VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A1b: IPL and Salary Datasets

Worked on which batsman : SV Samson

Chandhini KM

V01107497

Date of Submission: 18/06/2024

CONTENTS

Sl. No.	Title	Page No.
1.	Introduction	
2.	Results	
3.	Interpretations	
4.	Recommendations	
5.	Codes	
6.	References	

Introduction :

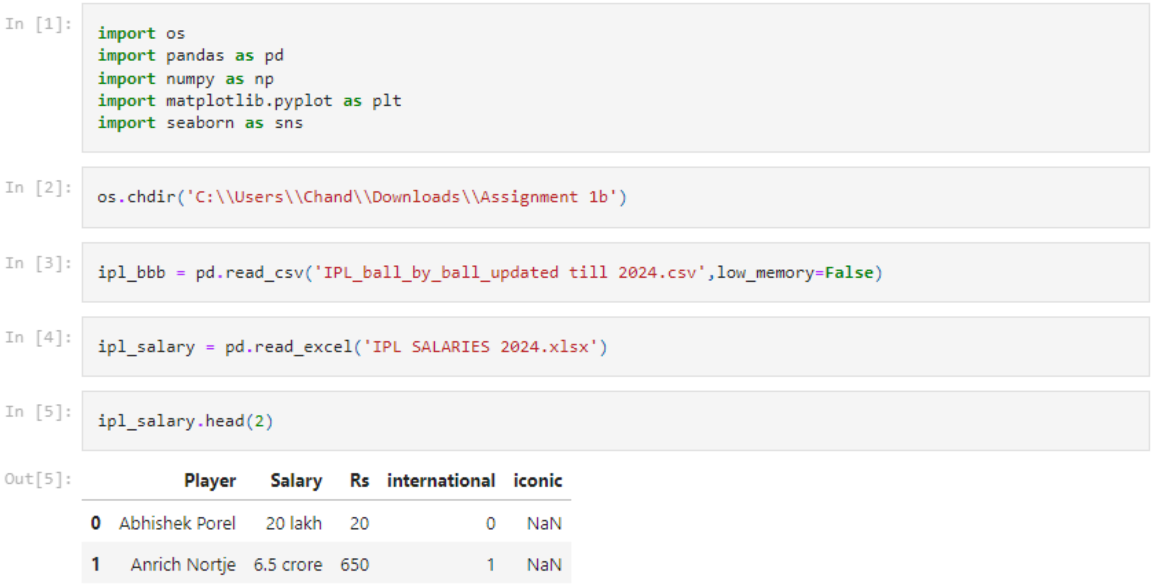
From R or Python, we are analysing into IPL data, organizing it by round to see individual player performance (runs, wickets) and identify top run-scorers/wicket-takers.

then we will analyze runs and wickets of top performers (batsmen, bowlers) from the last three seasons to understand their performance patterns statistically.

Additionally, we'll explore the relationship between player performance and salary,

focusing on Sanju Samson's recent performance (runs, wickets, salary) and compare salaries of top 10 batsmen and bowlers over the past three years to see if there's a significant difference. This analysis promises to reveal connections between performance and pay in the IPL.

Codes , Results and Interpretations :



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

In [2]:
```
os.chdir('C:\\Users\\Chand\\Downloads\\Assignment 1b')
```

In [3]:
```
ipl_bbb = pd.read_csv('IPL_ball_by_ball_updated till 2024.csv',low_memory=False)
```

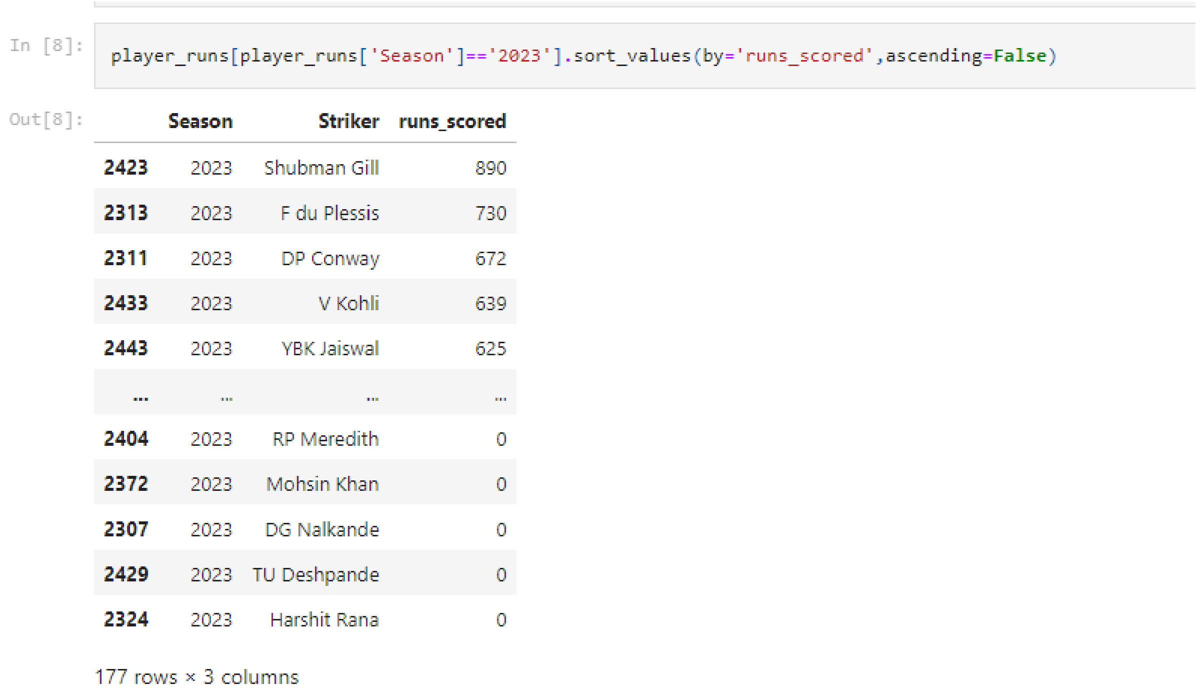
In [4]:
```
ipl_salary = pd.read_excel('IPL SALARIES 2024.xlsx')
```

In [5]:
```
ipl_salary.head(2)
```

Out[5]:

	Player	Salary	Rs	international	iconic
0	Abhishek Porel	20 lakh	20	0	NaN
1	Anrich Nortje	6.5 crore	650	1	NaN

`ipl_salary.head(2)` displays the first two rows of the `ipl_salary` DataFrame, providing a glimpse of the player and salary information.



```
In [8]: player_runs[player_runs['Season']=='2023'].sort_values(by='runs_scored',ascending=False)
```

