Module 6 - Team Project: Final Report

ALY6015: Intermediate Analytics

Module 6 — Team Project: Final Report

Group 3 Pei-Yu Jheng, Phuong Mai Dan Nguyen, Sumate Boonchitbanchong

Instructor: Vladimir Shapiro

Winter Semester Sec 09

Introduction

Success in competitive sports necessitates a combination of strategic thought, individual talent, and successful teamwork. Basketball, a riveting blend of strategy and skill, embodies the never-ending chase of win, with every dribble, pass, and shot having the potential to change the course of a game and shape an entire season.

This investigation delves into the complex dynamics of basketball performance to identify the keys to success. We use statistical analysis to understand how player performance, game conditions, and team dynamics impact athletic achievement.

Method

Table 1: Table 1 : Business Questions and Methods

Business Question	Method
BQ1: Test whether the mean points per minute are equal between close games and non-close games	t-test
BQ2: Test whether the mean points per minute differ across different opponent teams	ANOVA
BQ3: Test whether winning percentage by year is independent on game score levels	Chi-square of independence
BQ4: Predict the binary outcome variable'successYes/No'	Logistic Regression
BQ5: Predict the binary outcome variable 'win'	Logistic Regression
BQ6: Predict the points per minute variable	Lasso Regression

BQ1 allows us to compare the performance of Michael Jordan in close games versus non-close games.

A close game or non-close game compute by the "diff" variable. A close game defines as a game where the difference in points between the winning and losing teams is equal or less than 5 points.

M1: By conducting a t-test, we can determine if there is a statistically significant difference in mean points per minute between these two game situations.

Understanding these differences can shed light on how player perform under different levels of pressure or intensity.

BQ2 investigates how Michael Jordan's performance varies across different opponent teams.

M2: ANOVA is appropriate for comparing means across multiple groups, making it a suitable choice for assessing differences in points per minute among various opponents of Michael Jordan.

This information is valuable for understanding matchup dynamics and identifying teams that may pose particular challenges or opportunities for players.

BQ3 explores the relationship between winning percentage by year and its independence on game score levels.

M3 involves the Chi-square Test of independence, providing insights into how game score performance may depend on winning percentage by year.

BQ4 focuses on understanding the factors that contribute to long-term player success.

M4: Logistic Regression is applied to predict whether a player will achieve success based on their early career statistics.

By analyzing the predictors associated with success, such as performance metrics and game locations, we can identify key factors influencing a player's trajectory.

BQ5 predicts the binary outcome of individual game wins or losses.

M5: Logistic Regression is utilized to model the likelihood of a team winning a game based on various factors which are various variables within Michael Jordan's dataset.

By understanding the predictors associated with winning outcomes, teams can make informed decisions to improve their chances of success on the court.

BQ6 forecasts the continuous variable of points per minute.

M6: Lasso Regression allows us to not only predict points per minute but also to identify the most influential factors affecting a player's scoring efficiency during games.

By shrinking certain regression coefficients to zero, Lasso regression provides insight into the most impactful features to formulate strategies to maximize scoring output and optimize player performance for maximizing offensive potential.

Clears console

```r
cat("\014") # clears console
rm(list = ls()) # clears global environment
try(dev.off(dev.list()["RStudioGD"]), silent = TRUE) # clears plots
try(p_unload(p_loaded(), character.only = TRUE), silent = TRUE) # clears packages
options(scipen = 100) # disables scientific notation for entire R session
```

Import libraries

```r
library(pacman)
p_load(ggplot2)
p_load(ggthemes)
p_load(tidyverse)
p_load(lubridate)
p_load(ggeasy)
p_load(janitor)
p_load(dplyr)
p_load(tibble)
p_load(skimr)
p_load(cowplot)     #plot_grid
p_load(car)
p_load(corrplot)
```

```r
p_load(ggcorrplot)
p_load(RColorBrewer)

p_load(knitr)            #table caption
p_load(ROSE)             #ovun.sample

p_load(smbinning)        #smbinning
p_load(caret)            #createDataPartition function
p_load(glmnet)
p_load(Metrics)
p_load(pROC)             #area under the curve
p_load(MKclass)          #optCutoff()
```

Import Dataset

```r
#obtain the current work directory path
current_wd = getwd()

#add the dataset name
full_path_csv = paste(current_wd, "michael-jordan-nba-career-regular-season-stats-by-game.csv", sep="/")
#full_path_csv

#import csv
Michael_df <- read_csv(full_path_csv)
```

```
## Rows: 1072 Columns: 33
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): Date, Tm, Opp
## dbl (30): EndYear, Rk, G, Years, Days, Age, Home, Win, Diff, GS, MP, FG, FGA...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
Michael_df <- clean_names(Michael_df)

#Add missing data
#x3p_pct = 0
Michael_df$x3p_pct[is.na(Michael_df$x3p_pct) & Michael_df$x3pa == 0] <- 0

#ft_pct = 0
Michael_df$ft_pct[is.na(Michael_df$ft_pct) & Michael_df$fta == 0] <- 0

#Additional Variables
Michael_df <- Michael_df %>%
  mutate(close_game = ifelse(abs(diff) <= 5, 1, 0)) %>%
  mutate(pts_per_minute = pts / mp)

#Remove irrelevant variables
selected_columns <- c('rk', 'g', 'date')
Michael_df <- Michael_df[, !names(Michael_df) %in% selected_columns]
```

```r
#Add name variable
Michael_df$name <- 'Michael Jordan'
```

Analysis

EDA

```r
head(Michael_df, 10)
```

```
## # A tibble: 10 x 33
##    end_year years  days   age tm     home opp     win  diff    gs    mp    fg
##       <dbl> <dbl> <dbl> <dbl> <chr> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1985    21   252  21.7 CHI       1 WSB       1    16     1    40     5
##  2     1985    21   253  21.7 CHI       0 MIL       0    -2     1    34     8
##  3     1985    21   255  21.7 CHI       1 MIL       1     6     1    34    13
##  4     1985    21   256  21.7 CHI       0 KCK       1     5     1    36     8
##  5     1985    21   258  21.7 CHI       0 DEN       0   -16     1    33     7
##  6     1985    21   264  21.7 CHI       0 DET       1     4     1    27     9
##  7     1985    21   265  21.7 CHI       0 NYK       1    15     1    33    15
##  8     1985    21   267  21.7 CHI       0 IND       1     2     1    42     9
##  9     1985    21   270  21.7 CHI       1 SAS       1     3     1    43    18
## 10     1985    21   272  21.7 CHI       1 BOS       0   -20     1    33    12
## # i 21 more variables: fga <dbl>, fg_pct <dbl>, x3p <dbl>, x3pa <dbl>,
## #   x3p_pct <dbl>, ft <dbl>, fta <dbl>, ft_pct <dbl>, orb <dbl>, drb <dbl>,
## #   trb <dbl>, ast <dbl>, stl <dbl>, blk <dbl>, tov <dbl>, pf <dbl>, pts <dbl>,
## #   gm_sc <dbl>, close_game <dbl>, pts_per_minute <dbl>, name <chr>
```

```r
skim(Michael_df)
```

Table 2: Data summary

Name	Michael_df
Number of rows	1072
Number of columns	33
Column type frequency:	
character	3
numeric	30
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
tm	0	1	3	3	0	2	0
opp	0	1	3	3	0	33	0
name	0	1	14	14	0	1	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
end_year	0	1	1992.89	5.36	1985.00	1989.00	1992.00	1997.00	2003.00	
years	0	1	29.28	5.35	21.00	25.00	28.00	33.00	40.00	
days	0	1	201.63	137.58	0.00	42.00	274.00	320.00	365.00	
age	0	1	29.83	5.36	21.69	25.73	28.86	33.92	40.16	
home	0	1	0.50	0.50	0.00	0.00	0.50	1.00	1.00	
win	0	1	0.66	0.47	0.00	0.00	1.00	1.00	1.00	
diff	0	1	4.87	12.81	-44.00	-4.00	5.00	13.00	47.00	
gs	0	1	0.97	0.17	0.00	1.00	1.00	1.00	1.00	
mp	0	1	38.26	5.71	12.00	36.00	39.00	42.00	56.00	
fg	0	1	11.37	3.83	1.00	9.00	11.00	14.00	27.00	
fga	0	1	22.89	5.94	5.00	19.00	23.00	27.00	49.00	
fg_pct	0	1	0.50	0.11	0.11	0.42	0.50	0.57	0.83	
x3p	0	1	0.54	0.97	0.00	0.00	0.00	1.00	7.00	
x3pa	0	1	1.66	1.75	0.00	0.00	1.00	3.00	12.00	
x3p_pct	0	1	0.19	0.31	0.00	0.00	0.00	0.33	1.00	
ft	0	1	6.83	4.08	0.00	4.00	6.00	10.00	26.00	
fta	0	1	8.18	4.63	0.00	5.00	8.00	11.00	27.00	
ft_pct	0	1	0.81	0.21	0.00	0.75	0.83	1.00	1.00	
orb	0	1	1.56	1.44	0.00	0.00	1.00	2.00	8.00	
drb	0	1	4.67	2.57	0.00	3.00	4.00	6.00	14.00	
trb	0	1	6.22	3.02	0.00	4.00	6.00	8.00	18.00	
ast	0	1	5.25	2.72	0.00	3.00	5.00	7.00	17.00	
stl	0	1	2.35	1.66	0.00	1.00	2.00	3.00	10.00	
blk	0	1	0.83	1.01	0.00	0.00	1.00	1.00	6.00	
tov	0	1	2.73	1.73	0.00	1.00	3.00	4.00	9.00	
pf	0	1	2.60	1.39	0.00	2.00	3.00	4.00	6.00	
pts	0	1	30.12	9.75	2.00	23.00	30.00	36.00	69.00	
gm_sc	0	1	23.44	9.49	-1.40	16.80	23.45	29.60	64.60	
close_game	0	1	0.31	0.46	0.00	0.00	0.00	1.00	1.00	
pts_per_minute	0	1	0.79	0.23	0.05	0.63	0.78	0.93	1.57	

This section provides summary statistics including information on missing values, unique values, averages, minimums, maximums, and quartiles for both character and numerical variables in the Michael_df dataset.

```r
# Point Histogram (PTS):
ggplot(Michael_df, aes(x = pts)) +
  geom_histogram(binwidth = 5, fill = "blue", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Points (PTS)",
       x = "Points",
       y = "Frequency",
       caption = "Figure 1: Point Histogram (PTS)")
```

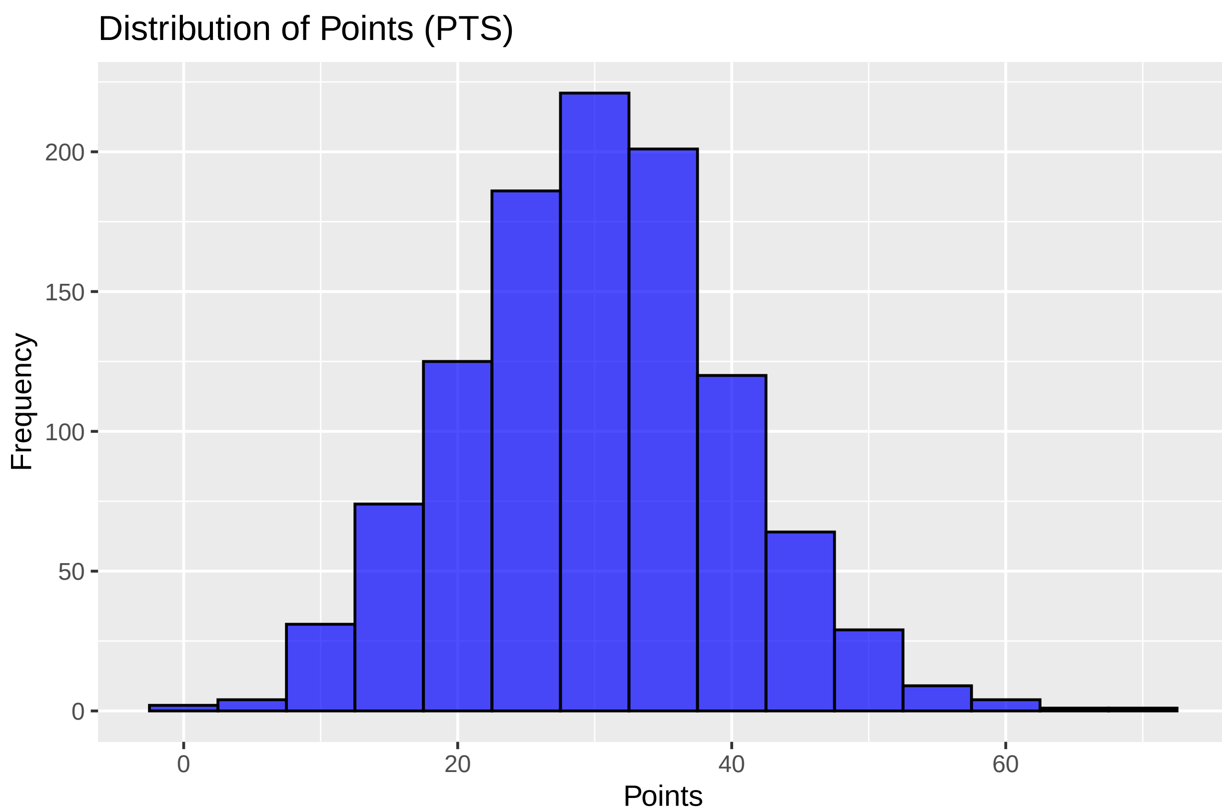


Figure 1: Point Histogram (PTS)

