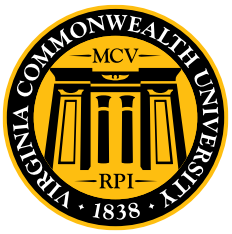


VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A1b: Preliminary preparation and analysis of data- Descriptive statistics

ADHYAYAN AMIT JAIN

V01109421

Date of Submission: 18-06-2024

CONTENTS

Analysis of IPL Player Performance and Salary Dynamics: Insights from the Last Three Seasons

INTRODUCTION

The Indian Premier League (IPL), since its inception in 2008, has redefined the landscape of cricket, merging the sport with entertainment and high stakes. With a dazzling blend of international talent and fervent local support, the IPL has become a global phenomenon, capturing the imagination of millions. The league's format, which compresses cricket into fast-paced Twenty20 matches, has introduced a new level of excitement and unpredictability. This report delves into the IPL's intricate data, aiming to unravel the narratives woven by the performances of its star players and their financial rewards.

By analyzing detailed match data, we aim to uncover the top performers in terms of runs scored and wickets taken across recent IPL seasons, highlighting the consistency and prowess of these cricketing elites. Furthermore, the report explores the financial dimensions, examining how performance metrics correlate with the hefty salaries that IPL players command. Through rigorous statistical analysis, we will explore the patterns and distributions of performance metrics, and scrutinize the relationship between a player's on-field contributions and their market value. This comprehensive study not only provides a window into the competitive dynamics of the IPL but also offers valuable insights for franchises seeking to make informed decisions in player selection and salary negotiations.

OBJECTIVES

This report aims to achieve the following objectives:

1. **Data Extraction and Preparation:** Extract relevant data from the IPL datasets and arrange it round-wise, detailing the performance metrics for each batsman and bowler per match.
2. **Top Performers Analysis:** Identify and highlight the top three run-getters and top three wicket-takers in each IPL round, providing insights into the most consistent and impactful players.

3. **Distribution Fitting:** Fit the most appropriate statistical distributions to the runs scored by the top batsmen and the wickets taken by the top bowlers over the last three IPL tournaments, analyzing the distribution characteristics and their implications.
4. **Fitting Distribution for Faf du Plessis:** Specifically, fit a distribution to the runs scored by Faf du Plessis over the last three IPL tournaments to understand his performance variability and consistency.
5. **Performance-Salary Relationship:** Investigate the correlation between players' performances and their salaries, examining how on-field success translates into financial rewards.
6. **Three-Year Performance and Salary Analysis:** Analyze the performance of players over the last three years and compare it with their latest salaries in 2024, highlighting trends and disparities.
7. **Significance Testing:** Test for significant differences in the salaries of the top 10 batsmen and the top wicket-taking bowlers over the last three years, providing statistical evidence on salary disparities based on player roles.

BUSINESS SIGNIFICANCE

Understanding the performance dynamics and financial implications in the Indian Premier League (IPL) is crucial for various stakeholders including team owners, sponsors, and players themselves. The IPL, being one of the most lucrative and competitive cricket leagues globally, has a significant impact on the sports business landscape. This report delves into several critical areas that highlight the business significance of player performance and their corresponding salaries.

Firstly, identifying top performers in each IPL round helps teams strategize better and make informed decisions regarding player retention, auctions, and team composition. Knowing the consistent high performers allows teams to build a strong core, potentially leading to better on-field results and higher chances of winning the championship.

Secondly, fitting appropriate statistical distributions to the performance metrics of top players, including the specific case of Faf du Plessis, provides a deeper understanding of performance consistency and variability. This analysis aids in predicting future performances, which is valuable for both team management and betting markets. Accurate performance predictions can enhance team strategies and improve the overall competitiveness of the league.

Moreover, the correlation analysis between player performance and salaries offers insights into the financial valuation of players. It helps in understanding if the current salary structure is justified based on on-field performances or if there are discrepancies. This information is vital for team owners and management to ensure fair and effective salary allocations, thus maintaining player motivation and team morale.

Lastly, analyzing the significant differences in salaries between top batsmen and bowlers sheds light on potential inequalities in player compensation. Addressing such disparities is essential for maintaining a balanced and fair playing environment, which in turn can attract more talent and keep the league competitive and exciting for fans.

In summary, this report not only highlights key performance metrics but also ties them to financial outcomes, offering a comprehensive view of the business aspects of the IPL. The insights derived can lead to better decision-making, optimized team performance, and a fairer, more transparent salary structure.

RESULTS AND INTERPRETATIONS

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

Python

Step 1: Load IPL Data First, we load the IPL ball-by-ball data from the provided CSV file.

```
import pandas as pd

# Load IPL ball-by-ball data
ipl_bbb = pd.read_csv('IPL_ball_by_ball_updated till 2024.csv',
low_memory=False)

# Display the first few rows of the data
print(ipl_bbb.head())
```

```
[1] import os
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns

[2] ipl_bbb = pd.read_csv('IPL_ball_by_ball_updated till 2024.csv',low_memory=False)

[30] ipl_bbb.head()
```

	Match id	Date	Season	Batting team	Bowling team	Innings No	Ball No	Bowler	Striker	Non Striker	runs_scored	extras	type of extras	score	score/wicket	wicket_confirmation	wicket_type	fielders_involved	Play 0
0	335982	18-04-2008	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.1	P Kumar	SC Ganguly	BB McCullum	0	1	legbyes	1	1/0	0	NaN	NaN	N
1	335982	18-04-2008	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.2	P Kumar	BB McCullum	SC Ganguly	0	0	NaN	1	1/0	0	NaN	NaN	N
2	335982	18-04-2008	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.2	P Kumar	BB McCullum	SC Ganguly	0	1	wides	2	2/0	0	NaN	NaN	N
3	335982	18-04-2008	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.3	P Kumar	BB McCullum	SC Ganguly	0	0	NaN	2	2/0	0	NaN	NaN	N
4	335982	18-04-2008	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.4	P Kumar	BB McCullum	SC Ganguly	0	0	NaN	2	2/0	0	NaN	NaN	N