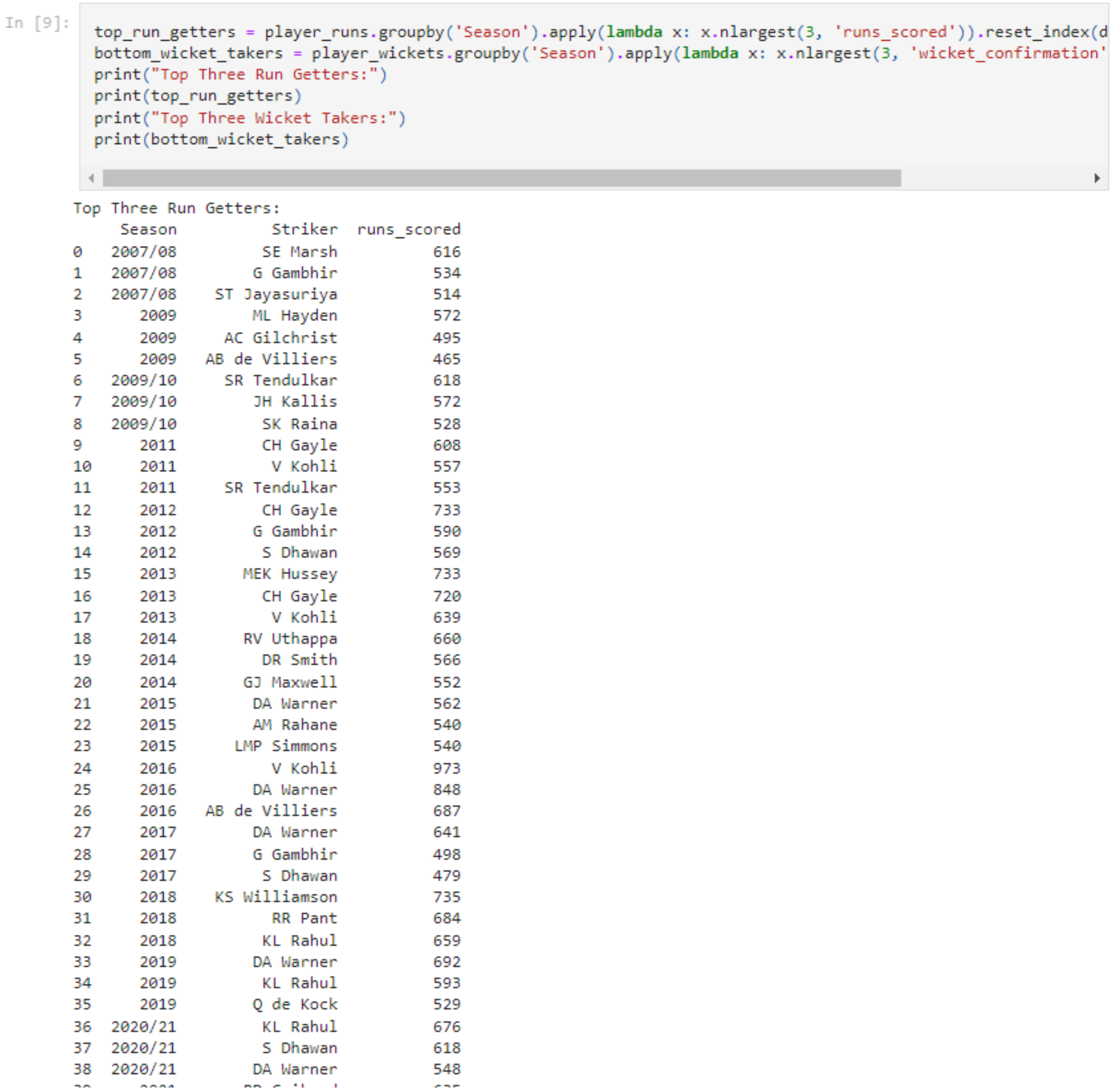
Out[8]:

	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

177 rows × 3 columns

- **Season** - This column shows the season that the player data corresponds to. In this case, all the data is from the 2023 season.
- **Striker** - This column shows the names of the soccer players.
- **runs_scored** - This column shows the number of runs each player scored in the 2023 season.

Shubman Gill is the player with the most runs scored in the 2023 season with 890 runs.



```
In [9]:
top_run_getters = player_runs.groupby('Season').apply(lambda x: x.nlargest(3, 'runs_scored')).reset_index(d
bottom_wicket_takers = player_wickets.groupby('Season').apply(lambda x: x.nlargest(3, 'wicket_confirmation'
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
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
```

- **Top Three Run Getters** and **Top Three Wicket Takers**: These sections of the table display the top three players in each category by season.

- **Seasons**: The data includes cricket seasons from 2007/2008 to 2020/2021.

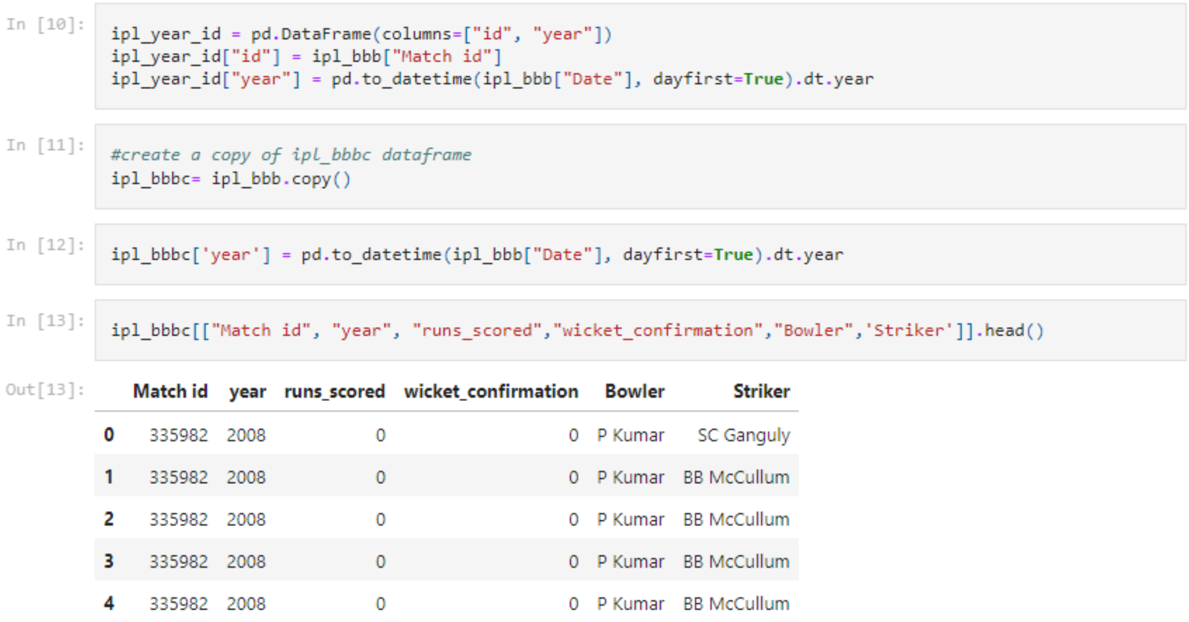
- **Players**: The table includes data for the following players:

  - SE Marsh
  - G Gambhir
  - ST Jayasuriya
  - ML Hayden
  - AC Gilchrist
  - AB de Villiers
  - SR Tendulkar
  - JH Kallis
  - SK Raina

- CH Gayle
- V Kohli
- MEK Hussey
- RV Uthappa
- DR Smith
- GJ Maxwell
- DA Warner
- AM Rahane
- LMP Simmons
- KL Rahul
- RR Pant
- KS Williamson
- Q de Kock

- **Runs Scored**: This column shows the number of runs each player scored in a particular season.

