg/ipl
wide_runs	The extra runs given due to wide ball or runs taken during a wide ball	Numeric	https://www.kaggle.com/manasgarg/ipl
bye_runs	Run's scored by the batsman when the ball has not been hit by the batsman and the ball has not hit the batsman's body.	Numeric	https://www.kaggle.com/manasgarg/ipl
legbye_runs	Run scored by the batsman when he did not hit the ball with his bat, but the ball hit his body or protective gear	Numeric	https://www.kaggle.com/manasgarg/ipl
noball_runs	It is a penalty against the fielding team, usually as a result of an illegal delivery by the bowler.	Numeric	https://www.kaggle.com/manasgarg/ipl
penalty_runs	Run scored by a means other than a batsman hitting the ball. Other than runs scored off the	Numeric	https://www.kaggle.com/manasgarg/ipl

	bat from a no-ball, a batsman is not given credit for extras and the extras are tallied separately on the scorecard and count only towards the team's score.		
batsman_runs	NO of runs scored by the batsman for a single ball in an over.	Numeric	https://www.kaggle.com/manasgarg/ipl
extra_runs	The sum of the extra runs scored in that ball	Numeric	https://www.kaggle.com/manasgarg/ipl
total_runs	Total runs scored in a particular Ball in an over	Numeric	https://www.kaggle.com/manasgarg/ipl
player_dismissed	If the player was bowled out in that particular over.	String	https://www.kaggle.com/manasgarg/ipl
dismissal_kind	How the player was dismissed in that particular over	String	https://www.kaggle.com/manasgarg/ipl
fielder	If the dismissal kind is "Caught", who was the fielder who Caught the ball.	String	https://www.kaggle.com/manasgarg/ipl

Batting Statistics

Column	Description	Datatype	Source
Player_Name	Name of the Player	String	Calculated from source file
Total_Innings	Total innings played by the player through IPL	Numeric	Calculated from source file
NO	The number of innings the batsmen were not out	Numeric	Calculated from source file
Runs	Total runs scored by the batsman	Numeric	Calculated from source file
Balls_faced	Total number of legal balls faced by the bowler.	Numeric	Calculated from source file
Hundreds	Number of centuries scored by the batsman through the IPL	Numeric	Calculated from source file
Fifties	Number of half centuries scored by the batsman through the IPL.	Numeric	Calculated from source file
Fours	Number of fours scored by the batsman through the IPL.	Numeric	Calculated from source file
Sixes	Number of sixes scored by the batsman through the IPL.	Numeric	Calculated from source file
Average	Average runs scored by the batsman per innings	Numeric	Calculated from source file
Strike_Rate	It's the average number of runs scored per 100 balls faced.	Numeric	Calculated from source file

sBowling Statistics

Column	Description	Datatype	Source
Bowler	Name of the Player	String	Calculated from source file
Runs_Given	Runs scored by the opponent batsman for balls bowled by the Bowler	Numeric	Calculated from source file
Balls_Delivered	Total number of legal balls delivered	Numeric	Calculated from source file
Wickets	Total number of wickets taken through the entire IPL	Numeric	Calculated from source file
Average	The average wicket taken per inning.	Numeric	Calculated from source file
Strike_Rate	The average number of balls bowled per wicket taken.	Numeric	Calculated from source file
Economy	Economy rate is the average number of runs conceded for each over bowled	Numeric	Calculated from source file

MODELING TECHNIQUES

Multiple Regression

Multiple regression is a technique through which we attempt to model the relationship between the target variable and the response variable by fitting a linear equation to the observed data. By performing multiple regression, we get the p-values corresponding to each predictor variable which determines the significance of that variable. Also, the coefficient of each predictor determines its contribution towards prediction of the dependent variable.

There are 3 major uses for multiple regression analysis.

- First, it can be used to identify the strength of the effect that the independent variables have on a dependent variable.
- Second, it can be used to forecast effects or impacts of changes. That is, multiple regression analysis helps us to understand how the dependent variable will change when we change the independent variables.
- Third, multiple regression analysis predicts trends and future values. The multiple regression analysis can be used to get point estimates.

Neural Networks

Neural networks are a series of algorithms that efforts to identify the relationships within a set of data by a process that imitates the way the human brain works. The Neural network has the ability to learn the dataset patterns and provide efficient classifications. The ability of neural networks to perform regression makes us to choose this model.

ASSUMPTIONS OF THE MODELING TECHNIQUES

Multiple Regression

- Linear relationship – Assumes that there is a linear relationship between the predictor and target variables.
- Normality of the residuals – Assumes that the residuals are normally distributed.
- No or little Multi-collinearity – Assumes that the predictor variables are not correlated to each other.
- No auto-correlation – Assumes that the data variable is not influenced by its own historical values.

- Homoscedasticity – Assumes that the variance around the regression line is same for all the values of the Independent variable.

Neural Networks

In theory, for Neural network we do not assume any latent pattern for the data, errors or targets, the patterns are captured during the course of building the neural network and are done without any assumptions. It only depends on the data and the configuration of the network.

MODEL GOALS

Multiple Regression

The objective of this model is to predict the top 10 Batsmen and Top 10 Bowlers in IPL 2017 by analyzing the data from 2008 to 2016. The target variable for determining the top 10 batsmen would be 'Runs', and the target variable to determine the top 10 Bowlers would be 'Wickets'. The predictor variables to determine the runs and the wickets are derived after performing the correlation analysis. In our dataset both the target and predictor variables are continuous and there are no categorical variables.

Multiple regression tells us how much predictive information is associated uniquely with each predictor, when you control for or partial out any overlap or correlation with all the other predictors.

Neural Network

The objective of the neural network is the same. To predict the top 10 Batsmen and Top 10 Bowlers in IPL 2017 by analyzing the data from 2008 to 2016. The target variable for determining the top 10 batsmen would be 'Runs', and the target variable to determine the top 10 Bowlers would be 'Wickets'. However, we need not determine the predictor variables.

Neural Network is a very efficient technique to compute data models. It takes a set of input vectors and produces a set of output vectors after performing a set of operations. Neural network is composed of many highly interconnected processing elements or neurons working in parallel to solve a specific problem.

DATA SPLITTING AND SUBSAMPLING

One of the first decisions to make when modeling is to decide which samples will be used to evaluate performance. Ideally, the model should be evaluated on samples that were not used to build or fine-tune the model, so that they provide an unbiased sense of model effectiveness. When a large amount of data is at hand, a set of samples can be set aside to evaluate the final model. The "training" data set is the general term for the samples used to create the model, while the "test" or "validation" data set is used to qualify performance.

To elaborate on the above portion, we could say that the training portion of the data is used to build a predictive model as the model sees this set of data while determining the best data transformation and to determine which predictors to include in the model and which one to eliminate. The training set is used only after the model is being build and it is used to compare predictive capabilities across different models and confident estimate evaluate the models performance.

Usually in a project, the data split is done as 70% and 30% chosen as the training and test data respectively. Since the objective of our project is to predict the top 10 batsman and bowlers, we are following an unusual way of data splitting which does not have any ratio as mentioned above. Instead, we are deriving statistics for batsmen and bowlers from the matches played from the season 2008 to 2016 as the training set and the matches played in the season 2017 as the testing set which helps in measuring the accuracy.

Reason for splitting testing and training data:

We could have gone with choosing statistics from the matches played between 2008 to 2013 as the training set and 2014 to 2017 as the testing set to ensure there is 50:50 split in the testing and training data. The reason why we chose to do like this is because, the more the number of records and statistics the training set has, the more will be the accuracy of the model. This shall be done as in iterative process. For example, if we are about to predict the top players for the season 2018, we could consolidate the data from the season 2008 to 2017 to build the model and apply it on 2018 to predict the players.

Since the training data has statistics from season 2008 to 2016, we are aggregating data whose match id is from 60 till the last match.