This histogram (Figure 1) displays the highest frequency of Michael Jordan scored within a specific point range. Each bar in the histogram represents a range of points.

```r
ggplot(Michael_df, aes(x = pts_per_minute)) +
  geom_histogram(fill = "blue", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Points per minute",
       x = "Points per minute",
       y = "Frequency",
       caption = "Figure 2: Point per minute Histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

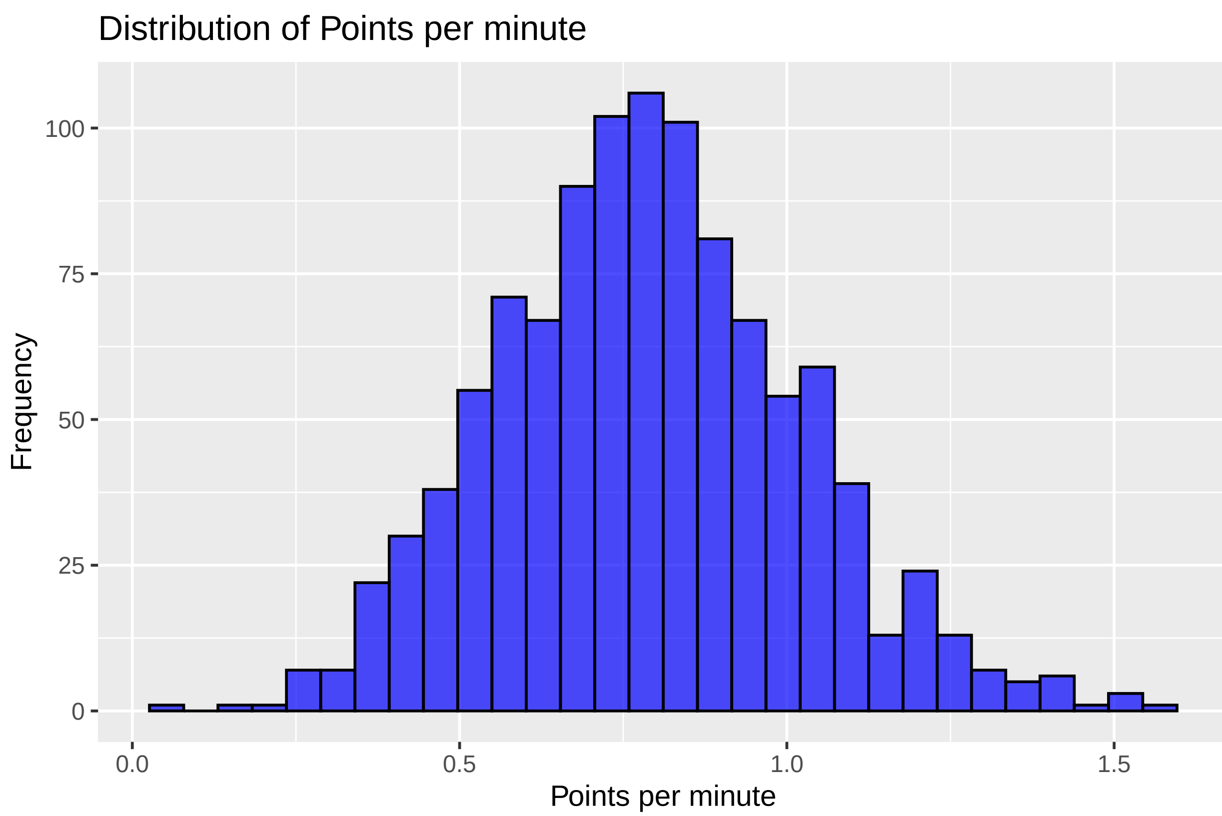
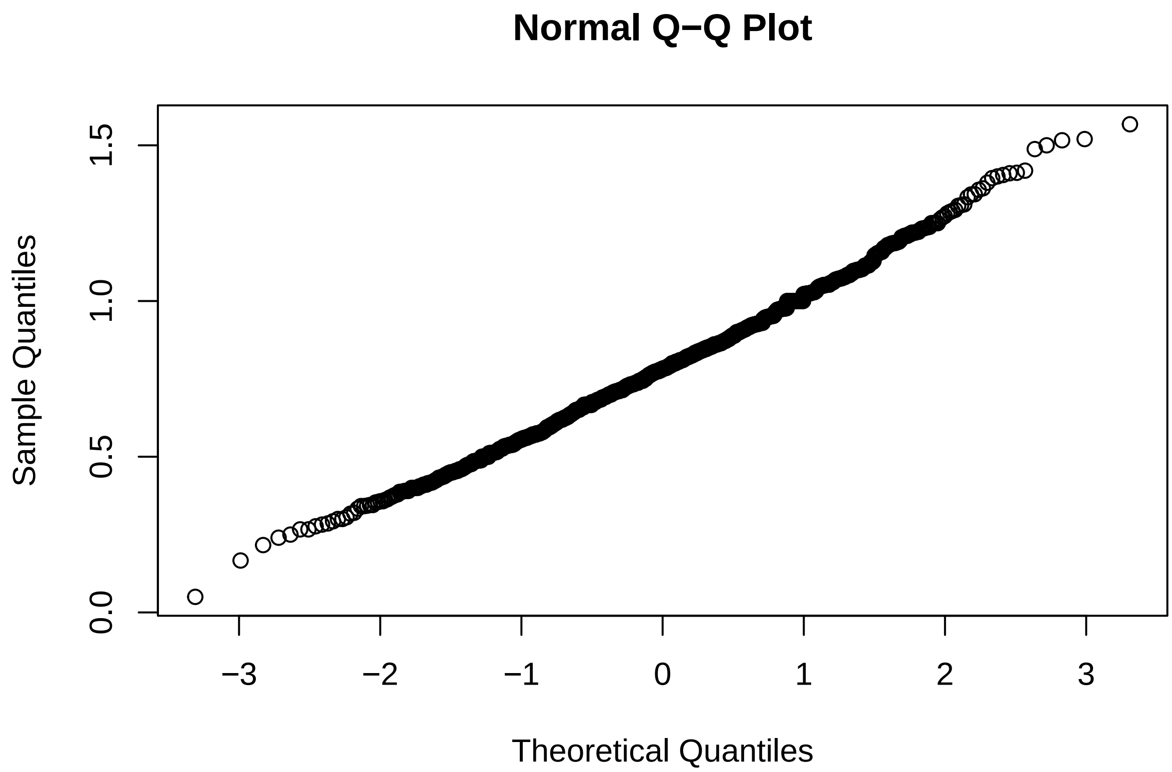
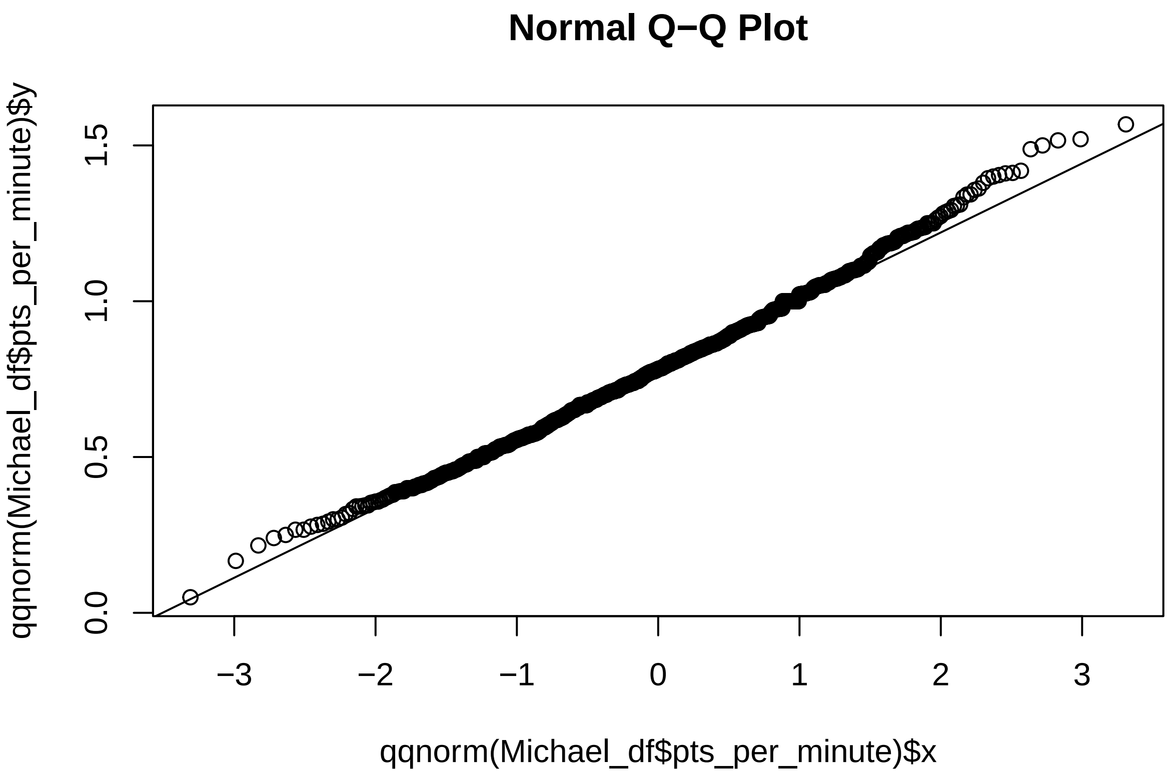


Figure 2: Point per minute Histogram

```
plot(qqnorm(Michael_df$pts_per_minute),
     main = "Normal Q-Q Plot")
```




```
qqline(Michael_df$pts_per_minute)
mtext("Figure 3: Normal Q-Q plot", side = 1, line = 4, cex = 0.8)
mtext("Normal Q-Q plot", side = 0, line = 0, cex = 0.8)
```



Q−Q plot

Figure 3: Normal Q−Q plot

Figures 2 and 3 depict the distribution of points per minute. Figure 2 shows a bell curve on the histogram with a mean around 0.7-0.8 points per minute, resembling a normal distribution. In addition, Figure 3 displays points forming a straight line on the Q-Q plot, suggesting that these observations may conform to a normal distribution.

```r
#Points box plot (PTS) by team:
ggplot(Michael_df, aes(x = tm, y = pts, fill = tm)) +
  geom_boxplot() +
  stat_summary(fun = mean,
               geom = "text", aes(label = round(..y.., 3)),
               size = 3,
               color = "black",
               vjust = -0.4) +
  labs(title = "Boxplot of Points (PTS) by Team",
       x = "Team",
       y = "Points",
       caption = "Figure 4: Boxplot of Points (PTS) by Team",
       fill = 'Team')
```

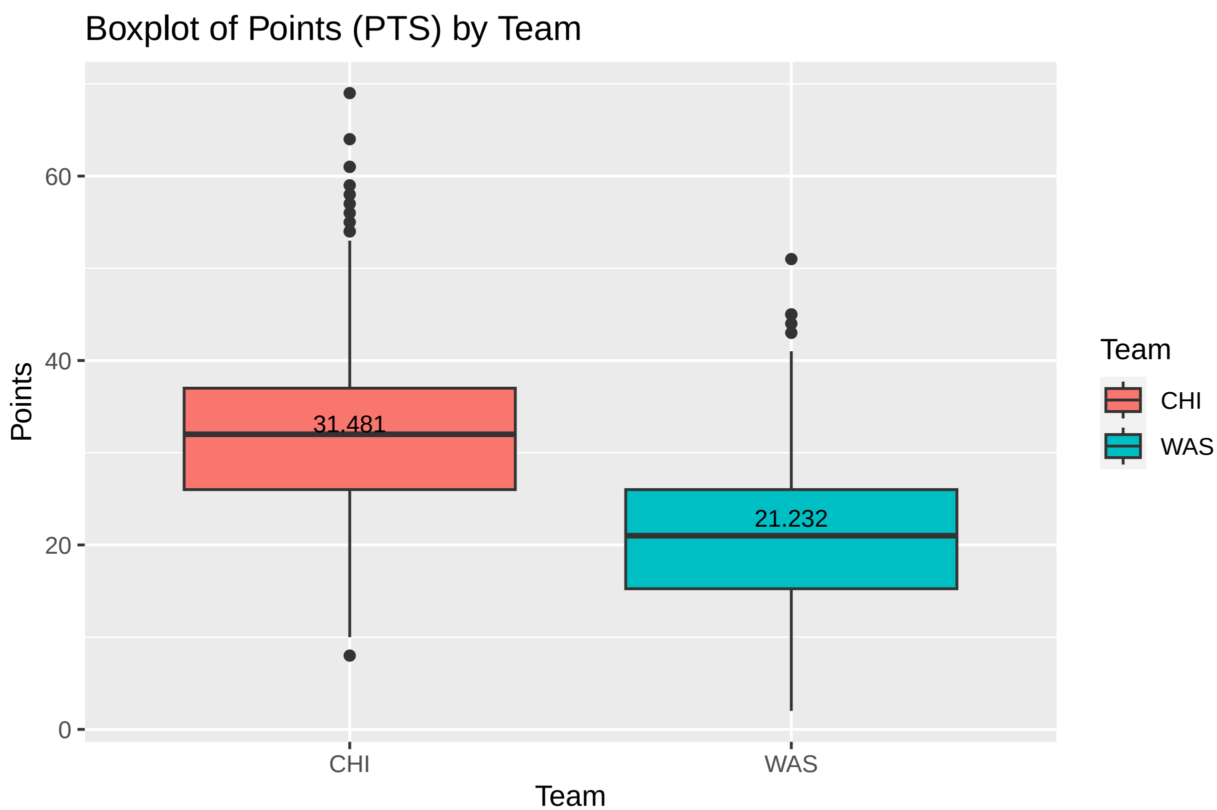


Figure 4: Boxplot of Points (PTS) by Team

This visualization (Figure 4) displays the distribution of points (pts) among different basketball teams (tm), enabling a comparison of their scoring performance by indicating the range and central tendency.

```
#Points per minute box plot by close game:
theGraph <- Michael_df %>%
  filter(tm == 'CHI') %>%
  ggplot(aes(x = as.factor(close_game),
             y = pts_per_minute,
             fill = as.factor(close_game))) +
  geom_boxplot() +
  stat_summary(fun = mean,
               geom = "text", aes(label = round(..y.., 4)),
               size = 3,
               color = "black",
               vjust = -0.4) +
  labs(title = "Boxplot of Points per minute by Close game and Non-close game",
       x = "Close game",
       y = "Points per Minute",
       caption = "Figure 5: Boxplot of Points per minute by Close game and Non-close game",
       fill = 'Close Game')

theGraph
```

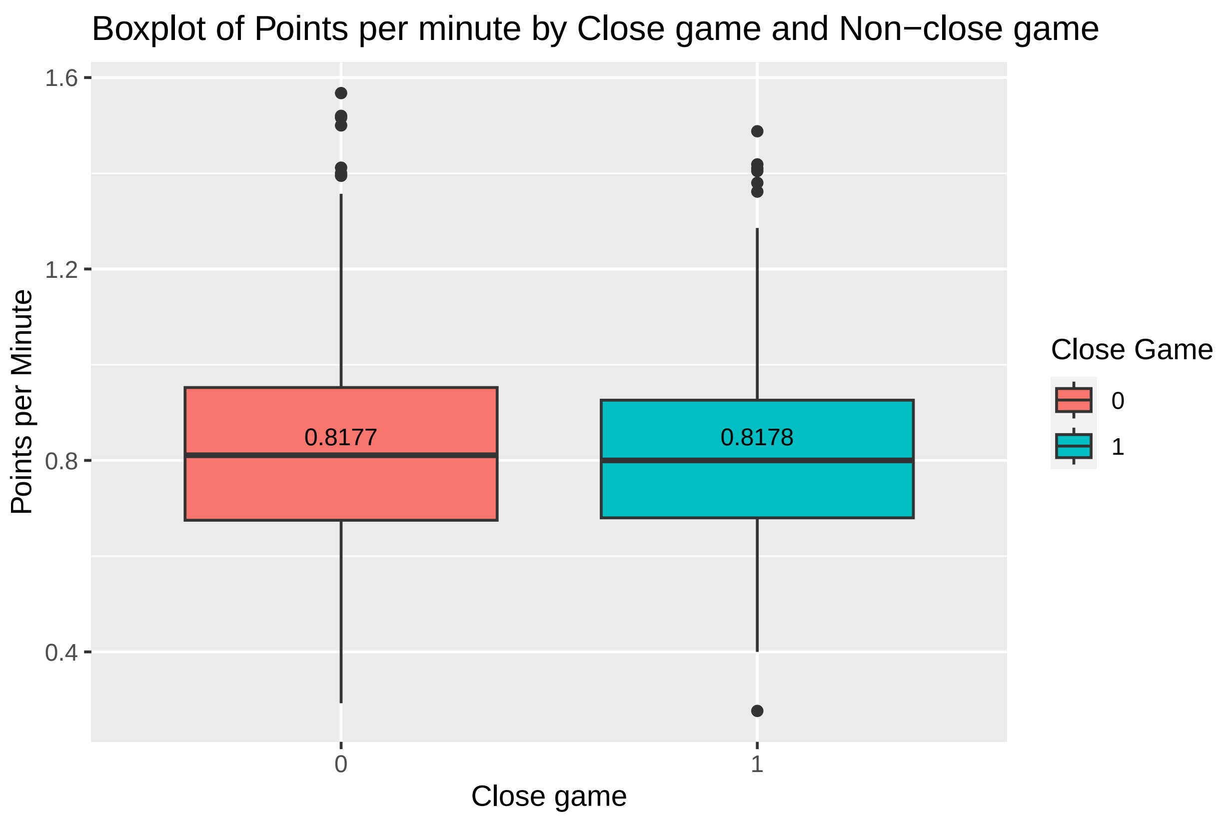


Figure 5: Boxplot of Points per minute by Close game and Non−close game

From Figure 5, the mean points per minute between close games and non-close games are closely similar, hovering around 0.82 points per minute.

```
#Points per minute box plot by opponent teams:
theGraph <- Michael_df %>%
  filter(tm == 'CHI') %>%
  ggplot(aes(x = as.factor(opp),
             y = pts_per_minute,
             fill = as.factor(opp))) +
  geom_boxplot() +
  labs(title = "Boxplot of Points per minute by Close game and Non-close game",
       x = "Opponent Team",
       y = "Points per Minute",
       caption = "Figure 6: Boxplot of Points per minute by Close game and Non-close game",
       fill = 'Close Game') +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))

theGraph
```

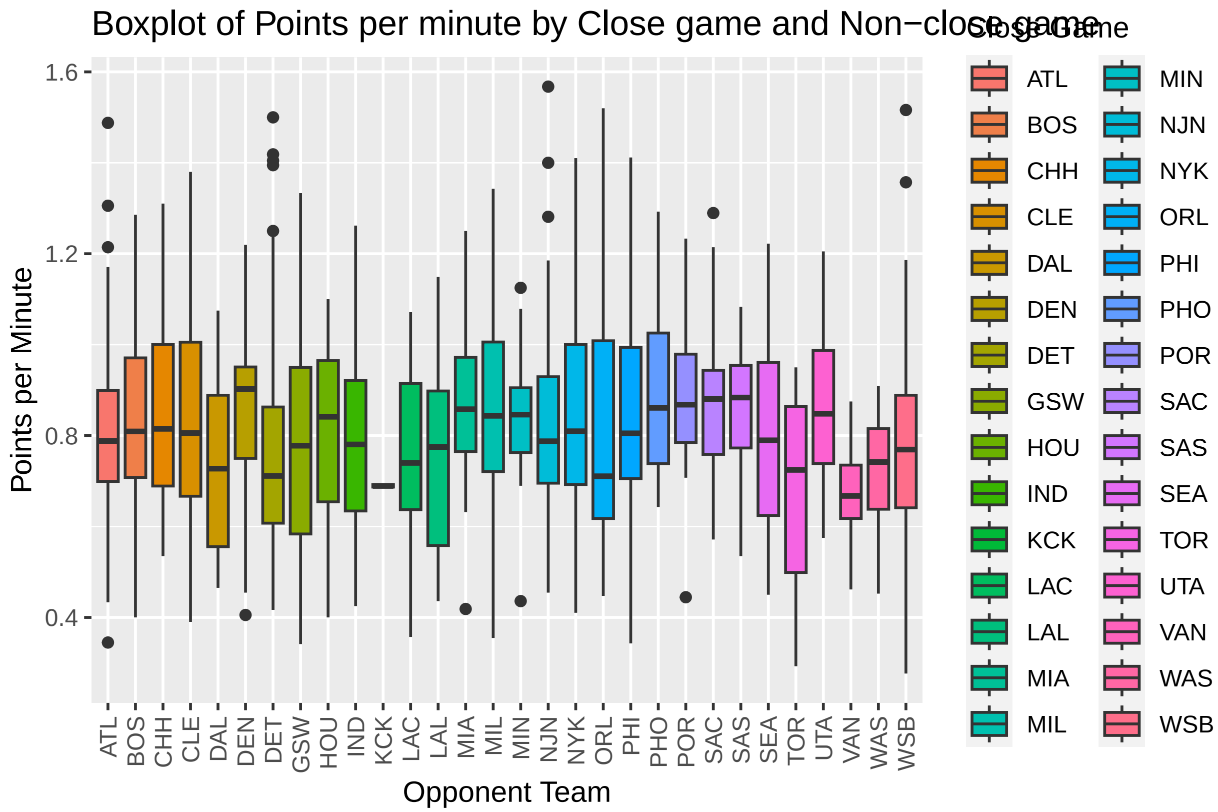


Figure 6: Boxplot of Points per minute by Close game and Non−close game

The box plot in Figure 6 illustrates the mean points per minute with different opponent teams, which varies around 0.7 to 0.9 points per minute.