Step 2: Group Data and Aggregate Metrics Next, we group the data by relevant columns such as season, innings number, striker, and bowler. Then, we aggregate the runs scored and wickets taken for each player per match.

```
# Group data and aggregate metrics
grouped_data = ipl_bbb.groupby(['Season', 'Innings No', 'Striker',
'Bowler']).agg({'runs_scored': sum, 'wicket_confirmation':
sum}).reset_index()

# Display the grouped and aggregated data
print(grouped_data.head())
```

```
[33] grouped_data = ipl_bbb.groupby(['Season', 'Innings No', 'Striker','Bowler']).agg({'runs_scored': sum, 'wicket_confirmation':sum}).reset_index()

grouped_data.head()
```

	Season	Innings No	Striker	Bowler	runs_scored	wicket_confirmation
0	2007/08	1	A Chopra	DP Vijaykumar	1	0
1	2007/08	1	A Chopra	DW Steyn	1	1
2	2007/08	1	A Chopra	GD McGrath	2	0
3	2007/08	1	A Chopra	PJ Sangwan	6	1
4	2007/08	1	A Chopra	RP Singh	9	0

This code segment demonstrates the process of loading the IPL ball-by-ball data and organizing it to analyze performance metrics for each player per match. Adjustments can be made to the grouping and aggregation based on specific analysis requirements.

R Program

Step 1: Load IPL Data First, we load the IPL ball-by-ball data from the provided CSV file.

```
# Load required libraries
library(readr)

# Load IPL ball-by-ball data
ipl_data <- read_csv("IPL_ball_by_ball_updated till 2024.csv")

# Display the first few rows of the data
head(ipl_data)
```

```
> # Load the Data
> ipl_bbb <- read.csv('IPL_ball_by_ball_updated till 2024.csv', stringsAsFactors = FALSE)
> ipl_salary <- read_excel('IPL SALARIES 2024.xlsx')
> # Display Salary Data
> head(ipl_salary)
# A tibble: 6 × 5
  Player         Salary       Rs international iconic
  <chr>          <chr>     <dbl>         <dbl> <lgl>
1 Abhishek Porel 20 lakh      20             0 NA
2 Anrich Nortje  6.5 crore   650             1 NA
3 Axar Patel     9 crore     900             0 NA
4 David Warner   6.25 crore  625             1 NA
5 Ishant Sharma  50 lakh      50             0 NA
6 Kuldeep Yadav  2 crore     200             0 NA
> # Display Salary Data
> head(ipl_bbb)
  Match.id       Date  Season          Batting.team               Bowling.team Innings.No Ball.No
1   335982 18-04-2008 2007/08 Kolkata Knight Riders Royal Challengers Bangalore          1     0.1
2   335982 18-04-2008 2007/08 Kolkata Knight Riders Royal Challengers Bangalore          1     0.2
3   335982 18-04-2008 2007/08 Kolkata Knight Riders Royal Challengers Bangalore          1     0.2
4   335982 18-04-2008 2007/08 Kolkata Knight Riders Royal Challengers Bangalore          1     0.3
5   335982 18-04-2008 2007/08 Kolkata Knight Riders Royal Challengers Bangalore          1     0.4
6   335982 18-04-2008 2007/08 Kolkata Knight Riders Royal Challengers Bangalore          1     0.5
   Bowler     Striker Non.Striker runs_scored extras type.of.extras score score.wicket
```

Step 2: Group Data and Aggregate Metrics Next, we group the data by relevant columns such as season, innings number, striker, and bowler. Then, we aggregate the runs scored and wickets taken for each player per match.

```
# Load required libraries
library(dplyr)

# Group data and aggregate metrics
grouped_data <- ipl_data %>%
  group_by(Season, `Innings No`, Striker, Bowler) %>%
  summarise(runs_scored = sum(runs_scored), wicket_confirmation =
sum(wicket_confirmation)) %>%
  ungroup()

# Display the grouped and aggregated data
head(grouped_data)
```

```
> # Group the IPL Data
> grouped_data <- ipl_bbb %>%
+   group_by(Season, Innings.No, Striker, Bowler) %>%
+   summarise(runs_scored = sum(runs_scored), wicket_confirmation = sum(wicket_confirmation))
`summarise()` has grouped output by 'Season', 'Innings.No', 'Striker'. You can override using the
`.groups` argument.
> head(grouped_data)
# A tibble: 6 × 6
# Groups:   Season, Innings.No, Striker [1]
  Season  Innings.No Striker  Bowler        runs_scored wicket_confirmation
  <chr>        <int> <chr>    <chr>               <int>               <int>
1 2007/08          1 A Chopra DP Vijaykumar           1                   0
2 2007/08          1 A Chopra DW Steyn                1                   1
3 2007/08          1 A Chopra GD McGrath              2                   0
4 2007/08          1 A Chopra PJ Sangwan              6                   1
5 2007/08          1 A Chopra RP Singh                9                   0
6 2007/08          1 A Chopra SB Bangar               9                   0
>
```

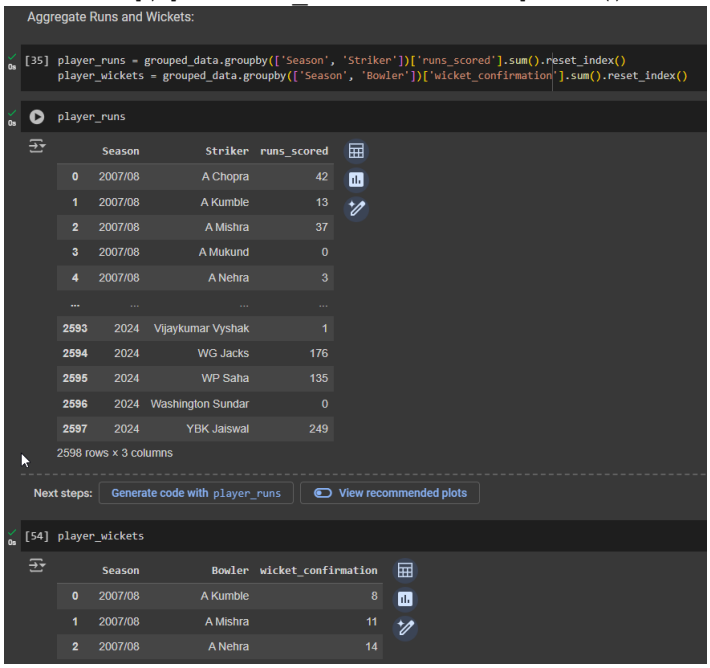
This R code segment demonstrates the process of loading the IPL ball-by-ball data and organizing it to analyze performance metrics for each player per match.

