VIRGINIA COMMONWEALTH UNIVERSITY

STATISTICAL ANALYSIS & MODELING

A1a: ANALYSIS OF IPL PLAYERS PERFORMANCE USING PYTHON AND R

SARATH S
V01109792

Date of Submission: 18/06/2024

CONTENTS

Analyzing IPL Players performance Using R and Python

INTRODUCTION

The Indian Premier League (IPL) is one of the most prominent and lucrative professional T20 cricket leagues globally, featuring elite players from around the world. The performance of players in the IPL is not only crucial for the success of their respective teams but also significantly impacts their market value and salary negotiations. This project aims to analyze the relationship between a player's on-field performance, specifically the runs scored by batsmen, and their salaries. Understanding this relationship can provide insights into how player performance influences their market value and can aid teams in making informed decisions during player auctions and contract renewals.

OBJECTIVES

a) Extract the files in R/Python
b) 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.
c) Fit the most appropriate distribution for runs scored and wickets taken by the top three batsmen and bowlers in the lost three IPL tournaments.
d) Find the relationship between a player’s performance and the salary he gets in your data.

BUSINESS SIGNIFICANCE

The business significance of this project lies in its potential impact on multiple stakeholders within the IPL ecosystem, including team owners, management, players, and sponsors. The findings from this analysis can be leveraged in the following ways:

1) Teams invest substantial amounts in acquiring and retaining players. By understanding the correlation between player performance and salary, teams can make more informed decisions regarding their investments, ensuring they get the best value for their money.

2) Players and their agents can use performance data to negotiate better contracts. Demonstrating a strong positive correlation between performance metrics and salary can justify higher pay for consistently high-performing players.
3) Accurate assessment of a player's market value based on performance metrics helps in setting appropriate base prices for auctions. This prevents overvaluation or undervaluation of players, ensuring a fair and competitive bidding process.
4) Teams can use the analysis to benchmark player performance against salary, identifying undervalued players who deliver high performance and could be potential targets for future auctions or trades.
5) Sponsors and endorsers can use performance-related salary data to identify key players who provide high visibility and value for money. This enhances the effectiveness of sponsorship deals and marketing campaigns.

RESULTS AND INTERPRETATION

a) Extract the files in R/Python

I performed this objective in both R and python. The codes are given below:

For Python,

```
os.chdir('C:\\Users\\SARATH\\OneDrive\\Desktop\\Bootcamp\\SCMA\\SCMA A1b')
```

In [3]:

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

In [4]:

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

b) 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:

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

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_runs['Season']=='2023'].sort_values(by='runs_scored',ascending=False)
```

	Season	Striker	runs_scored
2423	2023	Shubman Gill	890
2313	2023	F du Plessis	730
2311	2023	DP Conway	672
2433	2023	V Kohli	639
2443	2023	YBK Jaiswal	625
...	...	...	...
2404	2023	RP Meredith	0
2372	2023	Mohsin Khan	0
2307	2023	DG Nalkande	0
2429	2023	TU Deshpande	0
2324	2023	Harshit Rana	0