Training Set

```
######################################################### Reading Data#################################################################
# 1 Open File
datafile1 = read.table("C:\\Users\\Hi\\OneDrive - Oklahoma State University\\R and Python Project\\ds1\\deliveries.csv", header=T, sep=",")
datafile = datafile1[datafile1$is_super_over==0,]
datafile = datafile[datafile$match_id>59,]
```

Aggregating the other statistics for training set:

```
###############################################################
#================BAtting Statistics====================#######
###############################################################

#Try to find batsman overall stats, verify in cricbuzz
Runs = aggregate(datafile$batsman_runs, by=list(Category=datafile$batsman), FUN=sum)
names(Runs) <- c("Player_Name","Runs")

#Try to find number of 4s and 6s
da4 = datafile[datafile$batsman_runs==4,]
#aggregate(da4$batsman_runs/4, by=list(Category=da4$batsman), FUN=sum)
Fours = aggregate(da4$batsman_runs/4, by=list(Category=da4$batsman), FUN=sum)
names(Fours) <- c("Player_Name","Fours")

da6 = datafile[datafile$batsman_runs==6,]
Sixes = aggregate(da6$batsman_runs/6, by=list(Category=da6$batsman), FUN=sum)
names(Sixes) <- c("Player_Name","Sixes")

#Try to find scores greater than 50
da50=aggregate(datafile$batsman_runs, by=list(Category=datafile$match_id, Category=datafile$batsman), FUN=sum)
names(da50) <- c("Match_ID","Player_Name","Runs")
da50=da50[(da50$Runs>=50) & (da50$Runs<=99),]
Fifties = aggregate(da50$Runs, by=list(Category=da50$Player_Name), FUN=length)
names(Fifties) <- c("Player_Name","Fifties")


#Try to find scores greater than 100
da100=aggregate(datafile$batsman_runs, by=list(Category=datafile$match_id, Category=datafile$batsman), FUN=sum)
names(da100) <- c("Match_ID","Player_Name","Runs")
da100=da100[da100$Runs>=100,]
Hundreds = aggregate(da100$Runs, by=list(Category=da100$Player_Name), FUN=length)
names(Hundreds) <- c("Player_Name","Hundreds")
```

Writing the training set to the file called BattingStatistics16.csv

```
#ROund off Average ,and Strike Rate
final7$Strike_Rate <- round(final7$Strike_Rate,2)
final7$Average <- round(final7$Average,2)
final7 = final7[final7$Total_Innings!=0,]

#final7[which(!is.finite(final7))] <- 0
final7$Average <- ifelse(is.infinite((final7$Average)),final7$Runs,final7$Average)

#final4.to_csv(r'C:\\Users\\Hi\\Documents\\Project\\BattingStatistics.csv', sep=',', index=None)
write.table(final7, file = "BattingStatistics16.csv",row.names=FALSE, na="",col.names=TRUE, sep=",")
```

Testing set

```
########################################################## Reading Data####################################################################
# 1 Open File
datafile1 = read.table("C:\\Users\\Hi\\OneDrive - Oklahoma State University\\R and Python Project\\ds1\\deliveries.csv", header=T, sep=",")
datafile = datafile1[datafile1$is_super_over==0,]
datafile = datafile[datafile$match_id<60,]
```

Aggregating other statistics for Testing set:

```
##################################################################
#=================BAtting Statistics=====================#######
##################################################################

#Try to find batsman overall stats, verify in cricbuzz
Runs = aggregate(datafile$batsman_runs, by=list(Category=datafile$batsman), FUN=sum)
names(Runs) <- c("Player_Name","Runs")

#Try to find number of 4s and 6s
da4 = datafile[datafile$batsman_runs==4,]
#aggregate(da4$batsman_runs/4, by=list(Category=da4$batsman), FUN=sum)
Fours = aggregate(da4$batsman_runs/4, by=list(Category=da4$batsman), FUN=sum)
names(Fours) <- c("Player_Name","Fours")

da6 = datafile[datafile$batsman_runs==6,]
Sixes = aggregate(da6$batsman_runs/6, by=list(Category=da6$batsman), FUN=sum)
names(Sixes) <- c("Player_Name","Sixes")

#Try to find scores greater than 50
da50=aggregate(datafile$batsman_runs, by=list(Category=datafile$match_id, Category=datafile$batsman), FUN=sum)
names(da50) <- c("Match_ID","Player_Name","Runs")
da50=da50[(da50$Runs>=50) & (da50$Runs<=99),]
Fifties = aggregate(da50$Runs, by=list(Category=da50$Player_Name), FUN=length)
names(Fifties) <- c("Player_Name","Fifties")


#Try to find scores greater than 100
da100=aggregate(datafile$batsman_runs, by=list(Category=datafile$match_id, Category=datafile$batsman), FUN=sum)
names(da100) <- c("Match_ID","Player_Name","Runs")
da100=da100[da100$Runs>=100,]
Hundreds = aggregate(da100$Runs, by=list(Category=da100$Player_Name), FUN=length)
names(Hundreds) <- c("Player_Name","Hundreds")
```

Writing the testing set to a file called BattingStatistics17.csv

```
#ROund off Average ,and Strike Rate
final7$Strike_Rate <- round(final7$Strike_Rate,2)
final7$Average <- round(final7$Average,2)
final7 = final7[final7$Total_Innings!=0,]

#final7[which(!is.finite(final7))] <- 0
final7$Average <- ifelse(is.infinite((final7$Average)),final7$Runs,final7$Average)

#final4.to_csv(r'C:\\Users\\Hi\\Documents\\Project\\BattingStatistics.csv', sep=',', index=None)
write.table(final7, file = "BattingStatistics17.csv",row.names=FALSE, na="",col.names=TRUE, sep=",")
```

Similarly training and testing data was created for bowling statistics.

Comparison of Training and Testing Set

Batting

Training

```
> #Descriptive Statistics for Batting 2016
> describe(finaldata[,-c(1)])
              vars   n   mean     sd median trimmed    mad min     max   range  skew kurtosis    se
Total_Innings    1 439  20.02  28.17   8.00   13.42   8.90   1  143.00  142.00  2.19     4.47  1.34
NO               2 439   4.72   6.45   2.00    3.39   2.97   0   45.00   45.00  2.25     6.19  0.31
Runs             3 439 377.81 734.24  73.00  185.95 102.30   0 4110.00 4110.00  2.83     8.13 35.04
Balls_Faced      4 439 301.54 561.36  68.00  155.29  90.44   0 3151.00 3151.00  2.73     7.42 26.79
Hundreds         5 439   0.10   0.45   0.00    0.00   0.00   0    5.00    5.00  6.49    51.93  0.02
Fifties          6 439   1.85   4.80   0.00    0.55   0.00   0   32.00   32.00  3.64    14.65  0.23
Fours            7 439  35.10  71.14   6.00   16.29   8.90   0  422.00  422.00  2.92     8.53  3.40
Sixes            8 439  13.23  29.09   2.00    6.04   2.97   0  251.00  251.00  3.79    17.70  1.39
Average          9 439  15.48  10.78  13.83   14.70  11.61   0   55.67   55.67  0.63    -0.08  0.51
Strike_Rate     10 439 102.41  39.38 109.90  105.30  30.44   0  218.42  218.42 -0.67     0.67  1.88
```

Testing

```
> #Descriptive Statistics for Batting 2017
> describe(finaldata[,-c(1)])
              vars   n   mean     sd median trimmed   mad min    max  range  skew kurtosis    se
Total_Innings    1 144   6.35   4.75   5.00    5.95  5.93   1  16.00  15.00  0.54    -1.07  0.40
NO               2 144   1.42   1.62   1.00    1.17  1.48   0   9.00   9.00  1.45     2.51  0.13
Runs             3 144 124.35 146.84  50.00  102.20 71.16   0 641.00 641.00  1.09     0.16 12.24
Balls_Faced      4 144  93.25 105.63  40.50   76.96 55.60   0 452.00 452.00  1.16     0.45  8.80
Hundreds         5 144   0.03   0.22   0.00    0.00  0.00   0   2.00   2.00  6.85    50.40  0.02
Fifties          6 144   0.66   1.09   0.00    0.43  0.00   0   5.00   5.00  1.66     2.10  0.09
Fours            7 144  11.19  14.52   4.00    8.66  5.93   0  63.00  63.00  1.38     1.23  1.21
Sixes            8 144   4.90   6.52   1.00    3.66  1.48   0  26.00  26.00  1.41     1.10  0.54
Average          9 144  18.30  14.10  17.37   17.01 16.61   0  60.00  60.00  0.66    -0.17  1.18
Strike_Rate     10 144 111.38  46.25 120.47  113.75 36.31   0 233.33 233.33 -0.38     0.53  3.85
```

Bowling

Training

```
> #Descriptive Statistics for Bowling 2016
> describe(finaldata[,-c(1)])
                vars   n   mean     sd median trimmed    mad min  max range skew kurtosis    se
Runs_Given         1 332 516.98 676.30 236.00  364.71 278.73   0 3119  3119 1.95     3.28 37.12
Balls_Delivered    2 332 396.99 540.92 183.00  273.05 217.94   1 2673  2672 2.03     3.67 29.69
Total_Innings      3 332  20.68  26.02  10.00   14.93  11.86   1  123   122 1.90     3.06  1.43
Wickets            4 332  18.11  26.09   7.00   12.04   8.90   0  143   143 2.18     4.65  1.43
Average            5 332  29.84  20.44  28.02   28.03  11.23   0  136   136 1.28     3.98  1.12
Strike_Rate        6 332  21.29  12.99  21.23   20.76   8.25   0   85    85 0.63     2.07  0.71
economy            7 332   8.71   2.38   8.16    8.37   1.20   0   23    23 2.38     9.70  0.13
```

Testing

```
> #Descriptive Statistics for Bowling 2017
> describe(finaldata[,-c(1)])
                vars   n   mean     sd median trimmed    mad  min    max  range skew kurtosis    se
Runs_Given         1 107 172.21 127.40 144.00  161.80 140.85 9.00 507.00 498.00 0.61    -0.77 12.32
Balls_Delivered    2 107 125.14  97.99 107.00  117.09 112.68 6.00 356.00 350.00 0.63    -0.82  9.47
Total_Innings      3 107   6.59   4.46   6.00    6.36   5.93 1.00  16.00  15.00 0.38    -1.20  0.43
Wickets            4 107   6.07   6.07   4.00    5.28   4.45 0.00  26.00  26.00 1.07     0.40  0.59
Average            5 107  31.98  31.63  26.68   27.60  16.58 0.00 248.00 248.00 3.46    19.08  3.06
Strike_Rate        6 107  21.82  18.70  20.20   19.74   9.49 0.00 144.00 144.00 2.98    15.82  1.81
economy            7 107   9.06   2.18   8.77    8.86   1.68 3.75  16.36  12.61 1.02     1.89  0.21
```

Since the training set contains the statistics from 2008 to 2016 and the testing set contains the statistics for only 2017, we could observe a significant difference in the mean, standard deviation, median, skewness and kurtosis for different variables for batting as well as bowling.

