Course:
Data Exploration and Visualization

Work:
Final Assignment

Student:
Chen Shi
2022302091024

Behind the basketball data?
——Analysis and visualization of NBA data

Summary

In the field of NBA player value prediction, the topic of whether low-level data is more valuable to players than high-level data has triggered many discussions.

Low-level data, including basic stats like points, rebounds, assists, steals, and blocks, have long been crucial in gauging players' on-court performance. They're the "hard currency" of basketball, directly showing players' most visible contributions. Scoring numbers plainly display offensive firepower, with no need for complex conversions for fans, coaches, or management to gauge a player's scoring impact. Rebounding figures, be it defensive or offensive, quickly convey a player's interior presence and tenacity. Assists directly reflect a player's ability to spark the team's offense and connect teammates.

In contrast, high-level data, while calculated through complex algorithms with multiple variables (such as PER integrating various stats for comprehensive evaluation and WS quantifying contribution to victory from the team's win perspective), has a "threshold" for understanding and application. Non-professionals or those new to in-depth basketball analysis struggle to quickly grasp its meaning and interpret the hidden information.

In this article, we used existing data to conduct basic data analysis, time series analysis between teams, and used various machine learning prediction methods (Neural Networks, Gradient Boosting Decision Trees, Support Vector Machines and so on) to compare the differences between low-order data and high-order data, and concluded that:

<u>although high-order data calculations are more complex, low-order data has better predictive power for player value due to its intuitive features.</u>

Keyword： **Clustering , Dimensionality Reduction , Neural Networks , Gradient Boosting Decision Trees , Support Vector Machines , VAR**

Content

1. Fundamental analysis

1.1 Data description and processing

I found an NBA basketball dataset (playoffStat) on Kaggle. The original data has a dimension of (10648, 51), which is relatively complex and many of the data are quite old. In order to pay tribute to Michael Jordan's first NBA championship in 1991, I selected data from 1991 and later, and then selected some typical indicators from low to high order:

Variable	变量
Season	赛季年份
player	球员姓名
Pos	球员位置（C＝ 中锋，PF＝ 大前锋，SF＝ 小前锋，SG＝ 得分后卫，PG＝ 控球后卫）
age	球员年龄
team_id	球员所在球队的缩写
g	球员在赛季中的出场次数
gs	球员首发出场的次数
mp_per_g	球员平均每场比赛的出场时间（分钟）
fga_per_g	球员平均每场比赛的投篮尝试次数
fg_pct	球员的投篮命中率
fg3a_per_g	球员平均每场比赛的三分球尝试次数
fg3_pct	球员的三分球命中率
fg2a_per_g	球员平均每场比赛的两分球尝试次数
fg2_pct	球员的两分球命中率
fta_per_g	球员平均每场比赛的罚球尝试次数
ft_pct	球员的罚球命中率
ast_pct	球员的助攻百分比
blk_pct	球员的盖帽百分比
stl_pct	球员的抢断百分比
tov_pct	球员的失误百分比
trb_pct	球员的总篮板百分比
ts_pct	球员的真实投篮命中率
usg_pct	球员的使用率
dws	球员的防守胜利贡献
ows	球员的进攻胜利贡献
per	球员效率值
ws	球员的总胜利贡献
ws_per_48	球员每 48 分钟的胜利贡献

```r
library(tidyverse)
library(tidytext)
library(corrplot)
library(mlr3)
library(mlr3cluster)
library(mlr3learners)

playoffStats <- read.csv("D:/桌面/数据探索与可视化大作业/playoffStats.csv")

dim(playoffStats)
## [1] 10648     51

playoffStats_cleaned <- playoffStats |>
  filter(season >= 1991) |>
dplyr::select(season,player,pos,age,team_id,g,gs,mp_per_g,fga_per_g,fg_pct,fg3a_per_g,fg3_pct,
fg2a_per_g,fg2_pct,fta_per_g,ft_pct,trb_per_g,ast_per_g,stl_per_g,blk_per_g,tov_per_g,pf_per_g,
pts_per_g,ast_pct,blk_pct,stl_pct,tov_pct,trb_pct,ts_pct,usg_pct,dws,ows,per,ws,ws_per_48) |>
  drop_na()
```

```r
#查看数据结构
str(playoffStats_cleaned)
## 'data.frame':    3949 obs. of  35 variables:
##  $ season    : int  2022 2022 2022 2022 2022 2022 2022 2022 2022 2022 ...
##  $ player    : chr  "Omer Yurtseven" "Draymond Green" "Danny Green" "Devonte' G
raham" ...
##  $ pos       : chr  "C" "PF" "SF" "PG" ...
##  $ age       : int  23 31 34 26 26 33 28 28 27 20 ...
##  $ team_id   : chr  "MIA" "GSW" "PHI" "NOP" ...
##  $ g         : int  9 22 12 6 5 5 5 18 10 6 ...
##  $ gs        : int  0 22 12 0 5 3 0 18 10 6 ...
##  $ mp_per_g  : num  4.2 32 26.6 10 32 22.4 15.2 38.2 38.5 37.8 ...
##  $ fga_per_g : num  2 6.5 7.4 3 10.8 9 3 8.7 16.1 18.3 ...
##  $ fg_pct    : num  0.667 0.479 0.404 0.333 0.426 0.4 0.4 0.471 0.484 0.455 ...
##  $ fg3a_per_g: num  0.2 1.8 6.3 2.3 3 3 2.2 6 3.3 9.5 ...
##  $ fg3_pct   : num  0 0.205 0.408 0.357 0.2 0.267 0.364 0.426 0.212 0.404 ...
##  $ fg2a_per_g: num  1.8 4.8 1.1 0.7 7.8 6 0.8 2.7 12.8 8.8 ...
##  $ fg2_pct   : num  0.75 0.581 0.385 0.25 0.513 0.467 0.5 0.571 0.555 0.509 ...
##  $ fta_per_g : num  0.3 2.1 0.1 1.3 5.6 2.2 1 1.3 8.9 5.7 ...
##  $ ft_pct    : num  0.333 0.638 0 0.875 0.714 1 0.8 0.708 0.82 0.824 ...
##  $ trb_per_g : num  0.8 7.2 3.1 1.5 7.2 4.2 0.6 5.5 10.7 4.2 ...
##  $ ast_per_g : num  0.3 6.3 0.8 0.7 2.6 0.8 1.4 1.9 2.1 3 ...
##  $ stl_per_g : num  0 1.1 1 0.2 0.4 0.2 0.2 0.9 0.4 1.2 ...
##  $ blk_per_g : num  0.1 1 0.3 0.2 1.2 0 0 0.4 0.8 1.2 ...
##  $ tov_per_g : num  0 2.7 1.1 0.7 1.6 1 0.4 1 3.2 2.5 ...
##  $ pf_per_g  : num  0.2 4 1.9 0.5 2.8 1.2 1.4 2.9 3.4 3.7 ...
##  $ pts_per_g : num  2.8 8 8.6 4 13.8 10.2 4 11.7 23.6 25.2 ...
##  $ ast_pct   : num  16.6 25.7 4.9 9.4 12 6.4 12.4 7.8 9.9 14.8 ...
##  $ blk_pct   : num  2.9 3.2 1.2 1.4 3.9 0 0 0.9 1.9 2.6 ...
##  $ stl_pct   : num  0 1.8 2 0.9 0.6 0.5 0.7 1.3 0.5 1.4 ...
##  $ tov_pct   : num  0 26.4 12.7 15.7 10.8 9.1 10.4 9.7 13.8 10.7 ...
##  $ trb_pct   : num  10.8 12.7 7.1 8.9 13.9 11.3 2.4 8.5 17 6.1 ...
##  $ ts_pct    : num  0.647 0.534 0.576 0.558 0.52 0.512 0.581 0.63 0.59 0.604 ...
##  $ usg_pct   : num  22.6 14 15.1 18.3 20.2 22.5 11 12.8 28.4 26.4 ...
##  $ dws       : num  0 1 0.3 0 0 0 -0.1 0.3 0.4 0.2 ...
##  $ ows       : num  0.1 0.4 0 0.1 0.3 0 0.1 1.2 0.5 0.3 ...
##  $ per       : num  25.8 12.3 9.9 11.5 15.7 10.8 7.8 12.6 19.1 17.8 ...
##  $ ws        : num  0.2 1.4 0.3 0.1 0.2 0 0 1.5 0.9 0.5 ...
##  $ ws_per_48 : num  0.228 0.094 0.049 0.049 0.063 0.011 0.028 0.106 0.117 0.102
 ...
```

```
#按照每个球队所包含的球员的平均数（置信度95%）为界，剔除包含球员较少的球队
mean_players = playoffStats_cleaned |>
  group_by(team_id) |>
  count() |>
  pull(n) |>
  mean(trim = 0.05)

playoffStats_cleaned <- playoffStats_cleaned |>
  group_by(team_id)|>
  mutate(num_players = n()) |>
  filter(num_players >= mean_players)
playoffStats_cleaned |>
  distinct(team_id) |>
  pull()
##  [1] "MIA" "PHI" "DEN" "ATL" "DAL" "PHO" "CHI" "MIL" "UTA" "BOS" "TOR"
"LAL"
## [13] "LAC" "POR" "ORL" "HOU" "IND" "SAS" "DET" "CLE"
```

20 teams:

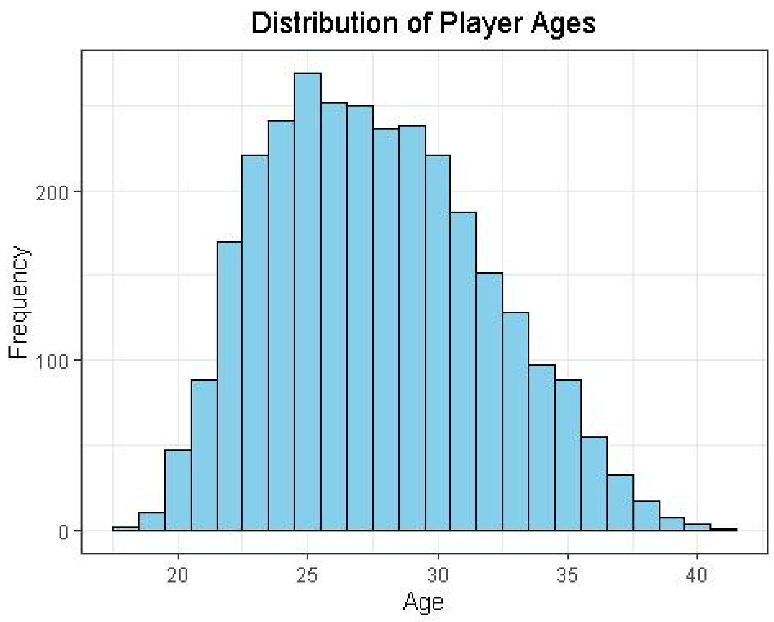
- **MIA**: Miami Heat（迈阿密热火）
- **PHI**: Philadelphia 76ers（费城 76 人）
- **DEN**: Denver Nuggets（丹佛掘金）
- **ATL**: Atlanta Hawks（亚特兰大老鹰）
- **DAL**: Dallas Mavericks（达拉斯独行侠）
- **PHO**: Phoenix Suns（菲尼克斯太阳）
- **CHI**: Chicago Bulls（芝加哥公牛）
- **MIL**: Milwaukee Bucks（密尔沃基雄鹿）

- **UTA**: Utah Jazz（犹他爵士）
- **BOS**: Boston Celtics（波士顿凯尔特人）
- **TOR**: Toronto Raptors（多伦多猛龙）
- **LAL**: Los Angeles Lakers（洛杉矶湖人）
- **LAC**: Los Angeles Clippers（洛杉矶快船）
- **POR**: Portland Trail Blazers（波特兰开拓者）
- **ORL**: Orlando Magic（奥兰多魔术）
- **HOU**: Houston Rockets（休斯顿火箭）
- **IND**: Indiana Pacers（印第安纳步行者）
- **SAS**: San Antonio Spurs（圣安东尼奥马刺）
- **DET**: Detroit Pistons（底特律活塞）
- **CLE**: Cleveland Cavaliers（克利夫兰骑士）

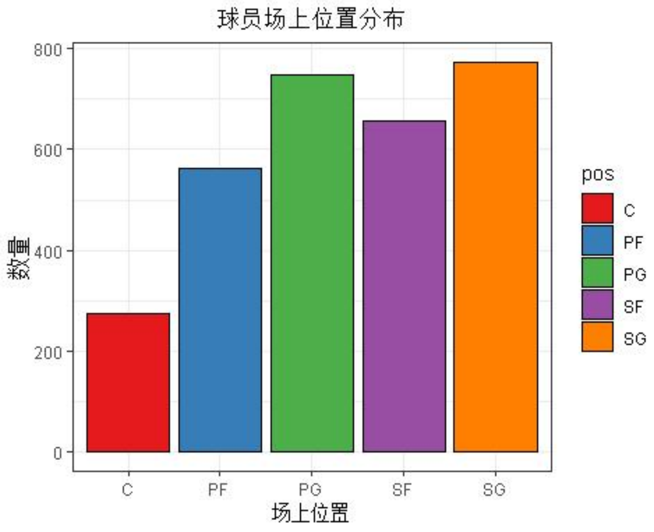
```r
write.csv(playoffStats_cleaned, "playoffStats_cleaned.csv")
```

1.2 Basic descriptive analysis

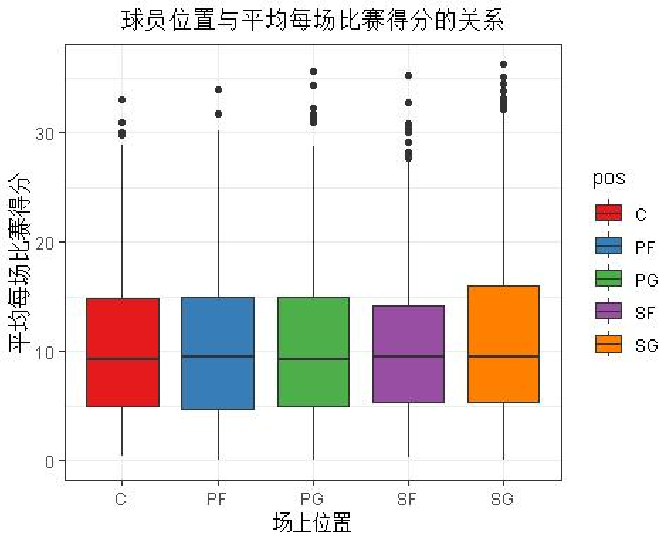
```r
#统计球员的年龄分布
ggplot(playoffStats_cleaned, aes(x = age)) +
  geom_histogram(aes(y =..count..), binwidth = 1, fill = "skyblue", color = "black") +
  labs(title = "Distribution of Player Ages", x = "Age", y = "Frequency") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5))
## Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
##    Please use `after_stat(count)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



