IPL Auction Analysis

- **Varun Arora**

Perquisites

- Created a database named ipl_auction_analysis

```
create database ipl_auction_analysis;
```

- Created table ipl_ball table and imported data from csv file.

```
create table ipl_ball (
id INT, inning INT, over INT, ball INT, batsman VARCHAR(255), non_striker VARCHAR(255),  bowler
VARCHAR(255), batsman_runs INT, extra_runs INT,
 total_runs INT, is_wicket INT,  dismissal_kind VARCHAR(255), player_dismissed VARCHAR(255),  fielder
VARCHAR(255), extras_type VARCHAR(255),
batting_team VARCHAR(255), bowling_team VARCHAR(255)
);

copy ipl_ball
		from 'C:\Program Files\PostgreSQL\16\pgAdmin 4\IPL Dataset\IPL_Ball.csv' delimiter ',' csv header;
```

Perquisites

- Created table ipl_match table and imported data from csv file.

```
CREATE TABLE ipl_matches (
id INT, city VARCHAR(255),  match_date DATE,  player_of_match VARCHAR(255),
venue VARCHAR(255),  neutral_venue INT, team1 VARCHAR(255),  team2 VARCHAR(255),  toss_winner
VARCHAR(255),  toss_decision VARCHAR(50),  winner VARCHAR(255),  result VARCHAR(50),  result_margin
VARCHAR(255),  eliminator VARCHAR(50),  method VARCHAR(50),  umpire1 VARCHAR(255),  umpire2
VARCHAR(255)
);

copy ipl_matches
		from 'C:\Program Files\PostgreSQL\16\pgAdmin 4\IPL Dataset\IPL_matches.csv' delimiter ',' csv
header;
```

Perquisites

- created a new table for ipl_ball table with year specified exracted from ipl_matches match_date

```
create table ipl_ball_years as
		select a.*,(Extract(year from b.match_date)) as ipl_year from ipl_ball
		as a left join ipl_matches as b
		on a.id = b.id;
```

Top Ten Aggressive Batsmen

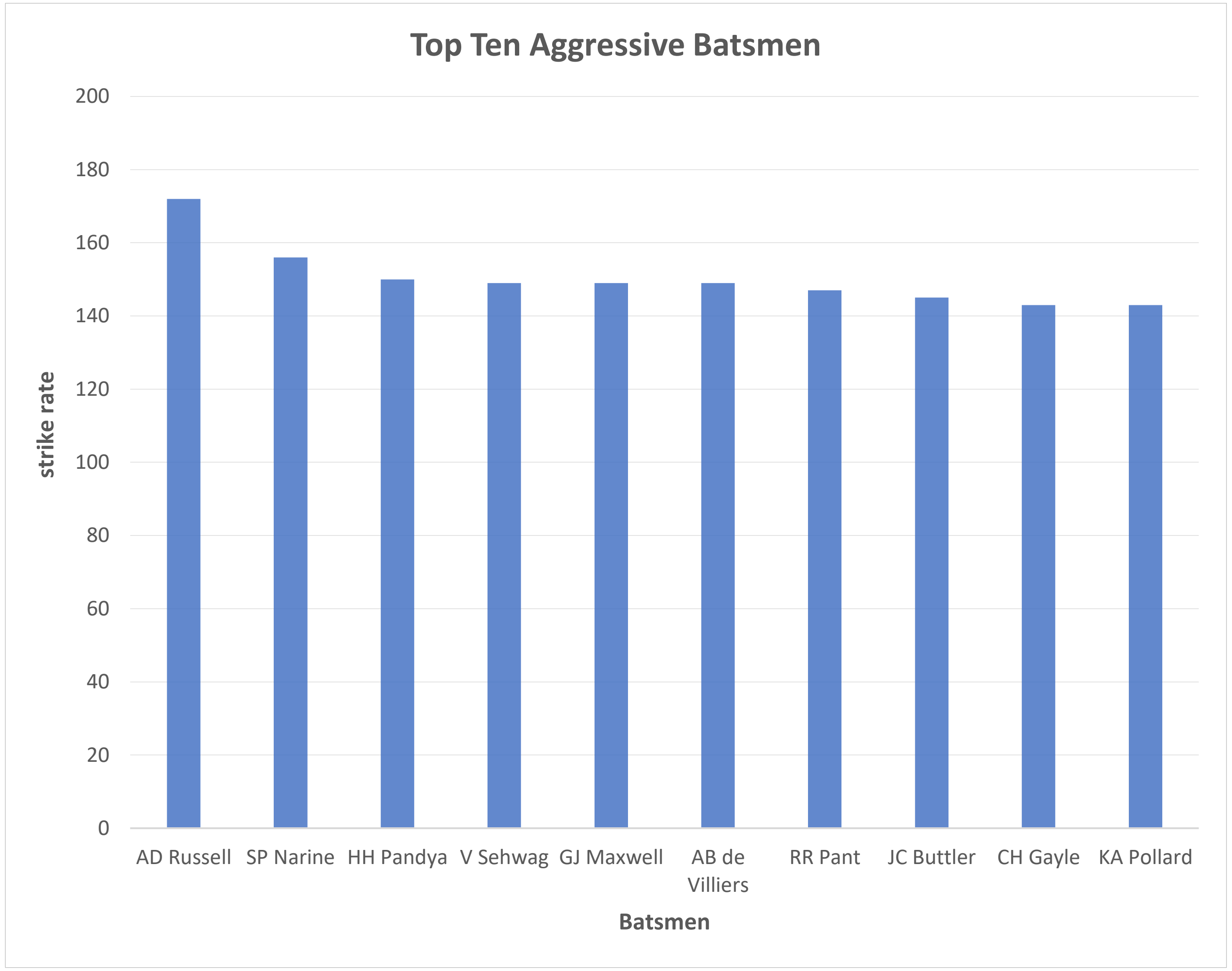
Query:-

```
SELECT BATSMAN,
SUM(BATSMAN_RUNS) AS TOTAL_RUN,
COUNT(BALL) AS TOTAL_BALLS,
ROUND((CAST(SUM(BATSMAN_RUNS) AS DECIMAL)
/COUNT(BALL) *100)) AS STRIKE_RATE,
COUNT(DISTINCT(IPL_YEAR)) AS
NUM_OF_SEASONS_PLAYED
FROM IPL_BALL_YEARS GROUP BY BATSMAN HAVING
COUNT(BALL) >=500 ORDER BY STRIKE_RATE DESC LIMIT 10;
```