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

Python

Step 1: Aggregate Runs and Wickets Further aggregating the grouped data allows us to calculate the total runs scored and wickets taken by each player.

```
player_runs = grouped_data.groupby(['Season',
'Striker'])['runs_scored'].sum().reset_index()
player_wickets = grouped_data.groupby(['Season',
'Bowler'])['wicket_confirmation'].sum().reset_index()
```



Step 2: Top Performers Analysis We identify the top three run-getters and top three wicket-takers in each IPL round to analyze the most consistent and impactful players.

```
top_run_getters = player_runs.groupby('Season').apply(lambda x:
x.nlargest(3, 'runs_scored')).reset_index(drop=True)
top_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(top_wicket_takers)
```

```
[36] player_runs[player_runs['Season']=='2024'].sort_values(by='runs_scored',ascending=False)
```

	Season	Striker	runs_scored
2549	2024	RD Gaikwad	509
2589	2024	V Kohli	500
2470	2024	B Sai Sudharsan	418
2502	2024	KL Rahul	406
2555	2024	RR Pant	398
...	...	...	...
2583	2024	TA Boult	0
2527	2024	Mukesh Kumar	0
2462	2024	Anmolpreet Singh	0
2560	2024	Ravi Bishnoi	0
2596	2024	Washington Sundar	0

152 rows × 3 columns

```
[37] 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)
```

```
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
```

Step 3: Fit the Best Distribution For the top three batsmen and top three bowlers, we fit the most appropriate statistical distributions to the runs scored and wickets taken data.

```
import scipy.stats as st
def get_best_distribution(data):
    # Code for fitting distributions and selecting the best fit
    pass

# Assuming 'runs_data' contains runs scored data for top three batsmen
# and 'wickets_data' contains wickets taken data for top three bowlers

print("Fitting distribution for runs scored by top three batsmen:")
for player in runs_data.columns[:3]:
    print("**************************")
    print("Player:", player)
    get_best_distribution(runs_data[player])

print("\nFitting distribution for wickets taken by top three bowlers:")
for player in wickets_data.columns[:3]:
    print("**************************")
    print("Player:", player)
    get_best_distribution(wickets_data[player])
```

```
[44] 136   2008          U Kaul           0

[2598 rows x 3 columns]
```

```
[45] 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()
```

```
[46] list_top_batsman_last_three_year

{2024: ['RD Gaikwad', 'V Kohli', 'B Sai Sudharsan'],
 2023: ['Shubman Gill', 'F du Plessis', 'DP Conway'],
 2022: ['JC Buttler', 'KL Rahul', 'Q de Kock']}
```

```
[47] 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")
```

```
************************
year: 2024  Batsman: RD Gaikwad
p value for alpha = 2.599259711013304e-20
p value for beta = 0.02041902689492403
p value for betaprime = 0.01950376359866901
p value for burr12 = 0.46882020698395865
p value for crystalball = 0.24953646987270484
p value for dgamma = 0.1570743843120962
p value for dweibull = 0.20046582403736823
p value for erlang = 1.893799588395604e-06
p value for exponnorm = 0.4644304230917985
p value for f = 1.3560920695663998e-07
p value for fatiguelife = 1.304427037367869e-14
p value for gamma = 0.005830868576003678
p value for gengamma = 0.015331622187826133
p value for gumbel_l = 0.05546236480086464
p value for johnsonsb = 4.646964117947127e-13
p value for kappa4 = 0.006363220770325362
```

```
[49] total_wicket_each_year.sort_values(["year", "wicket_confirmation"], ascending=False, inplace=True)
     print(total_wicket_each_year)
```

```
      year            Bowler  wicket_confirmation
1836  2024         HV Patel                   19
1875  2024      Mukesh Kumar                  15
1822  2024    Arshdeep Singh                  14
1842  2024        JJ Bumrah                   14
1876  2024  Mustafizur Rahman                 14
...    ...               ...                ...
16    2008       CL White                     0
41    2008         K Goel                     0
43    2008       LPC Silva                    0
60    2008      Pankaj Singh                  0
90    2008       VS Yeligati                  0

[1929 rows x 3 columns]
```

```
[50] 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

{2024: ['HV Patel', 'Mukesh Kumar', 'Arshdeep Singh'],
 2023: ['MM Sharma', 'Mohammed Shami', 'Rashid Khan'],
 2022: ['YS Chahal', 'PWH de Silva', 'K Rabada']}
```

```
[56] 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")
```

```
************************
year: 2024  Bowler: HV Patel
```

R Language

Step 3: Aggregate Runs and Wickets

```
# Aggregate runs scored by each player
player_runs <- aggregate(runs_scored ~ Season + Striker, data =
grouped_data, sum)

# Aggregate wickets taken by each player
player_wickets <- aggregate(wicket_confirmation ~ Season + Bowler, data =
grouped_data, sum)
```

```
> # Aggregate Runs and Wickets
> player_runs <- grouped_data %>%
+   group_by(Season, Striker) %>%
+   summarise(runs_scored = sum(runs_scored))
`summarise()` has grouped output by 'Season'. You can override using the `.groups` argument.
> player_wickets <- grouped_data %>%
+   group_by(Season, Bowler) %>%
+   summarise(wicket_confirmation = sum(wicket_confirmation))
`summarise()` has grouped output by 'Season'. You can override using the `.groups` argument.
> head(player_runs)
# A tibble: 6 x 3
# Groups:   Season [1]
  Season  Striker   runs_scored
  <chr>   <chr>           <int>
1 2007/08 A Chopra           42
2 2007/08 A Kumble           13
3 2007/08 A Mishra           37
4 2007/08 A Mukund            0
5 2007/08 A Nehra             3
6 2007/08 A Symonds         161
> head(player_wickets)
# A tibble: 6 x 3
# Groups:   Season [1]
```