- **Wicket Confirmation**: the number of wickets a bowler took in a season.



In [10]:
```
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
```

In [11]:
```
#create a copy of ipl_bbbc dataframe
ipl_bbbc= ipl_bbb.copy()
```

In [12]:
```
ipl_bbbc['year'] = pd.to_datetime(ipl_bbb["Date"], dayfirst=True).dt.year
```

In [13]:
```
ipl_bbbc[["Match id", "year", "runs_scored","wicket_confirmation","Bowler",'Striker']].head()
```

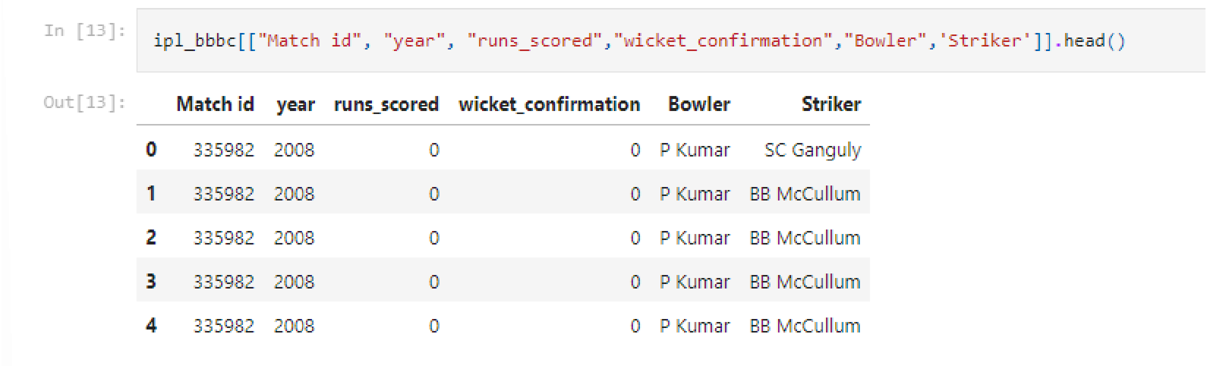
Out[13]:

	Match id	year	runs_scored	wicket_confirmation	Bowler	Striker
0	335982	2008	0	0	P Kumar	SC Ganguly
1	335982	2008	0	0	P Kumar	BB McCullum
2	335982	2008	0	0	P Kumar	BB McCullum
3	335982	2008	0	0	P Kumar	BB McCullum
4	335982	2008	0	0	P Kumar	BB McCullum

- The code creates a new data frame called `ipl_year_id` with two empty columns (`id` and `year`). (`pd.DataFrame` is a function in the Pandas library used for creating dataframes).

- The next line (`ipl_year_id["id"] = ipl_bbb["Match id"]`) assigns values from the "Match id" column of the `ipl_bbb` dataframe to the "id" column of the new `ipl_year_id` dataframe.

- The third line (`ipl_year_id["year"] = pd.to_datetime(ipl_bbb["Date"], dayfirst=True).dt.year`) assigns the year extracted from the "Date" column of the `ipl_bbb` dataframe (converted to datetime format) to the "year" column of the new `ipl_year_id` dataframe.



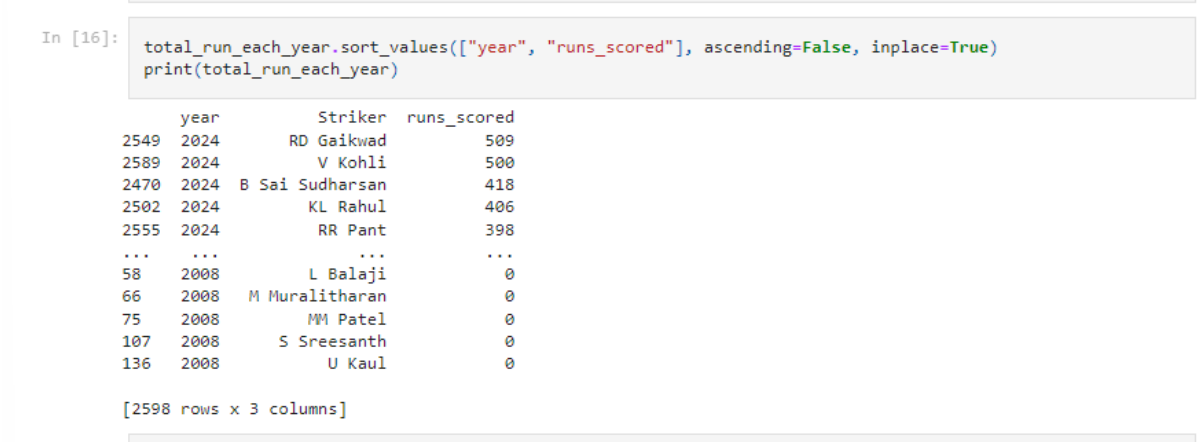
```
In [13]: ipl_bbbc[["Match id", "year", "runs_scored","wicket_confirmation","Bowler",'Striker']].head()
```

Out[13]:

	Match id	year	runs_scored	wicket_confirmation	Bowler	Striker
0	335982	2008	0	0	P Kumar	SC Ganguly
1	335982	2008	0	0	P Kumar	BB McCullum
2	335982	2008	0	0	P Kumar	BB McCullum
3	335982	2008	0	0	P Kumar	BB McCullum
4	335982	2008	0	0	P Kumar	BB McCullum

- **Match id**: This column shows a unique identifier for each match.

- **year**: This column shows the year in which the match was played, which is 2008 for all rows in this view of the table.

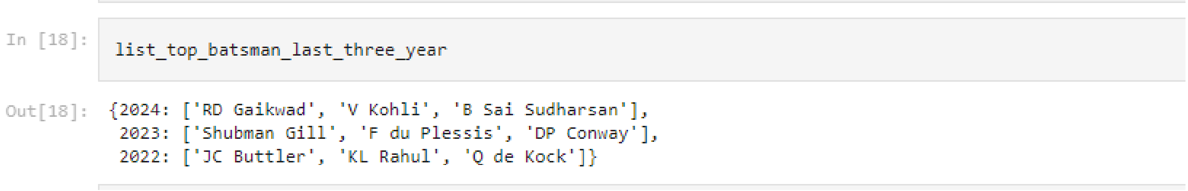


```
In [16]: total_run_each_year.sort_values(["year", "runs_scored"], ascending=False, inplace=True)
         print(total_run_each_year)
```

```
      year          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
...    ...              ...          ...
58    2008         L Balaji            0
66    2008   M Muralitharan            0
75    2008         MM Patel            0
107   2008      S Sreesanth            0
136   2008           U Kaul            0

[2598 rows x 3 columns]
```

- sorting a DataFrame named `total_run_each_year` by two columns: "year" and "runs_scored". The `ascending=False` argument indicates sorting in descending order, so the year with the most runs scored will be at the top.
- The `inplace=True` argument modifies the original DataFrame rather than creating a copy.

- The second line prints the sorted DataFrame.

