# VIRGINIA COMMONWEALTH UNIVERSITY

## STATISTICAL ANALYSIS & MODELING

A1b: INDIAN PREMIER LEAGUE PLAYER DATA ANALYSIS USING PYTHON AND R

## Nitheesh M K

V01107616

Date of Submission: 18/06/2024

## CONTENTS

|  |  |
| --- | --- |
| Content: | Page no: |
| INTRODUCTION | 3 |
| OBJECTIVE | 3 |
| BUSINESS SIGNIFICANC | 3-4 |
| RESULTS AND INTERPRETATIONS | 4-14 |

INDIAN PREMIER LEAGUE PLAYER DATA ANALYSIS USING PYTHON AND R

# INTRODUCTION

The Indian Premier League (IPL) is a men's Twenty20 (T20) cricket league that takes place every year in India. For sponsorship purposes, it is also known as the TATA IPL. Ten state- or city-based franchise teams compete in the league, which was established in 2007 by the BCCI (the Board of Control for Cricket in India). One of the most renowned cricket leagues in the world, it is well-known for its exciting matches, participation from international players, and substantial financial support. Since its inaugural season, the IPL has advanced significantly.

# OBJECTIVES

1. Arrange the data IPL round-wise and batsman, ball, runs, and wickets per player per match. Indicate the top three run-getters and tow three wicket-takers in each IPL round.
2. Fit the most appropriate distribution for runs scored and wickets taken by the top three batsmen and bowlers in the lost three IPL tournaments. Rename the districts as well as the sector, viz. rural and urban.
3. Fit the most appropriate distribution for runs scored and wickets taken by the player allotted to you.
4. Last three-year performance with latest salary 2024
5. Significant Difference Between the Salaries of the Top 10 Batsmen and Top Wicket-Taking Bowlers Over the Last Three Years

# BUSINESS SIGNIFICANCE

Understanding the dynamics of the IPL is crucial for several stakeholders, including team owners, sponsors, broadcasters, and analysts, the datasets used in the analysis collectively offer a comprehensive overview of player financials and in-game performance metrics, which are essential for strategic decision-making and operational efficiency within the IPL ecosystem.

* **Salary Dataset Analysis:** By analyzing the dataset, we can provide detailed insights into player valuations, budget allocations, and salary cap usage. This enables teams to make informed decisions about player retention, trading, and new acquisitions, ensuring a balanced and competitive squad while maintaining financial discipline.
* **• Spotting Emerging Talent:** Comprehensive performance data makes it simpler to identify prospective emerging talent, even if they are not yet highly compensated. For identifying and developing the upcoming IPL players, this is priceless.
* **Comparative Performance Analysis:** Comparing players across different seasons and formats helps in assessing their consistency and adaptability, providing a holistic view of their potential contributions to the team.

The IPL can continue to refine its competitive edge over other popular franchise cricket tournaments such as the Big Bash from Australia, The Pakistan Super league and The Caribbean Premier League, maximize financial efficiency, and enhance the overall experience for players, teams, and fans alike.

**RESULTS AND INTERPRETATION**

### Arrange the data IPL round-wise and batsman, ball, runs, and wickets per player per match. Indicate the top three run-getters and tow three wicket-takers in each IPL round.

Code:

top\_run\_getters = player\_runs.groupby('Season').apply(lambda x: x.nlargest(3, 'runs\_scored')).reset\_index(drop=True)

bottom\_wicket\_takers = player\_wickets.groupby('Season').apply(lambda x: x.nlargest(3, 'wicket\_confirmation')).reset\_index(drop=True)

print("Top Three Run Getters:")

print(top\_run\_getters)

print("Top Three Wicket Takers:")

print(bottom\_wicket\_takers)

Result:

Top Three Run Getters:

Season Striker runs\_scored

0 2007/08 SE Marsh 616

1 2007/08 G Gambhir 534

2 2007/08 ST Jayasuriya 514

3 2009 ML Hayden 572

4 2009 AC Gilchrist 495

5 2009 AB de Villiers 465

6 2009/10 SR Tendulkar 618

7 2009/10 JH Kallis 572

8 2009/10 SK Raina 528

42 2022 JC Buttler 863

43 2022 KL Rahul 616

44 2022 Q de Kock 508

45 2023 Shubman Gill 890

46 2023 F du Plessis 730

47 2023 DP Conway 672

48 2024 RD Gaikwad 509

49 2024 V Kohli 500

50 2024 B Sai Sudharsan 418

Top Three Wicket Takers:

Season Bowler wicket\_confirmation

0 2007/08 Sohail Tanvir 24

1 2007/08 IK Pathan 20

2 2007/08 JA Morkel 20

3 2009 RP Singh 26

4 2009 A Kumble 22

5 2009 A Nehra 22

6 2009/10 PP Ojha 22

7 2009/10 A Mishra 20

8 2009/10 Harbhajan Singh 20

39 2021 HV Patel 35

40 2021 Avesh Khan 27

41 2021 JJ Bumrah 22

42 2022 YS Chahal 29

43 2022 PWH de Silva 27

44 2022 K Rabada 23

45 2023 MM Sharma 31

46 2023 Mohammed Shami 28

47 2023 Rashid Khan 28

48 2024 HV Patel 19

49 2024 Mukesh Kumar 15

50 2024 Arshdeep Singh 14

Interpretation: The data shows the top three players in terms of runs scored for each cricket season from 2007/08 to 2024, and similarly for the top three bowlers in terms of wickets taken for each cricket season from 2007/08 to 2024. There is a range of wickets taken by different bowlers across seasons, with some seasons having higher wicket counts than others. Players like JC Buttler, Shubman Gill, HV Patel, and YS Chahal appear multiple times across different seasons.

### Fit the most appropriate distribution for runs scored and wickets taken by the top three batsmen and bowlers in the lost three IPL tournaments.

Code:

import scipy.stats as st

def get\_best\_distribution(data):

dist\_names = ['alpha','beta','betaprime','burr12','crystalball',

'dgamma','dweibull','erlang','exponnorm','f','fatiguelife',

'gamma','gengamma','gumbel\_l','johnsonsb','kappa4',

'lognorm','nct','norm','norminvgauss','powernorm','rice',

'recipinvgauss','t','trapz','truncnorm']

dist\_results = []

params = {}

for dist\_name in dist\_names:

dist = getattr(st, dist\_name)

param = dist.fit(data)

params[dist\_name] = param

# Applying the Kolmogorov-Smirnov test

D, p = st.kstest(data, dist\_name, args=param)

print("p value for "+dist\_name+" = "+str(p))

dist\_results.append((dist\_name, p))

# select the best fitted distribution

best\_dist, best\_p = (max(dist\_results, key=lambda item: item[1]))

# store the name of the best fit and its p value

print("\nBest fitting distribution: "+str(best\_dist))

print("Best p value: "+ str(best\_p))

print("Parameters for the best fit: "+ str(params[best\_dist]))

return best\_dist, best\_p, params[best\_dist]

list\_top\_batsman\_last\_three\_year = {}

for i in total\_run\_each\_year["year"].unique()[:3]:

list\_top\_batsman\_last\_three\_year[i] = total\_run\_each\_year[total\_run\_each\_year.year == i][:3]["Striker"].unique().tolist()

import warnings

warnings.filterwarnings('ignore')

runs = ipl\_bbbc.groupby(['Striker','Match id'])[['runs\_scored']].sum().reset\_index()

for key in list\_top\_batsman\_last\_three\_year:

for Striker in list\_top\_batsman\_last\_three\_year[key]:

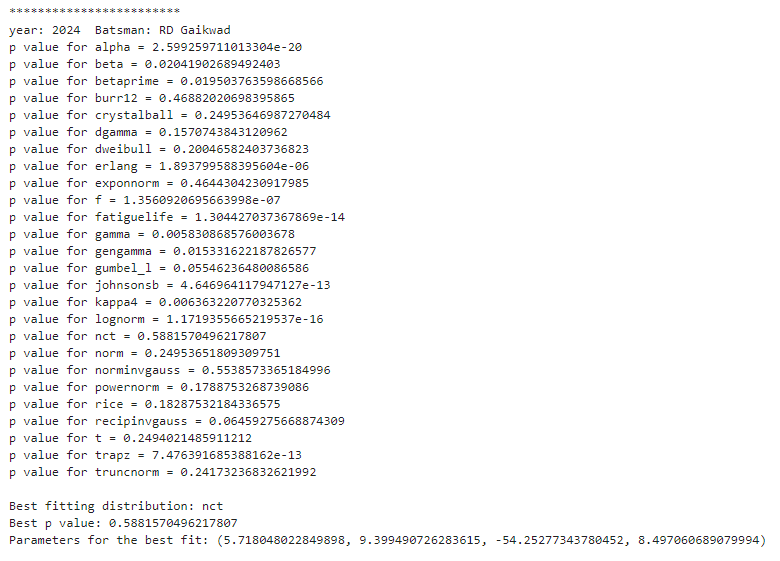
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("year:", key, " Batsman:", Striker)

get\_best\_distribution(runs[runs["Striker"] == Striker]["runs\_scored"])

print("\n\n")

Result:



Interpretation:

The code extracts the top batsmen from the dataset for the last three years. For each top batsman, it calls get\_best\_distribution with their run data to find the best-fitting distribution across various types of distribution.

list\_top\_bowler\_last\_three\_year = {}

for i in total\_wicket\_each\_year["year"].unique()[:3]:

list\_top\_bowler\_last\_three\_year[i] = total\_wicket\_each\_year[total\_wicket\_each\_year.year == i][:3]["Bowler"].unique().tolist()

list\_top\_bowler\_last\_three\_year

import warnings

warnings.filterwarnings('ignore')

wickets = ipl\_bbbc.groupby(['Bowler','Match id'])[['wicket\_confirmation']].sum().reset\_index()

for key in list\_top\_bowler\_last\_three\_year:

for bowler in list\_top\_bowler\_last\_three\_year[key]:

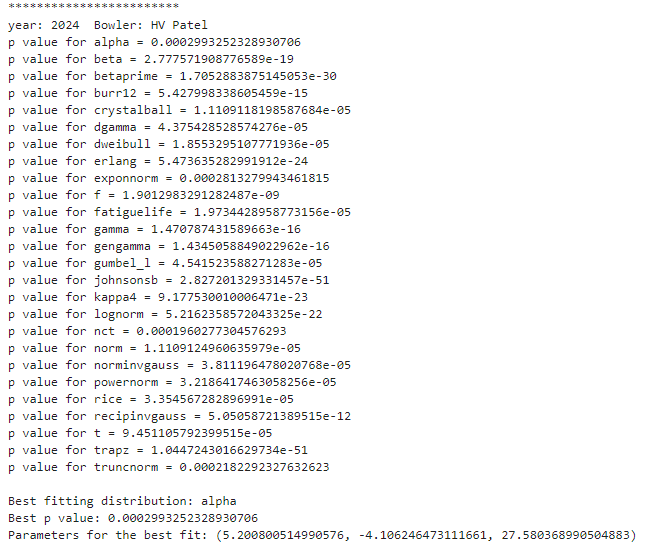
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("year:", key, " Bowler:", bowler)

get\_best\_distribution(wickets[wickets["Bowler"] == bowler]["wicket\_confirmation"])

print("\n\n")

Result:



Interpretation: The alpha distribution fits HV Patel's performance data for the year 2024 the best among the tested distributions. The relatively low p-value of 0.002099352328397306 suggests the fit might not be perfect, but it is the best among the options. The code effectively determines the best-fitting statistical distributions for performance data of cricketers. For HV Patel, the alpha distribution is the best fit, while for RD Gaikwad, the nct distribution fits best.