```
#统计球场各个位置的数量
ggplot(playoffStats_cleaned, aes(pos,fill = pos)) +
  geom_bar(color = "black") +
  labs(title = "球员场上位置分布", x = "场上位置", y = "数量")+
  theme_bw()+
    theme(plot.title = element_text(hjust = 0.5))+
    scale_fill_brewer(palette = "Set1")
```

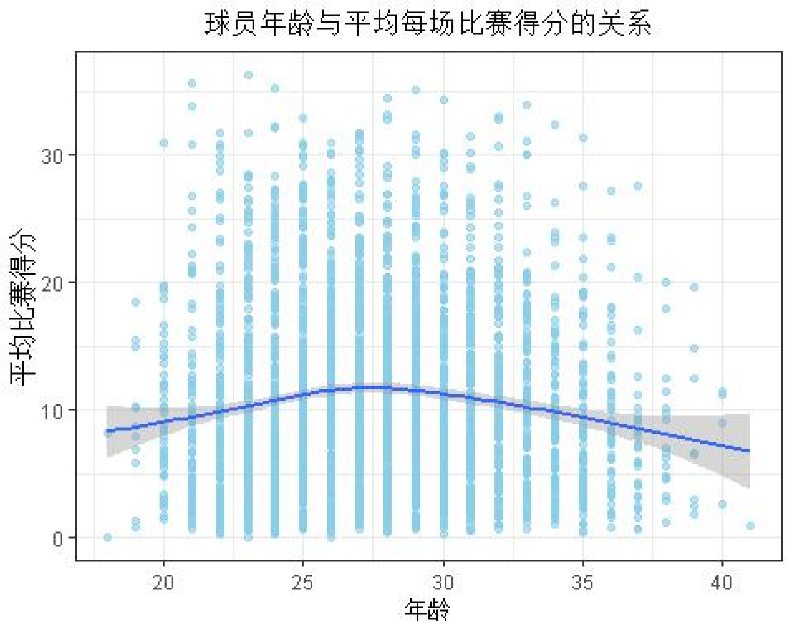


```
# 箱线图：球员位置与平均每场比赛得分的关系
ggplot(playoffStats_cleaned, aes(x = pos, y = pts_per_g,fill = pos)) +
  geom_boxplot() +
  labs(title = "球员位置与平均每场比赛得分的关系", x = "场上位置", y = "平均每场比
赛得分")+
  theme_bw()+
    theme(plot.title = element_text(hjust = 0.5))+
    scale_fill_brewer(palette = "Set1")
```



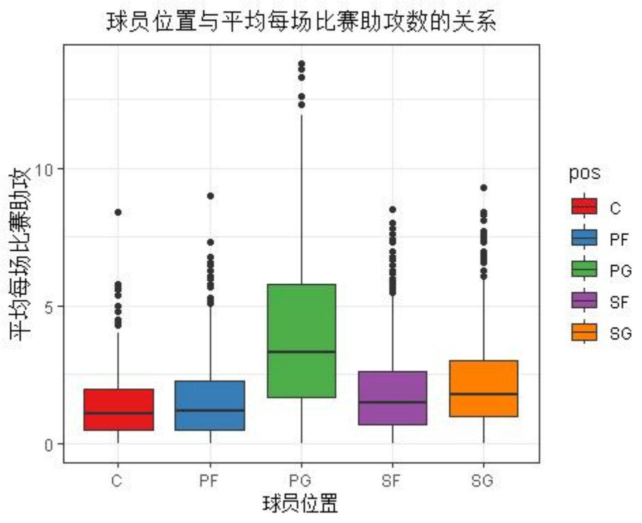
Due to the large amount of data, there is no significant difference in the scores between positions.

```
# 散点图：球员年龄与平均每场比赛得分的关系
ggplot(playoffStats_cleaned, aes(x = age, y = pts_per_g)) +
  geom_point(color = "skyblue", alpha = 0.6) +
  geom_smooth() +
  labs(title = "球员年龄与平均每场比赛得分的关系", x = "年龄", y = "平均比赛得分")+
  theme_bw()+
    theme(plot.title = element_text(hjust = 0.5))
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



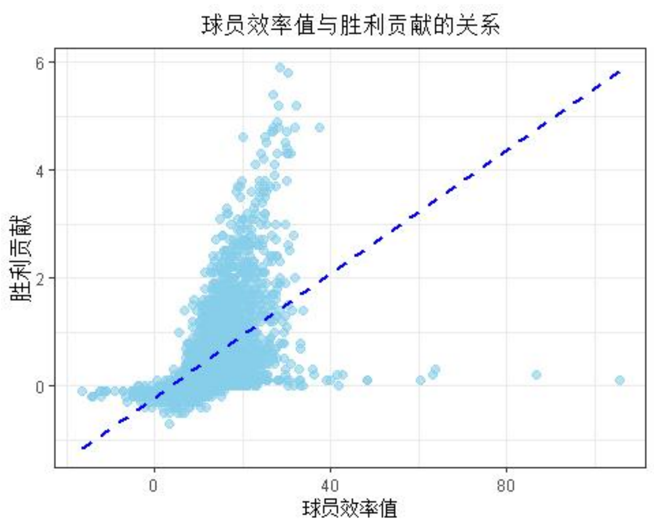
As players age, their average points per game tend to rise first and then fall, reaching a peak around age 27.

```
# 箱线图: 球员位置与平均每场比赛助攻数的关系
ggplot(playoffStats_cleaned, aes(x = pos, y = ast_per_g,fill = pos)) +
  geom_boxplot() +
  labs(title = "球员位置与平均每场比赛助攻数的关系", x = "球员位置", y = "平均每场比赛助攻") +
  theme_bw()+
    theme(plot.title = element_text(hjust = 0.5))+
    scale_fill_brewer(palette = "Set1")
```



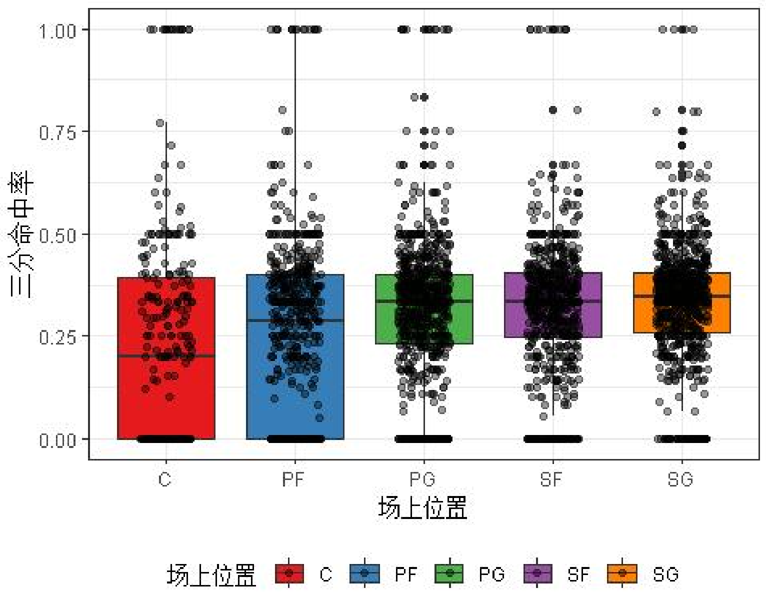
At this time, the box plot can reflect the significant difference. PG point guards tend to contribute more passes and connect the whole team.

```
# 散点图: 球员效率值与胜利贡献的关系
ggplot(playoffStats_cleaned, aes(x = per, y = ws)) +
  geom_point(color = "skyblue", alpha = 0.6, size = 2) +
  geom_smooth (method = "lm",color = "blue", linetype = "dashed", se = FALSE) +
  labs(
    title = "球员效率值与胜利贡献的关系",
    x = "球员效率值",
    y = "胜利贡献"
  ) +
  theme_bw() +
  theme(
    plot.title = element_text( hjust = 0.5),
  )+
    scale_fill_brewer(palette = "Set1")
## `geom_smooth()` using formula = 'y ~ x'
```



Win share is proportional to the player's efficiency value

```
#验证场上位置与三分命中率的直观关系
ggplot(playoffStats_cleaned, aes(x = pos, y = fg3_pct, fill = pos)) +
  geom_boxplot() +
  geom_jitter(alpha = 0.4, width = 0.2) +  # 添加散点图展示个体数据点，设置透明度和水平抖动幅度
  labs(x = "场上位置", y = "三分命中率", fill = "场上位置") +
  theme_bw() +
  theme(
    plot.title = element_text(hjust = 0.5),  # 标题居中
    legend.position = "bottom"  # 图例位置设置在底部
  ) +
  scale_fill_brewer(palette = "Set1")
```



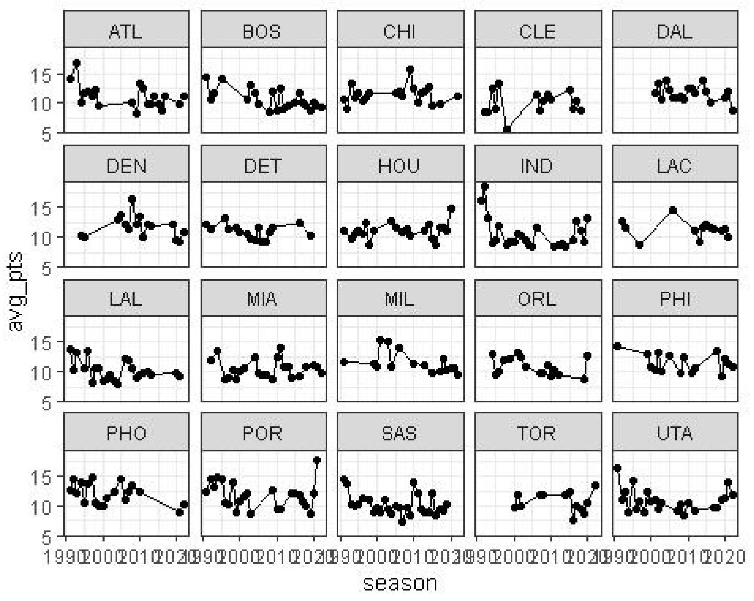
The three-point shooting percentage of centers is significantly lower than that of other positions, which is in line with the common sense of our classic play.

```
dim(playoffStats_cleaned)
## [1] 3017   36
```

Generate a dataset of team scores for time series analysis:

```
team_score_aggregated <- playoffStats_cleaned %>%
  group_by(season, team_id) %>%
  summarise(total_pts = sum(pts_per_g),
            avg_pts = mean(pts_per_g),
            min_pts = min(pts_per_g),
            max_pts = max(pts_per_g),
            num_games = n())
## `summarise()` has grouped output by 'season'. You can override using the
## `.groups` argument.
team_score_aggregated %>%
  filter(team_id == 'LAL')
## # A tibble: 23 × 7
## # Groups:   season [23]
##    season team_id total_pts avg_pts min_pts max_pts num_games
##     <int> <chr>       <dbl>   <dbl>   <dbl>   <dbl>     <int>
##  1   1991 LAL          95.5   13.6      1.9    21.8         7
##  2   1992 LAL          51.5   10.3      1.8    18.8         5
##  3   1993 LAL          80     13.3      6.8    18           6
##  4   1995 LAL          73.3   10.5      1.5    20           7
##  5   1996 LAL          93.7   13.4      9      19           7
##  6   1997 LAL          65.4    8.18     1.3    14.4         8
##  7   1998 LAL          62.8   10.5      6      17           6
##  8   1999 LAL          63.8   10.6      4.3    19.8         6
##  9   2000 LAL          66.5    8.31     2.4    21.1         8
## 10   2001 LAL          69.2    8.65     1.9    29.4         8

## #    13 more rows
write.csv(team_score_aggregated,"team_score_arrgregated.csv")
```

```
#球队平均每场得分随赛季变化的关系，时间序列图
ggplot(team_score_aggregated,aes(season,avg_pts))+
  facet_wrap(~team_id)+
  geom_point()+
  geom_line(color = "black", linetype = "solid")+
  theme_bw()
```



Next, we will explore machine learning.
First, we need to construct a binary variable to determine whether a player is **"great"**. Apart from the columns of various basic data and high-level data, the number of appearances g of a player is a very good intuitive variable:
if a player is "great", then he will have enough appearances;
therefore, we use the 75th percentile of the player's playing time, which means that the function will sort the data from small to large and find the value of the point where 75% of the data is less than or equal to this value.
In addition, we also removed 15% of the extreme data at both ends, and the requirements for "great" are extremely stringent.

```
#算出球员赛季出场次数的均值，并采取截尾操作
trimmed_mean_value <- quantile(playoffStats_cleaned$g,0.75,trim = 0.15)

playoffStats_trimmed <- playoffStats_cleaned |>
  mutate(is_great = ifelse(g > trimmed_mean_value ,1,0))
write.csv(playoffStats_trimmed,"playoddStats_trimmed.csv")
playoffStats_trimmed |>
  ungroup() |>
  count(is_great)
## # A tibble: 2 × 2
##   is_great     n
##      <dbl> <int>
## 1        0  2286
## 2        1   731
#可以看到，在较为严格的筛选下，只有731 个球员被分类play ==为"great"
```

All subsequent analysis will use playoffStats_trimmed as the benchmark!