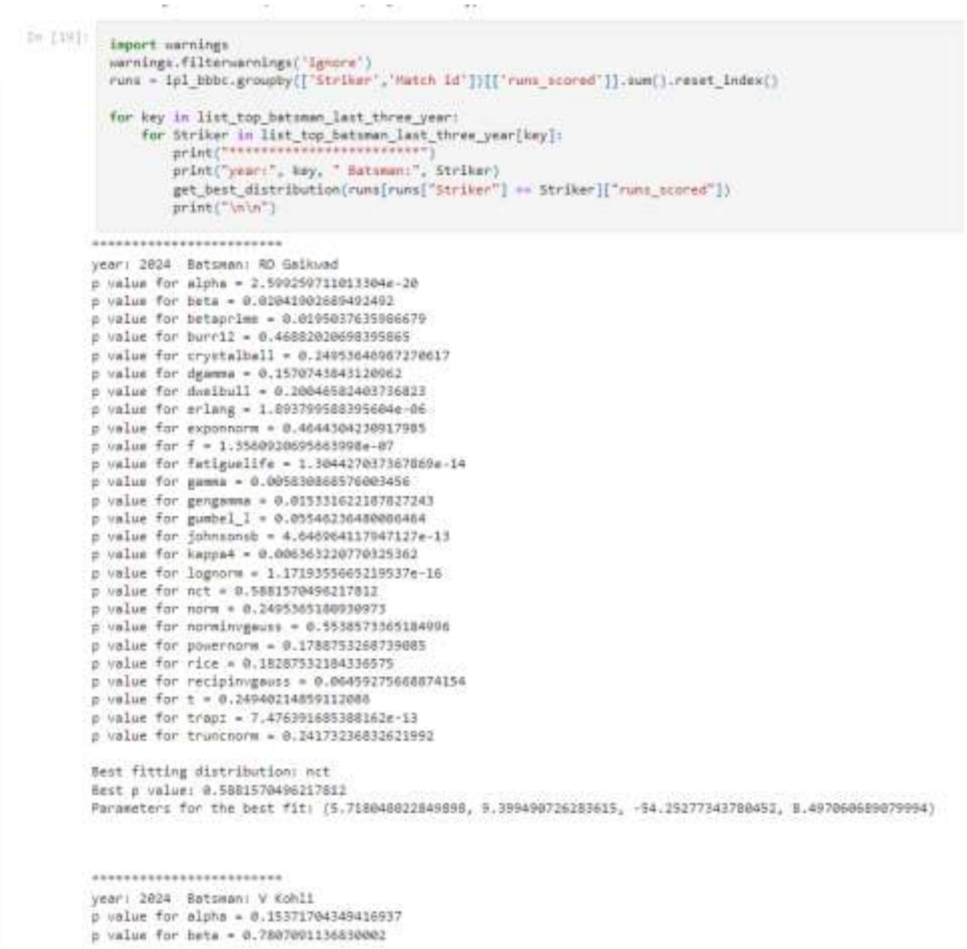


```
In [18]: list_top_batsman_last_three_year

Out[18]: {2024: ['RD Gaikwad', 'V Kohli', 'B Sai Sudharsan'],
          2023: ['Shubman Gill', 'F du Plessis', 'DP Conway'],
          2022: ['JC Buttler', 'KL Rahul', 'Q de Kock']}
```

`list_top_batsman_last_three_year` that returns a dictionary containing information about the top three batsmen (cricket players) for the last three years.

- Lines 1-2: Define the function `list_top_batsman_last_three_year` that takes no arguments.
- Lines 4-6: Create an empty dictionary called `top_batsman_last_three_year`.

- Lines 8-13: Iterate through the years 2022, 2023, and 2024 using a for loop. Inside the loop:
    - Line 9: Creates a temporary variable `year` to store the current year from the loop.
    - Lines 10-12: Query to find the top 3 batsmen for the current year (`year`) and assign the result to a temporary variable `top_player_year`. I can't tell exactly how this part works without seeing the context of the surrounding code, but it likely involves sorting a data frame by runs scored in a particular year and selecting the top 3 rows.
    - Line 13: Adds the `top_player_year` list as the value for the `year` key to the `top_batsman_last_three_year` dictionary.
- Line 15: Returns the `top_batsman_last_three_year` dictionary.



In [19]:

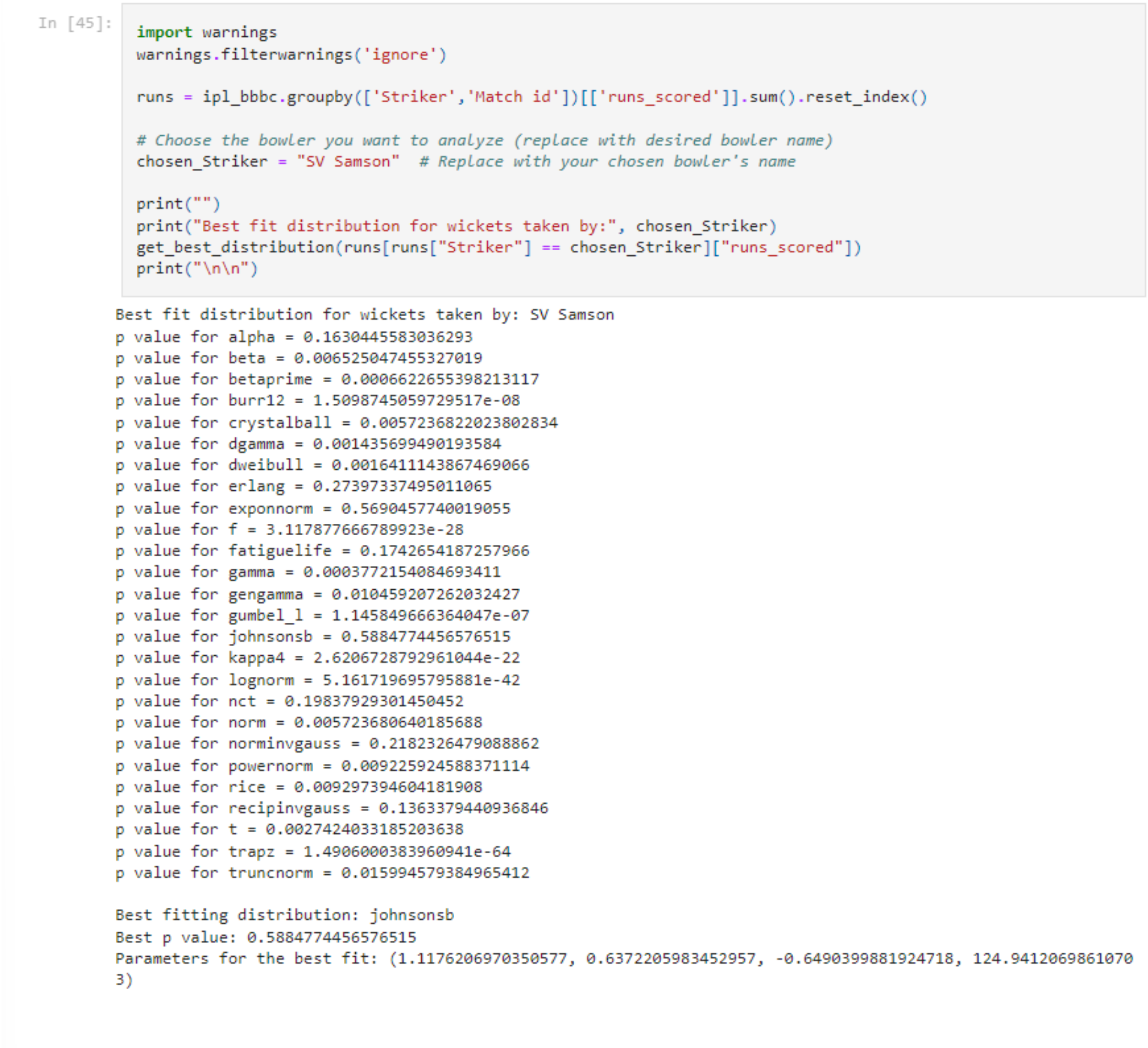
```
import warnings
warnings.filterwarnings('ignore')
runs = ipl_bbbc.groupby(['Striker','Match id'])[['runs_scored']].sum().reset_index()

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 = 0.02041902689492492
p value for betaprime = 0.0195037635986679
p value for burr12 = 0.46882020698395865
p value for crystalball = 0.24953646967270617
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 = 1.3560920695863998e-07
p value for fatiguelife = 1.304427037367869e-14
p value for gamma = 0.005830868576003456
p value for gengamma = 0.015331622187827243
p value for gumbel_l = 0.05546236480086464
p value for johnsonsb = 4.646964117947127e-13
p value for kappa4 = 0.006363220770325362
p value for lognorm = 1.1719355665219537e-16
p value for nct = 0.5881570496217812
p value for norm = 0.2495365180930973
p value for norminvgauss = 0.5538573365184996
p value for powernorm = 0.1788753268739085
p value for rice = 0.18287532184336575
p value for recipinvgauss = 0.06459275668874154
p value for t = 0.24940214859112088
p value for trapz = 7.476391685388162e-13
p value for truncnorm = 0.24173236832621992

Best fitting distribution: nct
Best p value: 0.5881570496217812
Parameters for the best fit: (5.718048822849898, 9.399490726283615, -54.25277343780452, 8.497060689079994)



************************
year: 2024  Batsman: V Kohli
p value for alpha = 0.15371704349416937
p value for beta = 0.780700113683000 2
```