Explanation:-

This query will return the list of 10 batsmen who have played more than 500 balls and have highest strike rate. Strike rate is calculated using formula no. of runs made by batsman over no. of balls played by batsman into 100.

Using order by and limit 10, we are getting 10 records with highest strike rate to lowest. I also added a the column with number of season they played to have a better insight with respect to strike rate in the table.



Top Ten Anchor Batsmen

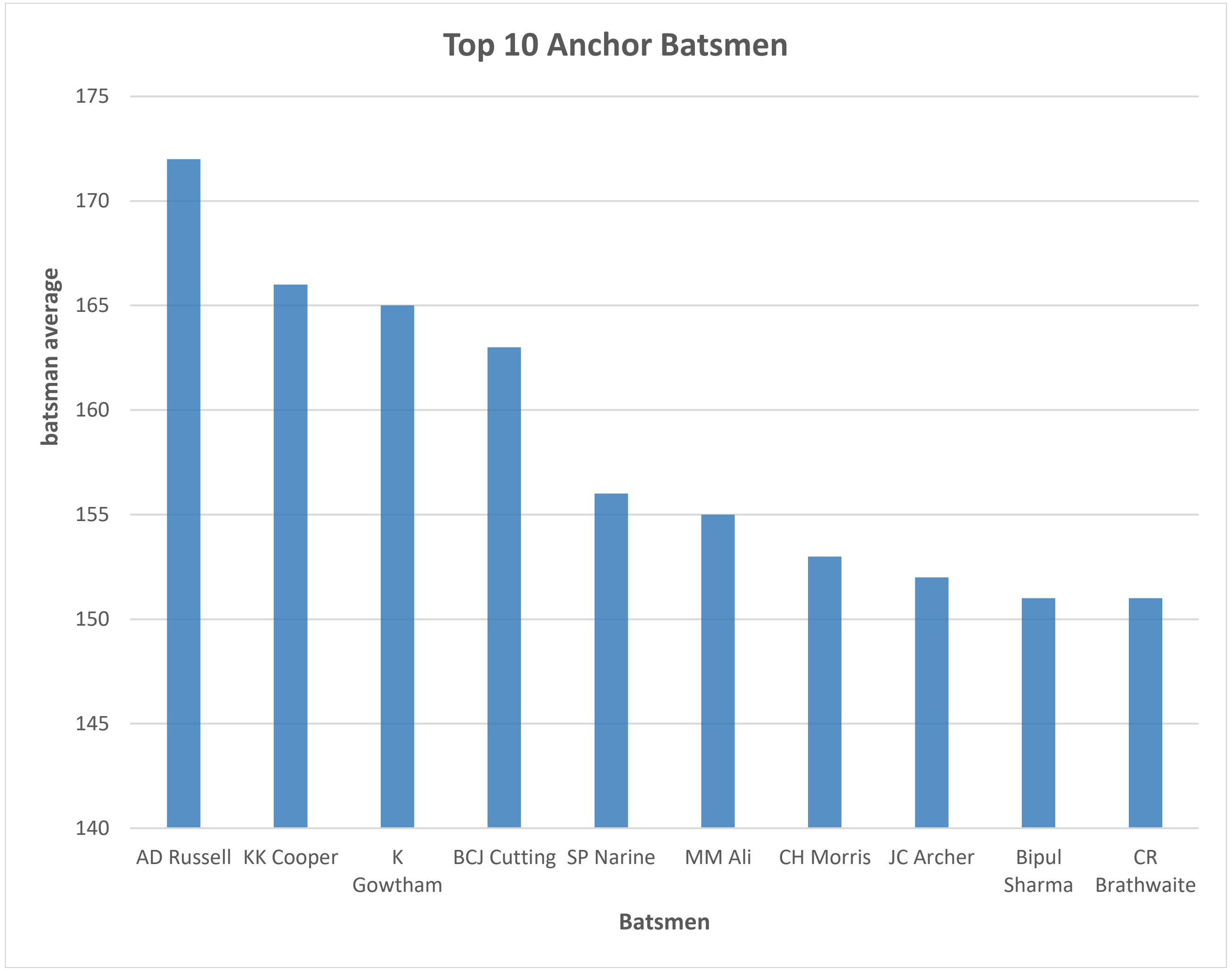
Query:-

```
SELECT  BATSMAN, TOTAL_RUNS, TOTAL_DISMISSALS,
NUM_OF_SEASONS_PLAYED,

ROUND((CAST(TOTAL_RUNS AS DECIMAL)/TOTAL_DISMISSALS * 100))
AS BATSMAN_AVERAGE

FROM (SELECT

BATSMAN,  SUM(BATSMAN_RUNS) AS TOTAL_RUNS,

COUNT(IS_WICKET) AS TOTAL_DISMISSALS,

COUNT(DISTINCT(IPL_YEAR)) AS NUM_OF_SEASONS_PLAYED

FROM IPL_BALL_YEARS GROUP BY BATSMAN HAVING
COUNT(IS_WICKET) > 0 AND COUNT(DISTINCT(IPL_YEAR)) > 2

   ) AS SUBQUERY

ORDER BY BATSMAN_AVERAGE DESC LIMIT 10;
```