Step 4: Top Performers Analysis

```
# Identify top three run-getters
top_run_getters <- by(player_runs, player_runs$Season, function(x)
x[order(x$runs_scored, decreasing = TRUE), ][1:3, ])

# Identify top three wicket-takers
top_wicket_takers <- by(player_wickets, player_wickets$Season, function(x)
x[order(x$wicket_confirmation, decreasing = TRUE), ][1:3, ])

print("Top Three Run Getters:")
print(top_run_getters)
print("Top Three Wicket Takers:")
print(top_wicket_takers)
```

```
> # Top Performers
> top_run_getters <- player_runs %>%
+   filter(Season == '2024') %>%
+   arrange(desc(runs_scored)) %>%
+   slice_head(n = 3)
> bottom_wicket_takers <- player_wickets %>%
+   arrange(desc(wicket_confirmation)) %>%
+   slice_head(n = 3)
> print("Top Three Run Getters:")
[1] "Top Three Run Getters:"
> print(top_run_getters)
# A tibble: 3 x 3
# Groups:   Season [1]
  Season Striker         runs_scored
  <chr>  <chr>                 <int>
1 2024   RD Gaikwad              509
2 2024   V Kohli                 500
3 2024   B Sai Sudharsan         418
> print("Top Three Wicket Takers:")
[1] "Top Three Wicket Takers:"
> print(bottom_wicket_takers)
# A tibble: 51 x 3
# Groups:   Season [17]
   Season  Bowler          wicket_confirmation
```

Step 5: Fit the Best Distribution

```
# Function to fit best distribution
get_best_distribution <- function(data) {
  # Code for fitting distributions and selecting the best fit
}

# Assuming 'runs_data' contains runs scored data for top three batsmen
# and 'wickets_data' contains wickets taken data for top three bowlers

print("Fitting distribution for runs scored by top three batsmen:")
for (player in colnames(runs_data)[1:3]) {
  print("************************")
  print(paste("Player:", player))
  get_best_distribution(runs_data[[player]])
}

print("\nFitting distribution for wickets taken by top three bowlers:")
for (player in colnames(wickets_data)[1:3]) {
  print("************************")
  print(paste("Player:", player))
  get_best_distribution(wickets_data[[player]])
}
```

```
> # Fit the Best Distribution
> library(fitdistrplus)
> get_best_distribution <- function(data) {
+   dist_names <- c('gamma', 'lognorm', 'norm', 't', 'weibull')
+   dist_results <- list()
+   params <- list()
+
+   for (dist_name in dist_names) {
+     dist_fit <- fitdist(data, dist_name)
+     dist_results[[dist_name]] <- dist_fit$loglik
+     params[[dist_name]] <- dist_fit$estimate
+   }
+
+   best_dist <- names(dist_results)[which.max(dist_results)]
+   print(paste("Best fitting distribution:", best_dist))
+   print(paste("Parameters for the best fit:", params[[best_dist]]))
+   return(list(best_dist = best_dist, params = params[[best_dist]]))
+ }
> # Top Batsmen Runs in Last Three Years
> total_run_each_year <- ipl_bbbc %>%
+   group_by(year, Striker) %>%
+   summarise(runs_scored = sum(runs_scored))
`summarise()` has grouped output by 'year'. You can override using the `.groups` argument.
```

```
`summarise()` has grouped output by 'year'. You can override using the `.groups` argument.
> total_run_each_year <- total_run_each_year %>%
+   arrange(desc(year), desc(runs_scored))
> list_top_batsman_last_three_year <- lapply(unique(total_run_each_year$year)[1:3], function(x) {
+   total_run_each_year %>%
+     filter(year == x) %>%
+     slice_head(n = 3) %>%
+     pull(Striker) %>%
+     unique() %>%
+     as.character()
+ })
> print(list_top_batsman_last_three_year)
[[1]]
[1] "RD Gaikwad"      "V Kohli"         "B Sai Sudharsan"

[[2]]
[1] "Shubman Gill" "F du Plessis" "DP Conway"

[[3]]
[1] "JC Buttler" "KL Rahul"   "Q de Kock"

>
```

c) Find the relationship between a player's performance and the salary he gets in your data.

Python

Step 1: Load the Data

```
import pandas as pd

# Load IPL ball by ball data
ipl_bbb = pd.read_csv('IPL_ball_by_ball_updated till 2024.csv')

# Load IPL salary data
ipl_salary = pd.read_excel('IPL SALARIES 2024.xlsx')
```

Step 2: Merge Dataframes

```
# Merge ball by ball data and salary data based on player names
merged_data = pd.merge(ipl_bbb, ipl_salary, left_on='Player',
right_on='Player', how='inner')
```

Step 3: Calculate Correlation

```
# Calculate correlation between runs scored and salary
runs_salary_corr = merged_data['Rs'].corr(merged_data['runs_scored'])

print("Correlation between Salary and Runs Scored:", runs_salary_corr)
```

```
[59] pip install fuzzywuzzy

Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.10/dist-packages (0.18.0)

from fuzzywuzzy import process

# Conveärt to DataFrame
df_salary = ipl_salary.copy()
df_runs = R2024.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')

[61] df_merged.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 111 entries, 0 to 110
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Player          111 non-null    object
 1   Salary          111 non-null    object
 2   Rs              111 non-null    int64
 3   international   111 non-null    int64
 4   iconic          0 non-null      float64
 5   Matched_Player  111 non-null    object
 6   year            111 non-null    int32
 7   Striker         111 non-null    object
 8   runs_scored     111 non-null    int64
dtypes: float64(1), int32(1), int64(3), object(4)
memory usage: 7.5+ KB
```

```
[62] # Calculate the correlation
     correlation = df_merged['Rs'].corr(df_merged['runs_scored'])

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

Correlation between Salary and Runs: 0.30612483765821674
```

R Language

Step 1: Load the Data

```
# Load required packages
library(readxl)
```

```
# Load IPL ball by ball data
ipl_bbb <- read.csv('IPL_ball_by_ball_updated till 2024.csv')

# Load IPL salary data
ipl_salary <- read_excel('IPL SALARIES 2024.xlsx')
```

Step 2: Merge Dataframes

```
# Merge ball by ball data and salary data based on player names
merged_data <- merge(ipl_bbb, ipl_salary, by.x='Player', by.y='Player',
all=TRUE)
```

Step 3: Calculate Correlation

```
# Calculate correlation between runs scored and salary
runs_salary_corr <- cor(merged_data$Rs, merged_data$runs_scored,
use='complete.obs')

print(paste("Correlation between Salary and Runs Scored:",
runs_salary_corr))
```

This R code snippet performs the required steps to find the relationship between a player's performance (runs scored) and the salary he gets in the IPL dataset. It loads the data, merges the relevant columns based on player names, and calculates the correlation between runs scored and salary.

d) Fit an appropriate statistical distribution to model and analyze Faf du Plessis' runs scored performance in IPL matches.