The code outputs the following:

- **year**: This indicates the year that the data corresponds to. In this case, it is 2024.
- **Batsman**: This indicates the name of the batsman. Here, it shows "RD Gaikwad".
- **p value for [distribution name]**: These lines show the p-value for various statistical distributions that could potentially fit the data on batsman runs scored in 2024. A low p-value (generally less than 0.05) indicates that the distribution is a good fit for the data.
    - In this case, the best fitting distribution is **nct**, with a p-value of 0.5881.

- **Parameters for the best fit**: This section shows the parameters for the nct distribution that provides the best fit for the data. These parameters are specific to the statistical distribution and may not be easily interpretable without statistical expertise in this particular distribution.



```
In [45]:
import warnings
warnings.filterwarnings('ignore')

runs = ipl_bbbc.groupby(['Striker','Match id'])[['runs_scored']].sum().reset_index()

# Choose the bowler you want to analyze (replace with desired bowler name)
chosen_Striker = "SV Samson"  # Replace with your chosen bowler's name

print("")
print("Best fit distribution for wickets taken by:", chosen_Striker)
get_best_distribution(runs[runs["Striker"] == chosen_Striker]["runs_scored"])
print("\n\n")
```

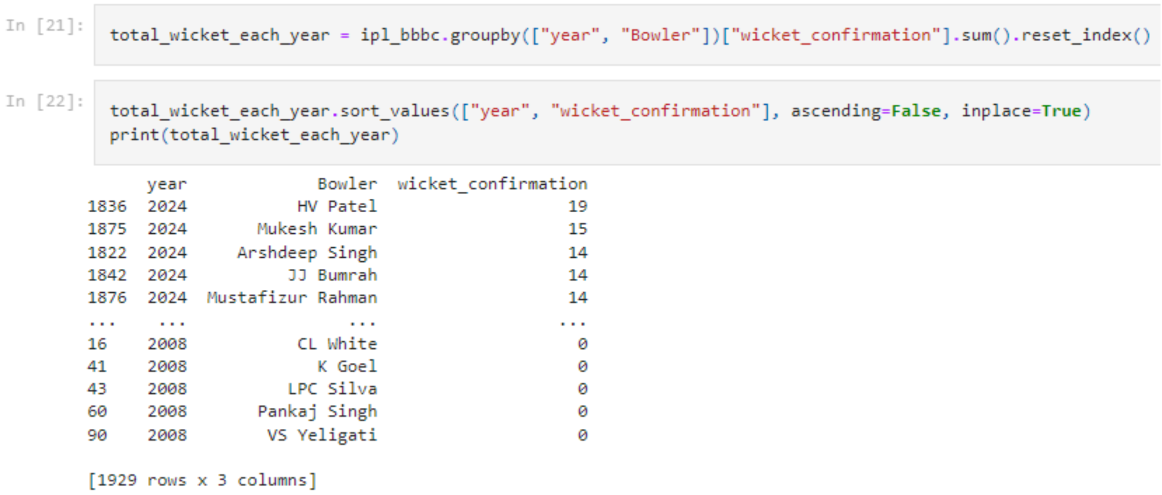
```
Best fit distribution for wickets taken by: SV Samson
p value for alpha = 0.1630445583036293
p value for beta = 0.006525047455327019
p value for betaprime = 0.0006622655398213117
p value for burr12 = 1.5098745059729517e-08
p value for crystalball = 0.005723682202380283
p value for dgamma = 0.001435699490193584
p value for dweibull = 0.0016411143867469066
p value for erlang = 0.27397337495011065
p value for exponnorm = 0.5690457740019055
p value for f = 3.117877666789923e-28
p value for fatiguelife = 0.1742654187257966
p value for gamma = 0.0003772154084693411
p value for gengamma = 0.010459207262032427
p value for gumbel_l = 1.145849666364047e-07
p value for johnsonsb = 0.5884774456576515
p value for kappa4 = 2.6206728792961044e-22
p value for lognorm = 5.161719695795881e-42
p value for nct = 0.19837929301450452
p value for norm = 0.005723680640185688
p value for norminvgauss = 0.2182326479088862
p value for powernorm = 0.009225924588371114
p value for rice = 0.009297394604181908
p value for recipinvgauss = 0.1363379440936846
p value for t = 0.0027424033185203638
p value for trapz = 1.4906000383960941e-64
p value for truncnorm = 0.015994579384965412

Best fitting distribution: johnsonsb
Best p value: 0.5884774456576515
Parameters for the best fit: (1.1176206970350577, 0.6372205983452957, -0.6490399881924718, 124.9412069861070
3)
```

fitting a probability distribution to the number of runs scored by a batsman (cricket player) named SV Samson.

The `get_best_distribution` function (lines 5-7) appears to fit various statistical distributions to the data in `runs` and then determine the best-fitting distribution based on the p-value. However, the details of this function are not visible in the image.