```
theGraph <- Michael_df %>%
  filter(tm == 'CHI') %>%
  ggplot(aes(x = mp,
             y = pts_per_minute,
             color = as.factor(win))) +
  geom_point(alpha = 0.7) +
  labs(title = "Scatter plot of Points per Minute vs. Minutes Played (MIN) for the Chicago Team",
      x = "Minutes Played",
      y = "Points per Minute",
      caption = "Figure 7: Scatter plot of Points per Minute vs. Minutes Played (MIN) for the Chicago '
      color = 'Win(1) / Lose(0)')

theGraph
```

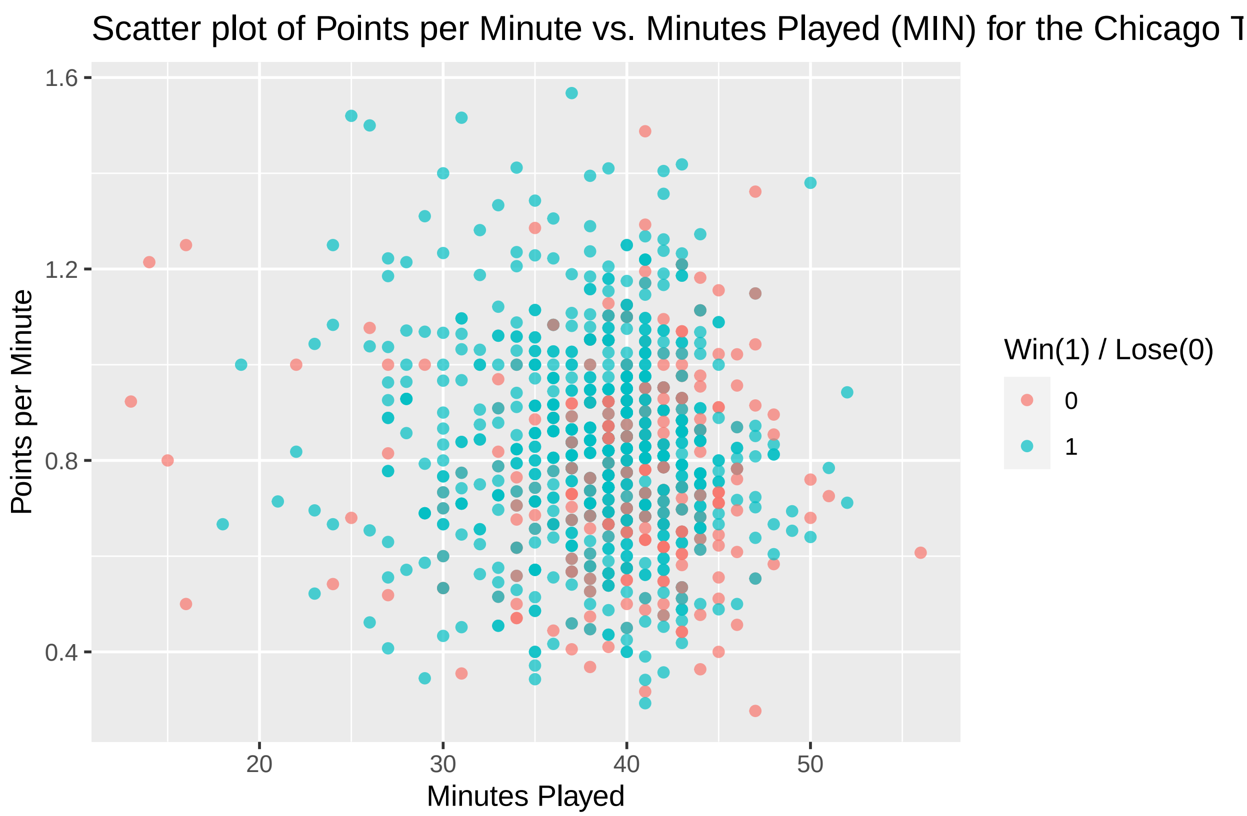


Figure 7: Scatter plot of Points per Minute vs. Minutes Played (MIN) for the Chicago Team

This scatter plot (Figure 7) illustrates the relationship between points per minute and minutes played (MP) during Michael Jordan's time with the Chicago team. It suggests that when he played more than 50 minutes, the points per minute were possibly lower compared to when he played around 35-45 minutes.

```
# Create a bar plot of average gm_sc by end_year
theGraph <- Michael_df %>%
  group_by(years) %>%
  summarise(average_gm_sc = mean(gm_sc, na.rm = TRUE)) %>%
  ggplot(aes(x = years, y = average_gm_sc)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Average Game Scores by Age Year",
       x = "Age Year",
       y = "Average gm_sc",
       caption = "Figure 8: Average Game Scores by Age Year")

theGraph
```

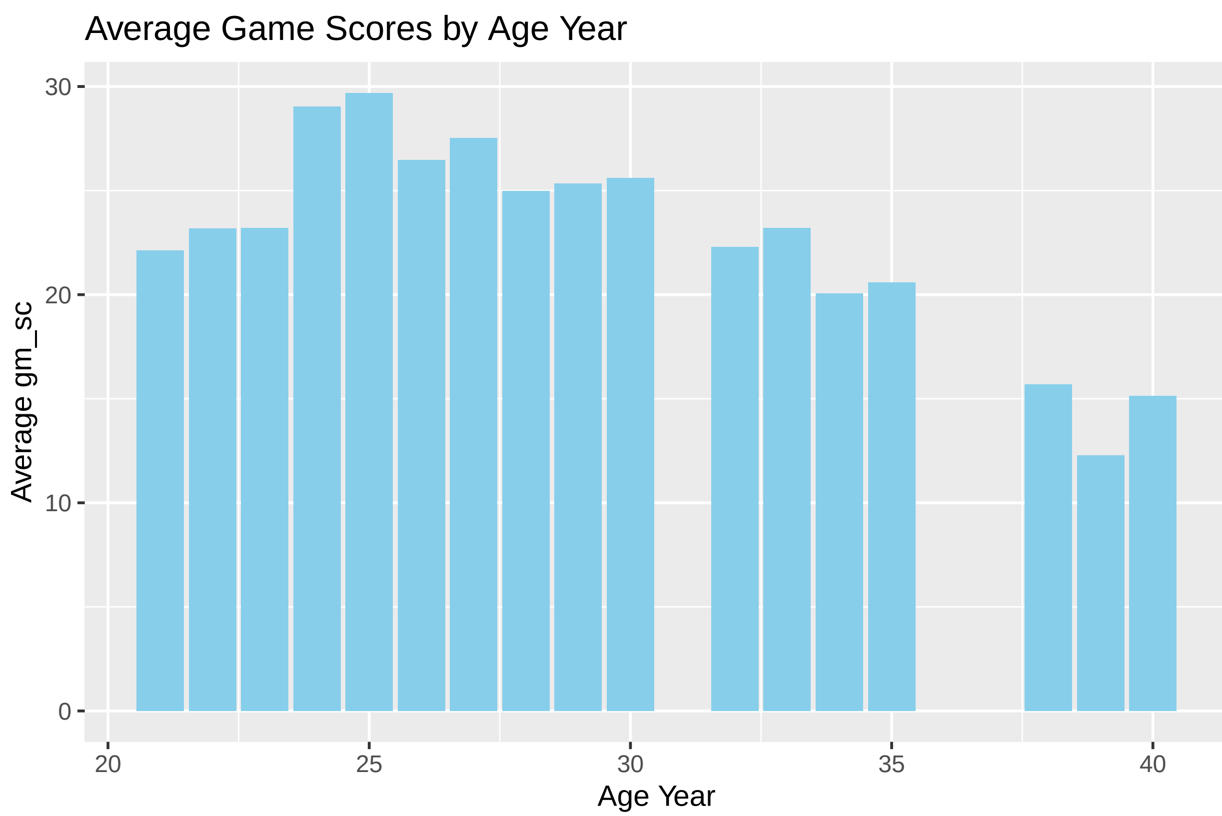


Figure 8: Average Game Scores by Age Year

The average game score of Michael Jordan (Figure 8) peaked when he was 24 and 25. The period when Michael Jordan was 21-23 will be used to create a prediction model for a new potential successful player in the future. Michael Jordan experienced two retirement periods, one when he was 32 and another when he was 36 to 37.

```
#correlation matrix
numeric_columns <- Michael_df %>%
  select_if(is.numeric) %>%
  select_if(~ !is.factor(.))

cor_matrix <- cor(numeric_columns)

cors <- cor(numeric_columns, use = 'pairwise')
#n=8    => 8 colors
#tl.cex => label size
#tl.col => label color
cor_df <- corrplot(cors, type ='upper',
                   col = brewer.pal(n=8, name='RdYlBu'),
                   tl.col = "blue",
                   tl.cex = 0.6)

# Add a caption using mtext
mtext("Figure 9: correlation plot of all the variables", side = 1, line = 4.2, cex = 0.7)
```

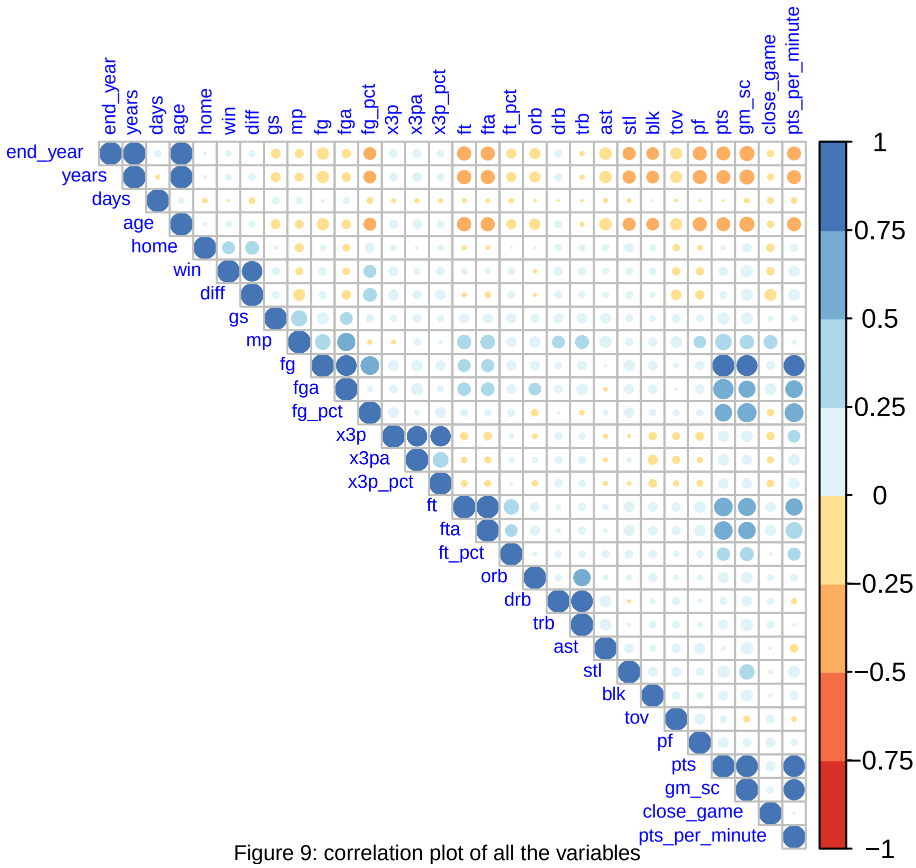
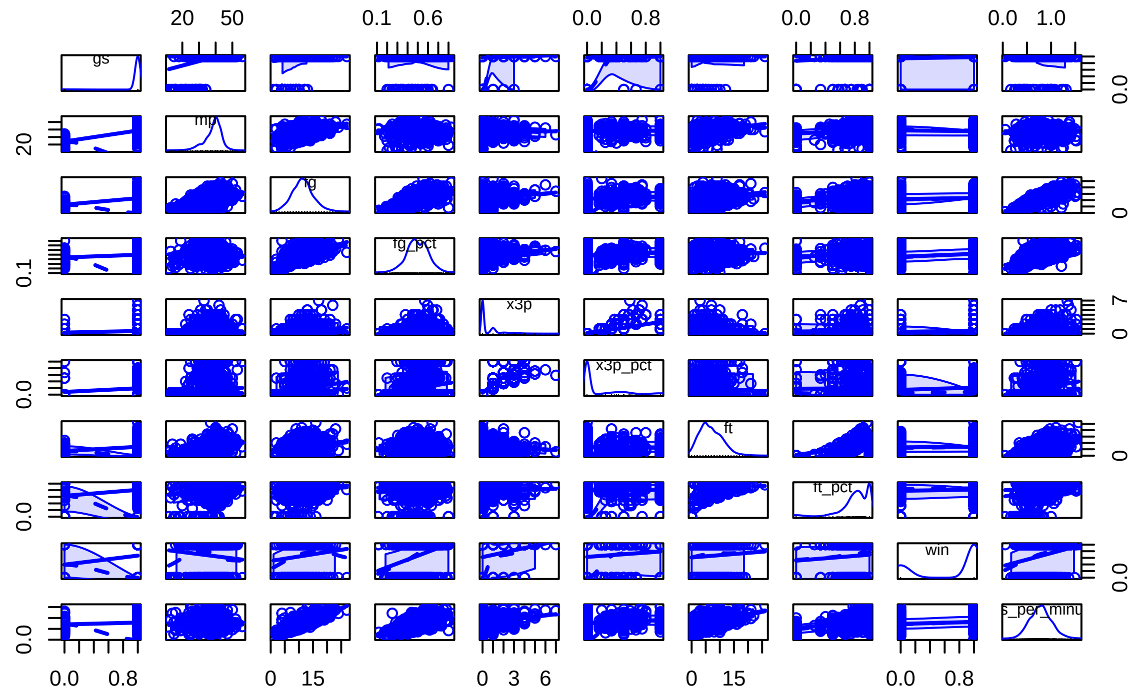


Figure 9: correlation plot of all the variables

From the correlation matrix (Figure 9), game scores show a strong positive correlation with field goals (FG) and points (PTS), while exhibiting a moderate negative correlation with age.

```r
#Scatter plot matrix
scatterplotMatrix(~ gs + mp + fg + fg_pct + x3p + x3p_pct + ft + ft_pct + win + pts_per_minute,
                  data = Michael_df,
                  spread = FALSE,
                  smoother.args = list(lty = 5),
                  main = "Figure 10: Scatter Plot Matrix")
```

Figure 10: Scatter Plot Matrix



From the Scatter Plot Matrix (Figure 10), points per minute show a strong positive relationship with field goals and free throws. Furthermore, the scatter plot for points per minute and 3-pointers forms a bell curve, indicating a peak point in the middle. After that point, points per minute increase, while 3-pointers decrease.

BQ 1: Test whether the mean points per minute are equal between close games and non-close games

Based on: Michael Jordan's statistics played with Chicago Bulls.

Using: t-test.

State the hypotheses -Null Hypothesis (H0): The mean points per minute in close games are equal to the mean points per minute in non-close games.

close = non-close

-Alternative Hypothesis (H1): The mean points per minute in close games are higher than the mean points per minute in non-close games. close <> non-close

```
Michael_CHI_df <- Michael_df %>%
  filter(tm == 'CHI')

Michael_CHI_df$tm   <- as.factor(Michael_CHI_df$tm)
Michael_CHI_df$opp  <- as.factor(Michael_CHI_df$opp)

#Remove irrelevant variables and duplicate variables
```

```r
selected_columns1 <- c('end_year', 'tm', 'age', 'days', 'gm_sc', 'diff', 'name')
Michael_CHI_df <- Michael_CHI_df[, !names(Michael_CHI_df) %in% selected_columns1]
```

```r
# Set the significance level
alpha <- 0.05

#Additional Variables
Michael_df <- Michael_df %>%
  mutate(close_game = ifelse(abs(diff) <= 5, 1, 0)) %>%
  mutate(pts_per_minute = pts / mp)


# t-test
#alternative follow H1 sign
result <- t.test(pts_per_minute ~ close_game, data = Michael_CHI_df, alternative = "two.sided")

# Print the result
print(result)
```

```
##
##  Welch Two Sample t-test
##
## data:  pts_per_minute by close_game
## t = -0.0073272, df = 576.3, p-value = 0.9942
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  -0.02934174  0.02912363
## sample estimates:
## mean in group 0 mean in group 1
##       0.8176988       0.8178078
```

```r


p.value <- result$p.value
p.value
```

Make a decision

```
## [1] 0.9941564
```

```r
#Compare the p-value to alpha and make the decision
ifelse(p.value > alpha,
       "Fail to reject the null hypothesis",
       "Reject the null hypothesis")
```

```
## [1] "Fail to reject the null hypothesis"
```

This indicates that there is not enough evidence to reject the null hypothesis (H0) and supports the claim that the mean points per minute in close games are equal to the mean points per minute in non-close games.

We want to test whether is there any difference performance of Michael Jordan whether the game is competitive or not.

Our result suggests that Jordan's scoring efficiency remained consistent regardless of the game scenario, whether facing pressure or in low-intensity situations, which will be valuable for the basketball team manager to decide who will be the best 5 players to face their opponents when the situation is a close game or a non-close game situation.

BQ1 Recommendation:

Another reason is that if a player is sensitive to a close game such as his performance drops when he plays in a close game situation, the team will decide to take him out and substitute with another player who has a higher tolerance for various situations. The high-tolerance player must be like Michael Jordan who can play consistently to earn more points.

Another situation will be when someone is extremely talented in earning more points in a close game. In order to maximize this player's performance we must put him to play for that period when the points difference between the two teams is less than 5 points.

BQ 2: Test whether the mean points per minute differ across different opponent teams

Based on: Michael Jordan's statistics played with Chicago Bulls.

Using: ANOVA.

State the hypotheses -Null Hypothesis (H0): There is no significant difference in the mean points per minute across different opponent teams.

1 = 2 = 3 = … = n

-Alternative Hypothesis (H1): There is a significant difference in the mean points per minute across different opponent teams. At least one is different