Explanation:-

This query will return the list of 10 batsmen who have highest batsman average which is computed with total run scored an total dismissals. Where condition matches two conditions first is batsman should have played in more than 2 IPL seasons and second, total dismissals should be more than 0 to avoid calculative errors which using it in batting average formula.

The subquery is used to aggregrate the data for batsman_runs, dismissals, seasons played and to use the alias name in main query from subquery.



Top Ten Hard Hitters

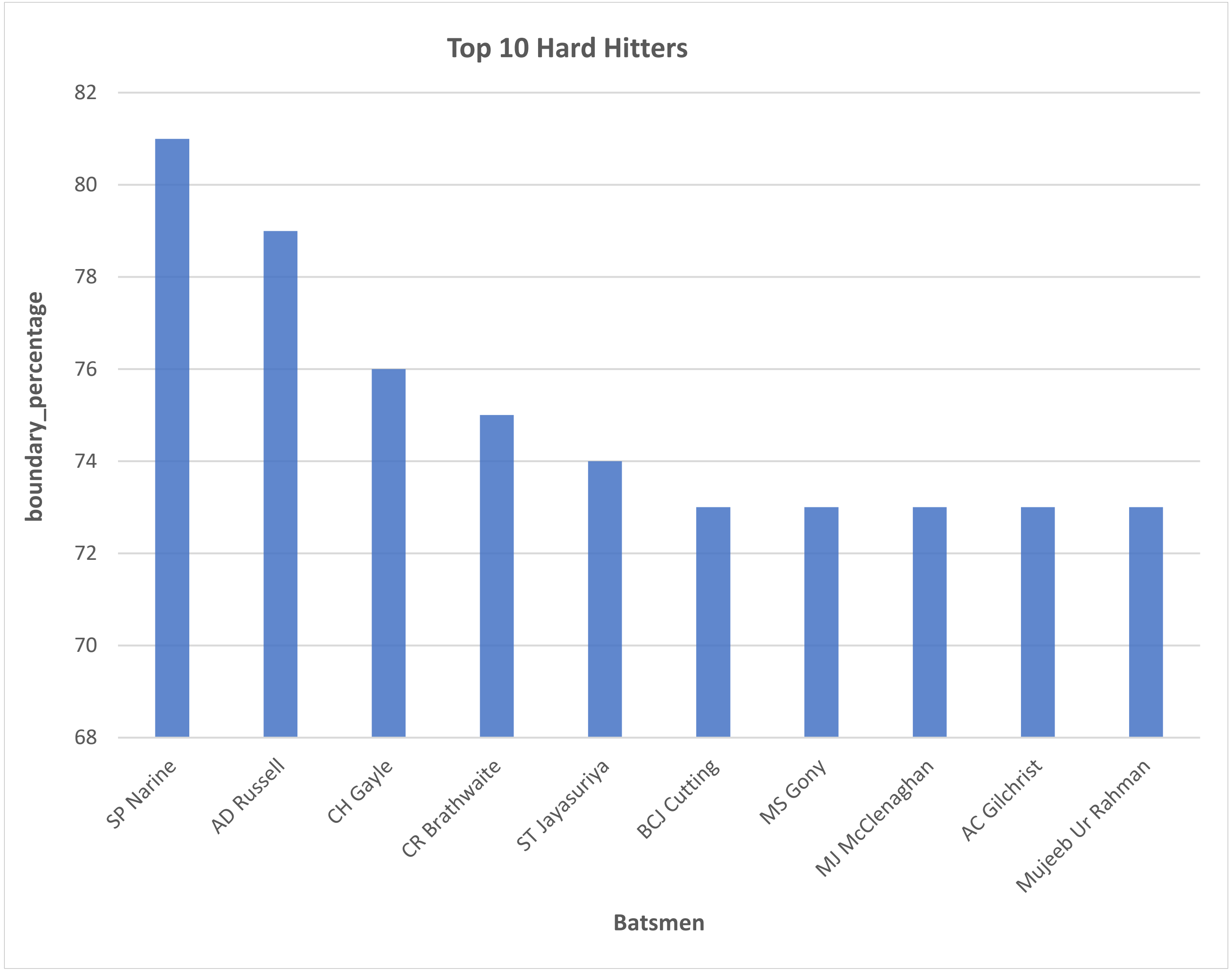
Query:-

```
SELECT
BATSMAN,  NUM_OF_SEASONS_PLAYED,  NUM_OF_FOURS, NUM_OF_SIXES,
(NUM_OF_FOURS + NUM_OF_SIXES) AS TOTAL_BOUNDARIES,
(NUM_OF_FOURS*4+NUM_OF_SIXES*6) AS BOUNDARY_RUNS, TOTAL_RUNS,
ROUND(CAST((NUM_OF_FOURS*4+NUM_OF_SIXES*6)AS DECIMAL)/TOTAL_RUNS
*100) AS BOUNDARY_PERCENTAGE
FROM (
SELECT BATSMAN,  SUM(BATSMAN_RUNS) AS TOTAL_RUNS,
COUNT(DISTINCT(IPL_YEAR)) AS NUM_OF_SEASONS_PLAYED,
SUM(CASE WHEN BATSMAN_RUNS = 4 THEN 1 ELSE 0 END) AS NUM_OF_FOURS,
SUM(CASE WHEN BATSMAN_RUNS = 6 THEN 1 ELSE 0 END) AS NUM_OF_SIXES
FROM IPL_BALL_YEARS GROUP BY BATSMAN)
AS SUBQUERY_ALIAS WHERE NUM_OF_SEASONS_PLAYED>2 ORDER BY
BOUNDARY_PERCENTAGE DESC LIMIT 10;
```

Explanation:-

This query will returns the top 10 hard-hitting batsmen who have played more than two IPL seasons, sorted by their boundary percentage. The inner subquery calculates aggregated stats for each batsman.

The outer query then calculates total boundaries (fours and sixes), boundary runs, and boundary percentage for each batsman. It filters the data of only those batsmen who have played more than 2 IPL seasons order by boundary_percentage in descending order.