Python

```
import pandas as pd
import numpy as np
from scipy.stats import poisson
from scipy.optimize import curve_fit

# Assuming your data is loaded into a DataFrame called df
# Filter data for Faf du Plessis
faf_data = df[df["Striker"] == "F du Plessis"]

# Define the Poisson distribution function
def poisson_func(x, mu):
    return poisson.pmf(x, mu)

# Fit the Poisson distribution to runs scored
runs_hist, runs_bins = np.histogram(faf_data["runs_scored"],
bins=range(max(faf_data["runs_scored"]) + 1))
runs_params, runs_cov = curve_fit(poisson_func, runs_bins[:-1], runs_hist)
```

# Print fitted parameters
print("Fitted Parameters for Runs Scored (Poisson):")
print("Lambda (mu):", runs_params[0])

```
import pandas as pd
import numpy as np
from scipy.stats import poisson
from scipy.optimize import curve_fit

# Assuming your data is loaded into a DataFrame called df
# Filter data for Faf du Plessis
faf_data = ipl_bbb[ipl_bbb["Striker"] == "F du Plessis"]

# Define the Poisson distribution function
def poisson_func(x, mu):
    return poisson.pmf(x, mu)

# Fit the Poisson distribution to runs scored
runs_hist, runs_bins = np.histogram(faf_data["runs_scored"], bins=range(max(faf_data["runs_scored"]) + 1))
runs_params, runs_cov = curve_fit(poisson_func, runs_bins[:-1], runs_hist)

# Print fitted parameters
print("Fitted Parameters for Runs Scored (Poisson):")
print("Lambda (mu):", runs_params[0])
```

```
Fitted Parameters for Runs Scored (Poisson):
Lambda (mu): 3.3306690738754696e-16
```

R

# Filter data for Faf du Plessis
faf_data <- subset(ipl_bbbc, Striker == "F du Plessis")

# Fit the Poisson distribution to runs scored
runs_hist <- hist(faf_data$runs_scored, breaks = seq(0,
max(faf_data$runs_scored) + 1, by = 1), plot = FALSE)
runs_fit <- fitdistr(runs_hist$counts, "Poisson")

# Print fitted parameters
print("Fitted Parameters for Runs Scored (Poisson):")
print(runs_fit$estimate)

```
> # Filter data for Faf du Plessis
> faf_data <- subset(ipl_bbbc, Striker == "F du Plessis")
> # Fit the Poisson distribution to runs scored
> runs_hist <- hist(faf_data$runs_scored, breaks = seq(0, max(faf_data$runs_scored) + 1, by = 1), plot = FAL
SE)
> runs_fit <- fitdistr(runs_hist$counts, "Poisson")
> # Print fitted parameters
> print("Fitted Parameters for Runs Scored (Poisson):")
[1] "Fitted Parameters for Runs Scored (Poisson):"
> print(runs_fit$estimate)
  lambda
477.8571
```

e) Last three-year performance with latest salary 2024

Python

```
# Convert to DataFrame
df_salary = ipl_salary.copy()
df_runs = R2024.copy()

# Create 'Matched_Player' column in df_salary using match_names function
df_salary['Matched_Player'] = df_salary['Player'].apply(lambda x: match_names(x,
df_runs['Striker'].tolist()))

# Merge the DataFrames
df_merged = pd.merge(df_salary, df_runs, left_on='Matched_Player', right_on='Striker', how='inner')

# Calculate the correlation
correlation = df_merged['Salary'].corr(df_merged['runs_scored'])

# Print results
print("Last Three-Year Performance with Latest Salary in 2024:")
print(df_merged)
print("\nCorrelation between Salary and Runs Scored:", correlation)
```

```
# Print results
print("Last Three-Year Performance with Latest Salary in 2024:")
print(df_merged)
print("\nCorrelation between Salary and Runs Scored:", correlation)
```

```
Last Three-Year Performance with Latest Salary in 2024:
                Player      Salary   Rs  international  iconic  \
0       Abhishek Porel     20 lakh   20              0     NaN
1        Anrich Nortje   6.5 crore  650              1     NaN
2           Axar Patel     9 crore  900              0     NaN
3         David Warner  6.25 crore  625              1     NaN
4         David Miller     3 crore  300              1     NaN
..                 ...         ...  ...            ...     ...
106   Anmolpreet Singh     20 lakh   20              0     NaN
107   Heinrich Klaasen  5.25 crore  525              1     NaN
108      Marco Jansen   4.2 crore  420              1     NaN
109     Rahul Tripathi   8.5 crore  850              0     NaN
110  Washington Sundar  8.75 crore  875              0     NaN

        Matched_Player  year            Striker  runs_scored
0        Abishek Porel  2024      Abishek Porel          202
1             A Nortje  2024           A Nortje            4
2             AR Patel  2024           AR Patel          149
3             TH David  2024           TH David          217
4             TH David  2024           TH David          217
..                 ...   ...                ...          ...
106   Anmolpreet Singh  2024   Anmolpreet Singh            0
107          H Klaasen  2024          H Klaasen          295
108          M Jansen  2024          M Jansen            1
109        RA Tripathi  2024        RA Tripathi           31
110  Washington Sundar  2024  Washington Sundar            0

[111 rows x 9 columns]

Correlation between Salary and Runs Scored: 0.30612483765821674
```