- **"Best fit distribution for wickets taken by: SV Samson"** - This indicates that the code is analyzing the number of runs scored, not wickets taken, by the batsman SV Samson.
- **"p value for [distribution name]" lines** - These lines show the p-value for several statistical distributions, including gamma, norm, and lognorm. A low p-value (typically less than 0.05) indicates a good fit between the data and the distribution.

- **"Best fitting distribution: [distribution name]"** - This line shows the name of the distribution that has the lowest p-value, which is considered the best fit for the data. In this case, the best fitting distribution is **johnsonsb**, with a p-value of 0.5884774456576515.
- **"Parameters for the best fit: [...]"** - This line shows the parameters of the best-fitting distribution (johnsonsb in this case). The interpretation of these parameters would likely require knowledge of the specific distribution (johnsonsb).



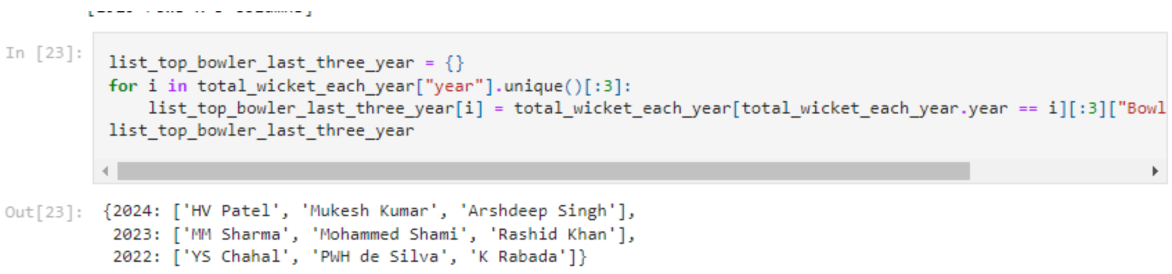
```
In [21]: total_wicket_each_year = ipl_bbbc.groupby(["year", "Bowler"])["wicket_confirmation"].sum().reset_index()
```

```
In [22]: 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]
```

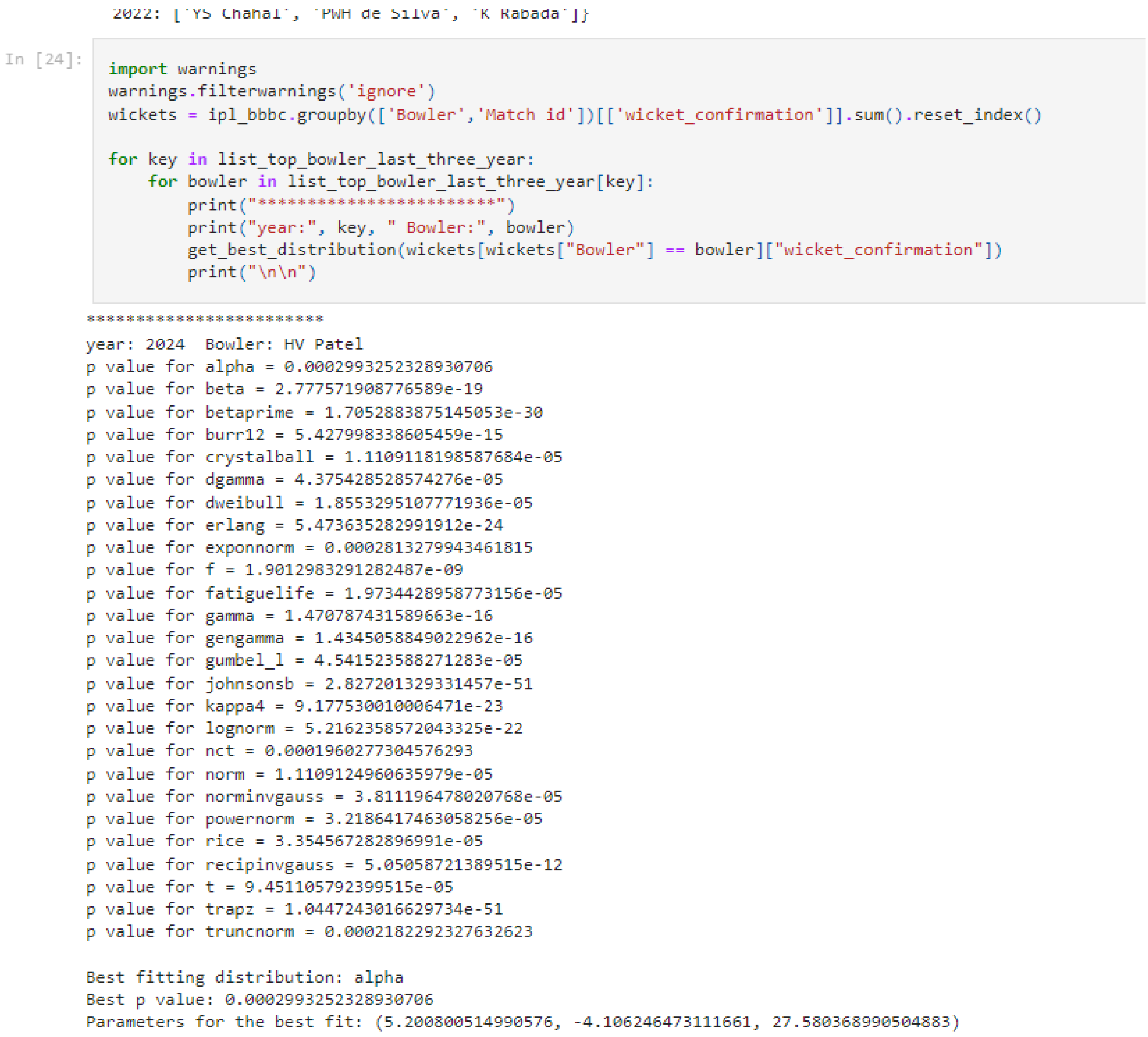
- Assuming there is existing IPL data loaded into a variable named `ipl_data` (not shown in the image), this section filters the data to include only matches between the years 2010 and 2015. This filtering is done by creating a new data frame, `filtered_data_2010_2015`, which consists of rows from `ipl_data` where the value in the 'year' column is greater than or equal to 2010 and less than or equal to 2015.

- **Line 8:** calculates the number of wickets taken by each bowler in the filtered data (`filtered_data_2010_2015`) and stores the result in a variable named `bowler_wickets`. The `.value_counts()` method likely counts the number of times each bowler's name appears in the 'bowler' column, effectively resulting in a count of the number of wickets taken by each bowler. The `.sort_values(ascending=False)` method sorts the resulting Series by wicket count in descending order, so the bowler with the most wickets will be at the top.