Top Ten Economy Bowlers

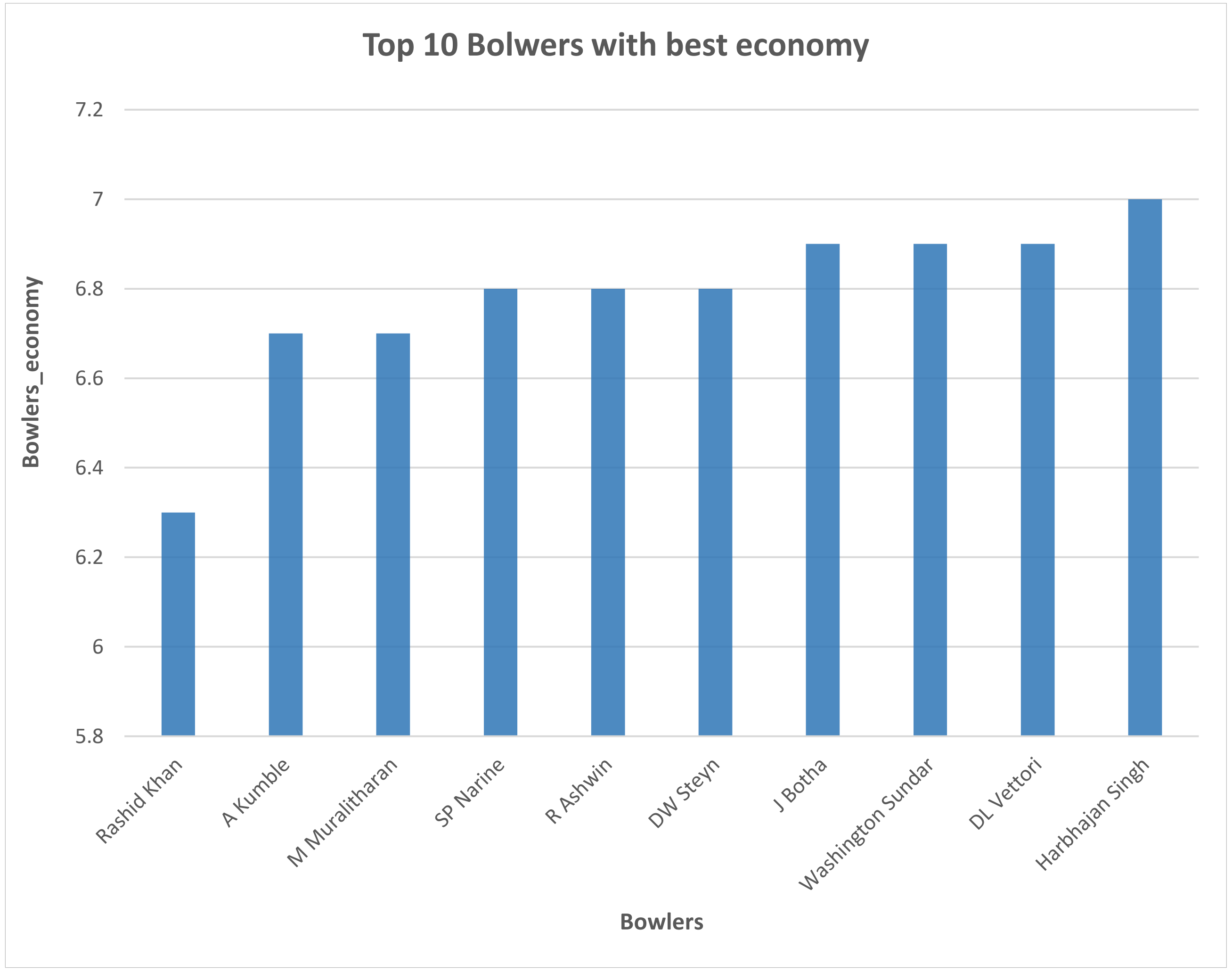
Query:-

```
SELECT
BOWLER, TOTAL_RUN,  TOTAL_BALLS,  TOTAL_OVERS,
ROUND((CAST((TOTAL_RUN) AS DECIMAL)/TOTAL_OVERS),1) AS
BOWLERS_ECONOMY
FROM (
SELECT
BOWLER, COUNT(BALL) AS TOTAL_BALLS,
COUNT(OVER)/6 AS TOTAL_OVERS, SUM(TOTAL_RUNS) AS TOTAL_RUN
FROM IPL_BALL_YEARS GROUP BY BOWLER
) AS SUB_QUERIES
WHERE TOTAL_BALLS >= 500 ORDER BY BOWLERS_ECONOMY LIMIT 10;
```

Explanation:-

This query will returns the top 10 economy bowlers. The inner query aggregates bowler's stats like total ball bowled using count function, total overs (dividing total balls with 6) and total runs.

The outer query then calculate the economy rate using total balls and total runs aggregated in inner query and then filter it to show only those records of those blowers who have bowled more than 500 balls order by economy in ascending order limiting records up to 10.