2. Time Series analysis

2.1 Introduction

We selected two teams from the Western Conference Pacific Division: **Los Angeles Lakers**, **Phoenix Suns.**

Los Angeles Lakers and Phoenix Suns have played against each other 332 times in the regular season since their first encounter in 1968 until November 2024. The Lakers have an advantage with 191 wins and 143 losses, 103 wins and 55 losses at home, and 88 wins and 82 losses at home for the Suns.

They have met 13 times in the playoffs, making them the most frequent team in the Western Conference playoffs. In the 1970s and 1980s, the Lakers repeatedly suppressed the Suns to advance, such as the 1970 Western Conference semifinals, where they reversed the game after falling behind 1-3. Classic battles also include the first round of the Western Conference in 2006, when the Lakers led 3-1 but were overturned by the Suns 4-3; the Lakers won 4-2 in the Western Conference Finals in 2010; the Suns defeated the Lakers 4-2 in the first round of the Western Conference in 2021, and Devin Booker played brilliantly to help the team advance. The feud and exciting confrontation between the two sides have lasted for many years.

In terms of predictive ability, the **VAR** model can accurately predict the future short-term scoring situation based on the past scoring data of the two teams. Secondly, in terms of variable relationship analysis, it can clearly present the mutual influence relationship between the scores of the two teams.

The impulse response function can be used to analyze the impact of the score shock of one side on the other side, and the source of the score fluctuations of the two teams can be determined through variance decomposition. In addition, the VAR model is data-driven and does not rely on a priori theory.

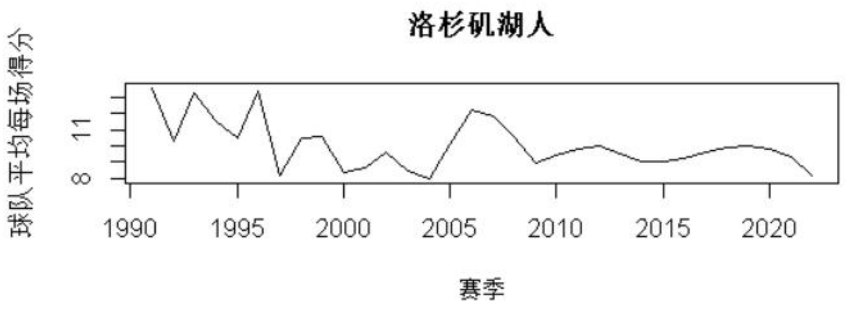
For the scoring data of the two teams in non-economic fields, there is no need to preset a strict causal relationship direction, and it can reveal the variable relationship more flexibly and objectively.

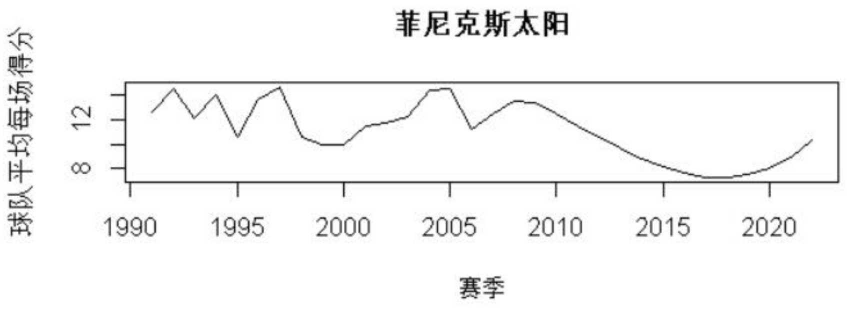
```r
library(tseries)
library(vars)
library(tidyverse)

team_score_aggregated <- read.csv("D:/桌面/数据探索与可视化大作业/team_score_arrgregate
d.csv")
```

```r
#选取了西部太平洋赛区的两支球队：洛杉矶湖人、菲尼克斯太阳、洛杉矶快船
team_selected <- team_score_aggregated |>
  filter(team_id %in% c('LAL','PHO','LAC')) |>
  arrange(team_id)

LAL_data <- team_selected |>
  filter(team_id == 'LAL')

PHO_data <- team_selected |>
  filter(team_id == 'PHO')
#得到 LAL_fulldata,PHO_fulldata
new_time <- seq(1991, 2022, by = 1)

spline_interpolation_1 <- spline(x = LAL_data$season, y = LAL_data$avg_pts, xout = new
_time)
LAL_fulldata <- data.frame(season = new_time, avg_pts = spline_interpolation_1$y)

spline_interpolation_2 <- spline(x = PHO_data$season, y = PHO_data$avg_pts, xout = new
_time)
PHO_fulldata <- data.frame(season = new_time, avg_pts = spline_interpolation_2$y)
```

2.2 Time Series analysis

```r
par(mfrow = c(2,1))
plot(LAL_fulldata$season,LAL_fulldata$avg_pts,type = "l",main = "洛杉矶湖人",xlab = "赛
季", ylab = "球队平均每场得分")
plot(PHO_fulldata$season,PHO_fulldata$avg_pts,type = "l",main = "菲尼克斯太阳", xlab =
"赛季", ylab = "球队平均每场得分")
```





```
LAL_pts <- ts(LAL_fulldata$avg_pts,start = c(1991),end = c(2022),frequency = 1)
PHO_pts <- ts(PHO_fulldata$avg_pts,start = c(1991),end = c(2022),frequency = 1)
```

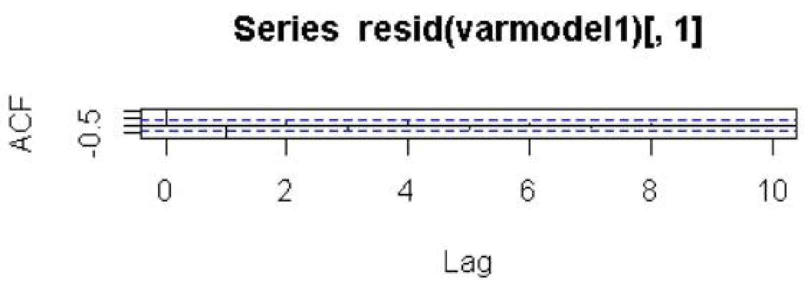
```
#LAL 的数据需要做二次差分
adf.test(diff(diff(LAL_pts)))
## Warning in adf.test(diff(diff(LAL_pts))): p-value smaller than printed p-value
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(diff(LAL_pts))
## Dickey-Fuller = -4.9438, Lag order = 3, p-value = 0.01
## alternative hypothesis: stationary
#PHO 的数据需要做二次差分
adf.test(diff(diff(PHO_pts)))
## Warning in adf.test(diff(diff(PHO_pts))): p-value smaller than printed p-value
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(diff(PHO_pts))
## Dickey-Fuller = -5.3591, Lag order = 3, p-value = 0.01
## alternative hypothesis: stationary
LAL_pts_diff <- diff(diff(LAL_pts))
PHO_pts_diff <- diff(diff(PHO_pts))
```

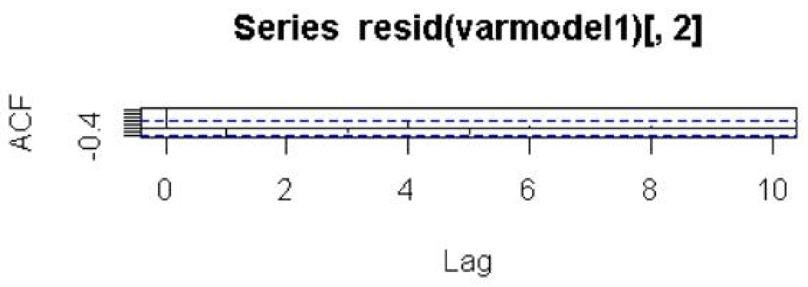
```
data <- cbind(LAL_pts_diff,PHO_pts_diff)
VARselect(data, lag.max = 10, type = 'both')
## $selection
## AIC(n)  HQ(n)  SC(n) FPE(n)
##      9      9      9     10
##
## $criteria
##                1          2          3           4          5           6
## AIC(n) 0.7811879 -0.2705050 -0.3940382 -1.03151749 -2.1731884 -3.46444327
## HQ(n)  0.8589388 -0.1538786 -0.2385363 -0.83714009 -1.9399355 -3.19231491
## SC(n)  1.1794808  0.3269343  0.4025476 -0.03578522 -0.9783097 -2.07041809
## FPE(n) 2.2080684  0.7926079  0.7412314  0.43418138  0.1651998  0.06110014
##                    7             8    9   10
## AIC(n) -8.2992793984 -1.066805e+01 -Inf -Inf
## HQ(n)  -7.9882755573 -1.031817e+01 -Inf -Inf
## SC(n)  -6.7061077608 -8.875730e+00 -Inf -Inf
## FPE(n)  0.0008211291  2.296003e-04  NaN    0
```

```
#选取滞后阶数 p = 9 或者 10 时，滞后项多项式的根有的位于单位圆内，综合考究 AIC 与多项式的根的限制，选取 p = 8
varmodel1 <- VAR(data, type = 'both', p = 8)
roots(varmodel1)
##  [1] 0.9162560 0.9162560 0.9028341 0.9011141 0.9011141 0.8983083 0.8983083
##  [8] 0.8973016 0.8973016 0.8779584 0.8448015 0.8448015 0.8414527 0.8414527
## [15] 0.7315983 0.7315983
summary(varmodel1)
##
## VAR Estimation Results:
## =========================
## Endogenous variables: LAL_pts_diff, PHO_pts_diff
## Deterministic variables: both
## Sample size: 22
## Log Likelihood: 60.782
## Roots of the characteristic polynomial:
## 0.9163 0.9163 0.9028 0.9011 0.9011 0.8983 0.8983 0.8973 0.8973 0.878 0.8448 0.8448
 0.8415 0.8415 0.7316 0.7316
## Call:
## VAR(y = data, p = 8, type = "both")
##
##
## Estimation results for equation LAL_pts_diff:
## ===========================================
## LAL_pts_diff = LAL_pts_diff.l1 + PHO_pts_diff.l1 + LAL_pts_diff.l2 + PHO_pts_diff.l
```

```
2 + LAL_pts_diff.l3 + PHO_pts_diff.l3 + LAL_pts_diff.l4 + PHO_pts_diff.l4 + LAL_pts_d
iff.l5 + PHO_pts_diff.l5 + LAL_pts_diff.l6 + PHO_pts_diff.l6 + LAL_pts_diff.l7 + PHO_pt
s_diff.l7 + LAL_pts_diff.l8 + PHO_pts_diff.l8 + const + trend
##
##                  Estimate Std. Error t value Pr(>|t|)
## LAL_pts_diff.l1  0.350314   0.084418   4.150 0.014263 *
## PHO_pts_diff.l1 -0.487243   0.116312  -4.189 0.013816 *
## LAL_pts_diff.l2 -0.930849   0.079603 -11.694 0.000306 ***
## PHO_pts_diff.l2 -0.482832   0.081713  -5.909 0.004106 **
## LAL_pts_diff.l3  0.573424   0.175023   3.276 0.030607 *
## PHO_pts_diff.l3  0.442534   0.070337   6.292 0.003260 **
## LAL_pts_diff.l4  0.407722   0.134084   3.041 0.038372 *
## PHO_pts_diff.l4 -0.259598   0.065060  -3.990 0.016263 *
## LAL_pts_diff.l5  0.203855   0.083062   2.454 0.070123 .
## PHO_pts_diff.l5  0.058866   0.070591   0.834 0.451240
## LAL_pts_diff.l6 -0.059695   0.067663  -0.882 0.427477
## PHO_pts_diff.l6 -0.777827   0.107671  -7.224 0.001947 **
## LAL_pts_diff.l7  0.088652   0.082522   1.074 0.343181
## PHO_pts_diff.l7 -0.351796   0.053906  -6.526 0.002847 **
## LAL_pts_diff.l8 -0.092221   0.043362  -2.127 0.100572
## PHO_pts_diff.l8 -0.383071   0.037583 -10.193 0.000522 ***
## const           -0.022029   0.170108  -0.129 0.903213
## trend           -0.004355   0.007821  -0.557 0.607303
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 0.1743 on 4 degrees of freedom
## Multiple R-Squared: 0.9963,  Adjusted R-squared: 0.9804
## F-statistic: 62.73 on 17 and 4 DF,  p-value: 0.0005533
##
##
## Estimation results for equation PHO_pts_diff:
## =============================================
## PHO_pts_diff = LAL_pts_diff.l1 + PHO_pts_diff.l1 + LAL_pts_diff.l2 + PHO_pts_diff.l
2 + LAL_pts_diff.l3 + PHO_pts_diff.l3 + LAL_pts_diff.l4 + PHO_pts_diff.l4 + LAL_pts_d
iff.l5 + PHO_pts_diff.l5 + LAL_pts_diff.l6 + PHO_pts_diff.l6 + LAL_pts_diff.l7 + PHO_pt
s_diff.l7 + LAL_pts_diff.l8 + PHO_pts_diff.l8 + const + trend
##
##                  Estimate Std. Error t value Pr(>|t|)
## LAL_pts_diff.l1 -0.448124   0.100731  -4.449  0.01126 *
## PHO_pts_diff.l1 -0.276764   0.138788  -1.994  0.11689
## LAL_pts_diff.l2  0.738642   0.094985   7.776  0.00147 **
## PHO_pts_diff.l2  0.483520   0.097503   4.959  0.00771 **
## LAL_pts_diff.l3  0.399042   0.208844   1.911  0.12864
## PHO_pts_diff.l3  0.043185   0.083928   0.515  0.63399
## LAL_pts_diff.l4 -0.290524   0.159994  -1.816  0.14357
## PHO_pts_diff.l4 -0.117261   0.077632  -1.510  0.20544
## LAL_pts_diff.l5 -0.196987   0.099112  -1.988  0.11778
## PHO_pts_diff.l5 -0.264688   0.084232  -3.142  0.03477 *
## LAL_pts_diff.l6  0.162441   0.080738   2.012  0.11454
## PHO_pts_diff.l6  0.165527   0.128477   1.288  0.26709
## LAL_pts_diff.l7  0.025535   0.098469   0.259  0.80819
## PHO_pts_diff.l7  0.185060   0.064322   2.877  0.04515 *
## LAL_pts_diff.l8 -0.208130   0.051741  -4.023  0.01583 *
## PHO_pts_diff.l8  0.165108   0.044845   3.682  0.02117 *
## const           -0.752200   0.202979  -3.706  0.02073 *
## trend            0.037830   0.009332   4.054  0.01543 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 0.208 on 4 degrees of freedom
## Multiple R-Squared: 0.9963,  Adjusted R-squared: 0.9807
## F-statistic: 63.77 on 17 and 4 DF,  p-value: 0.0005356
## Covariance matrix of residuals:
##              LAL_pts_diff PHO_pts_diff
## LAL_pts_diff      0.03038      0.03001
## PHO_pts_diff      0.03001      0.04325
##
## Correlation matrix of residuals:
##              LAL_pts_diff PHO_pts_diff
## LAL_pts_diff        1.000        0.828
## PHO_pts_diff        0.828        1.000
```

```
#残差检验
par(mfrow = c(2,1))
acf(resid(varmodel1)[,1], lag = 10)
acf(resid(varmodel1)[,2], lag = 10)
```