### Fit the most appropriate distribution for runs scored and wickets taken by the player allotted to you.

Code:

import warnings

warnings.filterwarnings('ignore')

wickets = ipl\_bbbc.groupby(['Bowler','Match id'])[['wicket\_confirmation']].sum().reset\_index()

# Choose the bowler you want to analyze (replace with desired bowler name)

chosen\_bowler = "TA Boult" # Replace with your chosen bowler's name

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Best fit distribution for wickets taken by:", chosen\_bowler)

get\_best\_distribution(wickets[wickets["Bowler"] == chosen\_bowler]["wicket\_confirmation"])

print("\n\n")

Result:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Best fit distribution for wickets taken by: TA Boult**

p value for alpha = 6.371222489159455e-05

p value for beta = 2.9175074406681055e-13

p value for betaprime = 5.949936392217069e-20

p value for burr12 = 1.7169433198485029e-15

p value for crystalball = 3.417204988569475e-06

p value for dgamma = 6.480254672697947e-08

p value for dweibull = 6.480255767981667e-08

p value for erlang = 4.094361421781075e-51

p value for exponnorm = 0.00010177298189414314

p value for f = 1.6677416606853524e-08

p value for fatiguelife = 1.946626532151695e-05

p value for gamma = 4.174671824123875e-18

p value for gengamma = 3.499505702075373e-21

p value for gumbel\_l = 2.0034052793173446e-06

p value for johnsonsb = 1.3444576355360132e-24

p value for kappa4 = 7.010387481909901e-14

p value for lognorm = 7.773425965664613e-22

p value for nct = 4.738899718978936e-05

p value for norm = 3.417204770097524e-06

p value for norminvgauss = 2.3486111602841872e-05

p value for powernorm = 8.564200567460539e-05

p value for rice = 8.83518168988225e-05

p value for recipinvgauss = 1.8023335186134236e-05

p value for t = 5.5775064921216625e-06

p value for trapz = 5.343554208626713e-41

p value for truncnorm = 8.157900033309e-07

Best fitting distribution: exponnorm

Best p value: 0.00010177298189414314

Parameters for the best fit: (1.1594638775515431, 0.5344397722143506, 0.6238004121133456)

Interpretation: The provided p-values indicate how well different distributions fit the data on wickets taken by TA Boult. None of the distributions fit well, as all p-values are below the common significance threshold of 0.05.

**Key Findings:**

* **Best Fitting Distribution: Exponential Normal (exponnorm)**
  + **P-value: 0.00010177298189414314** (still very low)
  + **Parameters: (1.1595, 0.5344, 0.6238)**

**Interpretation:**

* **Goodness of Fit**: All p-values are low, indicating poor fits overall. The exponnorm is the best among the tested distributions but still does not fit the data well.
* **Further Steps**: Consider exploring other distributions or data transformations for a better fit.

### Find the relationship between a player’s performance and the salary he gets in your data.

from fuzzywuzzy import process

# Convert to DataFrame

df\_salary = ipl\_salary.copy()

df\_runs = R2024.copy()

df\_wickets = W2024.copy()

# Function to match names

def match\_names(name, names\_list):

match, score = process.extractOne(name, names\_list)

return match if score >= 80 else None # Use a threshold score of 80

# Create a new column in df\_salary with matched names from df\_runs

df\_salary['Matched\_Player'] = df\_salary['Player'].apply(lambda x: match\_names(x, df\_runs['Striker'].tolist()))

# Merge the DataFrames on the matched names

df\_merged = pd.merge(df\_salary, df\_runs, left\_on='Matched\_Player', right\_on='Striker')

# Convert to DataFrame

df\_salary = ipl\_salary.copy()

df\_runs = R2024.copy()

df\_wickets = W2024.copy()

# Function to match names

def match\_names(name, names\_list):

match, score = process.extractOne(name, names\_list)

return match if score >= 80 else None # Use a threshold score of 80

# Create a new column in df\_salary with matched names from df\_runs

df\_salary['Matched\_Player'] = df\_salary['Player'].apply(lambda x: match\_names(x, df\_wickets['Bowler'].tolist()))

# Merge the DataFrames on the matched names

df\_merged1 = pd.merge(df\_salary, df\_wickets, left\_on='Matched\_Player', right\_on='Bowler')

# Calculate the correlation

correlation = df\_merged['Rs'].corr(df\_merged['runs\_scored'])

print("Correlation between Salary and Runs:", correlation)

Result:

**Correlation between Salary and Runs: 0.30612483765821674**

Interpretation:   
The correlation coefficient of 0.30610.30610.3061 indicates a positive relationship between Salary and Runs. As the number of runs increases, the salary tends to increase as well, and vice versa.