```
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
9	2011	CH Gayle	608
10	2011	V Kohli	557
11	2011	SR Tendulkar	553
12	2012	CH Gayle	733
13	2012	G Gambhir	590
14	2012	S Dhawan	569
15	2013	MEK Hussey	733
16	2013	CH Gayle	720
17	2013	V Kohli	639
18	2014	RV Uthappa	660
19	2014	DR Smith	566
20	2014	GJ Maxwell	552
21	2015	DA Warner	562
22	2015	AM Rahane	540
23	2015	LMP Simmons	540
24	2016	V Kohli	973
25	2016	DA Warner	848
26	2016	AB de Villiers	687
27	2017	DA Warner	641
28	2017	G Gambhir	498
29	2017	S Dhawan	479
30	2018	KS Williamson	735
31	2018	RR Pant	684
32	2018	KL Rahul	659
33	2019	DA Warner	692
34	2019	KL Rahul	593
35	2019	Q de Kock	529
36	2020/21	KL Rahul	676
37	2020/21	S Dhawan	618
38	2020/21	DA Warner	548
39	2021	RD Gaikwad	635
40	2021	F du Plessis	633
41	2021	KL Rahul	626
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
9	2011	SL Malinga	30
10	2011	MM Patel	22
11	2011	S Aravind	22
12	2012	M Morkel	30
13	2012	SP Narine	29
14	2012	SL Malinga	25
15	2013	DJ Bravo	34
16	2013	JP Faulkner	33
17	2013	R Vinay Kumar	27
18	2014	MM Sharma	26
19	2014	SP Narine	22
20	2014	B Kumar	21
21	2015	DJ Bravo	28
22	2015	SL Malinga	26
23	2015	A Nehra	25
24	2016	B Kumar	24
25	2016	SR Watson	23
26	2016	YS Chahal	22
27	2017	B Kumar	28
28	2017	JD Unadkat	27
29	2017	JJ Bumrah	23
30	2018	AJ Tye	28
31	2018	S Kaul	24
32	2018	Rashid Khan	23
33	2019	K Rabada	29
34	2019	Imran Tahir	26
35	2019	JJ Bumrah	23
36	2020/21	K Rabada	32
37	2020/21	JJ Bumrah	30
38	2020/21	TA Boult	26
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 showcases the top three batsmen by runs scored and the top three bowlers by wickets taken for each cricket season from 2007/08 to 2024. It reveals a fluctuation in total runs scored across seasons, with some years, such as 2023 and 2022, standing out for their high totals. Similarly, the number of wickets taken by bowlers varies by season, with certain years having higher counts. Notably, players like JC Buttler, Shubman Gill, HV Patel, and YS Chahal consistently rank among the top performers across multiple seasons, highlighting their sustained excellence.

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

For batsman:
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")
```

```
************************
year: 2024  Batsman: RD Gaikwad
p value for alpha = 2.599259711013304e-20
p value for beta = 0.02041902689492403
p value for betaprime = 0.019503763598668566
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 = 0.015331622187826577
p value for gumbel_l = 0.05546236480086586
p value for johnsonsb = 4.646964117947127e-13
p value for kappa4 = 0.006363220770325362
p value for lognorm = 1.1719355665219537e-16
p value for nct = 0.5881570496217807
p value for norm = 0.24953651809309751
p value for norminvgauss = 0.5538573365184996
p value for powernorm = 0.1788753268739086
p value for rice = 0.18287532184336575
p value for recipinvgauss = 0.06459275668874309
p value for t = 0.2494021485911212
p value for trapz = 7.476391685388162e-13
p value for truncnorm = 0.24173236832621992

Best fitting distribution: nct
Best p value: 0.5881570496217807
Parameters for the best fit: (5.718048022849898, 9.399490726283615, -54.25277343780452, 8.497060
689079994)
```

For bowler:

Code:

```
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")
```

```
year: 2024  Bowler: HV Patel
p value for alpha = 0.0002993252328930706
p value for beta = 2.777571908776589e-19
p value for betaprime = 1.7052883875145053e-30
p value for burr12 = 5.427998338605459e-15
p value for crystalball = 1.1109118198587684e-05
p value for dgamma = 4.375428528574276e-05
p value for dweibull = 1.8553295107771936e-05
p value for erlang = 5.473635282991912e-24
p value for exponnorm = 0.0002813279943461815
p value for f = 1.9012983291282487e-09
p value for fatiguelife = 1.9734428958773156e-05
p value for gamma = 1.470787431589663e-16
p value for gengamma = 1.4345058849022962e-16
p value for gumbel_l = 4.541523588271283e-05
p value for johnsonsb = 2.827201329331457e-51
p value for kappa4 = 9.177530010006471e-23
p value for lognorm = 5.2162358572043325e-22
p value for nct = 0.0001960277304576293
p value for norm = 1.1109124960635979e-05
p value for norminvgauss = 3.811196478020768e-05
p value for powernorm = 3.2186417463058256e-05
p value for rice = 3.354567282896991e-05
p value for recipinvgauss = 5.05058721389515e-12
p value for t = 9.451105792399515e-05
p value for trapz = 1.0447243016629734e-51
p value for truncnorm = 0.0002182292327632623