MULTIPLE REGRESSION MODEL BUILDING

Multiple Regression for Runs scored by the Batsmen

Fortunately, our dataset does not contain huge number of columns and there was no need to perform any variable reduction process before building the model. However, it is mandatory to see the correlation between the variables to choose the predictor variables for the target variable 'Runs'.

Correlation analysis for Runs scored by the batsmen

Below is the result we obtained after doing the correlation analysis.

```
> #Correlation
> cor(battingdata)
```

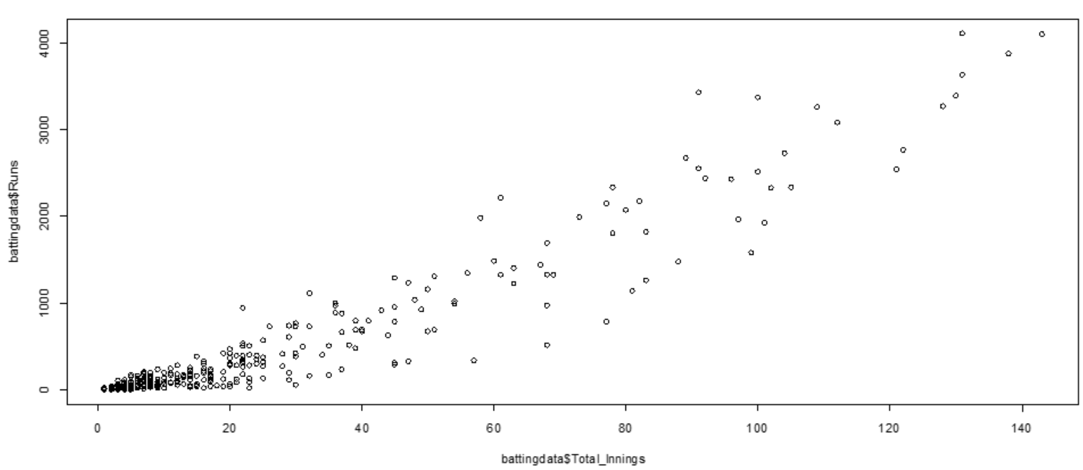
	Runs	Total_Innings	NO	Balls_Faced	Fours	Sixes	Average	Strike_Rate	Hundreds	Fifties
Runs	1.0000000	0.9495295	0.6145275	0.9950936	0.9816019	0.9168952	0.6033945	0.3272421	0.6334846	0.9494808
Total_Innings	0.9495295	1.0000000	0.7750200	0.9534945	0.9208684	0.8629516	0.5570554	0.3443660	0.5060868	0.8409590
NO	0.6145275	0.7750200	1.0000000	0.6114514	0.5304143	0.6035634	0.4043989	0.2695366	0.2625059	0.4681210
Balls_Faced	0.9950936	0.9534945	0.6114514	1.0000000	0.9828455	0.8835840	0.5993632	0.3128182	0.5920850	0.9431512
Fours	0.9816019	0.9208684	0.5304143	0.9828455	1.0000000	0.8498795	0.5838318	0.3119130	0.6151874	0.9520402
Sixes	0.9168952	0.8629516	0.6035634	0.8835840	0.8498795	1.0000000	0.5600984	0.3374925	0.7223605	0.8452465
Average	0.6033945	0.5570554	0.4043989	0.5993632	0.5838318	0.5600984	1.0000000	0.6380468	0.3754954	0.5549736
Strike_Rate	0.3272421	0.3443660	0.2695366	0.3128182	0.3119130	0.3374925	0.6380468	1.0000000	0.1875542	0.2667180
Hundreds	0.6334846	0.5060868	0.2625059	0.5920850	0.6151874	0.7223605	0.3754954	0.1875542	1.0000000	0.6176269
Fifties	0.9494808	0.8409590	0.4681210	0.9431512	0.9520402	0.8452465	0.5549736	0.2667180	0.6176269	1.0000000

Since Runs is the target variable, let us analyze the correlation between the other variables: Total_Innings is highly correlated with Runs, Not Outs(NO), Balls_Faced, Fours, Sixes and Fifties. This is pretty obvious because as the total innings played by a batsman increases, there is more chance of getting runs, and the number of balls faced by the batsman increases leading to more sixes and half centuries(Fifties)

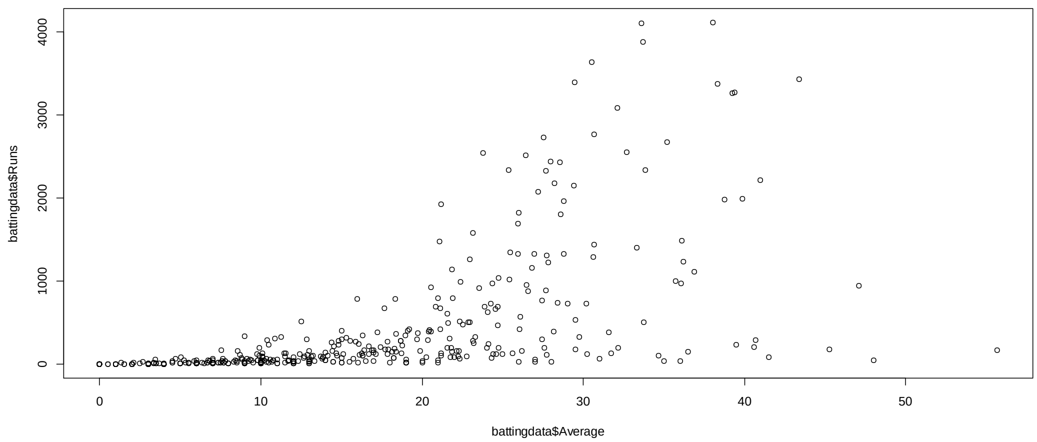
We could observe that Total_innings has less correlation with Average, Strike_Rate and Hundreds. Hence we could consider Total_Innings, Average, Strike_Rate and Hundreds as the predictor variables.

Linearity Check

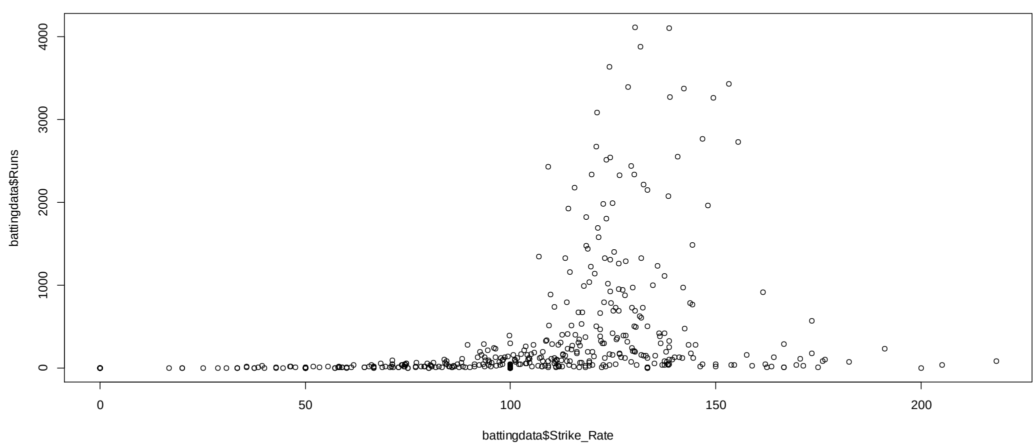
The second checkpoint before building the regression model is to assess the linear relationship between the predictor and the target variable:



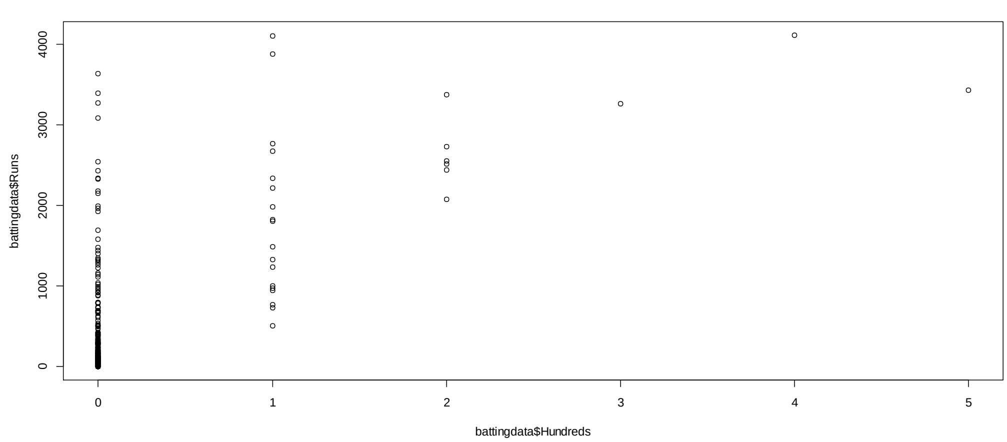
Runs vs Total_Innings



Runs Vs Average



Runs Vs Strike Rate



Hundreds Vs Runs

S NO	Predictor variables Vs Runs	Linearity
1	Total_Innings	Linear
2	Average	Non-Linear
3	Strike_Rate	Non-Linear
4	Hundreds	Non-Linear

We will choose this as the predictor variables and do the stepwise regression to find out which are the variables significant in predicting Runs.

```
> #Build Stepwise Regression based on correlation assessment.
> model.null = lm(Runs ~ 1, data=battingdata)
> model.full = lm(Runs ~ Total_Innings + Average + Strike_Rate + Hundreds, data=battingdata)
> step(model.null, scope = list(upper=model.full), direction="both", data=battingdata1)
Start:  AIC=5794.78
Runs ~ 1

                Df Sum of Sq       RSS    AIC
+ Total_Innings  1 212896837  23233783 4778.8
+ Hundreds       1  94759851 141370768 5571.6
+ Average        1  85971599 150159020 5598.0
+ Strike_Rate    1  25286619 210844001 5747.1
<none>                       236130619 5794.8

Step:  AIC=4778.84
Runs ~ Total_Innings

                Df Sum of Sq       RSS    AIC
+ Hundreds       1   7424978  15808804 4611.8
+ Average        1   1897905  21335877 4743.4
<none>                        23233783 4778.8
+ Strike_Rate    1        18  23233765 4780.8
- Total_Innings  1 212896837 236130619 5794.8

Step:  AIC=4611.8
Runs ~ Total_Innings + Hundreds

                Df Sum of Sq       RSS    AIC
+ Average        1   1061900  14746904 4583.3
<none>                        15808804 4611.8
+ Strike_Rate    1      1639  15807166 4613.8
- Hundreds       1   7424978  23233783 4778.8
- Total_Innings  1 125561964 141370768 5571.6


                Df Sum of Sq       RSS    AIC
+ Strike_Rate    1    599129  14147775 4567.1
<none>                        14746904 4583.3
- Average        1   1061900  15808804 4611.8
- Hundreds       1   6588973  21335877 4743.4
- Total_Innings  1  89896543 104643447 5441.5

Step:  AIC=4567.07
Runs ~ Total_Innings + Hundreds + Average + Strike_Rate

                Df Sum of Sq       RSS    AIC
<none>                        14147775 4567.1
- Strike_Rate    1    599129  14746904 4583.3
- Average        1   1659391  15807166 4613.8
- Hundreds       1   6273515  20421291 4726.2
- Total_Innings  1  90056855 104204630 5441.7

Call:
lm(formula = Runs ~ Total_Innings + Hundreds + Average + Strike_Rate,
    data = battingdata)

Coefficients:
  (Intercept)  Total_Innings       Hundreds        Average    Strike_Rate
      -78.775         21.014        313.090          8.474         -1.223
```

The final model has all the variables that we considered. Let us build the models as follows:

```
> Runsreg = lm(Runs~ Total_Innings + Average + Strike_Rate + Hundreds, data = battingdata)
> summary(Runsreg)

Call:
lm(formula = Runs ~ Total_Innings + Average + Strike_Rate + Hundreds,
    data = battingdata)

Residuals:
    Min      1Q  Median      3Q     Max
-802.77  -68.65   28.37   77.10  853.06

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -78.7753    24.2239  -3.252  0.00124 **
Total_Innings  21.0135     0.3998  52.560  < 2e-16 ***
Average         8.4742     1.1877   7.135 4.10e-12 ***
Strike_Rate    -1.2232     0.2853  -4.287 2.23e-05 ***
Hundreds      313.0902    22.5690  13.873  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 180.6 on 434 degrees of freedom
Multiple R-squared:  0.9401,    Adjusted R-squared:  0.9395
F-statistic:  1702 on 4 and 434 DF,  p-value: < 2.2e-16
```

Interpretation of the Model

Overall p-value of the model is less than 0.05 and hence the overall model is significant in predicting the target variable Runs.

The Multiple R squared value is .9401 which states the fact that the variables explain 94% of the variability in predicting the target variable Runs.

Interpreting the individual predictors:

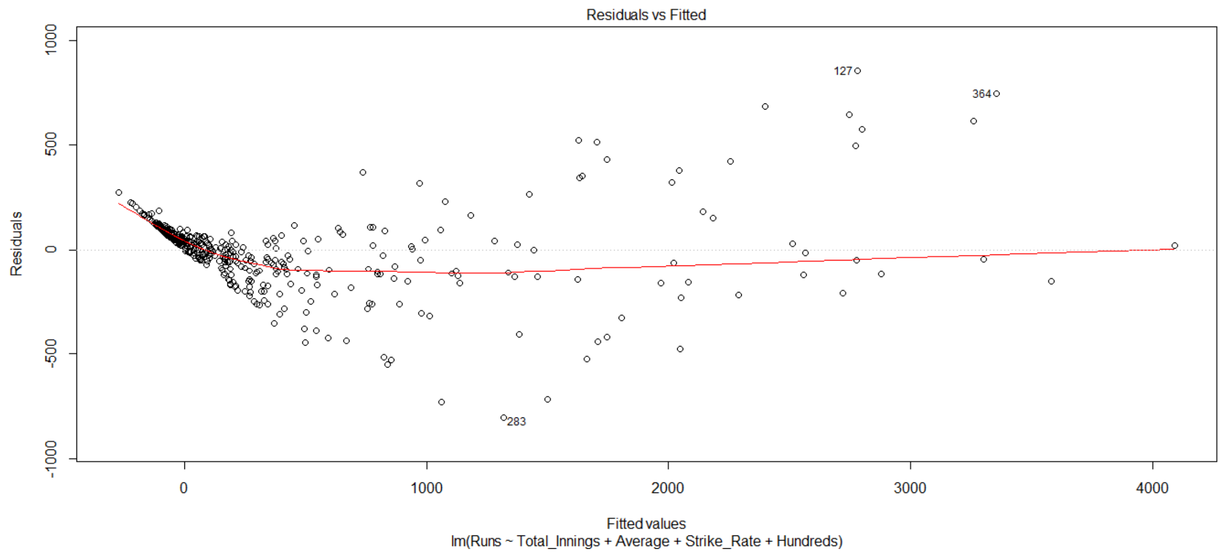
- The significance value of each variable individually is also less than 0.05 and hence they are significant in predicting the target variables Runs.
- Holding the effect of other variables constant, whenever there is a unit increase in the Total_Innings, the Runs scored by a batsman increases by 21.01
- Holding the effect of other variables constant, whenever there is a unit increase in Average, the Runs scored by a batsman increases by 8.47
- Holding the effect of other variables constant, whenever there is a unit increase in Strike_Rate, the Runs decreases by 1.22. This doesn't make sense because the strike rate is nothing but the number of runs by the batsman divided by the number of balls he faced. Since we don't have the balls_faced variable considered, the coefficient suggests that whenever there is an increase in the strike rate, the Runs scored by the batsmen decreases by 1.
- Holding the effect of other variables constant, whenever there is a unit increase in the variable Hundreds which represents the number of centuries scored by a batsman, the Runs scored by the batsman increases by 313. The reason why there is huge positive coefficient for the variable Hundreds is understandable because, T-20 being a shorter format of the cricket game, it is a rare event for a batsman to score a century.

Assessment of the Model

Accuracy of the model is a vital factor which determines how good is the classification done by the model. Higher the accuracy means the model has higher predictive ability.