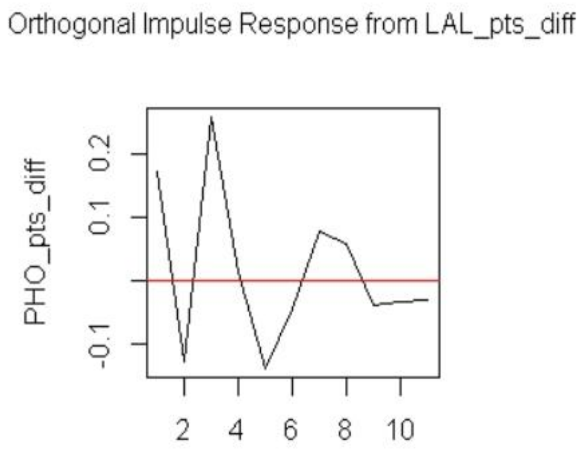


```
#Box.test 检验残差序列相关性，可以得到：在 10%的显著度下无法拒绝原假设，可以认为残差序列不存在相关性
Box.test(resid(varmodel1)[,1],lag = 10, type = 'Ljung')

##

##   Box-Ljung test

##

## data:  resid(varmodel1)[, 1]

## X-squared = 21.974, df = 10, p-value = 0.01524

Box.test(resid(varmodel1)[,2],lag = 10, type = 'Ljung')

##

##   Box-Ljung test

##

## data:  resid(varmodel1)[, 2]

## X-squared = 11.358, df = 10, p-value = 0.3303
```

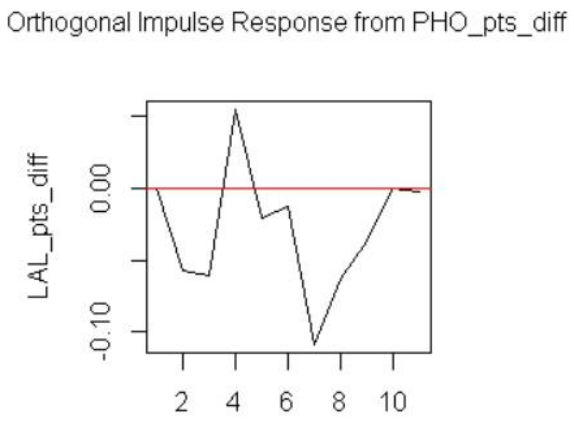
```
#格兰杰因果检验
causality(varmodel1, cause = 'LAL_pts_diff')
## $Granger
##
##  Granger causality H0: LAL_pts_diff do not Granger-cause PHO_pts_diff
##
## data:  VAR object varmodel1
## F-Test = 65.577, df1 = 8, df2 = 8, p-value = 1.718e-06
##
##
## $Instant
##
##  H0: No instantaneous causality between: LAL_pts_diff and PHO_pts_diff
##
## data:  VAR object varmodel1
## Chi-squared = 8.9484, df = 1, p-value = 0.002777
causality(varmodel1, cause = 'PHO_pts_diff')
## $Granger
##
##  Granger causality H0: PHO_pts_diff do not Granger-cause LAL_pts_diff
##
## data:  VAR object varmodel1
## F-Test = 46.882, df1 = 8, df2 = 8, p-value = 6.331e-06
##
##
## $Instant
##
##  H0: No instantaneous causality between: PHO_pts_diff and LAL_pts_diff
##
## data:  VAR object varmodel1
## Chi-squared = 8.9484, df = 1, p-value = 0.002777
```

Regardless of the casuality in the past or the casuality in the current period, the scoring data of the two teams are Granger causal, and the relationship is very close, because the p value is very significant. This is very consistent with the actual situation!

```
#脉冲效应
irf.LAL<- irf(varmodel1, impulse = 'LAL_pts_diff',
                  response = 'PHO_pts_diff', n.ahead = 10, boot = FALSE, seed = 12
34)
plot(irf.LAL)
```



```
irf.PHO<- irf(varmodel1, impulse = 'PHO_pts_diff',
                                    response = 'LAL_pts_diff', n.ahead = 10, boot = FALSE, seed = 12
34)
plot(irf.PHO)
```



The pulse effects between each other eventually tend to 0:

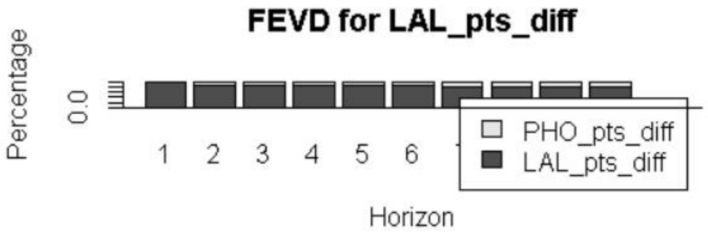
On the one hand, from the perspective of dynamic relationship stability, just like economic variables return to steady state after being impacted in the short term, teams may have mutual impacts in the short term due to key matches, player transfers and other events.

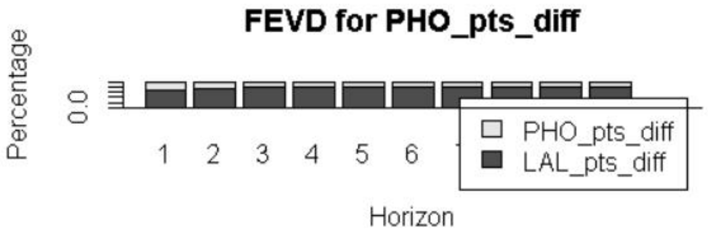
For example, a team introduces a superstar, which initially impacts the opponent's ticket sales and fan attention, but the opponent uses its own adjustments such as optimizing tactics and strengthening youth training to gradually eliminate the impact, and the long-term trend is stable, and the pulse tends to zero.

In addition, the team's internal adjustment mechanism plays a key role. When facing external shocks, the impact can be weakened by relying on the adaptive and feedback capabilities of coach tactical adjustments and player psychological and technical adjustments.

For example, the new defensive tactics of Team A impact the offensive efficiency of Team B. After the coach of Team B adjusts the offensive strategy, he adapts to the changes and makes the impact of the pulse gradually decrease to zero.

```
#方差分解
fevd.data <- fevd(varmodel1, n.ahead = 10)
fevd.data
## $LAL_pts_diff
##         LAL_pts_diff PHO_pts_diff
##  [1,]     1.0000000   0.00000000
##  [2,]     0.9054077   0.09459233
##  [3,]     0.9078046   0.09219542
##  [4,]     0.8795637   0.12043634
##  [5,]     0.8761177   0.12388233
##  [6,]     0.8836763   0.11632372
##  [7,]     0.8289620   0.17103803
##  [8,]     0.8302555   0.16974448
##  [9,]     0.8337132   0.16628682
## [10,]     0.8337313   0.16626869
##
## $PHO_pts_diff
##         LAL_pts_diff PHO_pts_diff
##  [1,]     0.6856121    0.3143879
##  [2,]     0.7564558    0.2435442
##  [3,]     0.8297938    0.1702062
##  [4,]     0.8145849    0.1854151
##  [5,]     0.8239947    0.1760053
##  [6,]     0.8257610    0.1742390
##  [7,]     0.8277120    0.1722880
##  [8,]     0.8307587    0.1692413
##  [9,]     0.8313110    0.1686890
## [10,]     0.8301718    0.1698282
plot(fevd.data)
```





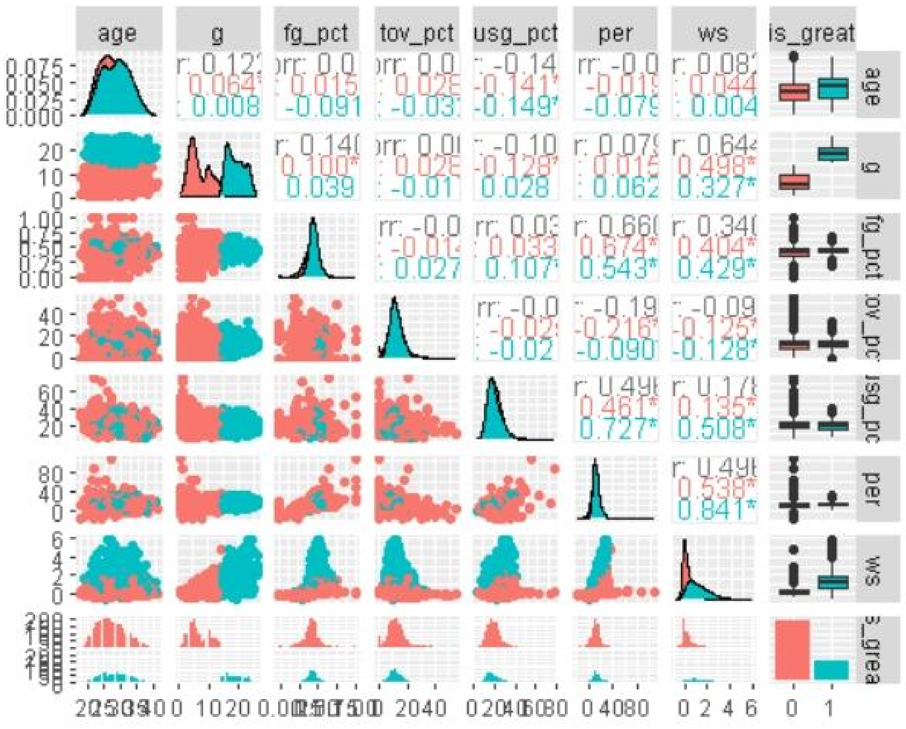
In the upper graph, in the early prediction stage (Horizon = 1), the Los Angeles Lakers' data itself explains nearly 100% of its variance, while the Phoenix Suns' data contributes almost 0. As the prediction time increases, the Los Angeles Lakers' data's self-explanatory degree gradually decreases, but it still dominates. In the lower graph, in the early prediction stage (Horizon = 1), the Phoenix Suns' data itself explains nearly 100% of its variance, while the Los Angeles Lakers' data contributes almost 0. As the time span increases, the Phoenix Suns' self-explanatory degree gradually decreases but still dominates. Overall, these two variables are greatly affected by themselves in the short term. As time goes by, although the influence of the other variable gradually increases, their own influence always dominates.

3. Dimensionality reduction and clustering

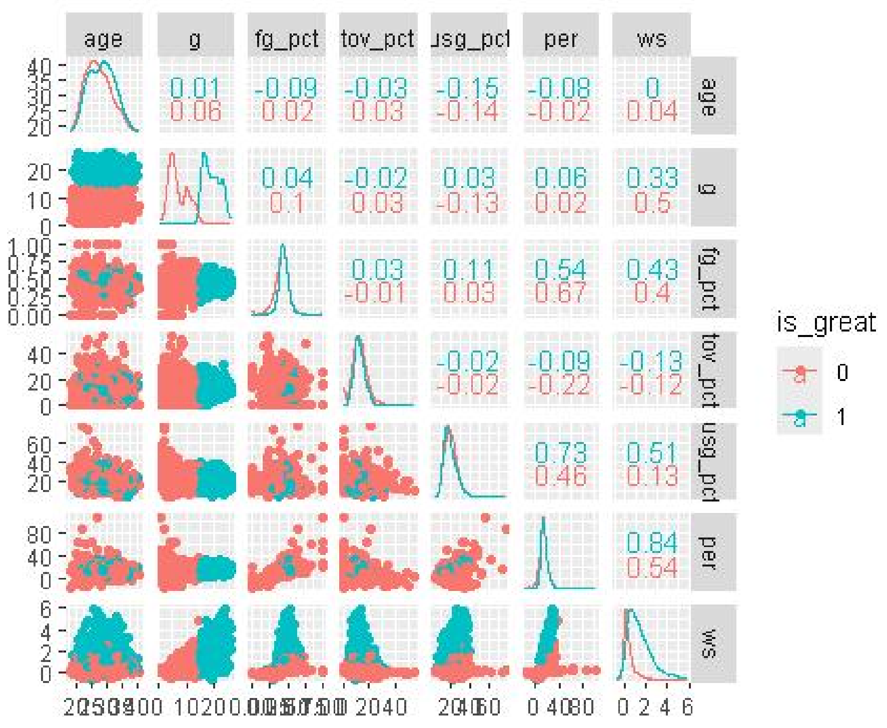
2. 1 PCA

```
library(tidyverse)
library(mclust)
library(psych)
library(GGally)
library(factoextra)
library(Rtsne)
library(umap)

playoffStats_trimmed <- read.csv("D:/桌面/数据探索与可视化大作业/playoddStats_trimmed.c
sv")
```

```
#playoffStats_PCA 选取了：年龄、出场次数、投篮命中率、失误百分比、球员的使用率、
球员的总胜利贡献、球员是否”great“
#其中 前三个为基础数据，后三个为高阶数据，is_great 是二分类指示变量
playoffStats_PCA <- playoffStats_trimmed |>
  dplyr::select(age,g,fg_pct,tov_pct,usg_pct,per,ws,is_great)
playoffStats_PCA$is_great <- as.factor(playoffStats_PCA$is_great)
#总体可视化
ggpairs(playoffStats_PCA,aes(col = is_great))
```