```
# Set the significance level
alpha <- 0.05

# t-test
#alternative follow H1 sign
anova <- aov(pts_per_minute ~ opp, data = Michael_CHI_df)
summary(anova)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## opp          29   1.72 0.05924   1.302  0.133
## Residuals   900  40.95 0.04550
```

```
# Print the result
summary(anova)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## opp          29   1.72 0.05924   1.302  0.133
## Residuals   900  40.95 0.04550
```

```
a.summay <- summary(anova)
p.value <- a.summay[[1]][[1, "Pr(>F)"]]
p.value
```

Make a decision

```
## [1] 0.133058
```

```
#Compare the F-value to alpha and make the decision
ifelse(p.value > alpha,
       "Fail to reject the null hypothesis",
       "Reject the null hypothesis")
```

```
## [1] "Fail to reject the null hypothesis"
```

This indicates that there is not enough evidence to reject the null hypothesis (H0) and supports the claim that there is no significant difference in the mean points per minute across different opponent teams.

In the realm of football, we've observed players who may not excel overall but exhibit exceptional performance when playing against particular teams. We aim to test the hypothesis that these players possess a unique ability or 'magic' specifically associated with certain teams. This investigation is not focused on overall player performance but rather on the extraordinary phenomenon of their scoring abilities, which seem to manifest only against specific teams (potentially more than one)

Our result suggests that Jordan's performance remained consistent regardless of the opposing team, challenging previous notions about matchup dynamics and highlighting his ability to perform consistently against any opponent.

BQ2 Recommendation:

-The basketball manager is seeking a player with consistent performance, akin to Michael Jordan, who consistently earned the same points regardless of the opposing teams.

-The consequence could be that teams might need to focus on overall improvement rather than tailoring strategies based on specific opponents, assuming that the performance doesn't significantly vary.

BQ 3: Test whether winning percentage by year is independent on game score levels

Based on: Michael Jordan's statistics played.

Using: Chi-square

A team winning percentage is defined as a measure that reflects the proportion of games won relative to the total number of games played.

State the hypotheses H0: Winning percentage by year is independent on game score levels.

H1: Winning percentage by year is dependent on game score levels.

Game score levels can be divided by 3 groups: < 24, 25-40, 41+

```
skim(Michael_df)
```

Table 5: Data summary

Name	Michael_df
Number of rows	1072
Number of columns	33

Column type frequency:	
character	3
numeric	30
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
tm	0	1	3	3	0	2	0
opp	0	1	3	3	0	33	0
name	0	1	14	14	0	1	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
end_year	0	1	1992.89	5.36	1985.00	1989.00	1992.00	1997.00	2003.00	
years	0	1	29.28	5.35	21.00	25.00	28.00	33.00	40.00	
days	0	1	201.63	137.58	0.00	42.00	274.00	320.00	365.00	
age	0	1	29.83	5.36	21.69	25.73	28.86	33.92	40.16	
home	0	1	0.50	0.50	0.00	0.00	0.50	1.00	1.00	
win	0	1	0.66	0.47	0.00	0.00	1.00	1.00	1.00	
diff	0	1	4.87	12.81	-44.00	-4.00	5.00	13.00	47.00	
gs	0	1	0.97	0.17	0.00	1.00	1.00	1.00	1.00	
mp	0	1	38.26	5.71	12.00	36.00	39.00	42.00	56.00	
fg	0	1	11.37	3.83	1.00	9.00	11.00	14.00	27.00	
fga	0	1	22.89	5.94	5.00	19.00	23.00	27.00	49.00	
fg_pct	0	1	0.50	0.11	0.11	0.42	0.50	0.57	0.83	
x3p	0	1	0.54	0.97	0.00	0.00	0.00	1.00	7.00	
x3pa	0	1	1.66	1.75	0.00	0.00	1.00	3.00	12.00	
x3p_pct	0	1	0.19	0.31	0.00	0.00	0.00	0.33	1.00	
ft	0	1	6.83	4.08	0.00	4.00	6.00	10.00	26.00	
fta	0	1	8.18	4.63	0.00	5.00	8.00	11.00	27.00	
ft_pct	0	1	0.81	0.21	0.00	0.75	0.83	1.00	1.00	
orb	0	1	1.56	1.44	0.00	0.00	1.00	2.00	8.00	
drb	0	1	4.67	2.57	0.00	3.00	4.00	6.00	14.00	
trb	0	1	6.22	3.02	0.00	4.00	6.00	8.00	18.00	
ast	0	1	5.25	2.72	0.00	3.00	5.00	7.00	17.00	
stl	0	1	2.35	1.66	0.00	1.00	2.00	3.00	10.00	
blk	0	1	0.83	1.01	0.00	0.00	1.00	1.00	6.00	
tov	0	1	2.73	1.73	0.00	1.00	3.00	4.00	9.00	
pf	0	1	2.60	1.39	0.00	2.00	3.00	4.00	6.00	
pts	0	1	30.12	9.75	2.00	23.00	30.00	36.00	69.00	
gm_sc	0	1	23.44	9.49	-1.40	16.80	23.45	29.60	64.60	
close_game	0	1	0.31	0.46	0.00	0.00	0.00	1.00	1.00	
pts_per_minute	0	1	0.79	0.23	0.05	0.63	0.78	0.93	1.57	

```
# Grouping data by end_year and calculating the team winning percentage
```

```r
team_win_percentage <- Michael_df %>%
  mutate(gm_sc_bin = cut(gm_sc, breaks = c(-Inf, 24, 40, Inf), labels = c("0-24", "25-40", "41+"))) %>%
  group_by(end_year, gm_sc_bin) %>%
  summarise(
    total_games = n(),                      # Total number of games played
    total_wins = sum(win),                  # Total number of games won
    team_winning_percentage = total_wins / total_games  # Winning percentage
  )
```

```
## `summarise()` has grouped output by 'end_year'. You can override using the
## `.groups` argument.
```

```r
# Viewing the resulting dataset with team winning percentage
print(team_win_percentage)
```

```
## # A tibble: 40 x 5
## # Groups:   end_year [15]
##    end_year gm_sc_bin total_games total_wins team_winning_percentage
##       <dbl> <fct>           <int>      <dbl>                   <dbl>
##  1     1985 0-24               49         18                   0.367
##  2     1985 25-40              29         17                   0.586
##  3     1985 41+                 4          3                   0.75
##  4     1986 0-24               16          8                   0.5
##  5     1986 25-40               2          1                   0.5
##  6     1987 0-24               32         10                   0.312
##  7     1987 25-40              42         23                   0.548
##  8     1987 41+                 8          7                   0.875
##  9     1988 0-24               24         10                   0.417
## 10     1988 25-40              51         33                   0.647
## # i 30 more rows
```

```r
# Create vectors for each row
r1 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1985 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1985 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1985 & team_win

r2 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1986 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1986 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1986 & team_win

r3 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1987 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1987 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1987 & team_win

r4 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1988 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1988 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1988 & team_win

r5 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1989 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1989 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1989 & team_win
```

```r
r6 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1990 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1990 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1990 & team_win

r7 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1991 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1991 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1991 & team_win

r8 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1992 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1992 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1992 & team_win

r9 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1993 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1993 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1993 & team_win

r10 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1995 & team_wi
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1995 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1995 & team_win

r11 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1996 & team_wi
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1996 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1996 & team_win

r12 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1997 & team_wi
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1997 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1997 & team_win

r13 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1998 & team_wi
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1998 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 1998 & team_win

r14 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 2002 & team_wi
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 2002 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 2002 & team_win

r15 <- c(sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 2003 & team_wi
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 2003 & team_win
        sum(team_win_percentage$team_winning_percentage[team_win_percentage$end_year == 2003 & team_win

# State the number of rows for the matrix
 rows <- 15

# Create a matrix from rows
mtrx <- matrix(c(r1, r2, r3, r4, r5,r6,r7,r8 ,r9 ,r10 ,r11 ,r12 ,r13, r14, r15), nrow = rows, byrow = TR

# Name the rows and columns
rownames(mtrx) <- c("1985", "1986", "1987", "1988", "1989", "1990", "1991", "1992", "1993", "1995", "199
colnames(mtrx) <- c("0-20", "21-40", "40+")

# Print the matrix
print(mtrx)
```

```
##           0-20     21-40       40+
## 1985 0.3673469 0.5862069 0.7500000
## 1986 0.5000000 0.5000000 0.0000000
## 1987 0.3125000 0.5476190 0.8750000
## 1988 0.4166667 0.6470588 1.0000000
## 1989 0.4500000 0.6274510 0.6000000
## 1990 0.4642857 0.7872340 0.7142857
## 1991 0.6875000 0.7708333 1.0000000
## 1992 0.8250000 0.8421053 1.0000000
## 1993 0.6578947 0.7941176 0.6666667
## 1995 0.6923077 1.0000000 0.0000000
## 1996 0.8461538 0.9024390 1.0000000
## 1997 0.7826087 0.9142857 1.0000000
## 1998 0.6833333 0.9545455 0.0000000
## 2002 0.4716981 0.7142857 0.0000000
## 2003 0.4054054 0.8750000 0.0000000
```

Calculation

```
result <- chisq.test(mtrx)
result
```

```
##
##  Pearson's Chi-squared test
##
## data:  mtrx
## X-squared = 4.3736, df = 28, p-value = 1
```

```
p.value <- result$p.value
p.value
```

Make a decision

```
## [1] 0.9999999
```

```
#Compare the p-value to alpha and make the decision
ifelse(p.value > 0.05,
       "Fail to reject the null hypothesis",
       "Reject the null hypothesis")
```

```
## [1] "Fail to reject the null hypothesis"
```

This indicates that there is not enough evidence to reject the null hypothesis (H0) and suggests that winning percentage by year is independent on game score level.

BQ3 Recommendation:

Therefore, we should explore other variables that may have a dependency on winning percentage.

BQ 4: Predict: The binary outcome variable 'successYes/No'

Based on: Statistics of successful players like Michael Jordan and non-successful players like Jabari Parker when they were 21-23, before becoming superstars. Irrelevant variables such as end_year, tm, opp, date, and gm_sc are excluded.

Using: Logistic Regression.

Import Jabari Parker's dataset, a non-successful player

```r
#add the dataset name
full_path_csv = paste(current_wd, "Jarabi_Parker.csv", sep="/")
full_path_csv
```

```
## [1] "C:/Users/user/Desktop/2024 Winter/ALY6015 Intermediate Analytics/Module 6/Jarabi_Parker.csv"
```

```r
#(source: Jabari Parker 2017-18 Game Log, n.d.)
#import csv
Jarabi_df <- read_csv(full_path_csv)
```

```
## Rows: 222 Columns: 34
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (4): Date, Tm, Opp, Name
## dbl (30): EndYear, Rk, G, Years, Days, Age, Home, Win, Diff, GS, MP, FG, FGA...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
head(Jarabi_df, 10)
```

```
## # A tibble: 10 x 34
##    EndYear    Rk     G Date      Years  Days   Age Tm     Home Opp     Win  Diff
##      <dbl> <dbl> <dbl> <chr>     <dbl> <dbl> <dbl> <chr> <dbl> <chr> <dbl> <dbl>
##  1    2015     5     1 11/04/20~    20   234  20.6 MIL       1 PHI       1     4
##  2    2015     6     2 11/06/20~    20   236  20.6 MIL       0 NYK       1     7
##  3    2015     7     3 11/07/20~    20   237  20.6 MIL       1 BRK       1     8
##  4    2015     8     4 11/10/20~    20   240  20.7 MIL       1 BOS       0   -16
##  5    2015    10     5 11/14/20~    20   244  20.7 MIL       1 CLE       1     3
##  6    2015    12     6 11/19/20~    20   249  20.7 MIL       0 CLE       0   -15
##  7    2015    13     7 11/21/20~    20   251  20.7 MIL       0 IND       0   -37
##  8    2015    14     8 11/23/20~    20   253  20.7 MIL       1 DET       1    21
##  9    2015    15     9 11/25/20~    20   255  20.7 MIL       1 SAC       0   -11
## 10    2015    16    10 11/27/20~    20   257  20.7 MIL       0 ORL       0   -24
## # i 22 more variables: GS <dbl>, MP <dbl>, FG <dbl>, FGA <dbl>, FG_PCT <dbl>,
## #   `3P` <dbl>, `3PA` <dbl>, `3P_PCT` <dbl>, FT <dbl>, FTA <dbl>, FT_PCT <dbl>,
## #   ORB <dbl>, DRB <dbl>, TRB <dbl>, AST <dbl>, STL <dbl>, BLK <dbl>,
## #   TOV <dbl>, PF <dbl>, PTS <dbl>, GmSc <dbl>, Name <chr>
```

```r
Jarabi_df <- clean_names(Jarabi_df)

#Add missing data
```

```r
#x3p_pct = 0
Jarabi_df$x3p_pct[is.na(Jarabi_df$x3p_pct) & Jarabi_df$x3pa == 0] <- 0

#ft_pct = 0
Jarabi_df$ft_pct[is.na(Jarabi_df$ft_pct) & Jarabi_df$fta == 0] <- 0

#Additional Variables
Jarabi_df <- Jarabi_df %>%
  mutate(close_game = ifelse(abs(diff) <= 5, 1, 0)) %>%
  mutate(pts_per_minute = pts / mp)

#Remove irrelevant variables
Jarabi_df <- Jarabi_df[, !names(Jarabi_df) %in% selected_columns]

skim(Jarabi_df)
```

Table 8: Data summary

Name	Jarabi_df
Number of rows	222
Number of columns	33
Column type frequency:	
character	3
numeric	30
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
tm	0	1	3	3	0	3	0
opp	0	1	3	3	0	30	0
name	0	1	13	13	0	1	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
end_year	0	1	2016.97	1.33	2015.00	2016.00	2017.00	2018.00	2019.00	
years	0	1	21.55	1.31	20.00	20.00	21.00	23.00	24.00	
days	0	1	242.55	114.47	0.00	234.25	274.00	319.75	364.00	
age	0	1	22.22	1.23	20.64	20.97	21.82	23.63	24.05	
home	0	1	0.50	0.50	0.00	0.00	0.50	1.00	1.00	
win	0	1	0.40	0.49	0.00	0.00	0.00	1.00	1.00	
diff	0	1	-3.06	12.96	-56.00	-11.00	-4.00	7.00	27.00	
gs	0	1	0.64	0.48	0.00	0.00	1.00	1.00	1.00	
mp	0	1	27.55	17.20	0.00	12.00	27.00	41.00	59.00	
fg	0	1	6.18	2.87	0.00	4.00	6.00	8.00	16.00	
fga	0	1	12.59	4.71	1.00	9.00	12.50	16.00	26.00	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
fg_pct	0	1	0.48	0.15	0.00	0.42	0.50	0.57	0.91	
x3p	0	1	0.75	1.09	0.00	0.00	0.00	1.00	5.00	
x3pa	0	1	2.20	1.91	0.00	0.25	2.00	3.00	8.00	
x3p_pct	0	1	0.22	0.29	0.00	0.00	0.00	0.50	1.00	
ft	0	1	2.29	2.14	0.00	0.00	2.00	4.00	10.00	
fta	0	1	3.08	2.61	0.00	1.00	2.00	5.00	11.00	
ft_pct	0	1	0.58	0.39	0.00	0.00	0.67	1.00	1.00	
orb	0	1	1.45	1.23	0.00	0.00	1.00	2.00	6.00	
drb	0	1	4.32	2.51	0.00	2.00	4.00	6.00	13.00	
trb	0	1	5.77	2.87	1.00	4.00	5.50	8.00	15.00	
ast	0	1	2.18	1.72	0.00	1.00	2.00	3.00	9.00	
stl	0	1	0.86	0.94	0.00	0.00	1.00	1.00	4.00	
blk	0	1	0.41	0.62	0.00	0.00	0.00	1.00	3.00	
tov	0	1	1.82	1.50	0.00	1.00	2.00	3.00	7.00	
pf	0	1	2.07	1.38	0.00	1.00	2.00	3.00	6.00	
pts	0	1	15.39	7.00	0.00	10.25	15.00	20.00	36.00	
gm_sc	0	1	11.06	6.73	-3.40	6.28	11.00	15.28	30.40	
close_game	0	1	0.31	0.46	0.00	0.00	0.00	1.00	1.00	
pts_per_minute	0	1	Inf	NaN	0.00	0.31	0.54	1.23	Inf	

Combine 2 dataset and create a variable called "successYes" to present the classification for logistic regression.

In the successYes variable, there will be 0 and 1: - 0 mean non-successful player Jarabi Parker; - 1 mean successful player Michael Jordan.