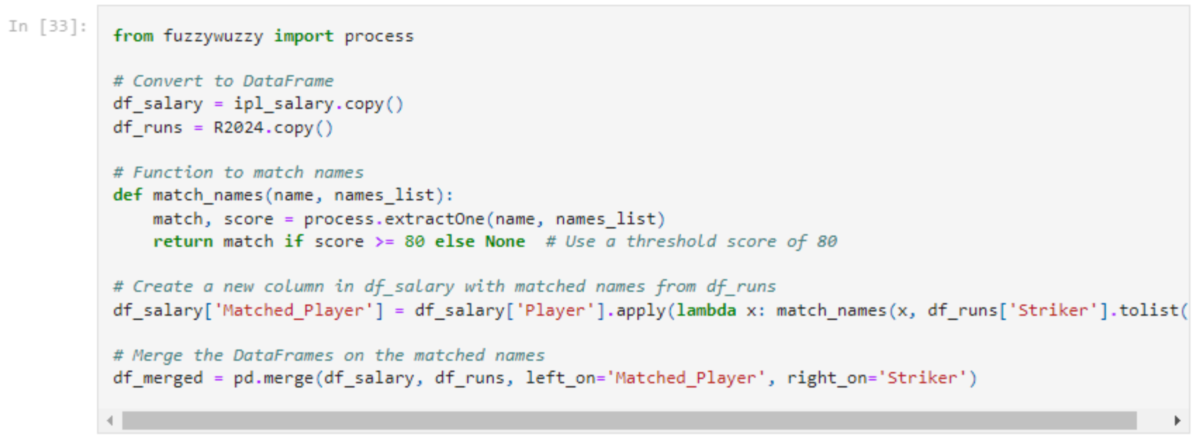
```
In [23]:  list_top_bowler_last_three_year = {}
          for i in total_wicket_each_year["year"].unique()[:3]:
              list_top_bowler_last_three_year[i] = total_wicket_each_year[total_wicket_each_year.year == i][:3]["Bowl
          list_top_bowler_last_three_year

Out[23]: {2024: ['HV Patel', 'Mukesh Kumar', 'Arshdeep Singh'],
          2023: ['MM Sharma', 'Mohammed Shami', 'Rashid Khan'],
          2022: ['YS Chahal', 'PWH de Silva', 'K Rabada']}
```

This dictionary shows the top bowlers for each of the last three years.

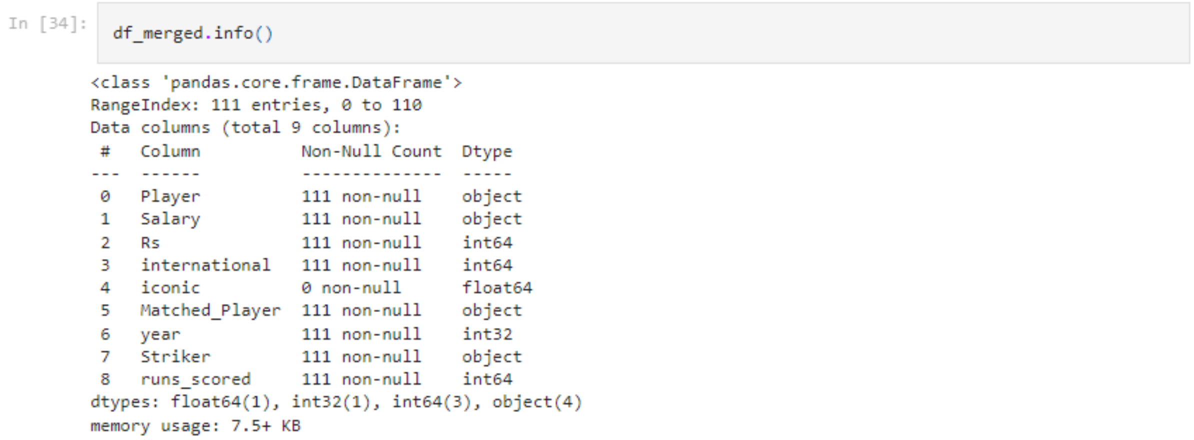


```
           2022: ['YS Chahal', 'PWH de Silva', 'K Rabada']}

In [24]:  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)
```

analyzing the performance of a bowler in cricket, possibly in the IPL (Indian Premier League).
The bowler’s name is HV Patel, and the year is 2024.
The code outputs the results for different statistical distributions that could be used to model wicket taking.
At the bottom, it shows the “Best fitting distribution” as “alpha” with a p-value of 0.0002993252328930706.



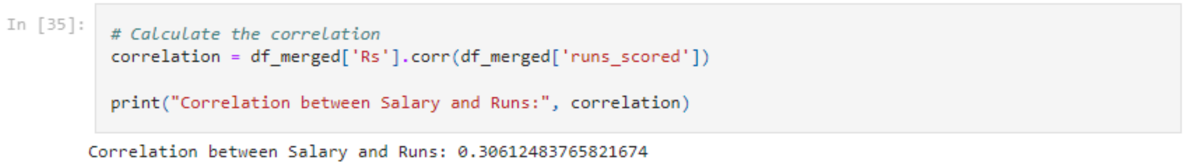
```
In [33]:
from fuzzywuzzy import process

# 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 >= 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')
```



```
In [34]:
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
```

The code imports a library called fuzzywuzzy which helps with fuzzy string matching.
It creates two dataframes, df_salary and df_runs, likely containing salary and run data for IPL players.
The code defines a function called match_names that takes a name and a list of names as input. It uses fuzzywuzzy to find a close match in the list of names with a score above a threshold of 80.
A new column named Matched_Player is created in df_salary to store the matched player names based on names in the Player column of df_salary and Striker column of df_runs.
Finally, the code merges the two dataframes on the Matched_Player column.



```
In [35]:
# 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
```