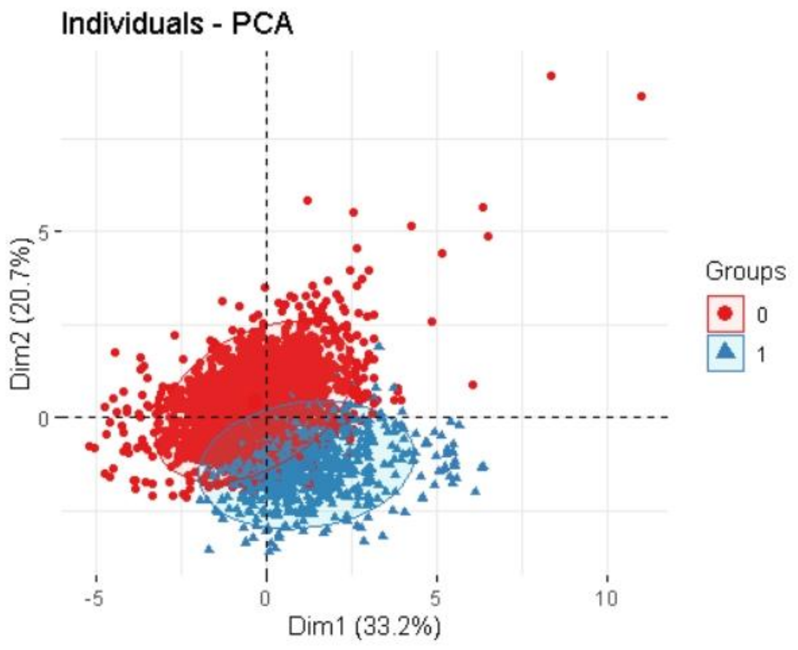
```
ggscatmat(playoffStats_PCA, color = "is_great")
```



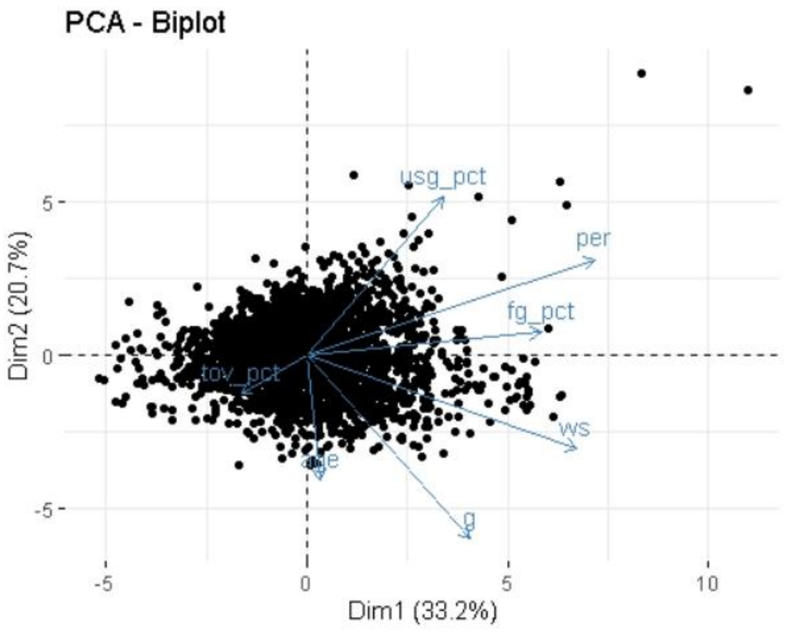
```
pca <- dplyr::select(playoffStats_PCA,-is_great) |>
  prcomp(center = T, scale = T)
#prcomp()用于主成分分析, center = T 表示在进行 PCA 之前将数据中心化, 即每个变量的均值将被减去。scale = T #表示在进行 PCA 之前将数据标准化, 即每个变量将被除以其标准差。这样做是为了确保每个变量对主成分分析的贡献#是等价的, 不受其量纲的影响
pca
## Standard deviations (1, .., p=7):
## [1] 1.5254660 1.2028651 0.9965048 0.9788394 0.9264104 0.5202179 0.3821752
##
## Rotation (n x k) = (7 x 7):
##                 PC1         PC2         PC3        PC4         PC5
  PC6
## age      0.02522457 -0.40756214  0.17084480  0.6388282 -0.62721340 -0.03822164
## g        0.31881902 -0.59830463 -0.05088975 -0.3872315  0.01738353 -0.58357116
## fg_pct   0.45870921  0.07315496 -0.14064235  0.4938623  0.47763674 -0.29415923
## tov_pct -0.12787427 -0.12719338 -0.95988966  0.1160263 -0.07882935  0.10950829
## usg_pct  0.26824398  0.51464955 -0.16246771 -0.2667929 -0.60337051 -0.32349881
## per      0.56434735  0.31009076  0.01481104  0.1977265 -0.01979728  0.12757950
## ws       0.52963141 -0.30557229 -0.02094608 -0.2726059 -0.08661604  0.66221906
##                 PC7
## age     -0.03331412
## g        0.21677297
## fg_pct  -0.45374828
## tov_pct   0.12005676
## usg_pct  -0.31128655
## per       0.72757967
## ws       -0.32454125
```

PC1 has the highest explained variance (about 0.33, not particularly high). In the linear expression of PC1, per and ws have the highest weights. This shows that under the premise of PCA analysis, **high-order data** is more explanatory of whether a player is "great".

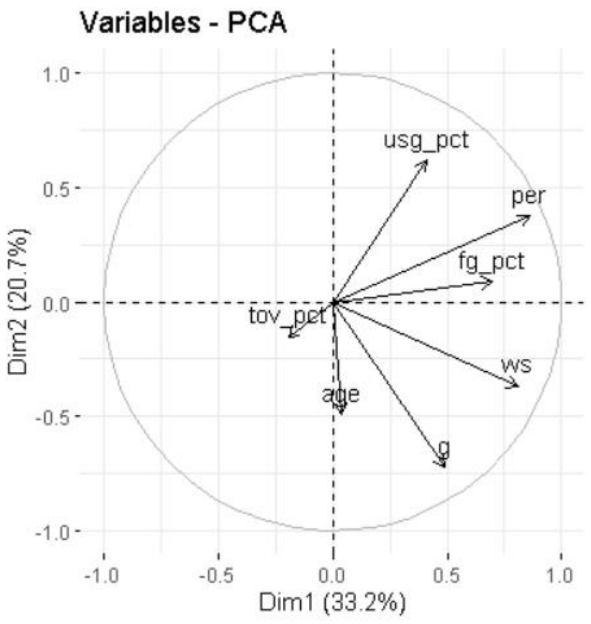
```
summary(pca)
## Importance of components:
##                            PC1    PC2    PC3    PC4    PC5     PC6     PC
7
## Standard deviation     1.5255 1.2029 0.9965 0.9788 0.9264 0.52022 0.38218
## Proportion of Variance 0.3324 0.2067 0.1419 0.1369 0.1226 0.03866 0.02087
## Cumulative Proportion  0.3324 0.5391 0.6810 0.8179 0.9405 0.97913 1.00000
pcaDat <- get_pca(pca)
pcaDat
## Principal Component Analysis Results for variables
##  ==================================================
##    Name        Description
## 1 "$coord"    "Coordinates for the variables"
## 2 "$cor"      "Correlations between variables and dimensions"
## 3 "$cos2"     "Cos2 for the variables"
## 4 "$contrib" "contributions of the variables"
fviz_pca_ind(pca, label = "none", habillage = playoffStats_PCA$is_great, addEllipses = T,
 ellipse.level = 0.9) +
  scale_color_brewer(palette = "Set1")
```



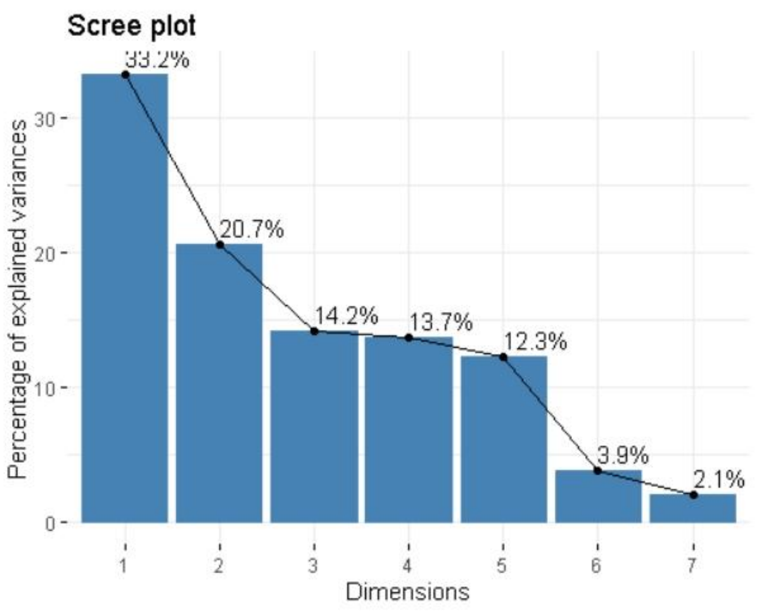
```
fviz_pca_biplot(pca, label = "var")
```



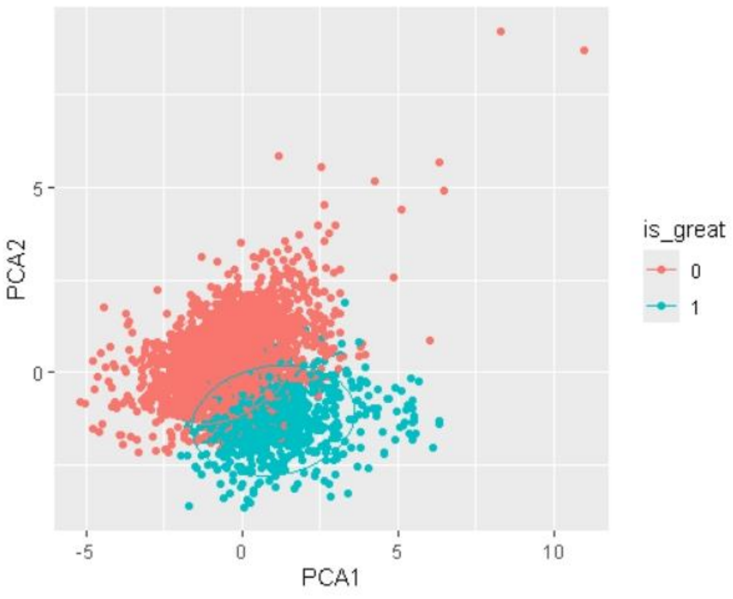
```
fviz_pca_var(pca)
```



```
fviz_screeplot(pca, addlabels = T, choice = "variance")
```



```
notePCA <- playoffStats_PCA %>%
  mutate(PCA1 = pca$x[,1], PCA2 = pca$x[,2]) #选择第一列和第二列进行可视化
ggplot(notePCA, aes(PCA1, PCA2, col = is_great)) +
  geom_point() +
  stat_ellipse(level = 0.9)
```



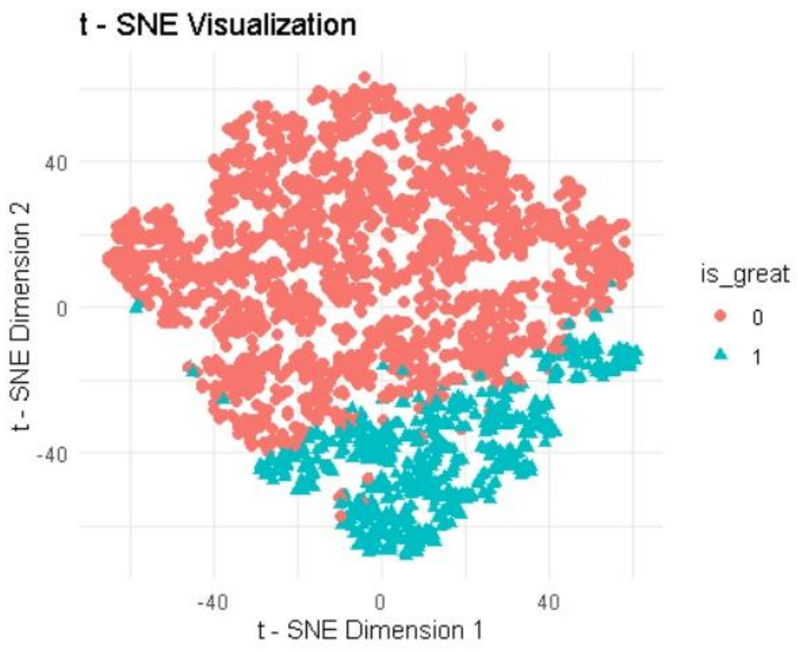
Since PCA is only a linear dimensionality reduction, although the data points of different colors have a certain separation trend, there are still a lot of overlapping areas. This means that only the first two principal components, PCA1 and PCA2, which account for the highest proportion of variance explanation, cannot fully distinguish the two types of data.