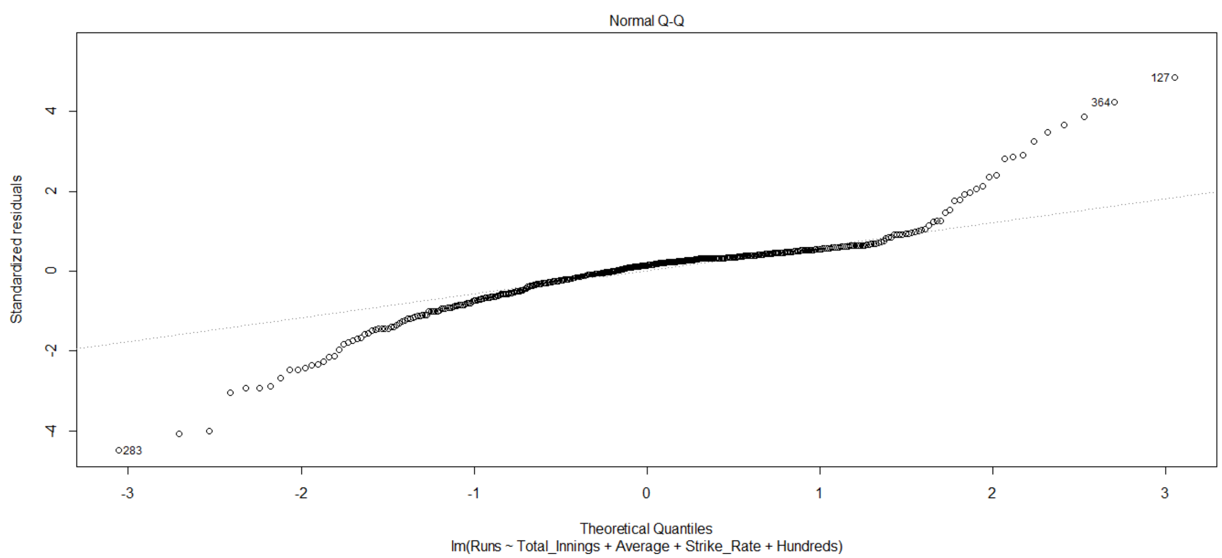
Post building the model, we checked for the homoscedasticity and normality of the residuals for the model built above.

Homoscedasticity:



From the above plot we can say that there seems to be a constant variance among the variables. Though it is not perfect, still overall it looks find and we can say it satisfies Homoscedasticity.

Q-Q Plot to assess Normality of the residuals:



Based on the above Q-Q plot, we can say that the distribution is over-dispersed relative to a Normal distribution similar to the previous data. Here we can see that the data has flatter distribution than a normal distributed data. Thus, we can say it has positive excess Kurtosis.

Muti-Collinearity and Auto-correlation:

```
> #Assessment
> vif(Runsreg)
Total_Innings       Average   Strike_Rate     Hundreds
     1.703638      2.204354      1.695883     1.374740
> durbinWatsonTest(Runsreg)
 lag Autocorrelation D-W Statistic p-value
   1      0.01750089      1.963979   0.688
 Alternative hypothesis: rho != 0
```

Since the VIF values for all the predictor variables is less than 5, we could say that there is no multi-collinearity between the predictor variables considered.

The p-value of the Durbin Watson Test is not significant. Hence it is safe to assume that there is no autocorrelation.

Prediction

Let us use the model built to do prediction on the testing data.

```
##############################
##### Take 2017 data #########
##############################

bat17 = read.table("BattingStatistics17.csv", header=T, sep=",")
bat17final = bat17[,c(4,2,3,5,8,9,10,11,6,7)]

bat17final2 = bat17[,c(2,10,11,6)]
topbatsman2 = predict(Runsreg, bat17final, type="response")
write.table(topbatsman2, file = "RunsPredicted.csv",row.names=FALSE, na="",col.names=TRUE, sep=",")
```

In order to measure the accuracy, after predicting, we have to compare the actual runs and the predicted runs to come up with the Mean Square Error, Mean Absolute Error and R2 Score.

<u>Note:</u> Since we will be building neural networks in Python, the same model was built in Python and MSE, MAE and R2 Score values were calculated after ensuring that Python also builds the same model.

```
In [69]: print('COEFFICIENTS\n',
    ...:       'Hundreds: ', linreg1.coef_[0],
    ...:       '\nStrike_Rate: ', linreg1.coef_[1],
    ...:       '\nAverage: ', linreg1.coef_[2],
    ...:      '\nTotal_Innings: ', linreg1.coef_[3],
    ...:       '\ninterc: ', linreg1.intercept_,
    ...:       '\nR-Square: ', rsquare)
COEFFICIENTS
 Hundreds:  313.090160268
Strike_Rate:  -1.22320242629
Average:  8.47421596465
Total_Innings:  21.0135362516
interc:  -78.7752647182
R-Square:  0.940084961274
```

```
In [95]: linpred1 = linreg1.predict(bat17_final)

In [96]: metrics.mean_absolute_error(bat17_data.Runs, linpred1)
Out[96]: 80.187755937274531

In [97]: metrics.mean_squared_error(bat17_data.Runs, linpred1)
Out[97]: 12234.57104432051

In [98]: metrics.r2_score(bat17_data.Runs, linpred1)
Out[98]: 0.42861053931294435
```

Multiple Regression for Wickets Taken by the Bowler

Fortunately, our dataset does not contain huge number of columns and there was no need to perform any variable reduction process before building the model. However, it is mandatory to see the correlation between the variables to choose the predictor variables for the target variable 'Wickets'.

Correlation analysis for Wickets taken by the bowler

Below is the result we obtained after doing the correlation analysis.

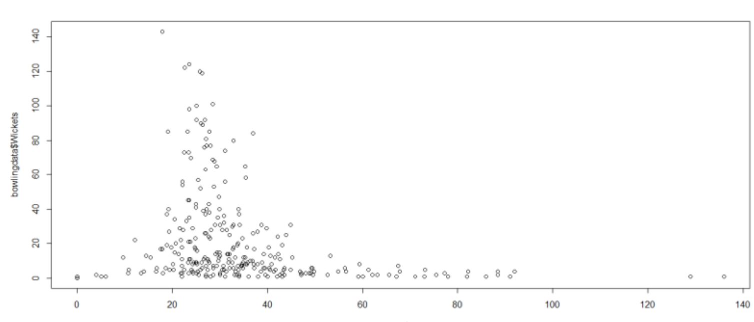
```
> bowlingcor = cor(bowlingdata)
> bowlingcor
                    Wickets  Runs_Given Balls_Delivered Total_Innings     Average Strike_Rate     economy
Wickets          1.00000000  0.97335681      0.97872936    0.95416743 -0.05291118  0.02553618 -0.27427826
Runs_Given       0.97335681  1.00000000      0.99525418    0.98436383  0.02120846  0.10183230 -0.25893391
Balls_Delivered  0.97872936  0.99525418      1.00000000    0.98442804  0.00493516  0.09359056 -0.27747617
Total_Innings    0.95416743  0.98436383      0.98442804    1.00000000  0.03882389  0.12788838 -0.27596525
Average         -0.05291118  0.02120846      0.00493516    0.03882389  1.00000000  0.96438462 -0.06590367
Strike_Rate      0.02553618  0.10183230      0.09359056    0.12788838  0.96438462  1.00000000 -0.20945261
economy         -0.27427826 -0.25893391     -0.27747617   -0.27596525 -0.06590367 -0.20945261  1.00000000
>
```

Here wickets taken is our target variable, we are thus going to find the correlation between the other independent variables:

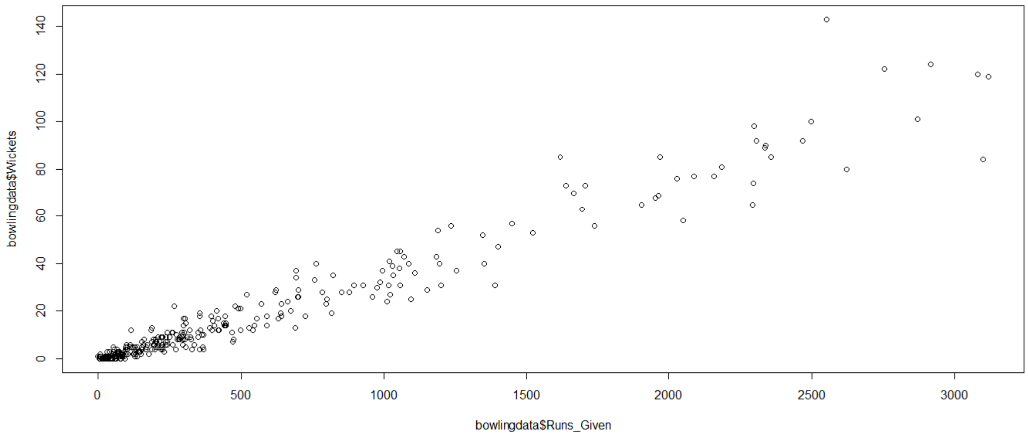
From the above correlation matrix, we can clearly see that the Runs_Given is highly correlated with Balls_Delivered and Total_innings. This makes sense because, the more balls we bowl and more number of innings played by the bowler, there is a more chance of giving runs. Runs_Given has less correlation with Average, Strike Rate and Economy. Hence these 4 variables will be considered as Predictor variables.