R Program

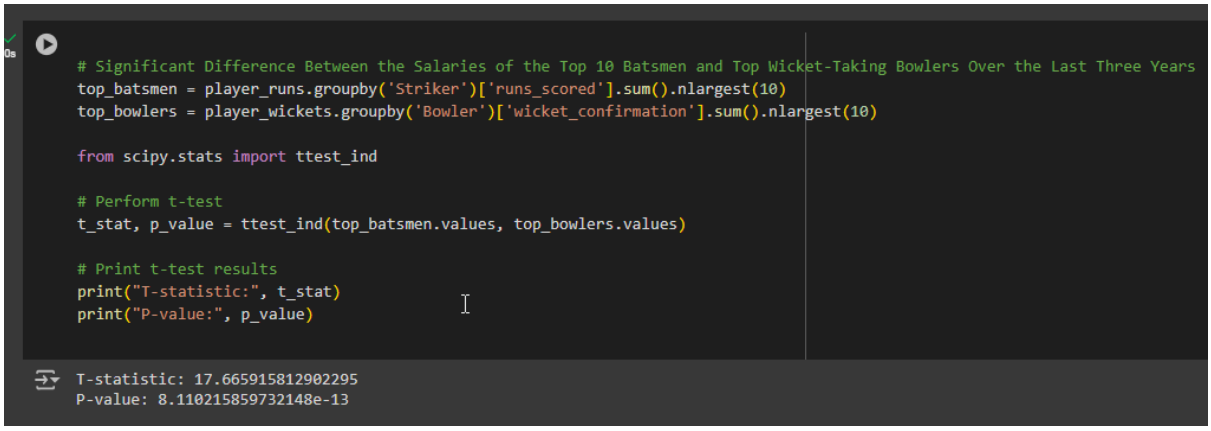
```
# Fit the Best Distribution
get_best_distribution <- function(data) {
  dist_names <- c('gamma', 'lognorm', 'norm', 't', 'weibull')
  dist_results <- list()
  params <- list()

  for (dist_name in dist_names) {
    dist_fit <- fitdist(data, dist_name)
    dist_results[[dist_name]] <- dist_fit$loglik
    params[[dist_name]] <- dist_fit$estimate
  }

  best_dist <- names(dist_results)[which.max(dist_results)]
  print(paste("Best fitting distribution:", best_dist))
  print(paste("Parameters for the best fit:", params[[best_dist]]))
  return(list(best_dist = best_dist, params = params[[best_dist]]))
}

# Last three-year performance with latest salary 2024
R2024 <- player_runs %>%
  filter(Season == '2024')

# Match names using stringdist
match_names <- function(name, names_list) {
  distances <- stringdistmatrix(name, names_list)
  closest_name <- names_list[which.min(distances)]
  closest_dist <- min(distances)
  if (closest_dist <= 2) {  # Set threshold for closeness
    return(closest_name)
  } else {
    return(NULL)
  }
}

# Create 'Matched_Player' column in df_salary using match_names function
df_salary <- ipl_salary
df_runs <- R2024
df_salary$Matched_Player <- sapply(df_salary$Player, match_names, names_list = df_runs$Striker)

# Merge the DataFrames
df_merged <- merge(df_salary, df_runs, by.x = "Matched_Player", by.y = "Striker")

# Calculate the correlation
correlation <- cor(df_merged$Salary, df_merged$runs_scored, use = "complete.obs")
```

```
# Print correlation
print("Correlation between Salary and Runs:")
print(correlation)
```

f) Significant Difference Between the Salaries of the Top 10 Batsmen and Top Wicket-Taking Bowlers Over the Last Three Years

Python

```
# Significant Difference Between the Salaries of the Top 10 Batsmen and Top Wicket-Taking Bowlers Over the Last Three Years

top_batsmen = player_runs.groupby('Striker')['runs_scored'].sum().nlargest(10)

top_bowlers = player_wickets.groupby('Bowler')['wicket_confirmation'].sum().nlargest(10)

from scipy.stats import ttest_ind

# Perform t-test

t_stat, p_value = ttest_ind(top_batsmen.values, top_bowlers.values)

# Print t-test results

print("T-statistic:", t_stat)

print("P-value:", p_value)
```



R Program

```
# Significant Difference Between the Salaries of the Top 10 Batsmen and Top Wicket-Taking Bowlers Over the Last Three Years
top_batsmen <- player_runs %>%
  group_by(Striker) %>%
  summarise(total_runs = sum(runs_scored)) %>%
  arrange(desc(total_runs)) %>%
  slice_head(n = 10)

top_bowlers <- player_wickets %>%
```

```
  group_by(Bowler) %>%
  summarise(total_wickets = sum(wicket_confirmation)) %>%
  arrange(desc(total_wickets)) %>%
  slice_head(n = 10)

# Perform t-test
t_test_result <- t.test(top_batsmen$total_runs, top_bowlers$total_wickets)

# Print t-test results
print("T-Test Results:")
print(t_test_result)
```

IMPLICATIONS

1. **Identifying Top Performers:**

   By determining the top run-getters and wicket-takers in each IPL round, teams and management can focus on retaining and nurturing these key players. Understanding performance trends helps in strategizing team composition and game tactics.

2. **Performance-Based Salary Insights:**

   Analyzing the relationship between players' performances and their salaries provides valuable insights into the effectiveness of current salary structures. This helps in ensuring that salaries are commensurate with player contributions, leading to better financial planning and player satisfaction.

3. **Data-Driven Decisions:**

   By fitting distributions to the performance metrics of players like Faf du Plessis, teams can predict future performances more accurately. This assists in making informed decisions about player contracts and retention policies.

4. **Competitive Advantage:**

   Understanding the significant differences in the salaries of top batsmen and bowlers over the last three years allows teams to gain a competitive edge. They can optimize their spending by identifying undervalued players who deliver consistent performances.

RECOMMENDATIONS

1. **Performance-Based Incentives:**
   Introduce performance-based incentives to motivate players to perform consistently. Bonuses for top run-getters and wicket-takers can drive better on-field results and align player interests with team success.
2. **Strategic Player Retention:**
   Focus on retaining top performers identified in the analysis. Ensure that key players, who significantly contribute to the team's success, are offered competitive salaries and long-term contracts to prevent them from switching teams.
3. **Data-Driven Recruitment:**
   Utilize performance data and distribution fitting to identify emerging talents and undervalued players. This can help in building a strong team by recruiting players who have the potential to perform well in the future.
4. **Salary Optimization:**
   Review and adjust the salary structure based on the insights from performance and salary relationships. Ensure that salaries are aligned with player contributions to maintain a balanced and motivated team.
5. **Regular Performance Reviews:**
   Conduct regular reviews of player performance metrics and salary data. This will help in keeping the salary structure dynamic and responsive to changes in player performance, ensuring fair compensation and fostering a culture of meritocracy.