Best fitting distribution: alpha
Best p value: 0.0002993252328930706
Parameters for the best fit: (5.200800514990576, -4.106246473111661, 27.580368990504883)
```

Interpretation:

The code includes a function, get_best_distribution, which is used to find the best-fitting distribution for the run data of each top batsman. This function is called for each of the top 3 wicket-takers in the last three IPL seasons. For HV Patel's performance data in 2024, the alpha distribution fits best among the tested distributions, although the relatively low p-value of 0.002099352328397306 indicates the fit might not be perfect. Nonetheless, it is the best among the options. The code effectively identifies the best-fitting statistical distributions for cricketers' perfo rmance data. For HV Patel, the alpha distribution is the best fit, while the nct distribution fits RD Gaikwad's perf ormance data best. The provided parameters can be used for further analysis or simulation of the players'

For the player allotted to me – HH Pandya

Code:

```
list_top_bowler_last_three_year = {}

# Loop through the unique years in the dataset, limited to the last three years
for i in total_wicket_each_year["year"].unique()[:3]:
    # Filter the dataset to include only records for HH Pandya in the current year
    HH_Pandya_data = total_wicket_each_year[(total_wicket_each_year["year"] == i) &
(total_wicket_each_year["Bowler"] == "HH Pandya")]
    # Get the unique list of years where HH Pandya appears in the filtered dataset
    list_top_bowler_last_three_year[i] = HH_Pandya_data["Bowler"].unique().tolist()

# Print the dictionary to verify the results
print(list_top_bowler_last_three_year)
{2024: ['HH Pandya'], 2023: ['HH Pandya'], 2022: ['HH Pandya']}
import warnings
warnings.filterwarnings('ignore')

# Group by Bowler and Match id, then sum the wickets
wickets = ipl_bbbc.groupby(['Bowler', 'Match id'])[['wicket_confirmation']].sum().reset_index()

# Loop through the dictionary to process HH Pandya's data for each year
for year, bowlers in list_top_bowler_last_three_year.items():
    for bowler in bowlers:
        if bowler == "HH Pandya":
            print("**************************")
            print("year:", year, " Bowler:", bowler)
            get_best_distribution(wickets[wickets["Bowler"] == bowler]["wicket_confirmation"])
            print("\n\n")
```

Result:

```
**************************
year: 2024  Bowler: HH Pandya
p value for alpha = 1.2068599860469232e-19
p value for beta = 1.2776165313170078e-22
p value for betaprime = 1.2776165324843997e-22
p value for burr12 = 1.2776165313170078e-22
p value for crystalball = 7.706458562309559e-08
```

p value for dgamma = 1.2236677905852817e-07
p value for dweibull = 3.0592386377377603e-06
p value for erlang = 1.2776165316086073e-22
p value for exponnorm = 1.612218423545063e-22
p value for f = 1.2776165313170078e-22
p value for fatiguelife = 1.4191464463856414e-21
p value for gamma = 1.2776165635218225e-22
p value for gengamma = 1.2972910342446806e-22
p value for gumbel_l = 2.429715488057867e-06
p value for johnsonsb = 1.2776165317693289e-22
p value for kappa4 = 1.27761653131764e-22
p value for lognorm = 2.970838205591996e-10
p value for nct = 1.3261890198333523e-19
p value for norm = 7.706458714257828e-08
p value for norminvgauss = 0.0
p value for powernorm = 3.31885050308089e-11
p value for rice = 3.7749230647091134e-11
p value for recipinvgauss = 1.2776251314946701e-22
p value for t = 1.2068599861496645e-19
p value for trapz = 6.060113302928463e-44
p value for truncnorm = 1.283705488374379e-22

Best fitting distribution: dweibull
Best p value: 3.0592386377377603e-06
Parameters for the best fit: (1.622973609465867, 0.6200805596021999, 0.8370434367442225)

Interpretation:

The above mentioned objective is to analyze the performance data of HH Pandya, both as a bowler and a batsman, over the last three years in IPL matches. For each year (2022, 2023, 2024), The best fitting d istribution consistently appears to be dweibull for wickets taken, with low p-values indicating statistical significance. The parameters of the best fit distribution (dweibull) provide insights into the typical perf ormance profile of HH Pandya in terms of wicket-taking ability across these years. Specifically, parame ters such as shape, location, and scale from the distribution fitting can give further insights into how con sistently HH Pandya performs as a wicket-taker.

Code:

```
list_top_batsman_last_three_year = {}

# Loop through the unique years in the dataset, limited to the last three years
for i in total_run_each_year["year"].unique()[:3]:
    # Filter the dataset to include only records for HH Pandya in the current year
    HH_Pandya_data = total_run_each_year[(total_run_each_year["year"] == i) &
(total_run_each_year["Striker"] == "HH Pandya")]
    # Get the unique list of years where HH Pandya appears in the filtered dataset
    list_top_batsman_last_three_year[i] = HH_Pandya_data["Striker"].unique().tolist()

# Print the dictionary to verify the results
print(list_top_batsman_last_three_year)
```

{2024: ['HH Pandya'], 2023: ['HH Pandya'], 2022: ['HH Pandya']}

```
import warnings
warnings.filterwarnings('ignore')

# Group by Batsman and Match id, then sum the wickets
wickets = ipl_bbbc.groupby(['Striker', 'Match id'])[['runs_scored']].sum().reset_index()

# Loop through the dictionary to process HH Pandya's data for each year
for year, strikers in list_top_batsman_last_three_year.items():
    for striker in strikers:
        if striker == "HH Pandya":
            print("**************************")
            print("year:", year, "batsman:", striker)
            get_best_distribution(runs[runs["Striker"] ==striker]["runs_scored"])
            print("\n\n")
```

Result:

```
**************************
year: 2024 batsman: HH Pandya
p value for alpha = 0.48083914965738106
p value for beta = 0.09725713558173044
p value for betaprime = 0.039515065298787344
p value for burr12 = 0.5218654996507346
p value for crystalball = 0.02502779105119146
p value for dgamma = 0.0371294565219874
p value for dweibull = 0.05137720758707631
p value for erlang = 0.007836274661668585
p value for exponnorm = 0.4437275682767443
p value for f = 9.384187705838297e-22
p value for fatiguelife = 0.3489676173122723
p value for gamma = 2.2938411686539194e-07
p value for gengamma = 5.3648900911177865e-06
p value for gumbel_l = 4.2272400444582196e-07
p value for johnsonsb = 0.5117357049308298
p value for kappa4 = 1.1044405047140013e-23
p value for lognorm = 6.9879684818504395e-34
p value for nct = 0.4041829427911746
p value for norm = 0.025027790041132758
p value for norminvgauss = 0.36139679343150344
p value for powernorm = 0.029549551861488
p value for rice = 0.03077821873090203
p value for recipinvgauss = 0.30848846184424883
p value for t = 0.023819875902041376
p value for trapz = 3.338856440448023e-63
p value for truncnorm = 0.19160129507989054

Best fitting distribution: burr12
Best p value: 0.5218654996507346
Parameters for the best fit: (1.0448465978155772, 334876.6991112063, -0.047685748895057625, 392
2583.832065367)
```

Interpretation:

The best fitting distribution appears to be burr12 for runs scored, with associated p-values indicating the goodness of fit. The parameters of the best fit distribution (burr12) offer insights into HH Pandya's typical scoring pattern across these years, including aspects like the shape and scale of the distribution.

To conclude, HH Pandya shows a consistent performance pattern both as a bowler (wickets taken) and as a batsman (runs scored) across the analyzed years. These analyses can help in understanding HH Pandya's strengths and weaknesses, and how these may have evolved or remained consistent over the last three IPL seasons.

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

Code:

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

#pip install fuzzywuzzy
from fuzzywuzzy import process

# Convert 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 >= 85 else None  # Use a threshold score of 85

# 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()
```

Result:

```
<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
```

```
# 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: 0.29009065196461536

Interpretation:

The correlation coefficient between salary (`Rs`) and runs scored in the year 2024 for players in our dataset is approximately 0.2900. To combine player salary data with their performance data (runs scored) for the year 2024, we had to use fuzzy string matching which uses Levenshtein Distance to calculate the differences between sequences. to match player names between the two datasets. Uses the fuzzywuzzy library to match player names from the salary data to the player names in the run data. Since the accuracy of the fuzzy score or threshold limit would directly impact the correlation, I have kept a high score of 85 to ensure there are no soft matches in the merged data frame. The value of 0.2900 indicates a moderate positive correlation between salary and runs scored. While not a very strong correlation, it suggests that, generally, players who earn higher salaries tend to score more runs.