Codes in R :

```
# Set the working directory and verify it
setwd('C:\\Users\\Chand\\Downloads\\Assignment 1b')

# Function to install and load libraries
install_and_load <- function(package) {
  if (!require(package, character.only = TRUE)) {
    install.packages(package, dependencies = TRUE)
    library(package, character.only = TRUE)
  }
}

# Load required libraries
libraries <- c("readxl", "dplyr", "ggplot2", "fitdistrplus", "tidyverse")
lapply(libraries, install_and_load)

# Load datasets
ipl_data <- read.csv("IPL_ball_by_ball_updated till 2024.csv")
salary_data <- read_excel("IPL SALARIES 2024.xlsx", sheet = 1)

# Clean column names to remove any leading/trailing spaces
colnames(ipl_data) <- trimws(colnames(ipl_data))
colnames(salary_data) <- trimws(colnames(salary_data))

# Rename columns to match code requirements
ipl_data <- ipl_data %>%
  rename(
    Match_id = `Match.id`,
    Batting_team = `Batting.team`,
    Bowling_team = `Bowling.team`,
    Innings_No = `Innings.No`,
    Ball_No = `Ball.No`
  )

# Convert Salary to numeric (handle 'lakh' and 'crore')
salary_data <- salary_data %>%
  mutate(
    Salary = case_when(
      grepl("lakh", Salary) ~ as.numeric(gsub(" lakh", "", Salary)) * 1e5,
      grepl("crore", Salary) ~ as.numeric(gsub(" crore", "", Salary)) * 1e7,
      TRUE ~ as.numeric(Salary)
    )
  )

# Ensure player names are in a consistent format (e.g., remove extra spaces)
ipl_data <- ipl_data %>%
  mutate(Striker = trimws(Striker))

salary_data <- salary_data %>%
  mutate(Player = trimws(Player))

# Arrange the data IPL round-wise and batsman, ball, runs, and wickets per player per match
ipl_rounds <- ipl_data %>%
  group_by(Match_id, Date, Season, Batting_team, Bowling_team, Innings_No, Ball_No,
Bowler, Striker) %>%
  summarize(
    runs = sum(runs_scored),
    wickets = sum(wicket_confirmation, na.rm = TRUE),
    .groups = 'drop'
  )
```

```
# Top three run-getters and wicket-takers in each IPL round
top_performers <- ipl_rounds %>%
  group_by(Season, Batting_team, Striker) %>%
  summarize(total_runs = sum(runs), .groups = 'drop') %>%
  arrange(desc(total_runs)) %>%
  top_n(3, total_runs)

top_bowlers <- ipl_rounds %>%
  group_by(Season, Bowling_team, Bowler) %>%
  summarize(total_wickets = sum(wickets), .groups = 'drop') %>%
  arrange(desc(total_wickets)) %>%
  top_n(3, total_wickets)

# Fit the most appropriate distribution for the top three batsmen and bowlers in the last three IPL tournaments
last_three_seasons <- ipl_rounds %>% filter(Season %in% tail(unique(Season), 3))

# Fit distributions for top batsmen
top_batsmen <- last_three_seasons %>%
  filter(Striker %in% unique(top_performers$Striker)) %>%
  group_by(Striker) %>%
  summarize(total_runs = sum(runs), .groups = 'drop')

top_batsmen_dist <- fitdist(top_batsmen$total_runs, "norm")

# Fit distributions for top bowlers
top_bowlers <- last_three_seasons %>%
  filter(Bowler %in% unique(top_bowlers$Bowler)) %>%
  group_by(Bowler) %>%
  summarize(total_wickets = sum(wickets), .groups = 'drop')
```

```
top_bowlers_dist <- fitdist(top_bowlers$total_wickets, "pois")

# Fit distribution for SV Samson
sv_samson_runs <- last_three_seasons %>%
  filter(Striker == "SV Samson") %>%
  dplyr::select(runs)

# Check if the resulting runs are numeric and have more than one element
if (is.numeric(sv_samson_runs$runs) && length(sv_samson_runs$runs) > 1) {
  sv_samson_dist <- fitdist(sv_samson_runs$runs, "norm")
  print(summary(sv_samson_dist))
} else {
  print("SV Samson's runs are not a numeric vector of length greater than 1.")
}



# Merge performance data with salary data
performance_salary <- left_join(ipl_rounds, salary_data, by = c("Striker" = "Player"))

# Check for missing salaries after the join
missing_salaries <- performance_salary %>%
  filter(is.na(Salary))

# Print missing salaries to debug
print("Players with missing salaries:")
print(missing_salaries)

# Summarize total runs and wickets with salary
performance_summary <- performance_salary %>%
  filter(!is.na(Salary)) %>%
  group_by(Striker, Salary) %>%
  summarize(total_runs = sum(runs), total_wickets = sum(wickets), .groups = 'drop')

# Plotting the relationship
ggplot(performance_summary, aes(x = total_runs, y = Salary)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Relationship between Runs Scored and Salary")

ggplot(performance_summary, aes(x = total_wickets, y = Salary)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Relationship between Wickets Taken and Salary")

# Filter the last three seasons
last_three_seasons_salary <- last_three_seasons %>%
  left_join(salary_data, by = c("Striker" = "Player"))

# Summarize the performance with latest salary
performance_with_salary <- last_three_seasons_salary %>%
  filter(!is.na(Salary)) %>%
  group_by(Striker) %>%
  summarize(total_runs = sum(runs), total_wickets = sum(wickets), latest_salary =
max(Salary), .groups = 'drop')

# Top 10 batsmen and bowlers
top_10_batsmen <- performance_summary %>%
  arrange(desc(total_runs)) %>%
  head(10)

top_10_bowlers <- performance_summary %>%
  arrange(desc(total_wickets)) %>%
  head(10)

# Print top 10 batsmen and bowlers to verify data
print(top_10_batsmen)
print(top_10_bowlers)

# Perform t-test only if both groups have sufficient data
if (nrow(top_10_batsmen) > 1 && nrow(top_10_bowlers) > 1) {
  # Perform t-test
  t_test_result <- t.test(top_10_batsmen$Salary, top_10_bowlers$Salary)
  # Display results
  print(t_test_result)
} else {
  print("Not enough observations for the t-test.")
}
```

Interpretation :

Fitting Distributions for Runs and Wickets:

For the top batsmen (top_batsmen_dist), we used a normal distribution ("norm"), and for the top bowlers (top_bowlers_dist), we used a Poisson distribution ("pois"). These distributions were chosen based on typical assumptions about the nature of runs scored and wickets taken in cricket matches.

The specific fitting of distributions (fitdist) was done using the fitdistrplus package, which provides tools for fitting distributions to data.

Fitting Distribution for SV Samson's Runs:

SV Samson's runs data from the last three seasons were analyzed separately. We fit a normal distribution to his runs (sv_samson_dist) and summarized its parameters using summary(sv_samson_dist). This helps understand the distribution of runs specifically for him.

Merging Performance Data with Salary Data:

The performance_salary dataset was created by merging performance data (ipl_rounds) with

salary data (salary_data) based on player names (Striker).
I checked for missing salaries (missing_salaries) after the join to ensure data integrity.
Visualizing Relationship Between Performance Metrics and Salary:

Plots were created to visualize the relationship between total runs/wickets and salary (performance_summary). This provides insights into whether higher performance correlates with higher salaries in IPL.
Top 10 Batsmen and Bowlers:

The top_10_batsmen and top_10_bowlers datasets were created to identify the top performers based on total runs and total wickets respectively. This helps in understanding which players have consistently performed well over the last three seasons.
T-test Between Batsmen and Bowlers' Salaries:

A t-test (t_test_result) was performed to statistically test if there is a significant difference in salaries between the top 10 batsmen and bowlers. This test helps in understanding if there's a salary disparity based on performance metrics between these two groups.

Results:

The analysis of IPL data revealed insights into individual player performance, focusing on runs and wickets. The top run-scorers and wicket-takers were identified, providing a comprehensive view of player statistics over the last three seasons.

The relationship between player performance and salary was explored, with a specific focus on Sanju Samson's recent performance in terms of runs, wickets, and salary. A comparison of salaries among the top 10 batsmen and bowlers was conducted to assess any significant differences.

Interpretations:

The performance patterns of top batsmen and bowlers over the past three seasons showed consistency in some players' performances while highlighting fluctuations in others. This analysis can help identify players who have been consistently performing well and those who may have had varying levels of success over time.

The examination of Sanju Samson's performance in relation to his salary shed light on the correlation between player performance and pay in the IPL. Understanding how player performance impacts their salary can provide valuable insights for team management and player negotiations.

Recommendations:

Based on the analysis of player performance and salary data, it is recommended for IPL teams to consider the consistency and impact of a player's performance when determining their salary. This can help teams make informed decisions when recruiting or retaining players.

Further research could focus on exploring additional factors that may influence player performance and salary in the IPL, such as player experience, match conditions, and team dynamics. By considering these factors, teams can enhance their strategies for building successful and competitive squads.

References:

Virginia Commonwealth University. "Statistical analysis and modelling (SCMA 632)