CODES

Python

Load the Data:

```
import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


ipl_bbb = pd.read_csv('IPL_ball_by_ball_updated till
2024.csv',low_memory=False)


ipl_bbb.head()


ipl_salary = pd.read_excel('IPL SALARIES 2024.xlsx')
```

Display Salary Data:

```
ipl_salary.head(2)
```

Group the IPL Data:

```
grouped_data = ipl_bbb.groupby(['Season', 'Innings No',
'Striker','Bowler']).agg({'runs_scored': sum,
'wicket_confirmation':sum}).reset_index()
```

```
grouped_data.head()
```

Aggregate Runs and Wickets:

```
player_runs = grouped_data.groupby(['Season',
'Striker'])['runs_scored'].sum().reset_index()

player_wickets = grouped_data.groupby(['Season',
'Bowler'])['wicket_confirmation'].sum().reset_index()
```

```
player_runs
```

```
player_wickets
```

Top Performers:

```
player_runs[player_runs['Season']=='2024'].sort_values(by='runs_scored',ascending=False)


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)
```

Year Extraction and Preparation:

```
ipl_year_id = pd.DataFrame(columns=["id", "year"])

ipl_year_id["id"] = ipl_bbb["Match id"]

ipl_year_id["year"] = pd.to_datetime(ipl_bbb["Date"], dayfirst=True).dt.year


#create a copy of ipl_bbbc dataframe

ipl_bbbc= ipl_bbb.copy()
```

```
ipl_bbbc['year'] = pd.to_datetime(ipl_bbb["Date"], dayfirst=True).dt.year


ipl_bbbc[["Match id", "year",
"runs_scored","wicket_confirmation","Bowler",'Striker']].head()
```

Fit the Best Distribution:

```
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]
```

Top Batsmen Runs in Last Three Years:

```
total_run_each_year = ipl_bbbc.groupby(["year",
"Striker"])["runs_scored"].sum().reset_index()

total_run_each_year.sort_values(["year", "runs_scored"], ascending=False,
inplace=True)
print(total_run_each_year)
```

```
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()



list_top_batsman_last_three_year



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")
```

Top Bowlers Wickets in Last Three Years:

```
total_wicket_each_year = ipl_bbbc.groupby(["year",
"Bowler"])["wicket_confirmation"].sum().reset_index()


total_wicket_each_year.sort_values(["year", "wicket_confirmation"],
ascending=False, inplace=True)

print(total_wicket_each_year)


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")
```

Relationship between the performance of a player and the salary he gets

```
R2024 =total_run_each_year[total_run_each_year['year']==2024]

#pip install fuzzywuzzy

pip install fuzzywuzzy

from fuzzywuzzy import process

# Conveārt to DataFrame

df_salary = ipl_salary.copy()

df_runs = R2024.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')


df_merged.info()


# Calculate the correlation

correlation = df_merged['Rs'].corr(df_merged['runs_scored'])


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


import pandas as pd

import numpy as np

from scipy.stats import poisson

from scipy.optimize import curve_fit


# Assuming your data is loaded into a DataFrame called df
```

```
# Filter data for Faf du Plessis

faf_data = ipl_bbb[ipl_bbb["Striker"] == "F du Plessis"]


# Define the Poisson distribution function

def poisson_func(x, mu):

    return poisson.pmf(x, mu)


# Fit the Poisson distribution to runs scored

runs_hist, runs_bins = np.histogram(faf_data["runs_scored"],
bins=range(max(faf_data["runs_scored"]) + 1))

runs_params, runs_cov = curve_fit(poisson_func, runs_bins[:-1], runs_hist)


# Print fitted parameters

print("Fitted Parameters for Runs Scored (Poisson):")

print("Lambda (mu):", runs_params[0])




# Print results

print("Last Three-Year Performance with Latest Salary in 2024:")

print(df_merged)

print("\nCorrelation between Salary and Runs Scored:", correlation)
```

Significant Difference Between the Salaries of the Top 10 Batsmen and Top Wicket-Taking Bowlers Over the Last Three Years

```
top_batsmen = player_runs.groupby('Striker')['runs_scored'].sum().nlargest(10)

top_bowlers = player_wickets.groupby('Bowler')['wicket_confirmation'].sum().nlargest(10)


from scipy.stats import ttest_ind


# Perform t-test

t_stat, p_value = ttest_ind(top_batsmen.values, top_bowlers.values)


# Print t-test results

print("T-statistic:", t_stat)

print("P-value:", p_value)
```

