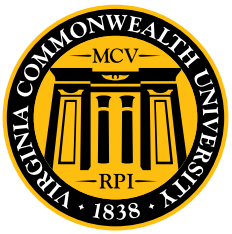


VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A1b: Preliminary preparation and analysis of data- Descriptive statistics

PRAMITT M PATIL

V01104754

Date of Submission: 18-06-2024

CONTENTS

Comprehensive Analysis of IPL Performance Metrics and Salary Correlations Over the Last Three Tournaments

1. Introduction

The analysis considers the rounds data-wise of IPL players and looks at how many matches a batsman plays, how he faces bowlers for balls, and how many runs the bowler takes for wickets player-wise. It identifies the top three scores and counts wickets taken in the innings. The two previous IPL tournaments, like distributions to runs and wickets of the most successful and unpopular computerised the study. This uses the relationship between a player's salary and performance, with a deeper dive into the player's 3-year performance and the last salary (2024). It also looks into the average salary differences between top ten batsmen and wicket-taking bowlers from the same time (D

2. Results and Interpretation

a) 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

i)Top 4 wicket-takers

	Bowler	Season	wicket
1	MM Sharma	2023	31
2	Mohammed Shami	2023	28
3	Rashid Khan	2023	28
4	TU Deshpande	2023	24

ii)Top 3 run getters

	Striker	Season	avg_runs
1	Shubman Gill	2023	890
2	F du Plessis	2023	730
3	DP Conway	2023	672
4	V Kohli	2023	639

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.

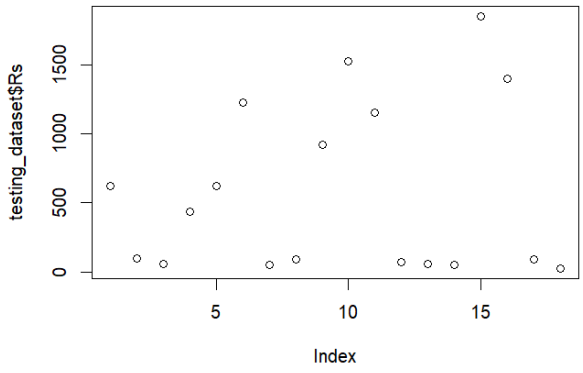
	Striker	Season	avg_runs
1	Shubman Gill	2023	890
2	JC Buttler	2022	863
3	F du Plessis	2023	730

	Bowler	Season	wicket
1	HV Patel	2021	35
2	MM Sharma	2023	31
3	YS Chahal	2022	29

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

	Player	Salary	Rs	international	iconic	Season.x	avg_runs	wicket	matched_player	performance
1	abishek porel	20 lakh	20	0	NA	NA	NA	NA	NA	NA
2	anrich nortje	6.5 crore	650	1	NA	NA	NA	NA	NA	NA
3	axar patel	9 crore	900	0	NA	NA	NA	NA	NA	NA
4	da warner	6.25 crore	625	1	NA	NA	NA	NA	NA	NA
5	ishant sharma	50 lakh	50	0	NA	NA	NA	NA	NA	NA
6	kuldeep yadav	2 crore	200	0	NA	2023	31	10	kuldeep yadav	156.0
7	lalit yadav	65 lakh	65	0	NA	2023	66	3	lalit yadav	103.5
8	lungi ngidi	5 crore	500	1	NA	NA	NA	NA	NA	NA
9	mitchell marsh	6.5 crore	650	1	NA	NA	NA	NA	NA	NA
10	mukesh kumar	5.5 crore	550	0	NA	2023	7	9	mukesh kumar	119.5

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



The scatter plot displays the relationship between player indices and their corresponding runs scored (Rs). The key observations are:

1. Wide Variability: There is significant variability in runs scored, with some players scoring very high (over 1000 runs) and many others scoring below 500 runs.

2. High Performers: A few players stand out with exceptionally high scores, particularly around indices 6, 10, and 16.

3. Low Scorers: A considerable number of players have relatively low scores, clustering below 500 runs.

This indicates a diverse range of performances among players, with a small number achieving high scores and many scoring relatively low. Further analysis could explore the central tendency, fit distributions, and investigate the correlation between runs scored and player salaries.