```
all_df <- rbind(Jarabi_df, Michael_df)
all_df <- all_df %>%
  mutate(successYes = ifelse(name == 'Jarabi Parker', 0,
                             ifelse(name == 'Michael Jordan', 1, NA)))

all_df$tm   <- as.factor(all_df$tm)
all_df$opp  <- as.factor(all_df$opp)
all_df$name <- as.factor(all_df$name)

#skim(all_df)
#compare Michael Jordan and Jarabi Parker while their age between 21-23
young_df <- all_df %>%
  filter(years >= 21 & years <= 23)

#Remove irrelevant variables and duplicate variables
selected_columns <- c('opp', 'name', 'end_year', 'tm', 'age', 'days', 'gm_sc', 'diff')
young_df <- young_df[, !names(young_df) %in% selected_columns]

#check the successYes variable turn into binary variable
head(all_df)
```

```
## # A tibble: 6 x 34
##   end_year years  days   age tm     home opp     win  diff    gs    mp    fg
##      <dbl> <dbl> <dbl> <dbl> <fct> <dbl> <fct> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     2015    20   234  20.6 MIL       1 PHI       1     4     1    21     1
## 2     2015    20   236  20.6 MIL       0 NYK       1     7     1    31     3
## 3     2015    20   237  20.6 MIL       1 BRK       1     8     1    11     4
```

```
## 4     2015    20   240  20.7 MIL       1 BOS       0   -16     1    33     4
## 5     2015    20   244  20.7 MIL       1 CLE       1     3     1     1     5
## 6     2015    20   249  20.7 MIL       0 CLE       0   -15     1    33     6
## # i 22 more variables: fga <dbl>, fg_pct <dbl>, x3p <dbl>, x3pa <dbl>,
## #   x3p_pct <dbl>, ft <dbl>, fta <dbl>, ft_pct <dbl>, orb <dbl>, drb <dbl>,
## #   trb <dbl>, ast <dbl>, stl <dbl>, blk <dbl>, tov <dbl>, pf <dbl>, pts <dbl>,
## #   gm_sc <dbl>, name <fct>, close_game <dbl>, pts_per_minute <dbl>,
## #   successYes <dbl>
```

Removed some variables:

- Duplicate variables:
  - 'years' represents the entire age, therefore 'age' and 'days' also pertain to age.
  - 'gm_sc' (game scores) is calculated from various player statistics like field goals and 3 points; thus, 'game score' has the same meaning as those variables.
  - 'diff' is already defined to indicate a close game.
- Irrelevant variables: 'end_year', 'name', 'opp', 'tm' are categorical variables unique to each player, so they might not be suitable for inclusion in the model.

```r
#Check Class bias
table(young_df$successYes)
```

```
##
##   0   1
## 150 149
```

```r
success_distribution <- round(prop.table(table(young_df$successYes)) * 100, 2)
success_distribution
```

```
##
##     0     1
## 50.17 49.83
```

```r
#histogram of successYes variable
ggplot(young_df, aes(x = as.factor(successYes), fill = as.factor(successYes))) +
  geom_bar() +
  labs(title = "Number of observations for success and non-success categories",
       x = "successYes",
       y = "Count",
       caption = 'Figure 11: Number of observations for success and non-success categories',
       fill = 'SuccessYes') +
  theme_minimal()
```

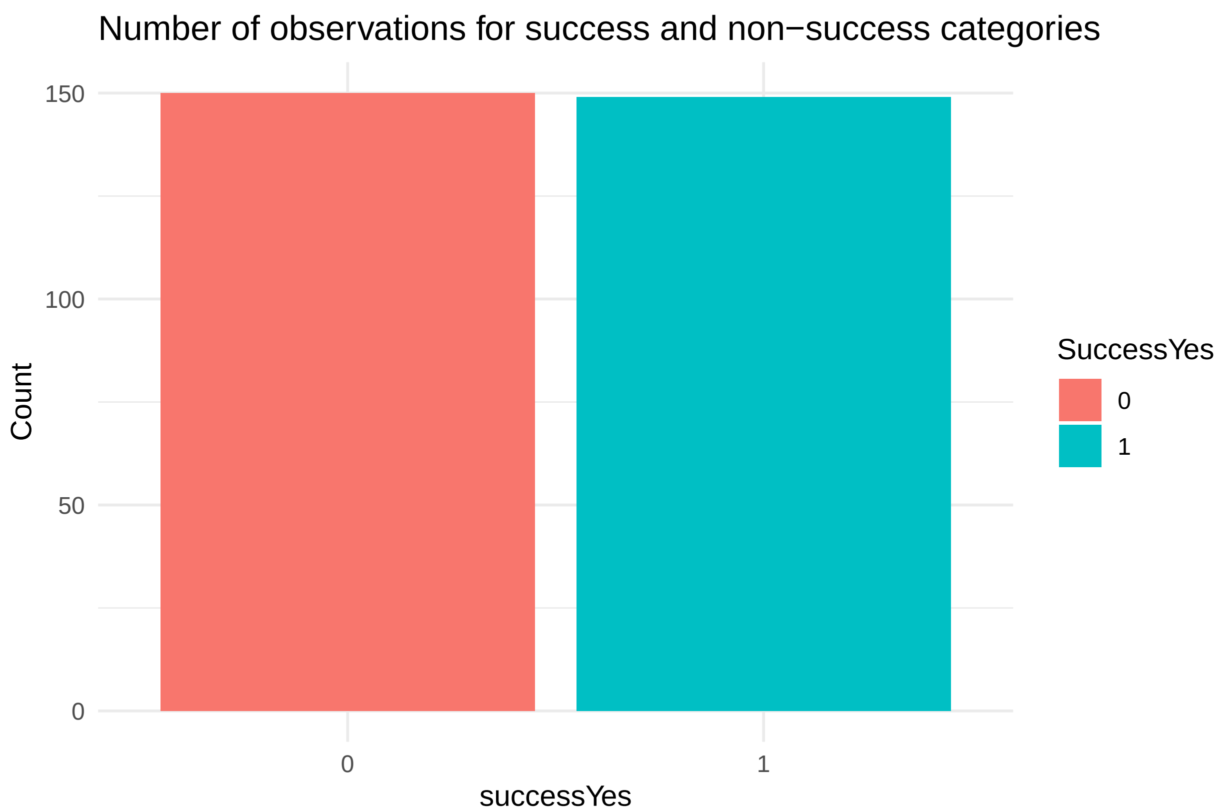


Figure 11: Number of observations for success and non−success categories

According to Figure 11, this dataset does not exhibit class bias as the number of observations for successful and unsuccessful players is almost the same.

```
#Split data into train and test sets
#Train 70%
#Test  30%

set.seed(5)
trainIndex <- createDataPartition(young_df$successYes, p = 0.70, list = FALSE, times = 1)

#don't forget ,
train <- young_df[trainIndex,]
test <- young_df[-trainIndex,]
```

Creating a training and testing dataset with a ratio of 70:30 will be employed to train and test the model, ensuring alignment with the training dataset. Additionally, the test dataset will be utilized to assess whether the model is overfitting or not.

```
#Create model
#successYes cannot be a factor
model <- glm(successYes ~., data = train, family = binomial(link = 'logit'))
summary(model)
```

```
##
## Call:
```

```
## glm(formula = successYes ~ ., family = binomial(link = "logit"),
##     data = train)
##
## Coefficients: (2 not defined because of singularities)
##                   Estimate  Std. Error z value Pr(>|z|)
## (Intercept)        -98.381  542520.763   0.000    1.000
## years               -1.105   42520.288   0.000    1.000
## home               -15.914   47608.304   0.000    1.000
## win                  3.718   82954.712   0.000    1.000
## gs                  -6.720   81941.103   0.000    1.000
## mp                  -1.388    5707.063   0.000    1.000
## fg                  -5.182   78336.219   0.000    1.000
## fga                  9.561   48899.701   0.000    1.000
## fg_pct             190.025 1017588.304   0.000    1.000
## x3p                 45.198  146748.618   0.000    1.000
## x3pa               -40.921   47966.060  -0.001    0.999
## x3p_pct            -60.587  266947.828   0.000    1.000
## ft                   5.093   34699.568   0.000    1.000
## fta                  2.376   32054.474   0.000    1.000
## ft_pct             -12.206  123201.088   0.000    1.000
## orb                -12.317   34228.844   0.000    1.000
## drb                 -9.373   11251.507  -0.001    0.999
## trb                     NA          NA      NA       NA
## ast                 11.761   22515.532   0.001    1.000
## stl                 -6.248   18567.836   0.000    1.000
## blk                  5.984   30804.783   0.000    1.000
## tov                 10.098   28152.246   0.000    1.000
## pf                  11.225   15437.845   0.001    0.999
## pts                     NA          NA      NA       NA
## close_game          -3.057   90433.570   0.000    1.000
## pts_per_minute     -49.971   98687.689  -0.001    1.000
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 290.950363932514  on 209  degrees of freedom
## Residual deviance:   0.000000024085  on 186  degrees of freedom
## AIC: 48
##
## Number of Fisher Scoring iterations: 25
```

Subset selection

```r
#Backward selection method
final_modelB <- step(model, direction = "backward")
```

```r
final_modelB
```

```
##
## Call:  glm(formula = successYes ~ home + fga + x3pa + ft + orb + drb +
##     ast + pts_per_minute, family = binomial(link = "logit"),
##     data = train)
##
## Coefficients:
```

```
##    (Intercept)           home            fga           x3pa             ft
##        -8092.4        -2585.0          749.2        -3714.4         1052.7
##            orb            drb            ast  pts_per_minute
##        -2717.8        -1235.6         2349.8        -2415.2
##
## Degrees of Freedom: 209 Total (i.e. Null);  201 Residual
## Null Deviance:       291
## Residual Deviance: 0.00005024    AIC: 18
```

```r
#Forward selection method
final_modelF <- step(model, direction = "forward")
```

```r
final_modelF
```

```
##
## Call:  glm(formula = successYes ~ years + home + win + gs + mp + fg +
##     fga + fg_pct + x3p + x3pa + x3p_pct + ft + fta + ft_pct +
##     orb + drb + trb + ast + stl + blk + tov + pf + pts + close_game +
##     pts_per_minute, family = binomial(link = "logit"), data = train)
##
## Coefficients:
##    (Intercept)          years           home            win             gs
##        -98.381         -1.105        -15.914          3.718         -6.720
##             mp             fg            fga         fg_pct            x3p
##         -1.388         -5.182          9.561        190.025         45.198
##           x3pa        x3p_pct             ft            fta         ft_pct
##        -40.921        -60.587          5.093          2.376        -12.206
##            orb            drb            trb            ast            stl
##        -12.317         -9.373             NA         11.761         -6.248
##            blk            tov             pf            pts     close_game
##          5.984         10.098         11.225             NA         -3.057
## pts_per_minute
##        -49.971
##
## Degrees of Freedom: 209 Total (i.e. Null);  186 Residual
## Null Deviance:       291
## Residual Deviance: 0.00000002409     AIC: 48
```

```r
#Step wise
final_modelS <- step(model, direction = "both")
```

```r
final_modelS
```

```
##
## Call:  glm(formula = successYes ~ home + fga + x3pa + ft + orb + drb +
##     ast + pts_per_minute, family = binomial(link = "logit"),
##     data = train)
##
## Coefficients:
##    (Intercept)           home            fga           x3pa             ft
##        -8092.4        -2585.0          749.2        -3714.4         1052.7
##            orb            drb            ast  pts_per_minute
```

```
##        -2717.8         -1235.6          2349.8         -2415.2
##
## Degrees of Freedom: 209 Total (i.e. Null);  201 Residual
## Null Deviance:       291
## Residual Deviance: 0.00005024    AIC: 18
```

```r
#Compare AIC
AIC <- c(AIC(final_modelB), AIC(final_modelF), AIC(final_modelS))
model_names <- c("Backward Selection", "Forward Selection", "Stepwise Selection")

best_model_index <- which(AIC == min(AIC))
best_model <- list(final_modelB, final_modelF, final_modelS)[best_model_index]
best_model
```

```
## [[1]]
##
## Call:  glm(formula = successYes ~ home + fga + x3pa + ft + orb + drb +
##     ast + pts_per_minute, family = binomial(link = "logit"),
##     data = train)
##
## Coefficients:
##    (Intercept)            home             fga            x3pa              ft
##        -8092.4         -2585.0           749.2         -3714.4          1052.7
##            orb             drb             ast  pts_per_minute
##        -2717.8         -1235.6          2349.8         -2415.2
##
## Degrees of Freedom: 209 Total (i.e. Null);  201 Residual
## Null Deviance:       291
## Residual Deviance: 0.00005024    AIC: 18
##
## [[2]]
##
## Call:  glm(formula = successYes ~ home + fga + x3pa + ft + orb + drb +
##     ast + pts_per_minute, family = binomial(link = "logit"),
##     data = train)
##
## Coefficients:
##    (Intercept)            home             fga            x3pa              ft
##        -8092.4         -2585.0           749.2         -3714.4          1052.7
##            orb             drb             ast  pts_per_minute
##        -2717.8         -1235.6          2349.8         -2415.2
##
## Degrees of Freedom: 209 Total (i.e. Null);  201 Residual
## Null Deviance:       291
## Residual Deviance: 0.00005024    AIC: 18
```

```r
print(coef(best_model)[1])
```

```
## NULL
```

```r
if (length(best_model_index) > 1) {
  cat("Best Models:", paste(model_names[best_model_index], collapse = ", "), "\n")
```

```r
} else {
  cat("Best Model:", model_names[best_model_index], "\n")
}
```

```
## Best Models: Backward Selection, Stepwise Selection
```

```r
#equation
#(Source: OpenAI, 2024)
coefficients <- round(coef(final_modelS),4)

equation <- paste("log(p / (1 - p)) =",
                  paste(c("Intercept", paste("(", coefficients[-1], "*", names(coefficients[-1]), ")",
                  sep = " ")
equation
```

Model equation

```
## [1] "log(p / (1 - p)) = Intercept + ( -2584.987 * home ) + ( 749.176 * fga ) + ( -3714.4082 * x3pa )
```

Create a confusion matrix

```r
#Train set prediction
set.seed(5)

#type = response => for
probabilities.train <- predict(final_modelS, newdata = train, type = 'response')

#optimal cutoff
optCutOff <- optCutoff(probabilities.train, train$win,  namePos = 1) [1]
cat("Optimal cutoff (threshold) is = ", optCutOff, "\n")
```

```
## Optimal cutoff (threshold) is =  1
```

```r
predicted.classes.min <- as.factor(ifelse(probabilities.train >= optCutOff, '1', '0'))

train$successYes <- factor(train$successYes, levels = levels(predicted.classes.min))

#Model accuracy
conf_matrix <- confusionMatrix(predicted.classes.min, train$successYes, positive = '1')
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 108   4
##          1   0  98
##
```

```
##                Accuracy : 0.981
##                  95% CI : (0.952, 0.9948)
##     No Information Rate : 0.5143
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.9618
##
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 0.9608
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9643
##              Prevalence : 0.4857
##          Detection Rate : 0.4667
##    Detection Prevalence : 0.4667
##       Balanced Accuracy : 0.9804
##
##        'Positive' Class : 1
##
```

In the confusion matrix of the training set, the model accurately predicts 'No' (not success) 108 times and 'Yes' (success) 98 times. However, it also makes errors, specifically in the case of False Negatives (Type II Error). These occur when the actual status is 'Yes' (Success), but the model predicts 'No', happening around 4 times.

```r
#Test set predictions
probabilities.test <- predict(final_modelS, newdata = test, type = 'response')
predicted.classes.min <- as.factor(ifelse(probabilities.test >= optCutOff, '1', '0'))

test$successYes <- factor(test$successYes, levels = levels(predicted.classes.min))

#Model accuracy
conf_matrix_test <- confusionMatrix(predicted.classes.min, test$successYes, positive = '1')
conf_matrix_test
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 40  4
##          1  2 43
##
##                Accuracy : 0.9326
##                  95% CI : (0.859, 0.9749)
##     No Information Rate : 0.5281
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.8651
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.9149
```

```
##              Specificity : 0.9524
##           Pos Pred Value : 0.9556
##           Neg Pred Value : 0.9091
##               Prevalence : 0.5281
##           Detection Rate : 0.4831
##     Detection Prevalence : 0.5056
##        Balanced Accuracy : 0.9336
##
##         'Positive' Class : 1
##
```

In the confusion matrix of the test set, the model accurately predicts 'No' (not success) 40 times and 'Yes' (success) 43 times. However, it makes errors in both categories: False Positives (Type I error) and False Negatives (Type II Error). False Positives occur when the actual status is 'No' (not success), but the model predicts 'Yes', happening around 2 times. False Negatives occur when the actual status is 'Yes' (Success), but the model predicts 'No', happening around 4 times. The model makes slightly more incorrect predictions on the test set compared to the training set.