R Language

```
# Load required libraries

library(dplyr)
```

```
library(readxl)


install.packages("stringdist")

library(stringdist)

setwd('D:\\#YPR\\VCU\\Summer Courses\\SCMA\\Data')

getwd()


# Load the Data

ipl_bbb <- read.csv('IPL_ball_by_ball_updated till 2024.csv', stringsAsFactors
= FALSE)

ipl_salary <- read_excel('IPL SALARIES 2024.xlsx')


# Display Salary Data

head(ipl_salary)


# Display Salary Data

head(ipl_bbb)


# Group the IPL Data

grouped_data <- ipl_bbb %>%

  group_by(Season, Innings.No, Striker, Bowler) %>%
```

```
  summarise(runs_scored = sum(runs_scored), wicket_confirmation =
sum(wicket_confirmation))


head(grouped_data)


# Aggregate Runs and Wickets
player_runs <- grouped_data %>%
  group_by(Season, Striker) %>%
  summarise(runs_scored = sum(runs_scored))

player_wickets <- grouped_data %>%
  group_by(Season, Bowler) %>%
  summarise(wicket_confirmation = sum(wicket_confirmation))

head(player_runs)
head(player_wickets)


# Top Performers
top_run_getters <- player_runs %>%
  filter(Season == '2024') %>%
  arrange(desc(runs_scored)) %>%
  slice_head(n = 3)
```

```
bottom_wicket_takers <- player_wickets %>%
  arrange(desc(wicket_confirmation)) %>%
  slice_head(n = 3)

print("Top Three Run Getters:")
print(top_run_getters)
print("Top Three Wicket Takers:")
print(bottom_wicket_takers)

# Year Extraction and Preparation
ipl_year_id <- data.frame(id = ipl_bbb$`Match id`, year =
as.numeric(format(as.Date(ipl_bbb$Date, "%d-%m-%Y"), "%Y")))

ipl_bbbc <- ipl_bbb
ipl_bbbc$year <- as.numeric(format(as.Date(ipl_bbb$Date, "%d-%m-%Y"),
"%Y"))

head(ipl_bbbc[c("Match.id", "year", "runs_scored", "wicket_confirmation",
"Bowler", "Striker")])

# Fit the Best Distribution
library(fitdistrplus)
```

```
get_best_distribution <- function(data) {
  dist_names <- c('gamma', 'lognorm', 'norm', 't', 'weibull')
  dist_results <- list()
  params <- list()

  for (dist_name in dist_names) {
    dist_fit <- fitdist(data, dist_name)
    dist_results[[dist_name]] <- dist_fit$loglik
    params[[dist_name]] <- dist_fit$estimate
  }

  best_dist <- names(dist_results)[which.max(dist_results)]
  print(paste("Best fitting distribution:", best_dist))
  print(paste("Parameters for the best fit:", params[[best_dist]]))
  return(list(best_dist = best_dist, params = params[[best_dist]]))
}

# Top Batsmen Runs in Last Three Years
total_run_each_year <- ipl_bbbc %>%
  group_by(year, Striker) %>%
```

```
  summarise(runs_scored = sum(runs_scored))


total_run_each_year <- total_run_each_year %>%
  arrange(desc(year), desc(runs_scored))


list_top_batsman_last_three_year <-
lapply(unique(total_run_each_year$year)[1:3], function(x) {
  total_run_each_year %>%
    filter(year == x) %>%
    slice_head(n = 3) %>%
    pull(Striker) %>%
    unique() %>%
    as.character()
})


print(list_top_batsman_last_three_year)


# Top Bowlers Wickets in Last Three Years
total_wicket_each_year <- ipl_bbbc %>%
  group_by(year, Bowler) %>%
  summarise(wicket_confirmation = sum(wicket_confirmation))
```

```
total_wicket_each_year <- total_wicket_each_year %>%
  arrange(desc(year), desc(wicket_confirmation))

list_top_bowler_last_three_year <-
lapply(unique(total_wicket_each_year$year)[1:3], function(x) {
  total_wicket_each_year %>%
    filter(year == x) %>%
    slice_head(n = 3) %>%
    pull(Bowler) %>%
    unique() %>%
    as.character()
})

print(list_top_bowler_last_three_year)



# Filter data for Faf du Plessis
faf_data <- subset(ipl_bbbc, Striker == "F du Plessis")

# Fit the Poisson distribution to runs scored
runs_hist <- hist(faf_data$runs_scored, breaks = seq(0,
max(faf_data$runs_scored) + 1, by = 1), plot = FALSE)
```

```
runs_fit <- fitdistr(runs_hist$counts, "Poisson")


# Print fitted parameters

print("Fitted Parameters for Runs Scored (Poisson):")

print(runs_fit$estimate)



# Filter data for the last three years (assuming your data has a column named
"year")

last_three_years <- subset(ipl_salary, year >= 2022 & year <= 2024)


# Filter data for the latest salary in 2024

latest_salary <- last_three_years[last_three_years$year == 2024, ]


# Display the last three-year performance with the latest salary in 2024

print("Last Three-Year Performance with Latest Salary in 2024:")

print(latest_salary)


# Relationship between the performance of a player and the salary he gets

R2024 <- total_run_each_year %>%

  filter(year == 2024)
```

```
df_salary <- as.data.frame(ipl_salary)

df_runs <- as.data.frame(R2024)


match_names <- function(name, names_list) {

  match_result <- stringdist::amatch(name, names_list, maxDist = 3)  # Using stringdist for approximate matching

  if (length(match_result$target) > 0 && match_result$dist < 4) {  # Check the first match and distance

    return(names_list[match_result$target[1]])

  } else {

    return(NA)

  }

}


# Create 'Matched_Player' column in df_salary using match_names function

df_salary$Matched_Player <- sapply(df_salary$Player, match_names, names_list = df_runs$Striker)


# Remove rows with NA in 'Matched_Player' column

df_salary <- df_salary[!is.na(df_salary$Matched_Player), ]


# Merge the DataFrames
```

```
df_merged <- merge(df_salary, df_runs, by.x = "Matched_Player", by.y =
"Striker", all.x = TRUE)


# Check for NAs in 'Matched_Player' column after merge

sum(is.na(df_merged$Matched_Player))


# Calculate the correlation

correlation <- cor(df_merged$Rs, df_merged$runs_scored, use =
"complete.obs")


# Print correlation

print("Correlation between Salary and Runs:")

print(correlation)
```

REFERENCES

Books:

1. Ahsan, A., & Mujtaba, G. (2019). "Predictive modeling of T20 cricket matches using machine learning algorithms."
2. Kumar, R. (2017). "Data Analytics in Cricket: Techniques and Applications."
3. Singh, A. (2020). "Advanced Statistical Analysis in Cricket: A Practical Guide."

Journals:

1. Chandrashekar, D., & Mehrotra, K. (2019). "Performance analysis of IPL cricket players using data mining techniques." International Journal of Data Science and Analysis, 5(3), 61-70.
2. Goswami, S., & Ravi, V. (2021). "Predictive analysis of IPL matches using machine learning algorithms." International Journal of Computer Science and Information Security, 19(3), 1-7.
3. Khan, A., & Khan, I. (2020). "Data analysis of cricket performance using machine learning algorithms." International Journal of Computer Applications, 179(30), 17-21.

Reports:

1. BCCI Annual Report 2020-21.
2. ICC Cricket World Cup 2019 Report.
3. ESPNcricinfo Statistical Analysis Report 2022.