Top Ten High Strike Bowlers

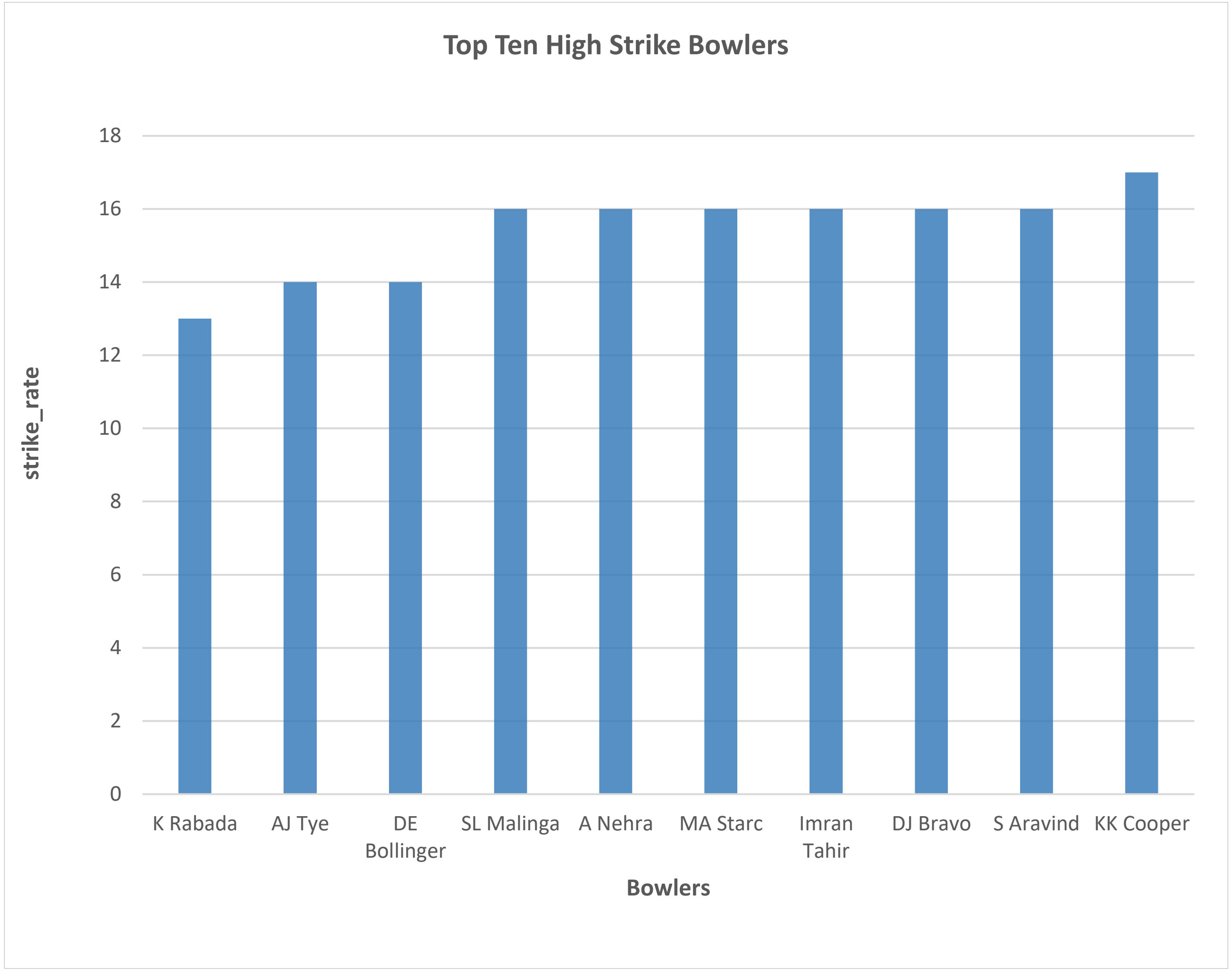
Query:-

```
SELECT
BOWLER,
COUNT(BALL) AS TOTAL_BALLS,
SUM(IS_WICKET) AS WICKET_TAKEN,
ROUND(CAST(COUNT(BALL) AS DECIMAL)/SUM(IS_WICKET))
AS STRIKE_RATE
FROM IPL_BALL_YEARS
GROUP BY BOWLER HAVING COUNT(BALL)>500
ORDER BY STRIKE_RATE LIMIT 10;
```

Explanation:-

This SQL query returns the top 10 bowlers with the best strike rates, having bowled at least 500 balls. The strike rate is calculated as the ratio of balls bowled to wickets taken.

The results are ordered in ascending order, with lower strike rates indicating higher efficiency.