3. 2 t-SNE

```
playoffStats_Tsne <- playoffStats_PCA

Tsne <- playoffStats_Tsne %>%
  dplyr::select(-is_great)
Tsne <- Rtsne(Tsne, perplexity = 30, theta = 0, max_iter = 1000, verbose = T)
## Performing PCA
## Read the 3017 x 7 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.000000
## Computing input similarities...
## Symmetrizing...
## Done in 12.20 seconds!
## Learning embedding...
## Iteration 50: error is 83.735954 (50 iterations in 4.91 seconds)
## Iteration 100: error is 78.521998 (50 iterations in 4.71 seconds)
## Iteration 150: error is 78.508640 (50 iterations in 4.78 seconds)
## Iteration 200: error is 78.508660 (50 iterations in 4.70 seconds)
## Iteration 250: error is 78.508661 (50 iterations in 4.76 seconds)
## Iteration 300: error is 2.231664 (50 iterations in 4.83 seconds)
## Iteration 350: error is 1.900658 (50 iterations in 4.78 seconds)
## Iteration 400: error is 1.749402 (50 iterations in 4.78 seconds)
## Iteration 450: error is 1.663589 (50 iterations in 4.91 seconds)
## Iteration 500: error is 1.610684 (50 iterations in 4.61 seconds)
## Iteration 550: error is 1.575342 (50 iterations in 4.71 seconds)
## Iteration 600: error is 1.550211 (50 iterations in 4.60 seconds)
## Iteration 650: error is 1.532333 (50 iterations in 4.70 seconds)
## Iteration 700: error is 1.518999 (50 iterations in 4.68 seconds)
## Iteration 750: error is 1.508815 (50 iterations in 4.64 seconds)
## Iteration 800: error is 1.500905 (50 iterations in 4.62 seconds)
## Iteration 850: error is 1.494647 (50 iterations in 4.66 seconds)
## Iteration 900: error is 1.489620 (50 iterations in 4.64 seconds)
## Iteration 950: error is 1.485527 (50 iterations in 4.66 seconds)
## Iteration 1000: error is 1.482157 (50 iterations in 4.65 seconds)
## Fitting performed in 94.34 seconds.
```

```
Tsne1 <- playoffStats_Tsne %>%
  mutate_if(.funs = scale, .predicate = is.numeric, scale = F) %>%
  mutate(tSNE1 = Tsne$Y[,1], tSNE2 = Tsne$Y[,2]) %>%
  pivot_longer(names_to = "Feature", values_to = "Value", c(-tSNE1, -tSNE2, -is_great))
```

```
ggplot(Tsne1, aes(tSNE1, tSNE2, shape = is_great, color = is_great)) +
  geom_point(size = 2) +
  theme_minimal() +
  labs(title = "t - SNE Visualization", x = "t - SNE Dimension 1", y = "t - SNE Dimen
sion 2")
```



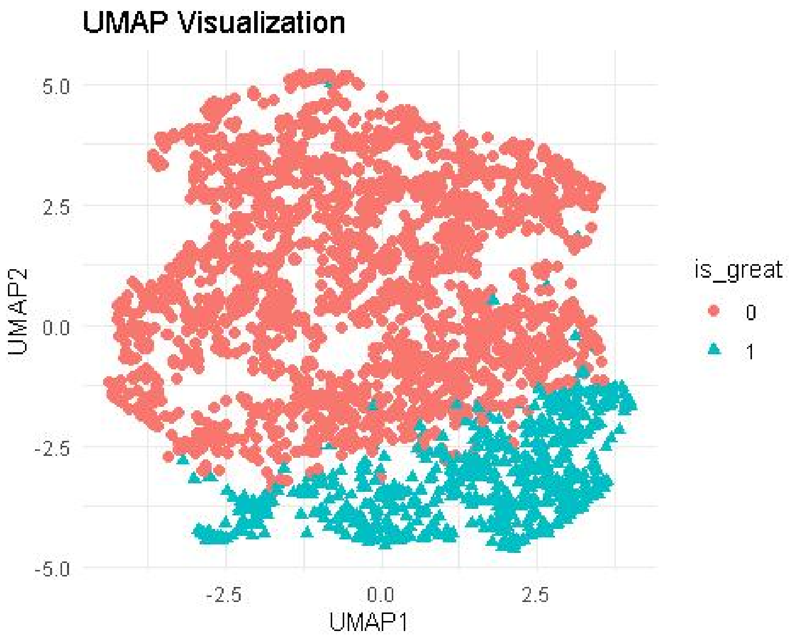
From the perspective of tSNE dimensionality reduction classification, the data points have a certain separation trend. However, its disadvantages are obvious. The two types of data points overlap seriously, especially in data-intensive areas, where red and cyan data points are mixed in large numbers and there is no clear classification boundary, which makes accurate classification more difficult.

3. 3 Umap

```
playoffStats_UMAP <- playoffStats_Tsne

Umap <- playoffStats_UMAP %>%
  dplyr::select(-is_great) %>%
  as.matrix() %>%
  umap(n_neibours = 7, min_dist = 0.1,
       metric = "manhattan", n_epochs = 200, verbose = T)
## [2024-11-30 02:04:22.110112]  starting umap
## [2024-11-30 02:04:22.221955]  creating graph of nearest neighbors
## [2024-11-30 02:04:29.046568]  creating initial embedding
## [2024-11-30 02:04:29.308585]  optimizing embedding
## [2024-11-30 02:04:37.161231]  done
```

```
Umap1 <-playoffStats_UMAP %>%
  mutate_if(.funs = scale, .predicate = is.numeric, scale = F) %>%
  mutate(UMAP1 = Umap$layout[,1], UMAP2 = Umap$layout[,2]) %>%
  pivot_longer(names_to = "Feature", values_to = "Value", c(-UMAP1, -UMAP2, -is_grea
t))
```

```
ggplot(Umap1, aes(UMAP1, UMAP2,shape = is_great,color = is_great)) +
  geom_point(size = 2) +
  labs(title = "UMAP Visualization", x = "UMAP1", y = "UMAP2") +
  theme_minimal()
```



The UMAP dimensionality reduction results show the distribution characteristics of the data to a certain extent. As can be seen from the figure, data points of different categories are separated to a certain extent, and the cyan (is_great = 1) and red (is_great = 0) data points each form a relatively concentrated area. However, there is still a lot of overlap between the two types of data points, especially in areas where the data is denser, which shows that although UMAP has reduced the dimensionality of the data, it has not completely separated the two types of data clearly. In general, UMAP dimensionality reduction can reveal the general structure of the data, but it still has certain limitations when distinguishing specific categories.

3. 4 Kmeans

```
pacman::p_load(mlr3verse,mlr3cluster,mlr3tuning,GGally,data.table)
theme_set(theme_bw())

playoffStats_clust <- playoffStats_UMAP
playoffStats_clust$is_great <- as.numeric(playoffStats_clust$is_great)
```

```
str(playoffStats_clust)
## 'data.frame':    3017 obs. of  8 variables:
##  $ age     : int  23 34 26 33 28 28 27 31 22 22 ...
##  $ g       : int  9 12 5 5 5 18 10 13 5 15 ...
##  $ fg_pct  : num  0.667 0.404 0.426 0.4 0.4 0.471 0.484 0.4 0.308 0.455 ...
##  $ tov_pct : num  0 12.7 10.8 9.1 10.4 9.7 13.8 14 10.2 12.5 ...
##  $ usg_pct : num  22.6 15.1 20.2 22.5 11 12.8 28.4 16.3 15 40.4 ...
##  $ per     : num  25.8 9.9 15.7 10.8 7.8 12.6 19.1 9.8 6 28.6 ...
##  $ ws      : num  0.2 0.3 0.2 0 0 1.5 0.9 0.2 -0.1 1.9 ...
##  $ is_great: num  1 1 1 1 1 2 1 1 1 2 ...
```

```
Scale <- as.data.table(scale(playoffStats_clust))
```

```
taskC = TaskClust$new("Scale", Scale)
class(taskC)
## [1] "TaskClust"         "TaskUnsupervised" "Task"              "R6"
mlr_learners$keys("clust")
##  [1] "clust.agnes"        "clust.ap"           "clust.cmeans"
##  [4] "clust.cobweb"       "clust.dbscan"       "clust.dbscan_fpc"
##  [7] "clust.diana"        "clust.em"           "clust.fanny"
## [10] "clust.featureless"  "clust.ff"           "clust.hclust"
## [13] "clust.hdbscan"      "clust.kkmeans"      "clust.kmeans"
## [16] "clust.MBatchKMeans" "clust.mclust"       "clust.meanshift"
## [19] "clust.optics"       "clust.pam"          "clust.SimpleKMeans"
## [22] "clust.xmeans"
```

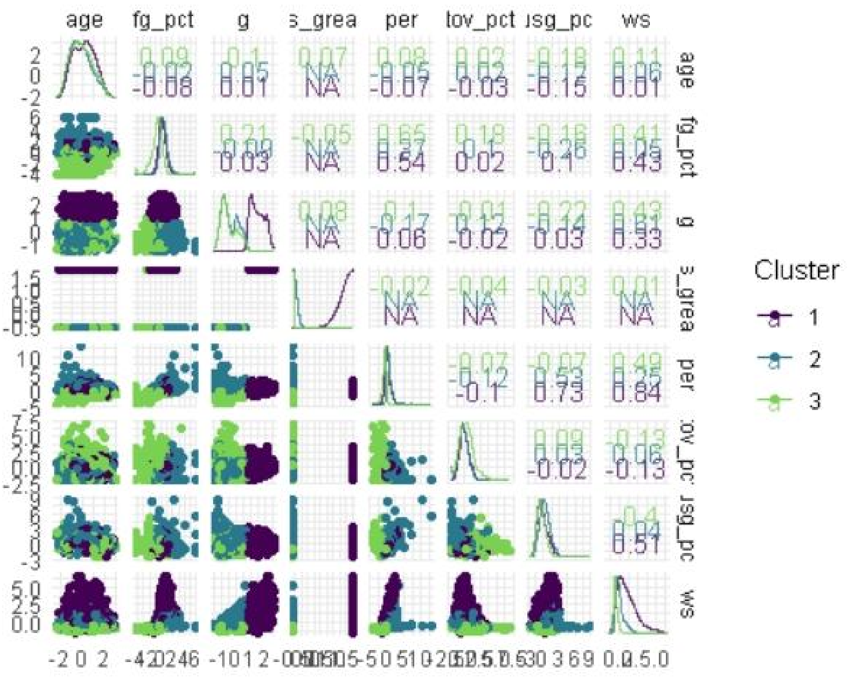
```
kmeans_lrn = lrn("clust.kmeans")
class(kmeans_lrn)
## [1] "LearnerClustKMeans" "LearnerClust"       "Learner"
## [4] "R6"
kmeans_lrn$param_set$params
## 索引(index): <id__cls__grouping>
##             id      cls                                    grouping     cargo
##         <char>   <char>                                      <char>    <list>
## 1:     centers ParamUty                                    ParamUty <list[2]>
## 2:   iter.max  ParamInt                                              ParamInt
## 3:  algorithm  ParamFct  "Forgy","Hartigan-Wong","Lloyd","MacQueen"
## 4:      nstart  ParamInt                                              ParamInt
## 5:       trace  ParamInt                                              ParamInt
##      lower  upper      tolerance                               levels  special_vals
##      <num> <num>          <num>                                <list>        <list>
## 1:      NA      NA              NA                                       <list[0]>
## 2:       1     Inf  1.490116e-08                                         <list[0]>
## 3:      NA      NA              NA  Hartigan-Wong,Lloyd,Forgy,MacQueen   <list[0]>
## 4:       1     Inf  1.490116e-08                                         <list[0]>
## 5:       0     Inf  1.490116e-08                                         <list[0]>
##              default  storage_type            .tags  .trafo  .requirements  .init_given
##               <list>        <char>           <list>  <list>         <list>       <lgcl>
## 1:  <NoDefault[0]>             list  required,train                               TRUE
## 2:                 10        integer            train                              FALSE
## 3:   Hartigan-Wong      character           train                              FALSE
## 4:                  1        integer            train                              FALSE
## 5:                  0        integer            train                              FALSE
##        .init
##      <list>
## 1:         2
## 2:
## 3:
## 4:
## 5:
```

```
kmeans_lrn$param_set$values = list(centers = 3, iter.max = 100, nstart = 10)
kmeans_lrn$train(taskC)
kmeans_lrn$state
```

```
kmeans_lrn$assignments
```

```
kmeans_lrn1 = kmeans_lrn$clone()
kmeans_lrn1$predict(taskC)
## <PredictionClust> for 3017 observations:
##       row_ids  partition
##             1          2
##             2          3
##             3          2
## ---
##          3015          2
##          3016          1
##          3017          3
```

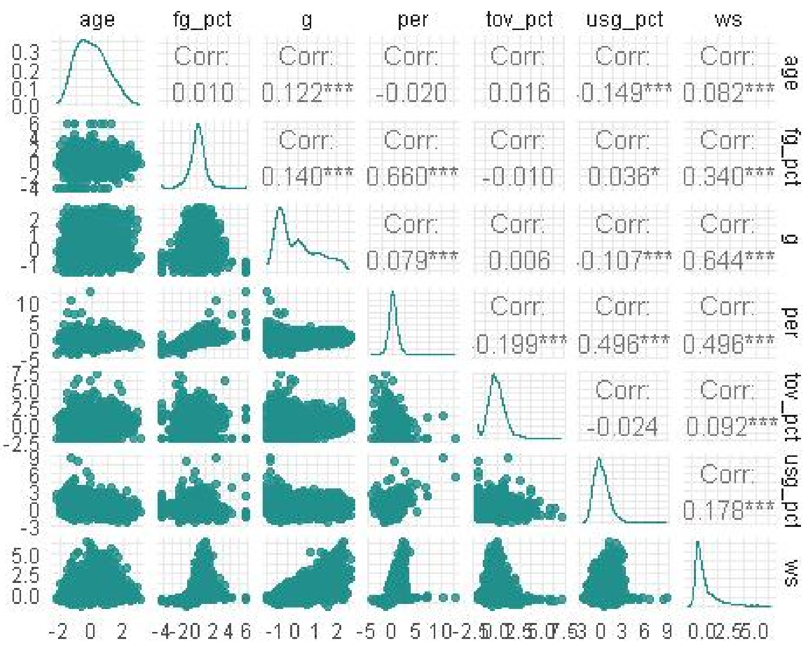
```
autoplot(kmeans_lrn1$predict(taskC),taskC,  type  =  "scatter"  )
```



The kmeans clustering results show the distribution of data on each variable through different colors. Its advantages are that it can intuitively present the general clustering structure and variable relationship of the data, and the clusters have good separation on some variables. The disadvantages are that there are many data points that are prone to overlap, the complex variable relationship is not fully displayed, and the clusters on some variables overlap seriously.