```r
#Create a dataframe
performance_df <- data.frame(
  Metric = c("Accuracy",
            "Pos Pred Value",
            "Sensitivity",
            "Specificity",
            "Accuracy",
            "Pos Pred Value",
            "Sensitivity",
            "Specificity"),
  Value = c(round(conf_matrix$overall['Accuracy'],4),
           round(conf_matrix$byClass['Pos Pred Value'],4),
           round(conf_matrix$byClass['Sensitivity'],4),
           round(conf_matrix$byClass['Specificity'],4),
           round(conf_matrix_test$overall['Accuracy'],4),
           round(conf_matrix_test$byClass['Pos Pred Value'],4),
           round(conf_matrix_test$byClass['Sensitivity'],4),
           round(conf_matrix_test$byClass['Specificity'],4)),
  Dataset = c('Train Set',
            'Train Set',
            'Train Set',
            'Train Set',
            'Test Set',
            'Test Set',
            'Test Set',
            'Test Set')
)

#Source(Shapiro and Sanchi, 2024)
# Create the line plot
theGraph <- performance_df %>%
  ggplot(aes(x = Metric,
             y = Value,
             group = Dataset,
             color = Dataset)) +
  geom_line() +
```

```
  geom_point(size = 3) +
  theme_minimal() +
  labs(title = "Performance Metrics: Train vs. Test Set",
       x = "Metric",
       y = "Value",
       caption = "Figure 12: Performance Metrics: Train vs. Test Set",)

theGraph
```

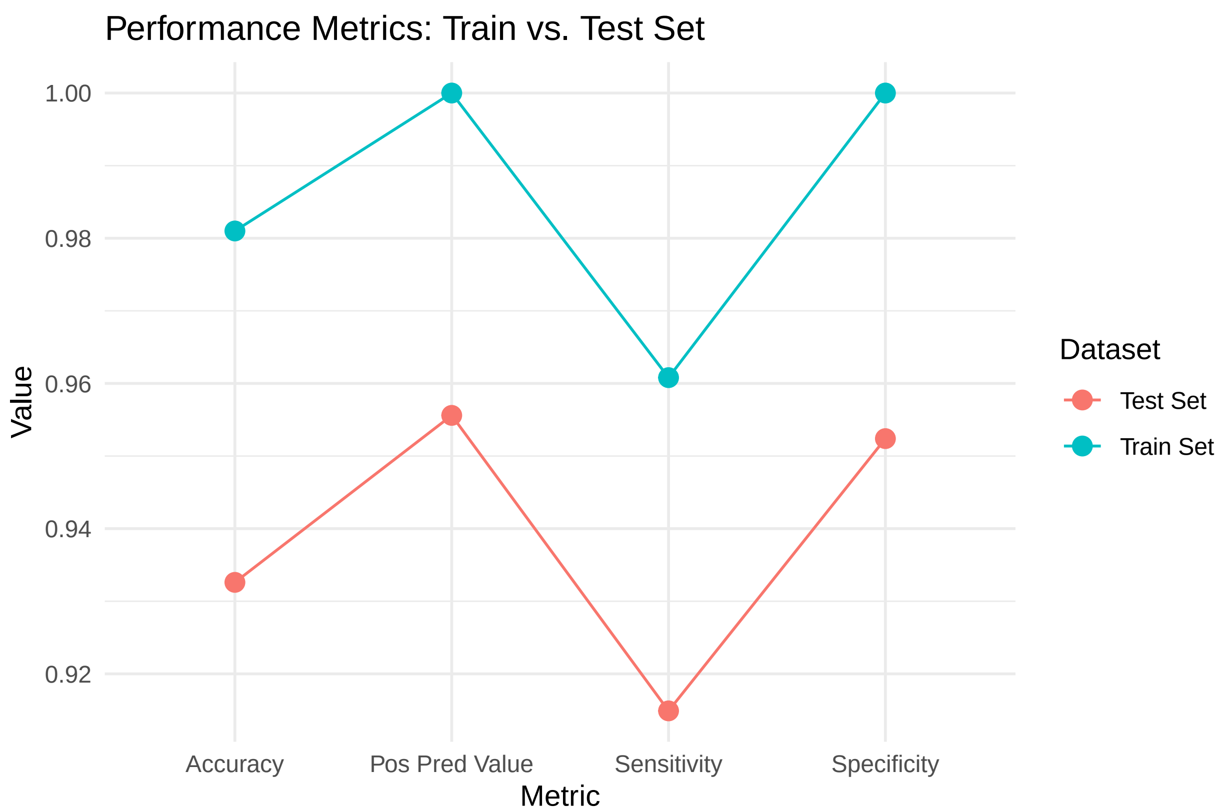


Figure 12: Performance Metrics: Train vs. Test Set

From performance metrics on Figure 12, overall performance on test set is lower than train set which is normal expectation but there is no significant different from 2 train and test set and no significant overfitting on train set.

Plot and interpret the ROC curve.

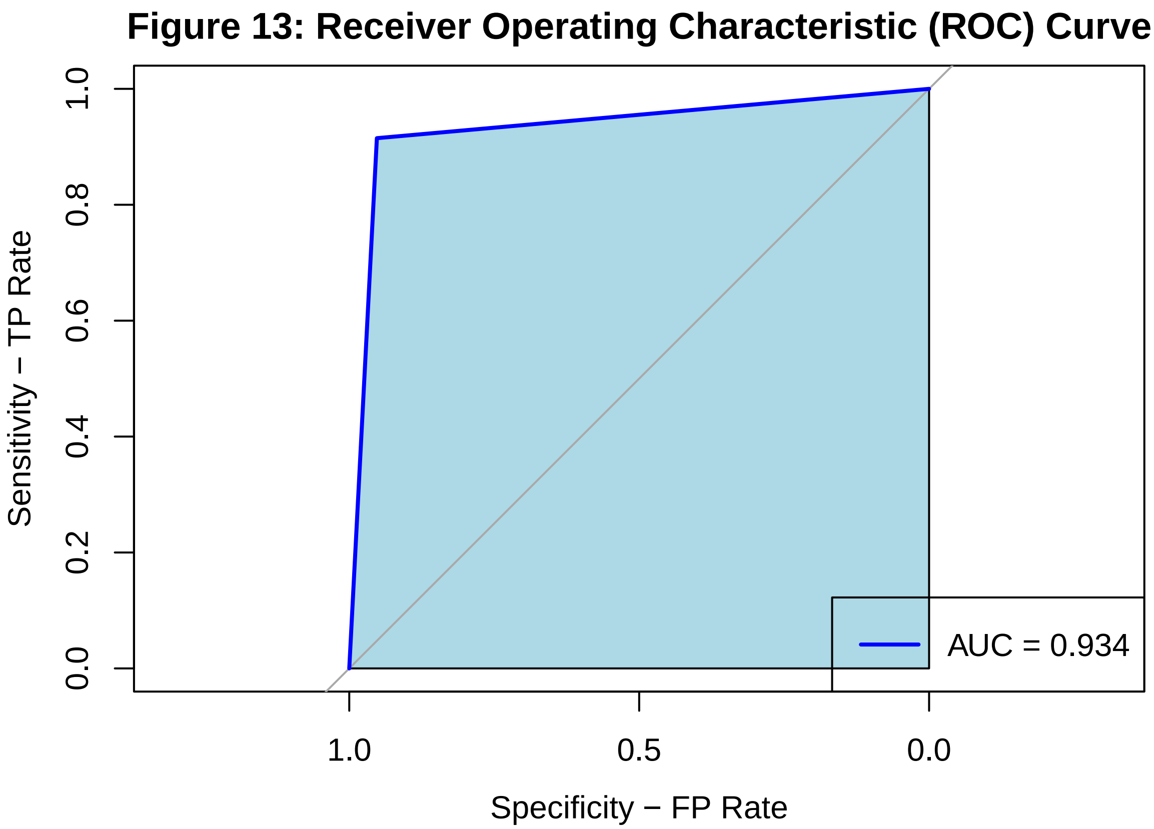
```
ROC1 <- roc(test$successYes, probabilities.test)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
auc <- round(auc(ROC1),3)

#Source(Sharanya and Sharanya, 2024)
plot(ROC1, col = 'blue',
```

```
      ylab= 'Sensitivity - TP Rate',
      xlab = 'Specificity - FP Rate',
      main = 'Figure 13: Receiver Operating Characteristic (ROC) Curve',
      auc.polygon.col = "lightblue",
      auc.polygon = TRUE)
legend("bottomright", legend = paste("AUC =", auc), col = "blue", lwd = 2)
```



In Figure 13, the AUC (area under the ROC curve) is a measure of the model's discrimination, with a value close to 1 indicating perfect discrimination In this case, the AUC is notably high at 0.934, suggesting a well-discriminating model.

BQ4 Recommendation:

Scouts use the model to discover new, talented basketball players and negotiate contracts at lower prices.

BQ 5: Predict: The binary outcome variable 'win' (0 or1)

Based on: Michael Jordan's statistics playing with Chicago Bulls, while excluding irrelevant variables such as end_year, tm, date, and gm_sc.

Using: Logistic Regression.

```
#Check Class bias
table(Michael_CHI_df$win)
```

```
##
```

```
##   0   1
## 291 639
```

```
success_distribution <- round(prop.table(table(Michael_CHI_df$win)) * 100, 2)
success_distribution
```

```
##
##     0     1
## 31.29 68.71
```

```
#histogram of successYes variable
ggplot(Michael_CHI_df, aes(x = as.factor(win), fill = as.factor(win))) +
  geom_bar() +
  labs(title = "Number of observations for win and lose categories",
       x = "Win/Lose",
       y = "Count",
       caption = 'Figure 14: Number of observations for win and lose categories',
       fill = 'Win') +
  theme_minimal()
```

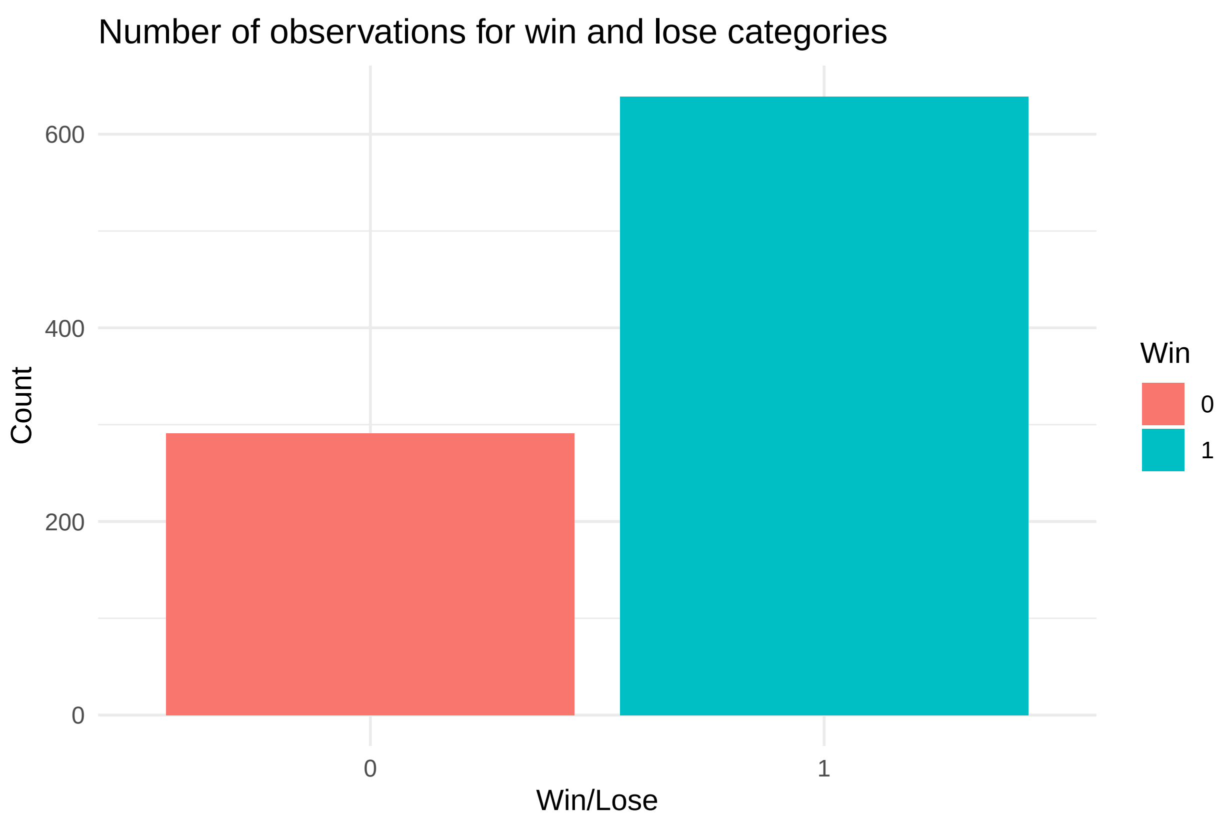


Figure 14: Number of observations for win and lose categories

In figure 14, there is a class bias with win/lose ratio of 31.29% to 68.71%. During the next sample selection process, it is important to ensure equal representation of this variable.

```r
#Increase the numbers of the minority class ('No')
#(Source: Klopper, 2019)
unbiased_df <- ovun.sample(win ~ .,
                           data = Michael_CHI_df,
                           method = "over",
                           seed = 123)$data

table(unbiased_df$win)
```

```
##
##   0   1
## 631 639
```

```r
#Split data into train and test sets
#Train 70%
#Test  30%

set.seed(5)
trainIndex <- createDataPartition(unbiased_df$win, p = 0.70, list = FALSE, times = 1)

#don't forget ,
train <- unbiased_df[trainIndex,]
test <- unbiased_df[-trainIndex,]
```

```r
model <- glm(win ~., data = train, family = binomial(link = 'logit'))
summary(model)
```

```
##
## Call:
## glm(formula = win ~ ., family = binomial(link = "logit"), data = train)
##
## Coefficients: (2 not defined because of singularities)
##                   Estimate Std. Error z value            Pr(>|z|)
## (Intercept)      -9.417353   3.356741  -2.806             0.00502 **
## years             0.228226   0.029269   7.797 0.00000000000000632 ***
## home              1.138027   0.184379   6.172 0.00000000067345779 ***
## oppBOS           -0.939412   0.518608  -1.811             0.07008 .
## oppCHH            1.229987   0.676380   1.818             0.06899 .
## oppCLE            0.063319   0.454864   0.139             0.88929
## oppDAL            1.091763   0.738277   1.479             0.13919
## oppDEN            0.113357   0.679053   0.167             0.86742
## oppDET            0.016472   0.456942   0.036             0.97124
## oppGSW            0.173585   0.674758   0.257             0.79698
## oppHOU           -1.189451   0.633875  -1.876             0.06059 .
## oppIND           -0.081072   0.456158  -0.178             0.85894
## oppKCK           17.400409 815.969694   0.021             0.98299
## oppLAC            2.282914   0.936901   2.437             0.01482 *
## oppLAL           -0.441478   0.573074  -0.770             0.44108
## oppMIA            0.061094   0.567213   0.108             0.91423
## oppMIL            0.740799   0.497039   1.490             0.13611
## oppMIN            1.826892   1.191003   1.534             0.12505
## oppNJN            0.035416   0.481870   0.073             0.94141
```

```
## oppNYK           0.640687   0.460023   1.393              0.16370
## oppORL          -0.430368   0.570549  -0.754              0.45067
## oppPHI          -0.271820   0.494456  -0.550              0.58250
## oppPHO          -0.112279   0.666242  -0.169              0.86617
## oppPOR          -0.092327   0.663029  -0.139              0.88925
## oppSAC           0.835840   0.731451   1.143              0.25316
## oppSAS           0.358595   0.638410   0.562              0.57432
## oppSEA           1.183411   0.671004   1.764              0.07779 .
## oppTOR          -0.446949   0.749758  -0.596              0.55109
## oppUTA          -0.004256   0.604950  -0.007              0.99439
## oppVAN          14.072296 537.626989   0.026              0.97912
## oppWAS          -0.560808   1.082827  -0.518              0.60452
## oppWSB           1.218832   0.539477   2.259              0.02387 *
## gs               1.753628   0.879082   1.995              0.04606 *
## mp              -0.023035   0.074511  -0.309              0.75721
## fg              -0.057437   0.217499  -0.264              0.79172
## fga             -0.026943   0.089486  -0.301              0.76335
## fg_pct           5.323707   3.982498   1.337              0.18130
## x3p              0.779309   0.259575   3.002              0.00268 **
## x3pa            -0.374768   0.089845  -4.171 0.0000302912013333 ***
## x3p_pct         -0.892425   0.497257  -1.795              0.07270 .
## ft               0.171134   0.138038   1.240              0.21506
## fta             -0.110928   0.103524  -1.072              0.28393
## ft_pct          -0.108366   0.836861  -0.129              0.89697
## orb             -0.069309   0.060705  -1.142              0.25357
## drb              0.029099   0.037602   0.774              0.43902
## trb                    NA         NA      NA                   NA
## ast              0.073711   0.035755   2.062              0.03925 *
## stl              0.055954   0.055087   1.016              0.30976
## blk              0.219149   0.093306   2.349              0.01884 *
## tov             -0.098674   0.055043  -1.793              0.07303 .
## pf              -0.164803   0.070180  -2.348              0.01886 *
## pts                    NA         NA      NA                   NA
## close_game      -0.614696   0.190191  -3.232              0.00123 **
## pts_per_minute   0.942805   3.466856   0.272              0.78566
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1232.16  on 888  degrees of freedom
## Residual deviance:  860.56  on 837  degrees of freedom
## AIC: 964.56
##
## Number of Fisher Scoring iterations: 14
```

```r
##Step wise selection method
final_modelS <- step(model, direction = "both")
```

```r
final_modelS
```

```
##
## Call:  glm(formula = win ~ years + home + opp + gs + mp + fga + fg_pct +
```

```
##     x3p + x3pa + x3p_pct + ft + fta + ast + blk + tov + pf +
##     close_game, family = binomial(link = "logit"), data = train)
##
## Coefficients:
## (Intercept)        years         home       oppBOS       oppCHH       oppCLE
##    -8.61386      0.23218      1.15807     -0.89279      1.15006      0.02108
##      oppDAL       oppDEN       oppDET       oppGSW       oppHOU       oppIND
##     1.05432      0.10297     -0.02306      0.15746     -1.18418     -0.08913
##      oppKCK       oppLAC       oppLAL       oppMIA       oppMIL       oppMIN
##    17.35795      2.31606     -0.49613      0.03307      0.74358      1.75565
##      oppNJN       oppNYK       oppORL       oppPHI       oppPHO       oppPOR
##    -0.02684      0.60322     -0.46559     -0.27442     -0.10932     -0.09629
##      oppSAC       oppSAS       oppSEA       oppTOR       oppUTA       oppVAN
##     0.89673      0.43820      1.15983     -0.62740     -0.01978     14.04345
##      oppWAS       oppWSB           gs           mp          fga       fg_pct
##    -0.55825      1.19201      1.69461     -0.04262     -0.03277      5.13004
##         x3p         x3pa      x3p_pct           ft          fta          ast
##     0.80982     -0.36957     -0.89717      0.18819     -0.10570      0.08424
##         blk          tov           pf   close_game
##     0.22997     -0.08738     -0.16111     -0.61178
##
## Degrees of Freedom: 888 Total (i.e. Null);  843 Residual
## Null Deviance:       1232
## Residual Deviance: 863.5     AIC: 955.5
```