Linearity Check:

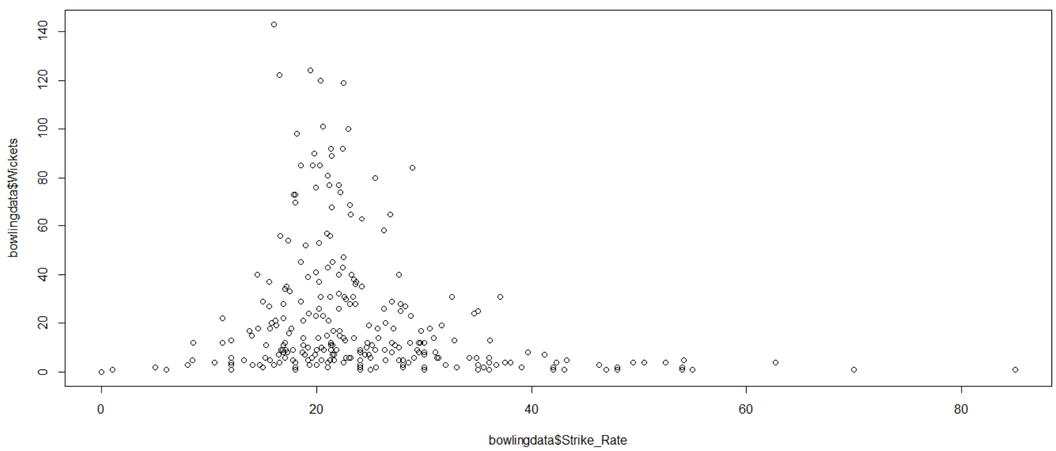
Before running the regression model, we need to assess the linear relationship between the predictor and the target variables.



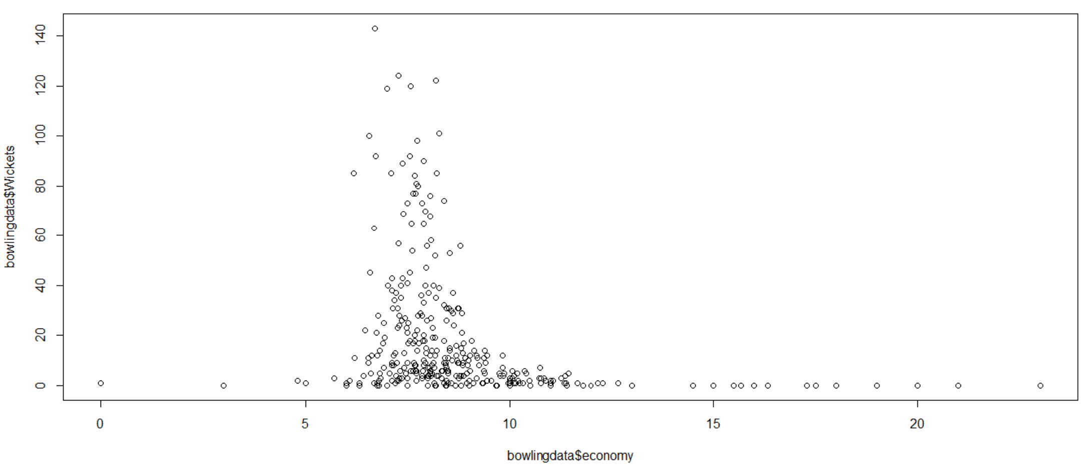
Average Vs Wickets



Runs_Given vs Wickets



Strike_Rate Vs Wickets



Economy Vs Wickets

S NO	Predictor variables Vs Runs	Linearity
1	Average	Non-Linear
2	Runs_Given	Linear
3	Strike_Rate	Non-Linear
4	Economy	Non-Linear

We will do a step wise Regression before building the model.

Step wise Regression:

We will choose these variables as the predictors and do the stepwise regression to find out which are the variables significant in predicting the Wickets taken.

```
> #Stepwise Regression
> model.null = lm(Wickets ~ 1, data=bowlingdata)
> model.full = lm(Wickets ~ Runs_Given + Average + Strike_Rate + economy, data=bowlingdata)
> step(model.null, scope = list(upper=model.full), direction="both", data=bowlingdata)
Start:  AIC=2166.71
Wickets ~ 1

              Df Sum of Sq    RSS    AIC
+ Runs_Given   1    213486  11847 1190.8
+ economy      1     16951 208381 2142.7
<none>                     225333 2166.7
+ Average      1       631 224702 2167.8
+ Strike_Rate  1       147 225186 2168.5

Step:  AIC=1190.8
Wickets ~ Runs_Given

              Df Sum of Sq    RSS    AIC
+ Strike_Rate  1      1233  10614 1156.3
+ Average      1      1220  10628 1156.7
+ economy      1       119  11728 1189.4
<none>                      11847 1190.8
- Runs_Given   1    213486 225333 2166.7

Step:  AIC=1156.32
Wickets ~ Runs_Given + Strike_Rate

              Df Sum of Sq    RSS    AIC
+ economy      1       322  10292 1148.1
<none>                      10614 1156.3
+ Average      1        14  10600 1157.9
- Strike_Rate  1      1233  11847 1190.8
- Runs_Given   1    214572 225186 2168.5


Step:  AIC=1148.09
Wickets ~ Runs_Given + Strike_Rate + economy

              Df Sum of Sq    RSS    AIC
<none>                      10292 1148.1
+ Average      1        33  10259 1149.0
- economy      1       322  10614 1156.3
- Strike_Rate  1      1436  11728 1189.4
- Runs_Given   1    197849 208141 2144.4

Call:
lm(formula = Wickets ~ Runs_Given + Strike_Rate + economy, data = bowlingdata)

Coefficients:
(Intercept)   Runs_Given  Strike_Rate      economy
    6.03896      0.03747     -0.16418     -0.43682
```

Based on the above results from the stepwise regression, Average is eliminated. Runs_Given, Strike_Rate and economy seems to be the predictor variables to do multiple regression.

```
> #Build Regression Model
> wicketstakenl = lm(Wickets ~ Runs_Given + Strike_Rate + economy, data = bowlingdata)
> summary(wicketstakenl)

Call:
lm(formula = Wickets ~ Runs_Given + Strike_Rate + economy, data = bowlingdata)

Residuals:
    Min      1Q  Median      3Q     Max 
-30.120  -2.247   0.294   1.693  46.914 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  6.0389612  1.4812087   4.077 5.73e-05 ***
Runs_Given   0.0374745  0.0004719  79.406  < 2e-16 ***
Strike_Rate -0.1641803  0.0242731  -6.764 6.18e-11 ***
economy     -0.4368234  0.1363152  -3.205  0.00149 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.602 on 328 degrees of freedom
Multiple R-squared:  0.9543,    Adjusted R-squared:  0.9539 
F-statistic:  2284 on 3 and 328 DF,  p-value: < 2.2e-16
```

Interpretation of the model

Overall p-value of the model is less than 0.05 and hence the overall model is significant in predicting the target variable, number of Wickets.

The Multiple R squared value is 0.9543 which implies that the variables considered, explain around 95% of the variability in predicting the target variable Wickets.

Interpreting the individual predictors:

- The significance value of each variable individually is also less than 0.05 and hence they are significant in predicting the target variables Wickets.
- Holding the effect of other variables constant, whenever there is a unit increase in the Runs_given, the Wickets taken by the batsman increases by 0.04 units
- Holding the effect of other variables constant, whenever there is a unit increase in economy, the wickets taken by the bowler decreases by 0.43 units.
- Holding the effect of other variables constant, whenever there is a unit increase in Strike_Rate, the wickets taken decreases by 0.16. Similar to what we observed in the regression model for batting, this doesn’t make sense because the strike rate is nothing but the number of wickets by the bowler divided by the number of balls he delivered. Since we don’t have the balls_ delivered variable being considered in the model, it does not give appropriate results.

The model equation can be represented as follows:

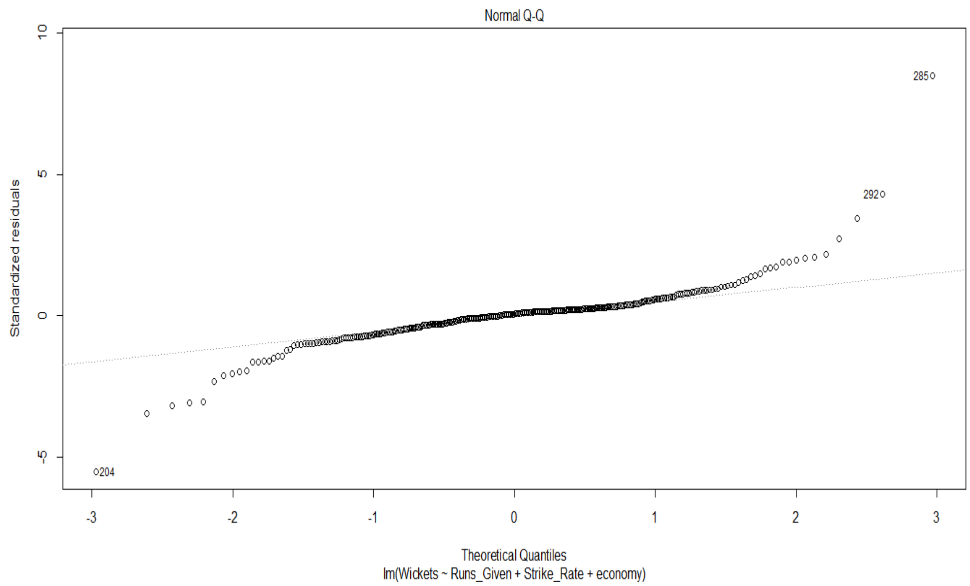
Wickets = 6.038 + (0.0374) Runs_Given - (0.164) Strike_Rate - (0.436) economy

Assessment of the model

Accuracy of the model is a vital factor which determines how good is the classification done by the model. Higher the accuracy means the model has higher predictive ability.