3. 5 DBSCAN

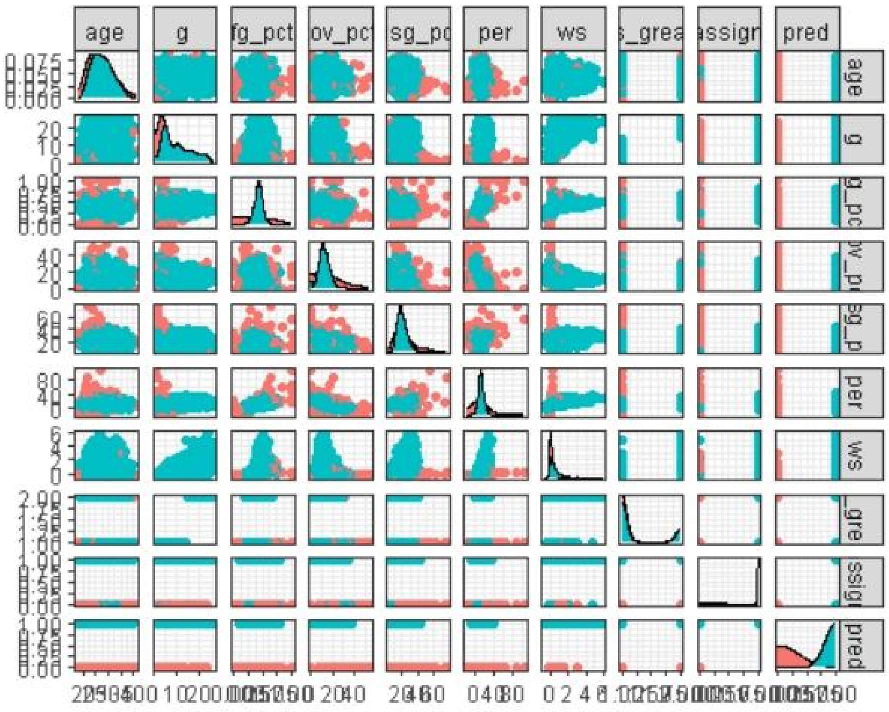
```
#建立任务
mlr_learners$keys("clust")
##  [1] "clust.agnes"         "clust.ap"            "clust.cmeans"
##  [4] "clust.cobweb"        "clust.dbscan"        "clust.dbscan_fpc"
##  [7] "clust.diana"         "clust.em"            "clust.fanny"
## [10] "clust.featureless"   "clust.ff"            "clust.hclust"
## [13] "clust.hdbscan"       "clust.kkmeans"       "clust.kmeans"
## [16] "clust.MBatchKMeans"  "clust.mclust"        "clust.meanshift"
## [19] "clust.optics"        "clust.pam"           "clust.SimpleKMeans"
## [22] "clust.xmeans"
data(playoffStats_clust, package = "mclust")
## Warning in data(playoffStats_clust, package = "mclust"):
## 没有'playoffStats_clust'这个数据集
bn = as.data.table(playoffStats_clust)[,-"is_great"]
bnScaled = as.data.table(scale(bn))

task = TaskClust$new(id = "playoffStats_scaled", backend = bnScaled)
# train_set = sample(task$nrow, 0.8 * task$nrow) test_set =
# setdiff(seq_len(task$nrow), train_set)

class(task)
## [1] "TaskClust"         "TaskUnsupervised"  "Task"              "R6"
autoplot(task)
```



```
#建立 learner，设置超参数
mlr_learners$keys("clust")
##  [1] "clust.agnes"         "clust.ap"            "clust.cmeans"
##  [4] "clust.cobweb"        "clust.dbscan"        "clust.dbscan_fpc"
##  [7] "clust.diana"         "clust.em"            "clust.fanny"
## [10] "clust.featureless"   "clust.ff"            "clust.hclust"
## [13] "clust.hdbscan"       "clust.kkmeans"       "clust.kmeans"
## [16] "clust.MBatchKMeans"  "clust.mclust"        "clust.meanshift"
## [19] "clust.optics"        "clust.pam"           "clust.SimpleKMeans"
## [22] "clust.xmeans"
dbscan_lrn = lrn("clust.dbscan")

class(dbscan_lrn)
## [1] "LearnerClustDBSCAN" "LearnerClust"       "Learner"
## [4] "R6"
dbscan_lrn$param_set$params
## 索引(index): <id__cls__grouping>
##              id      cls                                          grouping
##          <char>   <char>                                            <char>
## 1:          eps ParamDbl                                          ParamDbl
## 2:       minPts ParamInt                                          ParamInt
## 3: borderPoints ParamLgl                                          ParamLgl
## 4:      weights ParamUty                                          ParamUty
## 5:       search ParamFct                       "dist","kdtree","linear"
## 6:   bucketSize ParamInt                                          ParamInt
## 7:    splitRule ParamFct "FAIR","MIDPT","SL_FAIR","SL_MIDPT","STD","SUGGEST"
## 8:       approx ParamDbl                                          ParamDbl
##         cargo  lower  upper     tolerance                             levels
##        <list> <num>  <num>         <num>                               <lis
t>
## 1:               0    Inf  1.490116e-08
## 2:               0    Inf  1.490116e-08
## 3:              NA     NA            NA                              TRUE,F
ALSE
## 4: <list[2]>    NA     NA            NA

## 5:              NA     NA            NA                 kdtree,linear,dist
## 6:               1    Inf  1.490116e-08
## 7:              NA     NA            NA STD,MIDPT,FAIR,SL_MIDPT,SL_FAIR,S
UGGEST
## 8:            -Inf    Inf  1.490116e-08
##    special_vals        default storage_type            .tags  .trafo  .requirements
##          <list>         <list>       <char>           <list>  <list>         <list>
## 1:    <list[0]> <NoDefault[0]>      numeric   required,train
## 2:      <list[0]>                    5          integer              train
## 3:      <list[0]>               TRUE           logical              train
## 4:      <list[0]>  <NoDefault[0]>              list             train
## 5:      <list[0]>              kdtree       character              train
## 6:      <list[0]>                   10          integer              train                   <list[2]>
## 7:      <list[0]>            SUGGEST       character              train                   <list[2]>
## 8:      <list[0]>                    0          numeric              train
##      .init_given   .init
##               <lgcl>  <list>
## 1:            FALSE
## 2:            FALSE
## 3:            FALSE
## 4:            FALSE
## 5:            FALSE
## 6:            FALSE
## 7:            FALSE
## 8:            FALSE
dbscan_lrn$param_set$values = list(eps = 1.5, minPts = 9)
```

```
#训练，预测，结果可视化
dbscan_lrn$train(task)
dbscan_lrn$state
```

```
dbPred = dbscan_lrn$predict(task)
bnD = cbind(playoffStats_clust, assign = dbscan_lrn$assignments, pred = dbPred$partit
ion)

dplyr::count(bnD, is_great, pred)
##     is_great  pred     n
## 1            1       0    85
## 2            1       1  2201
## 3            2       0      5
## 4            2       1    726
ggpairs(bnD,aes(col = factor(pred)),upper = list(continuous = "points"))
```

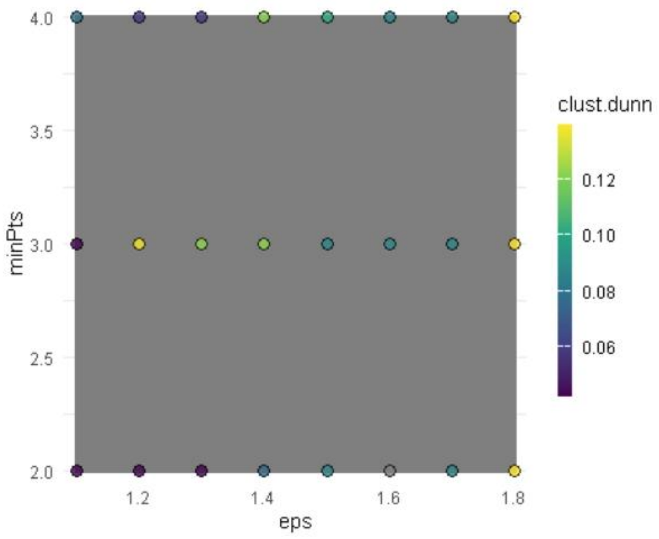


DBSCAN clustering produced diverse clustering results on these variables, reflecting the complexity and diversity of the data.

```
#超参数优化
instance = TuningInstanceBatchSingleCrit$new(
  task = task,
  learner = dbscan_lrn,
  resampling = rsmp("holdout"),
  measure = msr("clust.dunn"),
  search_space = ps(eps = p_dbl(1.1, 1.8),
                    minPts = p_int(2,4)),
  terminator = trm("none"))
tuner = tnr("grid_search", resolution = 8)

tuner$optimize(instance)
```

```
class(instance)
## [1] "TuningInstanceBatchSingleCrit" "OptimInstanceBatchSingleCrit"
## [3] "OptimInstanceBatch"            "OptimInstance"
## [5] "R6"
instance$archive
## <ArchiveBatchTuning> with 24 evaluations
##        eps minPts clust.dunn batch_nr warnings errors
##      <num>  <int>      <num>    <int>    <int>  <int>
##  1:    1.4      4      0.123        1        0      0
##   2:   1.7      4     0.087        2        0      0
##   3:   1.8      3     0.140        3        0      0
##   4:   1.6      2       Inf        4        0      0
##   5:   1.5      2     0.087        5        0      0
##   6:   1.1      4     0.078        6        0      0
##   7:   1.1      3     0.042        7        0      0
##   8:   1.3      4     0.059        8        0      0
##   9:   1.8      4     0.140        9        0      0
##  10:   1.2      4     0.059       10        0      0
##  11:   1.7      2     0.087       11        0      0
##  12:   1.4      3     0.123       12        0      0
##  13:   1.5      3     0.087       13        0      0
##  14:   1.7      3     0.087       14        0      0
##  15:   1.4      2     0.076       15        0      0
##  16:   1.3      3     0.123       16        0      0
##  17:   1.6      4     0.087       17        0      0
##  18:   1.3      2     0.043       18        0      0
##  19:   1.5      4     0.103       19        0      0
##  20:   1.2      3     0.139       20        0      0
##  21:   1.1      2     0.042       21        0      0
##  22:   1.6      3     0.087       22        0      0
##  23:   1.8      2     0.140       23        0      0
##  24:   1.2      2     0.043       24        0      0
##        eps  minPts  clust.dunn  batch_nr  warnings  errors
##        eps  minPts  clust.dunn  x_domain_eps  x_domain_minPts  batch_nr  warnings
##      <num>   <int>       <num>         <num>            <int>     <int>     <int>
##   1:   1.4      4       0.123          1.4                4         1         0
##   2:   1.7      4       0.087          1.7                4         2         0
##   3:   1.8      3       0.140          1.8                3         3         0
##   4:   1.6      2         Inf          1.6                2         4         0
##   5:   1.5      2       0.087          1.5                2         5         0
##   6:   1.1      4       0.078          1.1                4         6         0
##   7:   1.1      3       0.042          1.1                3         7         0
##   8:   1.3      4       0.059          1.3                4         8         0
##   9:   1.8      4       0.140          1.8                4         9         0
##  10:   1.2      4       0.059          1.2                4        10         0
##  11:   1.7      2       0.087          1.7                2        11         0
##  12:   1.4      3       0.123          1.4                3        12         0
##  13:   1.5      3       0.087          1.5                3        13         0
##  14:   1.7      3       0.087          1.7                3        14         0
##  15:   1.4      2       0.076          1.4                2        15         0
##  16:   1.3      3       0.123          1.3                3        16         0
##  17:   1.6      4       0.087          1.6                4        17         0
##  18:   1.3      2       0.043          1.3                2        18         0
## 19:    1.5      4        0.103          1.5                4        19        0
## 20:    1.2      3        0.139          1.2                3        20        0
## 21:    1.1      2        0.042          1.1                2        21        0
## 22:    1.6      3        0.087          1.6                3        22        0
## 23:    1.8      2        0.140          1.8                2        23        0
## 24:    1.2      2        0.043          1.2                2        24        0
##        eps  minPts  clust.dunn  x_domain_eps  x_domain_minPts  batch_nr  warnings
```

```
autoplot(instance, type = "surface")
```



4. Machine Learning Prediction

4.1 Neural Networks

```
library(tidyverse)
library(tidytext)
library(neuralnet)
library(caret)
library(gbm)
library(e1071)

playoffStats_trimmed <- read.csv("D:/桌面/数据探索与可视化大作业/playoddStats_trimmed.c
sv")
```

```
set.seed(123)

#playoffStats_neural 提出了 chr 变量，保留了所有的数值指标
playoffStats_neural <- playoffStats_trimmed |>
  dplyr::select(-X,-season,-pos,-player,-team_id)

n <- nrow(playoffStats_neural)

# 划分训练集和测试集
train_index <- sample(1:n, size = 0.8 * n)
train_data <- playoffStats_neural[train_index, ]
test_data <- playoffStats_neural[-train_index, ]
#除了最后一列是目标变量外，其他列都是输入变量，构建一个简单的神经网络

# 获取除最后一列之外的所有列名作为输入变量
input_cols <- names(playoffStats_neural)[1:(ncol(playoffStats_neural) - 1)]
# 构建公式
formula_str <- as.formula(paste("is_great ~", paste(input_cols, collapse = " + ")))

# 构建神经网络
nn <- neuralnet(formula = formula_str,
                data = train_data,
                hidden = c(5, 3), # 这里设置两个隐藏层，分别有 5 个和 3 个神经元
                linear.output = FALSE) # 因为是二分类问题，使用逻辑输出
# 对测试集进行预测
predictions <- compute(nn, test_data[, input_cols])$net.result

# 将预测结果转换为 0 或 1 （设 0.8 为阈值）
```

```r
#设定 0.8 作为阈值将预测概率转换为 0 或 1, 主要是为了根据具体应用场景的需求平衡假阳性和假阴性的风险、考虑错误成本、优化特定性能指标以及应对数据不平衡问题。这样做有助于使模型的决策更贴合实际应用的要求。简而言之，这是为了通过调整分类边界来达到最佳的应用效果。
predicted_classes <- ifelse(predictions > 0.8, 1, 0)

# 获取真实的测试集标签
true_classes <- test_data$is_great

# 计算准确率
accuracy <- sum(predicted_classes == true_classes) / length(true_classes)
print(paste("Accuracy:", accuracy))
## [1] "Accuracy: 0.990066225165563"
```