```r
#equation
#(Source: OpenAI, 2024)
coefficients <- round(coef(final_modelS),4)

equation <- paste("log(p / (1 - p)) =",
                  paste(c("Intercept", paste("(", coefficients[-1], "*", names(coefficients[-1]), ")",
                  sep = " ")
equation
```

Model equation

```
## [1] "log(p / (1 - p)) = Intercept + ( 0.2322 * years ) + ( 1.1581 * home ) + ( -0.8928 * oppBOS ) +
```

Create a confusion matrix

```r
#Train set prediction
set.seed(5)

#type = response => for
probabilities.train <- predict(final_modelS, newdata = train, type = 'response')

#optimal cutoff
optCutOff <- optCutoff(probabilities.train, train$win,  namePos = 1) [1]
cat("Optimal cutoff (threshold) is = ", optCutOff, "\n")
```

```
## Optimal cutoff (threshold) is =  0.4852535
```

```r
predicted.classes.min <- as.factor(ifelse(probabilities.train >= optCutOff, '1', '0'))

train$win <- factor(train$win, levels = levels(predicted.classes.min))

#Model accuracy
conf_matrix <- confusionMatrix(predicted.classes.min, train$win, positive = '1')
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 360 116
##          1  92 321
##
##                Accuracy : 0.766
##                  95% CI : (0.7368, 0.7935)
##     No Information Rate : 0.5084
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.5315
##
##  Mcnemar's Test P-Value : 0.1108
##
##             Sensitivity : 0.7346
##             Specificity : 0.7965
##          Pos Pred Value : 0.7772
##          Neg Pred Value : 0.7563
##              Prevalence : 0.4916
##          Detection Rate : 0.3611
##    Detection Prevalence : 0.4646
##       Balanced Accuracy : 0.7655
##
##        'Positive' Class : 1
##
```

In the confusion matrix of the training set, the model accurately predicts 'No' (Lose) 360 times and 'Yes' (Win) 321 times. However, it exhibits a moderate error rate in both categories: False Positives (Type I error) and False Negatives (Type II Error), resulting in an accuracy rate on the training set of around 77%, which could determine whether the prediction model is reliable enough or not or needing more player statistics.

Test set

```r
#Test set predictions
probabilities.test <- predict(final_modelS, newdata = test, type = 'response')
predicted.classes.min <- as.factor(ifelse(probabilities.test >= optCutOff, '1', '0'))
probabilities.test
```

```
##          1          4          9         11         14         18         19
## 0.69485724 0.08213066 0.51280047 0.54611110 0.06096489 0.72871870 0.85031295
##         21         22         26         29         35         36         38
## 0.30098216 0.78185322 0.85436317 0.04260856 0.23204667 0.62746748 0.24220857
##         46         47         50         52         53         55         59
```

```
## 0.84554839 0.17075573 0.24322332 0.24223441 0.67836402 0.70573612 0.42000057
##         60         64         65         76         80         82         84
## 0.48793401 0.45796127 0.49882731 0.39416979 0.91652967 0.29996207 0.95519784
##         87         88         95         96        101        104        106
## 0.52669428 0.68192147 0.02166314 0.28117671 0.74768446 0.87264650 0.87020072
##        109        115        124        131        136        140        144
## 0.94928176 0.26054452 0.53616236 0.59721035 0.24888415 0.68411917 0.60466301
##        145        147        149        150        151        160        162
## 0.40454405 0.64063042 0.50609244 0.75851903 0.36870034 0.84715709 0.93406574
##        165        166        167        168        173        179        180
## 0.75385058 0.46312040 0.63936608 0.73142263 0.95130196 0.63736884 0.46952449
##        182        186        190        193        195        196        198
## 0.97315487 0.48427694 0.09293967 0.80358693 0.71198080 0.91884192 0.85309908
##        200        203        205        209        210        211        217
## 0.48506860 0.40844762 0.36599949 0.13348542 0.97323441 0.51442695 0.70195116
##        218        221        223        229        230        236        238
## 0.24075615 0.16114135 0.74133094 0.78350604 0.26615157 0.70732751 0.54652497
##        241        244        254        256        257        267        268
## 0.38676683 0.94808843 0.94365843 0.79679711 0.65904955 0.36395128 0.93708397
##        270        271        273        275        276        277        279
## 0.76494990 0.26352519 0.81834409 0.50793256 0.83769222 0.97516950 0.67420603
##        283        284        288        292        293        294        297
## 0.81975739 0.94221792 0.90538800 0.75301639 0.79005630 0.53270822 0.90767013
##        301        302        306        311        315        316        322
## 0.69238254 0.65284311 0.94482961 0.81612861 0.86121168 0.55852440 0.40665248
##        330        331        335        336        338        342        349
## 0.85907965 0.42581390 0.50324542 0.95888725 0.83797629 0.68205205 0.90956444
##        351        354        366        367        368        371        372
## 0.24134125 0.70941537 0.63377397 0.91517246 0.32181216 0.27704381 0.92642711
##        376        377        379        383        384        386        388
## 0.64730660 0.79316175 0.85965905 0.37378003 0.98807077 0.14962420 0.78169736
##        390        393        399        403        405        406        408
## 0.88944095 0.69590441 0.55947523 0.79486044 0.90733173 0.36524293 0.94621823
##        411        413        415        416        418        428        430
## 0.94510937 0.48659294 0.93381077 0.22957278 0.17380910 0.59004543 0.69318473
##        431        451        453        456        460        462        464
## 0.65663250 0.91682167 0.59869881 0.38978948 0.62370382 0.59166344 0.96127002
##        465        467        475        476        477        478        486
## 0.96572508 0.72363860 0.02563356 0.80654880 0.57425651 0.59335873 0.98746243
##        487        488        493        498        501        502        504
## 0.84116041 0.88692375 0.56363961 0.99225073 0.97065465 0.15382894 0.92960359
##        507        514        519        520        521        525        526
## 0.98450270 0.61491781 0.77534553 0.75855896 0.95916835 0.96420774 0.99335662
##        530        536        543        545        551        552        553
## 0.83290494 0.86648661 0.96790920 0.95542981 0.94716703 0.33657833 0.84222791
##        554        557        560        562        566        568        572
## 0.96106958 0.91347775 0.91738334 0.59013677 0.79901375 0.88937456 0.95865445
##        574        579        580        583        587        589        592
## 0.95147603 0.49404625 0.65249124 0.71983629 0.44534120 0.93834356 0.90175235
##        593        600        602        603        614        619        621
## 0.74570672 0.76358454 0.74280640 0.93767606 0.92371096 0.95714243 0.97611791
##        623        625        626        630        632        634        642
## 0.93125165 0.85714092 0.59584386 0.50917388 0.86108598 0.99113085 0.18709637
##        648        655        656        660        662        663        668
```

```
## 0.06410097 0.03348516 0.23336063 0.04840316 0.10650521 0.29288743 0.35785155
##        674        683        689        690        696        698        700
## 0.06367428 0.29288743 0.20014802 0.14951153 0.12457003 0.67233054 0.77265270
##        706        707        717        719        720        722        726
## 0.03348516 0.84722833 0.08718683 0.42153682 0.15015729 0.04682345 0.13106718
##        729        733        734        736        738        745        752
## 0.10801380 0.83013319 0.14835792 0.28294336 0.41258195 0.13060848 0.57620924
##        756        757        761        763        766        768        770
## 0.36438090 0.41621882 0.34111005 0.17546401 0.19805414 0.43751707 0.80413542
##        774        776        777        778        782        786        787
## 0.14790324 0.62362965 0.42106563 0.11306667 0.04758851 0.57849292 0.64492726
##        790        791        792        793        794        804        807
## 0.36054660 0.77265270 0.04017532 0.06797509 0.11306667 0.83013319 0.21706876
##        811        815        827        829        830        832        837
## 0.13877893 0.65561750 0.09844910 0.71068206 0.03122424 0.11182859 0.45444039
##        839        841        850        854        857        858        861
## 0.32843198 0.58789329 0.71068206 0.62270691 0.68054782 0.15015729 0.45369316
##        865        870        873        874        875        876        882
## 0.84722833 0.13350509 0.59782667 0.14367866 0.27439438 0.45444039 0.63755303
##        883        886        890        893        895        897        903
## 0.66111108 0.05050362 0.47710127 0.30659686 0.19901326 0.51917495 0.11306667
##        906        909        910        915        920        925        927
## 0.09733606 0.57620924 0.18614173 0.39075845 0.83013319 0.23340391 0.49788443
##        929        931        932        934        936        939        944
## 0.45812435 0.33223858 0.29246389 0.65561750 0.44382957 0.33859541 0.18822026
##        951        952        956        961        963        966        971
## 0.30659686 0.41525231 0.56186148 0.01035612 0.59782667 0.48700415 0.14835792
##        989        991       1002       1003       1007       1018       1019
## 0.02217197 0.88933218 0.24183042 0.11708076 0.29005366 0.06640941 0.05075299
##       1020       1021       1023       1026       1036       1038       1043
## 0.23549455 0.46800518 0.44001107 0.40301323 0.09561316 0.89309615 0.05325384
##       1045       1046       1053       1054       1055       1060       1062
## 0.75417417 0.29288743 0.11557951 0.06797509 0.64892511 0.49788443 0.19805414
##       1065       1068       1072       1073       1080       1084       1088
## 0.83563710 0.67658051 0.03873374 0.02669748 0.06051225 0.23336063 0.17117886
##       1103       1110       1113       1114       1116       1121       1122
## 0.58783507 0.33223858 0.49760507 0.29546358 0.62270691 0.27797061 0.45930377
##       1127       1128       1130       1139       1140       1144       1152
## 0.29288743 0.49760507 0.06410097 0.07043474 0.01035612 0.09733606 0.14512792
##       1153       1155       1160       1161       1162       1166       1167
## 0.19710104 0.33859541 0.20014802 0.47835081 0.62270691 0.51370104 0.66437207
##       1171       1182       1185       1187       1188       1194       1200
## 0.45930377 0.35486224 0.64833160 0.23865487 0.18907280 0.12761262 0.76927292
##       1201       1204       1207       1210       1212       1214       1217
## 0.09733606 0.27439438 0.14835792 0.63309812 0.07043474 0.10092509 0.63297245
##       1227       1229       1235       1239       1243       1244       1247
## 0.52432048 0.05248509 0.27298274 0.62270691 0.42153682 0.03437649 0.67658051
##       1249       1250       1251       1254       1256       1257       1258
## 0.16244820 0.05248509 0.06367428 0.39616882 0.14512792 0.27797061 0.10255532
##       1261       1265       1267
## 0.22352768 0.03122424 0.29005366
```

```r
test$win <- factor(test$win, levels = levels(predicted.classes.min))
```

```r
#Model accuracy
conf_matrix_test <- confusionMatrix(predicted.classes.min, test$win, positive = '1')
conf_matrix_test
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 129  48
##          1  50 154
##
##                Accuracy : 0.7428
##                  95% CI : (0.6958, 0.7859)
##     No Information Rate : 0.5302
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.4834
##
##  Mcnemar's Test P-Value : 0.9195
##
##             Sensitivity : 0.7624
##             Specificity : 0.7207
##          Pos Pred Value : 0.7549
##          Neg Pred Value : 0.7288
##              Prevalence : 0.5302
##          Detection Rate : 0.4042
##    Detection Prevalence : 0.5354
##       Balanced Accuracy : 0.7415
##
##        'Positive' Class : 1
##
```

In the confusion matrix of the test set, the model accurately predicts 'No' (not success) 129 times and 'Yes' (success) 154 times. However, it remains errors rate in both categories as same as train set: False Positives (Type I error) and False Negatives (Type II Error).

```r
#Create a dataframe
performance_df <- data.frame(
  Metric = c("Accuracy",
             "Pos Pred Value",
             "Sensitivity",
             "Specificity",
             "Accuracy",
             "Pos Pred Value",
             "Sensitivity",
             "Specificity"),
  Value = c(round(conf_matrix$overall['Accuracy'],4),
            round(conf_matrix$byClass['Pos Pred Value'],4),
            round(conf_matrix$byClass['Sensitivity'],4),
            round(conf_matrix$byClass['Specificity'],4),
            round(conf_matrix_test$overall['Accuracy'],4),
            round(conf_matrix_test$byClass['Pos Pred Value'],4),
            round(conf_matrix_test$byClass['Sensitivity'],4),
```

```r
           round(conf_matrix_test$byClass['Specificity'],4)),
  Dataset = c('Train Set',
             'Train Set',
             'Train Set',
             'Train Set',
             'Test Set',
             'Test Set',
             'Test Set',
             'Test Set')
)

#Source(Shapiro and Sanchi, 2024)
# Create the line plot
theGraph <- performance_df %>%
  ggplot(aes(x = Metric,
             y = Value,
             group = Dataset,
             color = Dataset)) +
  geom_line() +
  geom_point(size = 3) +
  theme_minimal() +
  labs(title = "Performance Metrics: Train vs. Test Set",
       x = "Metric",
       y = "Value",
       caption = "Figure 15: Performance Metrics: Train vs. Test Set",)

theGraph
```

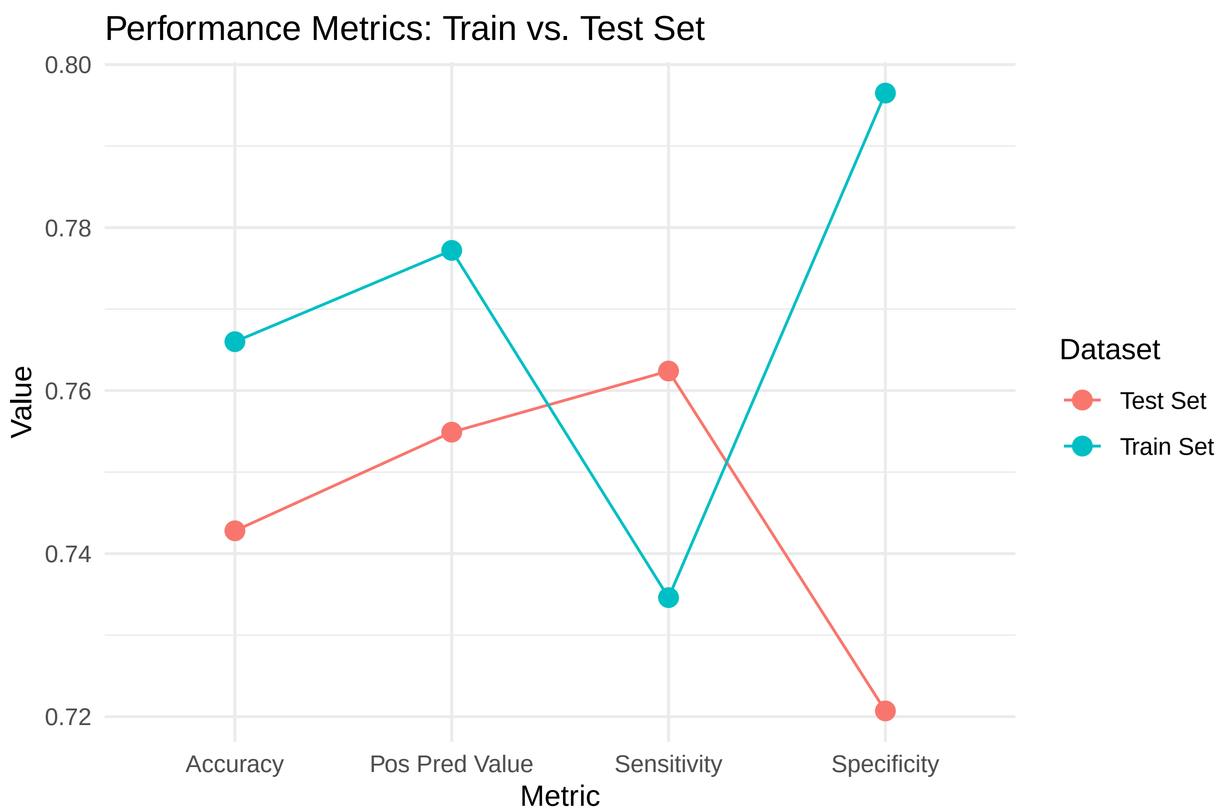


Figure 15: Performance Metrics: Train vs. Test Set

In Figure 15, the accuracy rate on the test set is lower than that on the training set, as expected. The specificity rate, focusing on the accuracy of negative predictions on the training set, is higher than on the test set, albeit with a trade-off resulting in a lower sensitivity rate. Overall, there is no significant difference between the two sets, indicating no significant overfitting on the training set.

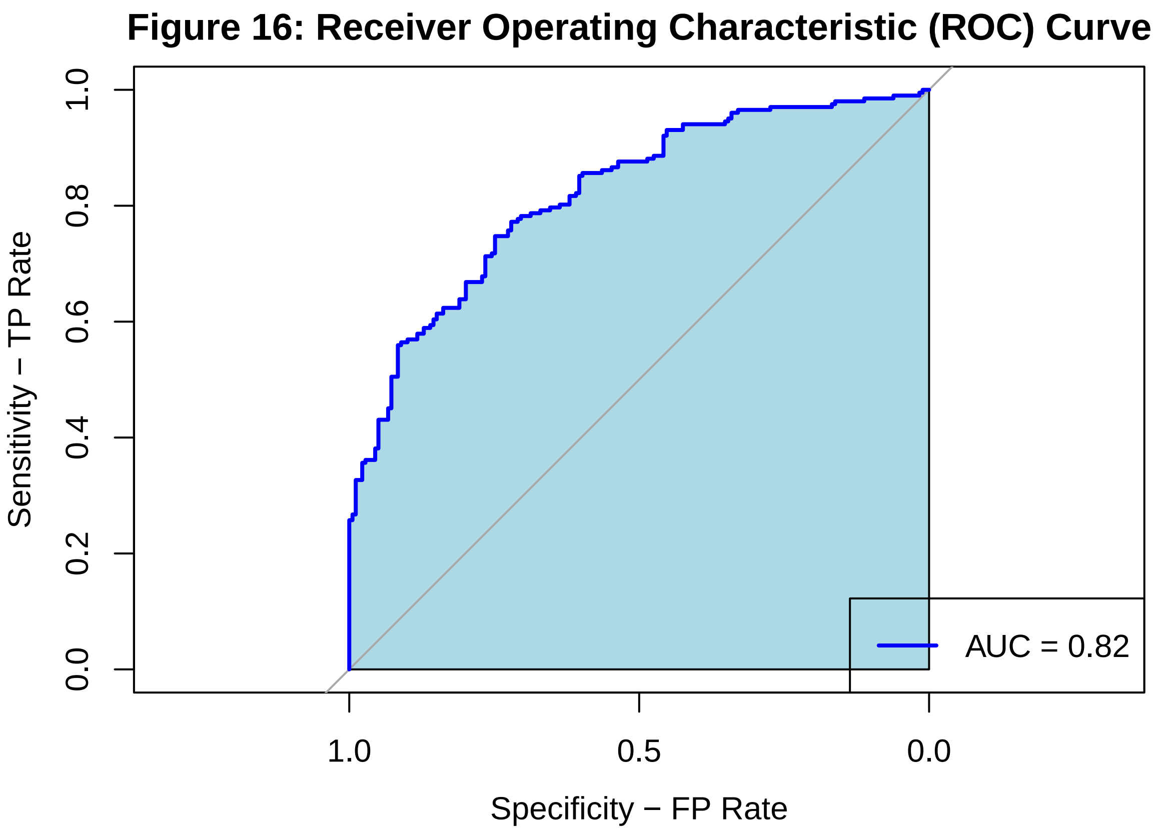
Plot and interpret the ROC curve.