CODES

i) R

```
 setwd("D:\\E drive\\SCMA 632\\data")
df_p = read.csv('IPL ball by ball updated till 2024.csv')
library(readxl)
df_s = read_excel('IPL SALARIES 2024.xlsx')


#install.packages('dplyr')
library(dplyr)
dput(names(df_p))
unique(df_p$wicket_confirmation)

library(dplyr)

# Print the column names of df_p to verify
print("Column names of df_p:")
print(colnames(df_p))

# Assuming the column names in df_p are correct as used in the select function
# If the column names differ, update them accordingly in the select function

# Perform the operations
df_bat <- df_p %>%
  select('Match.id', 'Season', 'Bowler', 'Striker', 'runs_scored', 'wicket_confirmation') %>%
  filter(Season %in% c('2023', '2022', '2021')) %>%
  group_by(Striker, Season) %>%
  summarise(avg_runs = sum(runs_scored, na.rm = TRUE)) %>%
```

```
  arrange(desc(avg_runs))

# Print the resulting data frame
print("Resulting df_bat:")
print(df_bat)

df_bat <- df_p %>%
  select(Match.id,Season,Bowler,Striker,runs_scored,wicket_confirmation)%>%
  filter((Season=='2023')|(Season=='2022')|(Season=='2021'))%>%
  group_by(Striker, Season) %>%
  summarise(avg_runs = sum(runs_scored, na.rm = TRUE))%>%
  arrange(desc(avg_runs))

head(df_bat,25)
df_bat$Striker


df_bow <- df_p %>%
  select(Match.id,Season,Bowler,Striker,runs_scored,wicket_confirmation)%>%
  filter((Season=='2023')|(Season=='2022')|(Season=='2021'))%>%
  group_by(Bowler, Season) %>%
  summarise(wicket = sum(wicket_confirmation, na.rm = TRUE))%>%
  arrange(desc(wicket))

dim(df_bat)
dim(df_bow)

head(df_bow)
# View the result
df_bat
unique(df_bat$Striker)


unique(df_bow$Bowler)

df_s

names(df_s)
head(df_s$Player)

bat = df_bat[df_bat$Season=='2023',]

bow = df_bow[df_bow$Season=='2023',]
head(bat )


dim(bow)
dim(df_s)
```

```
head(bat)
head(bow)

unique(df_s$Player)
unique(bat$Striker)
unique(bow$Bowler)

# Load necessary library
library(dplyr)


# Perform a full join
joined_df <- full_join(bat, bow, by = c("Striker" = "Bowler"))

# Print the result
print(joined_df)
dim(joined_df)
write.csv(joined_df, 'joined_df.csv')
#HARMONISE THE NAMES IN THE TWO FILES

names(joined_df)
df = joined_df %>%
select(Striker,Season.x,avg_runs,wicket)
View(df)

library(dplyr)
library(stringdist)

# Normalize the names
df$Striker <- tolower(trimws(df$Striker))
df_s$Player <- tolower(trimws(df_s$Player))


# Define a function to map names using fuzzy matching
match_names <- function(name, choices, threshold = 0.1) {
  distances <- stringdist::stringdist(name, choices, method = "jw")
  min_dist <- min(distances)
  if (min_dist <= threshold) {
    return(choices[which.min(distances)])
  } else {
    return(NA)
  }
}
# Create a list of choices from the second data frame
choices <- df_s$Player
```

```
# Apply the matching function to the first data frame
df <- df %>%
  mutate(matched_player = sapply(Striker, match_names, choices = choices ))

# Create a mapping dictionary if needed
name_mapping <- df %>%
  filter(!is.na(matched_player)) %>%
  select(Striker, matched_player)

# If needed, update the names in the original data frames
df<- df%>%
  mutate(Striker = ifelse(!is.na(matched_player), matched_player, Striker ))

df_s <- df_s%>%
  mutate(Player = ifelse(Player %in% name_mapping$matched_player,
                         name_mapping$Striker[match(Player, name_mapping$matched_player)],
                         Player))



# Ungroup the name_mapping to simplify the join
name_mapping <- name_mapping %>% ungroup()

# Perform a left join to safely replace player names
df_s <- df_s %>%
  left_join(name_mapping, by = c("Player" = "matched_player")) %>%
  mutate(Player = ifelse(is.na(Striker), Player, Striker)) %>%
  select(-Striker)


# Load necessary library
library(dplyr)

# Assuming df_s has already been processed
df_s <- df_s %>%
  left_join(name_mapping, by = c("Player" = "matched_player")) %>%
  mutate(Player = ifelse(is.na(Striker), Player, Striker)) %>%
  select(-Striker)

# Merge df_s with df using Player in df_s and Striker in df as keys
df_combined <- df_s %>%
  left_join(df, by = c("Player" = "Striker"))

# Print the resulting combined data frame
print(df_combined)
names(df_combined)
# Save the updated data frames if needed
```

```
write.csv(df_combined, 'df_combined.csv', row.names = FALSE)


max(df_combined$avg_runs, na.rm=TRUE)
max(df_combined$wicket, na.rm=TRUE)

quantile(df_combined$avg_runs, na.rm=TRUE,.9)
quantile(df_combined$wicket, na.rm=TRUE,.9)

348.7/28
# 12.45

df_combined$performance = df_combined$avg_runs+ 12.5*df_combined$wicket
any(is.na(df_combined$performance))
sum(is.na(df_combined$performance))




# Replace NA with 0 in the performance column
df_new <- df_combined %>%
  mutate(performance = ifelse(is.na(performance), 0, performance))

names(df_new)
str(df_new)


hist(df_new$performance, prob=TRUE)
lines(density(df_new$performance), na.rm=TRUE)

library(fitdistrplus)
descdist(df_new$performance)
head(df_new)
sum(is.null(df_new))
summary(df_new)
names(df_new)

fit = lm(Rs ~ avg_runs + wicket , data=df_new)
summary(fit)

library(car)
vif(fit)
library(lmtest)
bptest(fit)


fit1 = lm(Rs ~ avg_runs + wicket+ I(avg_runs*wicket), data=df_new)
summary(fit1)
```

```
fit2 = lm(Rs ~ performance + international , data=df_new)
summary(fit2)


library(dplyr)
# creating a data frame


training_dataset <- df_new %>% sample_frac(0.7)

testing_dataset <- anti_join(df_new, training_dataset, by = 'Rs')


dim(training_dataset)
dim(testing_dataset)
dim(df_new)


fit_t = lm(Rs ~ performance + international , data=training_dataset)
summary(fit_t)

pred <- predict(fit_t,testing_dataset)
pred
testing_dataset$Rs

rmse = sum(((pred-testing_dataset$Rs)^2/testing_dataset$Rs))
sqrt(rmse)
lines(pred)
plot(testing_dataset$Rs)
```

ii) Python

python code too long , uploaded on github and canvas