Top Ten All Rounders

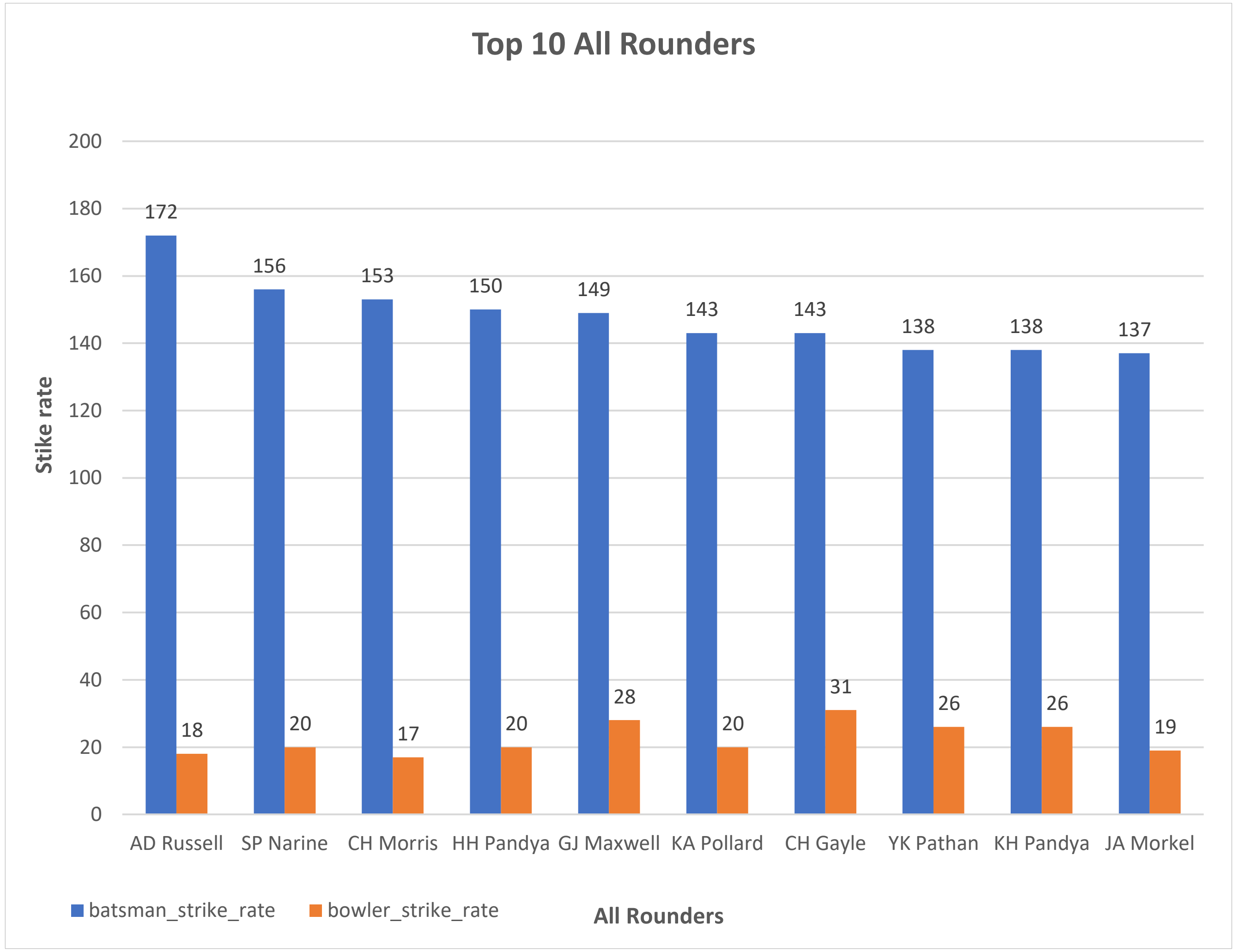
QUERY:-

```
SELECT A.NAME AS PLAYER,  A.RUNS AS RUNS_SCORED, A.BALLS AS BALL_PLAYED,

B.BALLS AS BALLS_BOWLED,  B.TOTAL_WICKETS AS TOTAL_WICKETS_TAKEN,

ROUND(CAST(A.RUNS AS DECIMAL)/A.BALLS*100) AS BATSMAN_STRIKE_RATE,

ROUND(CAST(B.BALLS AS DECIMAL)/B.TOTAL_WICKETS) AS BOWLER_STRIKE_RATE

FROM

(SELECT BATSMAN AS NAME,  SUM(BATSMAN_RUNS) AS RUNS, COUNT(BALL) AS BALLS

FROM IPL_BALL_YEARS GROUP BY BATSMAN) AS A INNER JOIN

 (SELECT BOWLER AS NAME, COUNT(BALL) AS BALLS, SUM(IS_WICKET) AS TOTAL_WICKETS
FROM IPL_BALL_YEARS GROUP BY BOWLER)  AS B

ON A.NAME = B.NAME WHERE A.BALLS>300 AND B.BALLS>500

ORDER BY BATSMAN_STRIKE_RATE DESC, BOWLER_STRIKE_RATE ASC LIMIT 10;
```

Explanation:-

This SQL query returns the list of 10 all rounders based on their both batting an bowling strike rate. I used to inner join to combine two table coming out to subquery which provide batsmen stats like name, total runs, balls played and bowlers stats like name, total wicket taken and total balls bowled respectively.

Inner join combine both subqueries based on common column 'name'. It is then filters with where clause to get those names who have bowled played more than 500 balls and played 300 balls atleast. Records are sorted in descending batsmen strike rate and ascending bowler strike rate limiting result to 10 only.