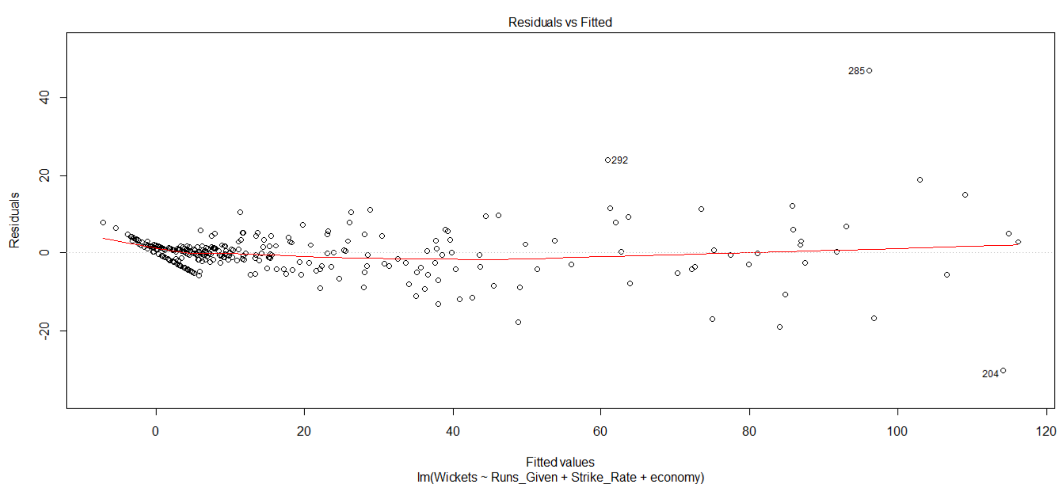
Post building the model, we checked for the homoscedasticity and normality of the residuals for the model built above.

Normality of Residuals:



Based on the above Q-Q plot, we can say that the distribution is over-dispersed relative to a Normal distribution similar to the previous data. Here we can see that the data has flatter distribution than a normal distributed data. Thus, we can say it has positive excess Kurtosis.

Homoscedasticity:



From the above plot we can say that there seems to be a constant variance among the variables. Though it is not perfect, still overall it looks find and we can say it satisfies Homoscedasticity.

Multi-Collinearity and Auto-Correlation:

```
> vif(wicketstaken1)
 Runs_Given Strike_Rate     economy
   1.074594    1.048546    1.112246
> durbinWatsonTest(wicketstaken1)
 lag Autocorrelation D-W Statistic p-value
   1      0.08790908      1.823004   0.082
 Alternative hypothesis: rho != 0
```

Since the VIF values for all the predictor variables are less than 5, we could say that there is less multi-collinearity between the variables.

Since the p-value of Durbin Watson test is not significant, it is safe to assume that there is no autocorrelation.

Prediction

Let us use the model built to do prediction on the testing data.

```
##############################
##### Take 2017 data #########
##############################

bowl17 = read.table("BowlingStatistics17.csv", header=T, sep=",")
bowl17final = bowl17[,c(2,7,8)]

topbowler = predict(wicketstaken1, bowl17final, type="response")
head(order(-topbowler),10)
write.table(topbowler, file = "WicketsPredicted.csv",row.names=FALSE, na="",col.names=TRUE, sep=",")
```

To measure the accuracy, after predicting, we have to compare the actual wickets and the predicted wickets to come up with the Mean Square Error, Mean Absolute Error and R2 Score.

<u>**Note**</u>: Since we will be building neural networks in Python, the same model was built in Python and MSE, MAE and R2 Score values were calculated after ensuring that Python also yields the same result.

```
In [52]: print('COEFFICIENTS\n',
    ...:       'Runs_Given: ', linreg1.coef_[0],
    ...:       '\nStrike_Rate: ', linreg1.coef_[1],
    ...:       '\neconomy: ', linreg1.coef_[2],
    ...:       '\ninterc: ', linreg1.intercept_,
    ...:       '\nR-Square: ', rsquare)
COEFFICIENTS
 Runs_Given:  0.0374744914033
Strike_Rate:  -0.164180333596
economy:  -0.43682339557
interc:  6.03896118922
R-Square:  0.954324660351

In [20]: linpred1 = linreg1.predict(bowl17_final)

In [21]: metrics.mean_absolute_error(bowl17_data.Wickets, linpred1)
Out[21]: 2.07420754852029

In [22]: metrics.mean_squared_error(bowl17_data.Wickets, linpred1)
Out[22]: 9.1271791622265468

In [23]: metrics.r2_score(bowl17_data.Wickets, linpred1)
Out[23]: 0.75018987480855703
```

NEURAL NETWORKS MODEL BUILDING AND ASSESSMENT

After performing multiple regression over the data, we have decided to create other models using neural networks. For this purpose, we have divided the dataset into training data and testing data. The data before 2017 was used for training purpose and 2017 data was used for testing purpose.

Neural Networks Model for Runs Scored by the Batsmen

Training data:

```
In [41]: bat16_data = pd.read_table('BattingStatistics16.csv', sep=',')

In [42]: bat16_data.dtypes
Out[42]:
Player_Name       object
Total_Innings      int64
NO                 int64
Runs               int64
Balls_Faced        int64
Hundreds           int64
Fifties            int64
Fours              int64
Sixes              int64
Average          float64
Strike_Rate      float64
dtype: object

In [43]: bat16_data.columns
Out[43]:
Index(['Player_Name', 'Total_Innings', 'NO', 'Runs', 'Balls_Faced', 'Hundreds',
       'Fifties', 'Fours', 'Sixes', 'Average', 'Strike_Rate'],
      dtype='object')

In [44]: bat16_data = bat16_data[['Runs', 'Total_Innings', 'NO', 'Balls_Faced', 'Hundreds', 'Fifties',
'Fours', 'Sixes', 'Average', 'Strike_Rate']]
```

Testing data:

```
In [45]: bat17_data = pd.read_table('BattingStatistics17.csv', sep=',')

In [46]: bat17_data.dtypes
Out[46]:
Player_Name       object
Total_Innings      int64
NO                 int64
Runs               int64
Balls_Faced        int64
Hundreds           int64
Fifties            int64
Fours              int64
Sixes              int64
Average          float64
Strike_Rate      float64
dtype: object

In [47]: bat17_data.columns
Out[47]:
Index(['Player_Name', 'Total_Innings', 'NO', 'Runs', 'Balls_Faced', 'Hundreds',
       'Fifties', 'Fours', 'Sixes', 'Average', 'Strike_Rate'],
      dtype='object')

In [48]: bat17_data = bat17_data[['Runs', 'Total_Innings', 'NO', 'Balls_Faced', 'Hundreds', 'Fifties',
'Fours', 'Sixes', 'Average', 'Strike_Rate']]
    ...: bat17_final = bat17_data[['Hundreds', 'Strike_Rate', 'Average', 'Total_Innings']]
```

Neural Networks regression model was built over training data. After testing the model with various hidden layer sizes, (20, 20) gave a good model.

```
In [52]: nnreg1 = MLPRegressor(activation='logistic', solver='sgd',
    ...:                       hidden_layer_sizes=(20,20),
    ...:                       early_stopping=True)

In [53]: nnreg1.fit(bat16_data, bat16_data.Runs)
Out[53]:
MLPRegressor(activation='logistic', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=True, epsilon=1e-08,
       hidden_layer_sizes=(20, 20), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='sgd', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

Assessment of Model

Once the model was built, prediction was done on testing data, i.e, the data of 2017 and MAE, MSE and R2 score value were obtained.