The accuracy of predictions for all numerical data is 0.758278145695364

In order to compare the difference in prediction accuracy between low-order data and high-order data, we need to divide the data set: (ensuring that the number of columns of low-order data and high-order data is the same)

```r
playoffStats_neural_low <- playoffStats_neural |>
  dplyr::select(age,g,gs,mp_per_g,fga_per_g,fg_pct,fg3a_per_g,fg3_pct,fg2a_per_g,fg2_pct,fta_per_g,ft_pct,is_great)

playoffStats_neural_high <- playoffStats_neural |>
  dplyr::select(ast_pct,blk_pct,stl_pct,tov_pct,trb_pct,ts_pct,usg_pct,dws,ows,per,ws,ws_per_48,is_great)
#针对低阶数据

# 划分训练集和测试集
train_index_low <- sample(1:n, size = 0.8 * n)
train_data_low <- playoffStats_neural_low[train_index_low, ]
test_data_low <- playoffStats_neural_low[-train_index_low, ]

# 获取除最后一列之外的所有列名作为输入变量
input_cols_low <- names(playoffStats_neural_low)[1:(ncol(playoffStats_neural_low) - 1)]
# 构建公式
formula_str_low <- as.formula(paste("is_great ~", paste(input_cols_low, collapse = " + ")))

# 构建神经网络
nn_low <- neuralnet(formula = formula_str_low,
                    data = train_data_low,
                    hidden = c(5, 3),
                       linear.output = FALSE)

# 对测试集进行预测
predictions_low <- compute(nn_low, test_data_low[, input_cols_low])$net.result

# 将预测结果转换为 0 或 1（设 0.5 为阈值）
predicted_classes_low <- ifelse(predictions_low > 0.8, 1, 0)

# 获取真实的测试集标签
true_classes_low <- test_data_low$is_great

# 计算准确率
accuracy_low <- sum(predicted_classes_low == true_classes_low) / length(true_classes_lo
w)
print(paste("Low Order Data Accuracy:", accuracy_low))
## [1] "Low Order Data Accuracy: 0.996688741721854"
```

```
#针对高阶数据

# 划分训练集和测试集
train_index_high <- sample(1:n, size = 0.8 * n)
train_data_high <- playoffStats_neural_high[train_index_high, ]
test_data_high <- playoffStats_neural_high[-train_index_high, ]

# 获取除最后一列之外的所有列名作为输入变量
input_cols_high <- names(playoffStats_neural_high)[1:(ncol(playoffStats_neural_high) - 1)]
# 构建公式
formula_str_high <- as.formula(paste("is_great ~", paste(input_cols_high, collapse = " +
")))
# 构建神经网络
nn_high <- neuralnet(formula = formula_str_high,
                       data = train_data_high,
 #                      hidden = c(10, 5),   # 改变隐藏层节点数
                       linear.output = TRUE)
# 对测试集进行预测
predictions_high <- compute(nn_high, test_data_high[, input_cols_high])$net.result

# 将预测结果转换为 0 或 1（设 0.5 为阈值）
predicted_classes_high <- ifelse(predictions_high > 0.8, 1, 0)

# 获取真实的测试集标签
true_classes_high <- test_data_high$is_great

# 计算准确率
```

```
accuracy_high <- sum(predicted_classes_high == true_classes_high) / length(true_classes_hi
gh)
print(paste("High Order Data Accuracy:", accuracy_high))

## [1] "High Order Data Accuracy: 0.768211920529801"
```

The prediction accuracy using low-order data is as high as 0.993377483443709, while the prediction accuracy using **high-order data is lower**, at 0.768211920529801; we continue to try different machine learning prediction methods.

4.2 Gradient Boosting Decision Trees

```
playoffStats_tree <- playoffStats_neural

# 设置随机种子以确保结果可重现
set.seed(123)

# 划分训练集和测试集
index <- createDataPartition(playoffStats_tree$is_great, p = 0.8, list = FALSE)
train_data_tree <- playoffStats_tree[index, ]
test_data_tree <- playoffStats_tree[-index, ]

# 分离出目标变量和自变量
train_target_tree <- train_data_tree$is_great
train_input_tree <- train_data_tree[, -ncol(train_data_tree)]

test_target_tree <- test_data_tree$is_great
test_input_tree <- test_data_tree[, -ncol(test_data_tree)]
# 构建梯度提升决策树模型
gbm_model <- gbm(train_target_tree ~.,
                 data = train_data_tree,
                 distribution = "bernoulli",  # 因为是二分类问题，选择伯努利分布
                 n.trees = 50,
                 interaction.depth = 3,
                 shrinkage = 0.01,
                 verbose = FALSE)
# 对测试集进行预测，得到概率值
test_pred_proba <- predict(gbm_model, newdata = test_data_tree, type = "response")
## Using 50 trees...
# 将概率预测转换为类别预测
test_pred_class <- ifelse(test_pred_proba > 0.8, 1, 0)

# 计算准确率
accuracy <- sum(test_pred_class == test_target_tree) / length(test_target_tree)
print(paste("Accuracy:", accuracy))
## [1] "Accuracy: 0.757877280265 34"
```

```
#针对低阶数据

playoffStats_tree_low <- playoffStats_neural_low

# 划分训练集和测试集
index <- createDataPartition(playoffStats_tree_low$is_great, p = 0.8, list = FALSE)
```

```
train_data_tree_low <- playoffStats_tree_low[index, ]
test_data_tree_low <- playoffStats_tree_low[-index, ]

# 分离出目标变量和自变量
train_target_tree_low <- train_data_tree_low$is_great
train_input_tree_low<- train_data_tree_low[, -ncol(train_data_tree_low)]

test_target_tree_low <- test_data_tree_low$is_great
test_input_tree_low <- test_data_tree_low[, -ncol(test_data_tree_low)]

# 构建梯度提升决策树模型
gbm_model_low <- gbm(train_target_tree_low ~.,
                     data = train_data_tree_low,
                     distribution = "bernoulli",  # 因为是二分类问题，选择伯努利分布
                     n.trees = 100,
                     interaction.depth = 3,
                     shrinkage = 0.01,
                     verbose = FALSE)

# 对测试集进行预测，得到概率值
test_pred_proba_low <- predict(gbm_model_low, newdata = test_data_tree_low, type = "res
ponse")
## Using 100 trees...
# 将概率预测转换为类别预测
test_pred_class_low <- ifelse(test_pred_proba_low > 0.8, 1, 0)

# 计算准确率
accuracy_low <- sum(test_pred_class_low == test_target_tree_low) / length(test_target_tree_l
ow)
print(paste("Low Order Data Accuracy:", accuracy_low))
## [1] "Low Order Data Accuracy: 0.764610281923715"
```

```
#针对高阶数据

playoffStats_tree_high <- playoffStats_neural_high

# 划分训练集和测试集
index <- createDataPartition(playoffStats_tree_high$is_great, p = 0.8, list = FALSE)
train_data_tree_high <- playoffStats_tree_high[index, ]
test_data_tree_high <- playoffStats_tree_high[-index, ]

# 分离出目标变量和自变量
train_target_tree_high <- train_data_tree_high$is_great
train_input_tree_high <- train_data_tree_high[, -ncol(train_data_tree_high)]
```

```
test_target_tree_high <- test_data_tree_high$is_great
test_input_tree_high <- test_data_tree_high[, -ncol(test_data_tree_high)]

# 构建梯度提升决策树模型
gbm_model_high <- gbm(train_target_tree_high ~.,
                      data = train_data_tree_high,
                      distribution = "bernoulli",  # 因为是二分类问题，选择伯努利分布
                      n.trees = 100,
                      interaction.depth = 3,
                      shrinkage = 0.01,
                      verbose = FALSE)

# 对测试集进行预测，得到概率值
test_pred_proba_high <- predict(gbm_model_high, newdata = test_data_tree_high, type = "r
esponse")
## Using 100 trees...
# 将概率预测转换为类别预测
test_pred_class_high <- ifelse(test_pred_proba_high > 0.8, 1, 0)

# 计算准确率
accuracy_high <- sum(test_pred_class_high == test_target_tree_high) / length(test_target_tre
e_high)
print(paste("High Order Data Accuracy:", accuracy_high))
## [1] "High Order Data Accuracy: 0.744510779436153"
```

The accuracy of high-order data prediction is lower than that of low-order data prediction

4.3 Support Vector Machines

```
playoffStats_SVM <- playoffStats_neural

set.seed(123)

index <- createDataPartition(playoffStats_SVM$is_great, p = 0.8, list = FALSE)
train_data_SVM <- playoffStats_SVM[index, ]
test_data_SVM <- playoffStats_SVM[-index, ]

# 分离出目标变量和自变量
train_target_SVM <- train_data_SVM$is_great
train_input_SVM <- train_data_SVM[, -ncol(train_data_SVM)]

test_target_SVM <- test_data_SVM$is_great
test_input_SVM <- test_data_SVM[, -ncol(test_data_SVM)]
# 构建 SVM 模型
svm_model <- svm(train_target_SVM ~.,
                 data = train_data_SVM,
                 kernel = "radial",  # 这里选择径向基核函数，你也可以尝试其他核函数
                 C = 1,
                 gamma = 1 / ncol(train_data_SVM)
                 )
# 对测试集进行预测
test_pred_SVM <- predict(svm_model, newdata = test_data_SVM)

test_pred_class_SVM <- ifelse(test_pred_SVM > 0.9, 1, 0)

# 计算准确率
accuracy = sum(test_pred_class_SVM == test_target_SVM) / length(test_target_SVM)
print(paste("Accuracy:", accuracy))
## [1] "Accuracy: 0.975124378109453"
```

```
#针对低阶数据
playoffStats_SVM_low <- playoffStats_neural_low

index <- createDataPartition(playoffStats_SVM_low$is_great, p = 0.8, list = FALSE)
train_data_SVM_low <- playoffStats_SVM_low[index, ]
test_data_SVM_low <- playoffStats_SVM_low[-index, ]

# 分离出目标变量和自变量
train_target_SVM_low <- train_data_SVM_low$is_great
train_input_SVM_low <- train_data_SVM_low[, -ncol(train_data_SVM_low)]
```

```
test_target_SVM_low <- test_data_SVM_low$is_great
test_input_SVM_low <- test_data_SVM_low[, -ncol(test_data_SVM_low)]

# 构建 SVM 模型
svm_model_low <- svm(train_target_SVM_low ~.,
                     data = train_data_SVM_low,
                     kernel = "radial",  # 这里选择径向基核函数，你也可以尝试其他核函
数
                     C = 1,
                     gamma = 1 / ncol(train_data_SVM_low)
                     )

# 对测试集进行预测
test_pred_SVM_low <- predict(svm_model_low, newdata = test_data_SVM_low)

test_pred_class_SVM_low <- ifelse(test_pred_SVM_low > 0.9, 1, 0)

# 计算准确率
accuracy_low = sum(test_pred_class_SVM_low == test_target_SVM_low) / length(test_targe
t_SVM_low)
print(paste("Low Order Data Accuracy:", accuracy_low))
## [1] "Low Order Data Accuracy: 0.995024875621891"
```

```
#针对高阶数据
playoffStats_SVM_high <- playoffStats_neural_high

index <- createDataPartition(playoffStats_SVM_high$is_great, p = 0.8, list = FALSE)
train_data_SVM_high <- playoffStats_SVM_high[index, ]
test_data_SVM_high <- playoffStats_SVM_high[-index, ]

# 分离出目标变量和自变量
train_target_SVM_high <- train_data_SVM_high$is_great
train_input_SVM_high <- train_data_SVM_high[, -ncol(train_data_SVM_high)]

test_target_SVM_high <- test_data_SVM_high$is_great
test_input_SVM_high <- test_data_SVM_high[, -ncol(test_data_SVM_high)]

# 构建 SVM 模型
svm_model_high <- svm(train_target_SVM_high ~.,
                     data = train_data_SVM_high,
                     kernel = "radial",  # 这里选择径向基核函数，你也可以尝试其他核函
数
                     C = 1,
                        gamma = 1 / ncol(train_data_SVM_high)
                        )

# 对测试集进行预测
test_pred_SVM_high <- predict(svm_model_high, newdata = test_data_SVM_high)

test_pred_class_SVM_high <- ifelse(test_pred_SVM_high > 0.9, 1, 0)

# 计算准确率
accuracy_high = sum(test_pred_class_SVM_high == test_target_SVM_high) / length(test_tar
get_SVM_high)
print(paste("High Order Data Accuracy:", accuracy_high))
## [1] "High Order Data Accuracy: 0.985074626865672"
```

The accuracy of high-order data is also lower than that of low-order data prediction, which further confirms our conclusion:

Low-level statistics are better predictors of player value than high-level statistics

5. Conclusion

From the perspective of predicting player value, low-level data often show advantages in reflecting the player's "hard power". For young players or role players, stable and excellent basic data can basically show that they have solid foundation and special expertise. For example, a blue-collar center who averages double-digit rebounds per game, even if his comprehensive score is not outstanding due to factors such as tactical position and playing time allocation under high-level data algorithms, low-level rebounding data is enough to show that he is indispensable in interior defense and rebounding. At the same time, low-level data is less affected by the team's tactical system and game style. Whether in a fast-paced running and gunning team or a traditional strong team that focuses on positional offense and rigorous defense, the collection standards of basic data such as scoring and rebounding are unified, the meaning is clear, and the comparison is strong, which is more conducive to measuring the basic ability and value stability of players across teams and seasons.

Although high-level data aims to dig deep into the comprehensive influence of players and fit the team's contribution, it is easily interfered by external factors such as team tactical inclination, teammate strength level, and game sample size. In a team full of superstars, the high-level data of a player with good strength may be "diluted" due to the distribution of ball rights and the tactics centered around the core, making its value mediocre; and in small sample games, occasional highlights or lows will excessively affect the overall evaluation under the complex calculation of high-level data, resulting in large fluctuations and difficulty in accurately anchoring the long-term stable value of players. Therefore, considering many aspects, low-level data does have unique advantages in predicting player value due to its intuitive presentation, less interference from the outside world, and accurate reflection of basic abilities. It can often give a concise and reliable preliminary judgment, laying a solid foundation for subsequent in-depth analysis combined with high-level data and providing key referen