All Rounders	batsman_strike_rate	bowler_strike_rate
AD Russell	172	18
SP Narine	156	20
CH Morris	153	17
HH Pandya	150	20
GJ Maxwell	149	28
KA Pollard	143	20
CH Gayle	143	31
YK Pathan	138	26
KH Pandya	138	26
JA Morkel	137	19

Top Five Wicketkeepers

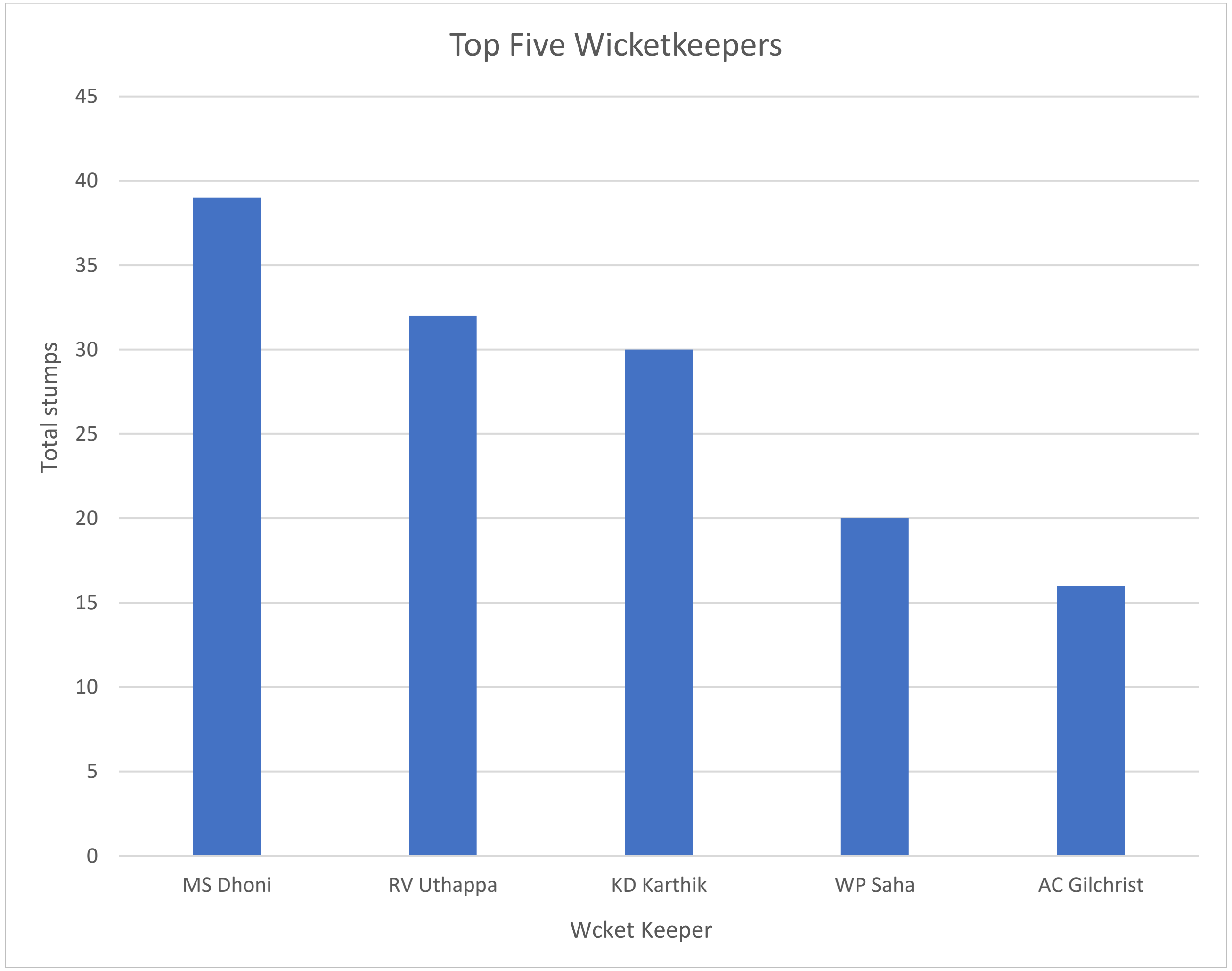
Query:-

```
SELECT
FIELDER AS WICKET_KEEPER,
COUNT(DISMISSAL_KIND) AS TOTAL_STUMPS
FROM IPL_BALL_YEARS
WHERE DISMISSAL_KIND = 'STUMPED’
GROUP BY FIELDER
ORDER BY TOTAL_STUMPS DESC LIMIT 5;
```

Explanation:-

This query returns list of 5 wicketkeepers who have taken most stumps throughout the seasons. We counted total stumps from the dismissal_kind column and fielder name from fielder column who stump the wicket.

Then the list is sorted in descending total_stumps order limit results up to 5.



Thank You !