```
In [54]: nnpred1 = nnreg1.predict(bat17_data)

In [55]: nnreg1.n_layers_
Out[55]: 4

In [57]: metrics.mean_absolute_error(bat17_data.Runs, nnpred1)
Out[57]: 251.242772757236

In [58]: metrics.mean_squared_error(bat17_data.Runs, nnpred1)
Out[58]: 77932.659956226911

In [59]: metrics.r2_score(bat17_data.Runs, nnpred1)
Out[59]: -2.6396781203839357

In [60]: nnreg1.score(bat17_data, bat17_data.Runs)
Out[60]: -2.6396781203839357
```

Neural Networks Model for Wickets Taken by the Bowler

Training Data:

```
In [3]: bowl16_data = pd.read_table('BowlingStatistics16.csv', sep=',')

In [4]: bowl16_data.dtypes
Out[4]:
Bowler              object
Runs_Given           int64
Balls_Delivered      int64
Total_Innings        int64
Wickets              int64
Average            float64
Strike_Rate        float64
economy            float64
dtype: object

In [5]: bowl16_data.columns
Out[5]:
Index(['Bowler', 'Runs_Given', 'Balls_Delivered', 'Total_Innings', 'Wickets',
       'Average', 'Strike_Rate', 'economy'],
      dtype='object')

In [6]: bowl16_data = bowl16_data[['Runs_Given', 'Balls_Delivered', 'Total_Innings', 'Wickets','Average',
'Strike_Rate', 'economy']]
```

Testing Data:

```
In [7]: bowl17_data = pd.read_table('BowlingStatistics17.csv', sep=',')

In [8]: bowl17_data.dtypes
Out[8]:
Bowler              object
Runs_Given           int64
Balls_Delivered      int64
Total_Innings        int64
Wickets              int64
Average            float64
Strike_Rate        float64
economy            float64
dtype: object

In [9]: bowl17_data.columns
Out[9]:
Index(['Bowler', 'Runs_Given', 'Balls_Delivered', 'Total_Innings', 'Wickets',
       'Average', 'Strike_Rate', 'economy'],
      dtype='object')

In [10]: bowl17_data = bowl17_data[['Runs_Given', 'Balls_Delivered', 'Total_Innings', 'Wickets','Average',
'Strike_Rate', 'economy']]

In [11]: bowl17_final = bowl17_data[['Balls_Delivered', 'Strike_Rate', 'economy']]
```

Neural networks model was built on training data and predictions were made on testing data.

```
In [24]: nnreg1 = MLPRegressor(activation='logistic', solver='sgd',
    ...:                        hidden_layer_sizes=(20,20),
    ...:                        early_stopping=True)
    ...: nnreg1.fit(bowl17_data, bowl17_data.Wickets)
Out[24]:
MLPRegressor(activation='logistic', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=True, epsilon=1e-08,
       hidden_layer_sizes=(20, 20), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='sgd', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

Assessment of Model

```
In [25]: nnpred1 = nnreg1.predict(bowl17_data)

In [26]: nnreg1.n_layers_
Out[26]: 4

In [27]: nnreg1.coefs_
Out[27]:
[array([[-0.18622334, -0.1018837 ,  0.21851199, -0.01880297,  0.06326426,
          0.00301736,  0.00131334,  0.09880918, -0.040052  ,  0.02998673,
         -0.0775485 , -0.08414117,  0.07048037,  0.25247375, -0.16602745,
          0.07449638, -0.04908536,  0.01173884,  0.02078189,  0.1945439 ],
        [ 0.02618535, -0.05977819,  0.23599858,  0.03781307,  0.22640452,

In [28]: metrics.mean_absolute_error(bowl17_data.Wickets, nnpred1)
Out[28]: 3.1723575285644405

In [29]: metrics.mean_squared_error(bowl17_data.Wickets, nnpred1)
Out[29]: 18.248918668716524

In [30]: metrics.r2_score(bowl17_data.Wickets, nnpred1)
Out[30]: 0.50052863253662272

In [31]: nnreg1.score(bowl17_data, bowl17_data.Wickets)
Out[31]: 0.50052863253662272
```

STRENGTHS AND WEAKNESS OF THE MODEL

MULTIPLE REGRESSION

Strength

- The biggest advantage of multiple regression is that it has the ability to determine the relative influence of one or more predictor variables.
- Multiple regression allows us to add as many predictor variables as possible which cannot be achieved using linear regression.
- It also yields an understanding of the association of all of the factors as a whole with the outcome, and the associations between the various predictor variables themselves.
- Multiple regression analysis identifies the outliers and anomalies.

Weakness

- Disadvantage of multiple regression is that we cannot use incomplete data. If incomplete data is used it is going give inaccurate model

NEURAL NETWORKS

Strengths

- Neural Network doesn't need any model that would act as a prerequisite. This means that you can directly build a neural network model without any structure for the model. This is one of the greatest advantages that we have with the neural networks.
- Creation of errors in parameter estimation is very less as it is a non-parametric method.
- The neural network can understand and train itself to assess and predict the pattern of the runs hits by the batsman and wickets taken by the bowler using the predictor variables.
- This model helped us to use independent variables that are not linearly related to the target variable as predictor variables.

Weakness

- The accuracy of neural networks is less when compared to the accuracy of other models.
- It is very hard to explain the output of neural networks and also extracting the knowledge from it is not an easy task.
- Neural Network is like black box and we do not know what happens inside. We just send the inputs and get the output. This does not allow us to add any new data into the model.

COMPARISON OF THE MODELS

A better model is picked based on the error values. Following are the errors that we have considered:

Mean Absolute Error is a model evaluation metric used with regression models. The mean absolute error of a model with respect to a data set is the mean of the absolute values of the individual prediction errors on over all instances in the data set. Each prediction error is the difference between the true value and the predicted value for the instance.

Mean Squared Error (MSE) or **Mean Squared Deviation** (MSD) of an estimator measures the average of the squares of the errors or deviations i.e. the difference between the estimator and what is estimated. MSE is a risk function, corresponding to the expected value of the squared error loss or quadratic loss. The difference occurs because of randomness or because the estimator doesn't account for information that could produce a more accurate estimate.

The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

R^2 (coefficient of determination) **regression score function**. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

Batting

	Neural Networks	Multiple Regression
Mean Absolute Error	251.243	80.188
Mean Squared Error	77932.660	12234.571
R2_Score	-2.640	0.429

On examining the absolute error, mean squared error and r2_score, it is evident that multiple regression model performs better on comparison with neural networks model. A negative R2_score indicates that it is a bad model. While the R2_score of Multiple regression is comparatively closer to zero, making it a better model.

Following is the comparison between list of Batsmen the model predicted, and the actual position as listed on Circbuzz.

Player_Name	Position
Hashim Amla	1
David Warner	2
Sanju Samson	3
Ben Stokes	4
Manish Pandey	5
Goutham Gambhir	6
Steven Smith	7
Suresh Raina	8
Moises Henriques	9
Shikar Dhawan	10

POS	PLAYER
1	David Warner
2	Gautam Gambhir
3	Shikhar Dhawan
4	Steven Smith
5	Suresh Raina
6	Hashim Amla
7	Manish Pandey
8	Parthiv Patel
9	Rahul Tripathi
10	Robin Uthappa

Bowling

	Neural Networks	Multiple Regression
Mean Absolute Error	3.1723575285644405	2.2121668164678181
Mean Squared Error	18.248918668716524	10.592058357195022
R2_Score	0.50052863253662272	0.71009625458031733

In case of bowling, both the models are performing almost similar. Their mean absolute error and r2_score are almost close. However, multiple regression model has lower mean_squared_error, making it a better model.

Following is the comparison between list of Bowlers the model predicted, and the actual position as listed on Circbuzz.

Bowler	Position
Mitchell McClenaghan	1
Jasprit Bumrah	2
Umesh Yadav	3
Bhuvneshwar Kumar	4
Sandeep Sharma	5
Chris Woakes	6
Mohit Sharma	7
Imran Tahir	8
Basil Thampi	9
Pat Cummins	10

POS	PLAYER
1	Bhuvneshwar Kumar
2	Jaydev Unadkat
3	Jasprit Bumrah
4	Mitchell McClenaghan
5	Imran Tahir
6	Rashid Khan
7	Sandeep Sharma
8	Umesh Yadav
9	Chris Woakes
10	Pawan Negi

CONCLUSION

We being very passionate about cricket, were very excited when we got hold of the IPL dataset. We started off by analyzing the dataset and as part of our first deliverable we did descriptive analytics. As the model deals with different players from all over the world and has data for a period of 10 years. During the course of 10 years few players got retired or did not play may matches. Therefore, we considered such players as exemptions and had a varying data set from year to year. We used multiple regression and neural networks to build models and came to conclusion that multiple regression performs better. Using this multiple regression model, we have predicted top ten batsmen and bowler. However, the difficulty here was that the retired/non-playing batsmen were also getting predicted. Hence, we removed them and picked up the next best players. We verified our results with the 2017 top ten batsmen and bowlers and found that 7 out of 10 players were predicted correctly.

From the business perspective this model, the methodology we used to build this model and the information we get from it can be used by the investors during IPL players auctioning. They can put their best bids on the top players and ultimately incur profits. Not just auctions, this model can also be used for Dream 11 - an online game, where users create a virtual team of real-life players and earn points based on the performances of these players in real matches. A user who scores the maximum points in his joined contest attains the first rank on the leaderboard. A similar method can be followed, and models can be built for various other leagues like Big Bash League, Caribbean Premier league.

REFERENCES

- https://en.wikipedia.org/wiki/Dream11
- https://sciencing.com/advantages-disadvantages-multiple-regression-model-12070171.html
- https://en.wikipedia.org/wiki/Mean_squared_error
- https://en.wikipedia.org/wiki/Indian_Premier_League
- http://www.cricbuzz.com/cricket-series/2568/indian-premier-league-2017/stats
- http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html