```
ROC1 <- roc(test$win, probabilities.test)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
auc <- round(auc(ROC1),3)

#Source(Shapiro and Sharanya, 2024)
plot(ROC1, col = 'blue',
     ylab= 'Sensitivity - TP Rate',
     xlab = 'Specificity - FP Rate',
     main = 'Figure 16: Receiver Operating Characteristic (ROC) Curve',
     auc.polygon.col = "lightblue",
     auc.polygon = TRUE)
legend("bottomright", legend = paste("AUC =", auc), col = "blue", lwd = 2)
```



In Figure 16, the AUC (area under the ROC curve) is a measure of the model's discrimination, with a value close to 1 indicating perfect discrimination. In this case, the AUC is notably high at 0.82, suggesting a moderately discriminating model.

BQ5 Recommendation:

It is crucial to acknowledge that when making win or lose predictions for future games, if we only use past game statistics to train the model, the model will be based on the next game statistics to predict the next win. This is nonsense and unnecessary because when we get the next game statistics, we already know the result of the game.

Predicting wins or losses should also involve current information before the game start, such as player injuries, team dynamics (including age, the first 5 players, substitute players, and player fit), coaching, and tactic strategy from both internal and opposing teams to improve this model.

In the meantime, our focus on real-time analytics in the current dataset may help with coaching staff can use this prediction model to identify individual impactful factors that influence winning outcomes, thereby increasing the chances of winning future games.

BQ 6: Predict: The points per minute variable

Based on: Michael Jordan's statistics playing with Chicago Bulls, while excluding irrelevant variables such as end_year, tm, date, and gm_sc.

Using: Linear Regression.

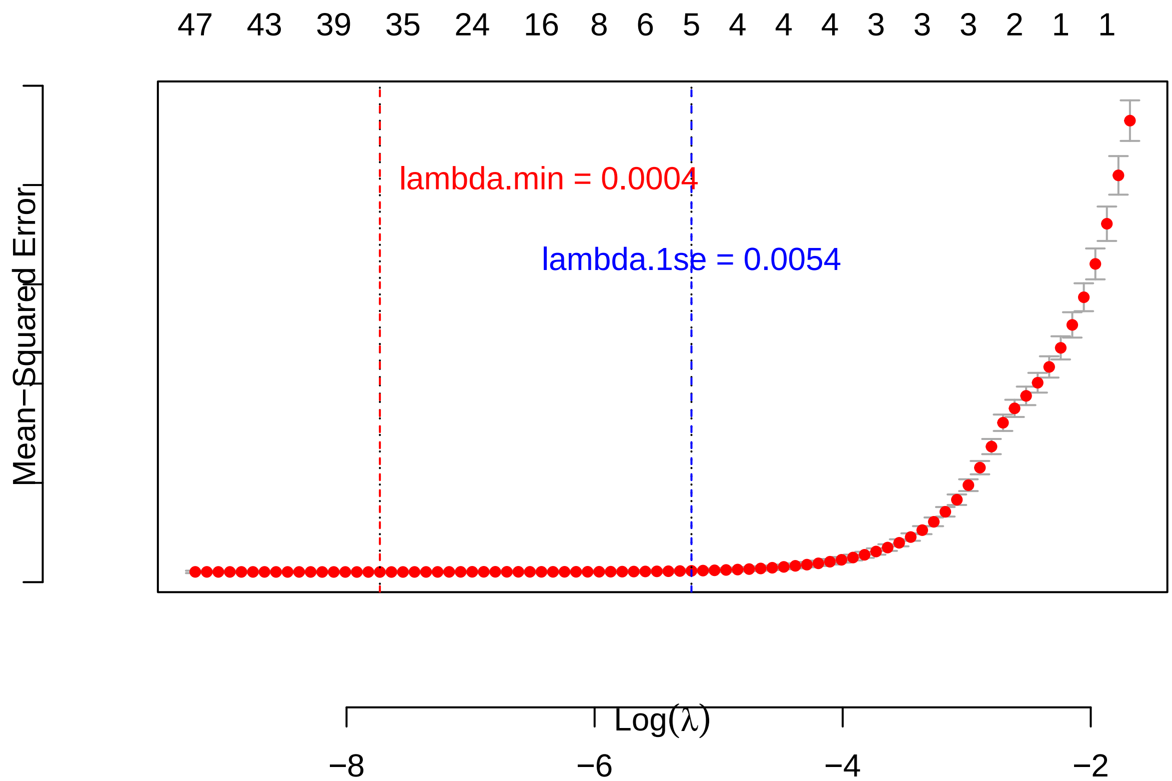
Split data into train and test sets

```r
#Split data into train and test sets
#Train 70%
#Test  30%

set.seed(5)
trainIndex <- createDataPartition(Michael_CHI_df$pts_per_minute, p = 0.70, list = FALSE, times = 1)

#don't forget ,
train <- Michael_CHI_df[trainIndex,]
test <- Michael_CHI_df[-trainIndex,]

#cv.glmnet requires matric
train_x <- model.matrix(pts_per_minute ~., train)[,-1]
test_x  <- model.matrix(pts_per_minute ~., test)[,-1]

train_y <- train$pts_per_minute
test_y  <- test$pts_per_minute
```

Estimate the lambda.min and lambda.1se

```r
lasso_cv <- cv.glmnet(x = train_x, y = train_y, alpha = 1, nfolds = 10)

lasso_lambda_table <- data.frame(
  Lambda_Type = c("lambda.min", "lambda.1se"),
  Lambda_Value = c(lasso_cv$lambda.min, lasso_cv$lambda.1se)
)

plot(lasso_cv,
     main = "Figure 17: Lasso Regression Cross-Validation Results",
     line = 3)

#Source(Shapiro and Sankalp, Monika, Sanchi, Pei-Yu, Phuong, Pengxiang, Zeyu, 2024)
abline(v = log(lasso_cv$lambda.min), col = "red", lty = 2)
abline(v = log(lasso_cv$lambda.1se), col = "blue", lty = 2)


# Annotations with lambda.min and lambda.1se values
text(x = log(lasso_cv$lambda.min), y = par("usr")[4] * 0.8, labels = paste("lambda.min =", round(lasso_c
text(x = log(lasso_cv$lambda.1se), y = par("usr")[4] * 0.7, labels = paste("lambda.1se =", round(lasso_c
```

Figure 17: Lasso Regression Cross−Validation Results



In Figure 17, for lambda.min, there are around 40 variables, while the number of variables on lambda.1se decreases to only 5-6.

```
lasso_model_min <- glmnet(train_x, train_y, alpha = 1, lambda = lasso_cv$lambda.min)
lasso_model_min
```

```
##
## Call:  glmnet(x = train_x, y = train_y, alpha = 1, lambda = lasso_cv$lambda.min)
##
##    Df  %Dev   Lambda
## 1 37  98.23 0.000439
```

```
lasso_model_1se <- glmnet(train_x, train_y, alpha = 1, lambda = lasso_cv$lambda.1se)
lasso_model_1se
```

```
##
## Call:  glmnet(x = train_x, y = train_y, alpha = 1, lambda = lasso_cv$lambda.1se)
##
##    Df  %Dev   Lambda
## 1  5 97.73 0.005412
```

```
lasso_coef_table <- data.frame(
  Variable = colnames(train_x),
  Coefficient.min = coef(lasso_model_min)[-1],
  Coefficient.1se = coef(lasso_model_1se)[-1]
```

```r
)
lasso_coef_table <- lasso_coef_table[order(-lasso_coef_table$Coefficient.1se),]

knitr::kable(lasso_coef_table, caption = "Table 2: Coefficient Values lasso Regression")
```

Table 11: Table 2: Coefficient Values lasso Regression

	Variable	Coefficient.min	Coefficient.1se
52	pts	0.0221621	0.0245381
37	fg_pct	0.1544594	0.0160688
43	ft_pct	0.0417028	0.0069628
1	years	0.0001370	0.0000000
2	home	-0.0022127	0.0000000
3	oppBOS	0.0025315	0.0000000
4	oppCHH	0.0051968	0.0000000
5	oppCLE	-0.0009737	0.0000000
6	oppDAL	0.0000000	0.0000000
7	oppDEN	-0.0021185	0.0000000
8	oppDET	0.0089512	0.0000000
9	oppGSW	0.0000000	0.0000000
10	oppHOU	0.0040088	0.0000000
11	oppIND	0.0045874	0.0000000
12	oppKCK	0.0000000	0.0000000
13	oppLAC	0.0000000	0.0000000
14	oppLAL	-0.0011331	0.0000000
15	oppMIA	0.0001420	0.0000000
16	oppMIL	0.0000000	0.0000000
17	oppMIN	0.0000000	0.0000000
18	oppNJN	0.0069153	0.0000000
19	oppNYK	-0.0061439	0.0000000
20	oppORL	-0.0118285	0.0000000
21	oppPHI	0.0011024	0.0000000
22	oppPHO	0.0025821	0.0000000
23	oppPOR	-0.0000252	0.0000000
24	oppSAC	0.0042820	0.0000000
25	oppSAS	-0.0046496	0.0000000
26	oppSEA	0.0048494	0.0000000
27	oppTOR	-0.0157549	0.0000000
28	oppUTA	0.0011540	0.0000000
29	oppVAN	-0.0156606	0.0000000
30	oppWAS	0.0000000	0.0000000
31	oppWSB	0.0075033	0.0000000
32	win	0.0031879	0.0000000
35	fg	0.0000000	0.0000000
36	fga	0.0033934	0.0000000
38	x3p	0.0074153	0.0000000
39	x3pa	-0.0010438	0.0000000
40	x3p_pct	0.0000000	0.0000000
41	ft	0.0000000	0.0000000
42	fta	0.0024139	0.0000000
44	orb	0.0000000	0.0000000
45	drb	0.0001317	0.0000000

	Variable	Coefficient.min	Coefficient.1se
46	trb	0.0000000	0.0000000
47	ast	0.0000000	0.0000000
48	stl	0.0014691	0.0000000
49	blk	0.0000000	0.0000000
50	tov	0.0000000	0.0000000
51	pf	-0.0009871	0.0000000
53	close_game	0.0000000	0.0000000
34	mp	-0.0208700	-0.0195614
33	gs	-0.1773310	-0.1201632

In Table 2, many coefficients of variables on Lambda.1se have been reduced to zero.

Determine the performance of the fit model against the training set by calculating the root mean square error (RMSE)

```r
preds_lasso_train_min <- predict(lasso_model_min, newx = train_x)
lasso_train_min_rmse <- rmse(train_y, preds_lasso_train_min)

preds_lasso_train_1se <- predict(lasso_model_1se, newx = train_x)
lasso_train_1se_rmse <- rmse(train_y, preds_lasso_train_1se)
```

Determine the performance of the fit model against the test set by calculating the root mean square error (RMSE)

```r
preds_lasso_test_min <- predict(lasso_model_min, newx = test_x)
lasso_test_min_rmse <- rmse(test_y, preds_lasso_test_min)

preds_lasso_test_1se <- predict(lasso_model_1se, newx = test_x)
lasso_test_1se_rmse <- rmse(test_y, preds_lasso_test_1se)

lasso_rmse_table <- data.frame(
  Metric = c("Lasso (Lambda.min)",
             "Lasso (Lambda.min)",
             "Lasso (Lambda.1se)",
             "Lasso (Lambda.1se)"),
  RMSE = c(lasso_train_min_rmse,
           lasso_test_min_rmse,
           lasso_train_1se_rmse,
           lasso_test_1se_rmse),
  Dataset = c('Train Set',
              'Test Set',
              'Train Set',
              'Test Set')
)

# Set the order of levels
lasso_rmse_table <- lasso_rmse_table %>%
  mutate(Dataset = factor(Dataset, levels = c('Train Set', 'Test Set')))

#Source(Shapiro and Sanchi, 2024)
# Create the line plot
```

```r
theGraph <- lasso_rmse_table %>%
  ggplot(aes(x = Dataset,
             y = RMSE,
             group = Metric,
             color = Metric)) +
  geom_line() +
  geom_point(size = 3) +
  theme_minimal() +
  labs(title = "Performance Metrics: Train vs. Test Set",
       x = "Metric",
       y = "Value",
       caption = "Figure 18: Performance Metrics: Train vs. Test Set")

theGraph
```

Figure 18: Performance Metrics: Train vs. Test Set

From Figure 18, the RSME values for the train set and test set are around 0.03-0.04, indicating that these lasso models can make predictions with new observations and yield similar results as the training set, suggesting that they are not overfitting to the training dataset.

```r
lasso_coef_1se_table <- data.frame(
  Variable = colnames(train_x),
  Coefficient.1se = coef(lasso_model_1se)[-1]
)
```

```
lasso_coef_1se_table <- lasso_coef_1se_table %>%
  filter(Coefficient.1se != 0)

lasso_coef_1se_table$Coefficient.1se <- round(lasso_coef_1se_table$Coefficient.1se, 5)

equation <- paste("Points Per Minute =", paste("(", lasso_coef_1se_table$Coefficient.1se, "*", lasso_coe
equation
```

Model equation

```
## [1] "Points Per Minute = ( -0.12016 * gs ) + ( -0.01956 * mp ) + ( 0.01607 * fg_pct ) + ( 0.00696 * 
```

BQ6 Recommendations:

- The points per minute prediction model reveals the five key factors that impact Jordan's scoring efficiency in games. Basketball trainers can adjust training schedules based on these impactful factors to enhance player performance and unlock player potential.
- Utilizing points per minute predictions to forecast when a player is likely to score during a basketball match is beneficial. By converting points per minute to the time it takes for a player to score 2 or 3 points, this information helps identify when exciting moments may occur. It can be used to automatically capture video highlights during a game, assisting short video creators in generating content that showcases thrilling shots and can be immediately shared on social platforms.

Conclusion

Through careful analysis of the research questions (BQ1-BQ6) and their corresponding methodologies (M1-M6), we have obtained significant insights into different facets of Michael Jordan's performance and how it influences team success.

We employed various statistical techniques, including t-tests and ANOVA, to assess the mean points per minute in different situations, identifying significant variations. Chi-square analysis was utilized to examine the independence of winning percentage by year and game score level (two categorical variables). Logistic regression was employed for predicting binary outcomes, specifically to identify the potential next successful player in the future using step-wise subsetting. Finally, Lasso regression was applied to predict points per minute.

Using the successful statistical records of Michael Jordan before his peak performance served as a foundation for creating prediction models to classify who might emerge as the next successful player in the future. Additionally, hypothesis testing played a crucial role in identifying differences in player performance through statistical calculations, ensuring an unbiased assessment.

To summarize, the thorough analyses carried out have provided insights into several aspects of Michael Jordan's performance and their impact on the success of the team. By comprehending Jordan's performance across many game scenarios, against diverse adversaries, and in correlation with team achievements, we can extract useful lessons for both individual player advancement and team strategy design. These insights can be utilized to optimize decision-making processes with the goal of increasing player performance and improving team competitiveness on the basketball court.

References

OpenAI. (2022, November 30). *ChatGPT*. Chat.openai.com; OpenAI. https://chat.openai.com

Jabari Parker 2017-18 Game Log. (n.d.). Basketball-Reference.com. Retrieved February 1, 2024, from https://www.basketball-reference.com/players/p/parkeja01/gamelog/2018

Hornets Beware: 7 Biggest busts at pick No. 2 in NBA Draft history. (2023, June 11). Swarm and Sting. https://swarmandsting.com/2023/06/11/hornets-beware-7-biggest-busts-pick-no-2-nba-draft-history/3/

Shapiro, V. (2024, January 31). ALY6015_Module3_R-HallOfFame_WinterA_2024. Https://Northeastern.instructure.com/. https://northeastern.instructure.com/courses/164824/files/25916160?wrap=1, Sharanya, Sanchi