PUBG Finish Placement Prediction

What’s the best strategy to win in PUBG ?

- 11712621 张子怡
- 11712706 刘明硕
- 11712804 郭明磊
- 11710810 刘晨希

SUSTech
Southern University of Science and Technology

Abstract

PlayerUnknown's BattleGrounds (PUBG) has enjoyed massive popularity. With over 50 million copies sold, it's the fifth best selling game of all time, and has millions of active monthly players. What's the best strategy to win in PUBG? Should you sit in one spot and hide your way into victory, or do you need to be the top shot? Let's let the data do the talking!

In this project, the relation between winPlacePerc and 13 features in solo mode is modeled through the linear model (LS, ridge, lasso, PCA) by data processing, parameter estimation, variable selection and quadratic terms addition. Because of the deficiency of the tests of residuals by linear regression, we propose other algorithms as well, such as KNN, SVM and neural network. Besides, we use the App — Regression Learner in MATLAB, which can assist us run the frequently-used models and choose the best one.

The results of the validation set indicates that the prediction accuracy according to the linear model with quadratic terms shows best R^2 both on training and test set with values 0.956 and 0.922, perhaps because we have dealt well with the multicollinearity problem during our predictors' analysis.

Key Words

R, MATLAB

LS, Ridge, Lasso, PCA, KNN, SVM, Neural Network, Regression Learner

Contents

1 Introduction

Data Source: *Kaggle_PUBG Finish Placement Prediction*

The *PUBG Finish Placement data set* used in this project comes from *Kaggle* published in 2018. The team at PUBG has made official game data available for the public to explore and scavenge outside of *"The Blue Circle."* And *Kaggle* collected data through the PUBG Developer API with the training set *train_V2.csv* and the test set *test_V2.csv*.

In a PUBG game, up to 100 players start in each match. Players can be on teams which get ranked at the end of the game (winPlacePerc) based on how many other teams are still alive when they are eliminated. In game, players can pick up different munitions, revive downed-but-not-out (knocked) teammates, drive vehicles, run, shoot, and experience all of the consequences — such as falling too far or running themselves over and eliminating themselves.

Because of too many observations in the training set (more than 4 million), hard to run a model for a laptop. For simplification, we focus on the solo mode. Approximately 4 thousand training observations are selected and 1 thousand test observations are chosen. Because some predictors are meaningful only we'll create models which predict players' finishing placement based on their final stats, on a scale from 1 (first place) to 0 (last place) and choose the best one.

2 Data Processinig and Analysis

2.1 Data Importing and Missing Data Dedection

```r
library(readxl)
library(gvlma)
library(reshape2)
library(ggplot2)
library(corrplot)
library(car)
library(MASS)
library(leaps)
library(gvlma)
library(car)
library(class)
library(psych)
library(glmnet)
library(e1071)
library(hydroGOF)
library(neuralnet)

setwd("D:/Collection_NUT/SUSTech_31/R/HW/Project/data")  # delete
data <- read_excel("train_data.xlsx")
sum(is.na(data))

## [1] 0
```

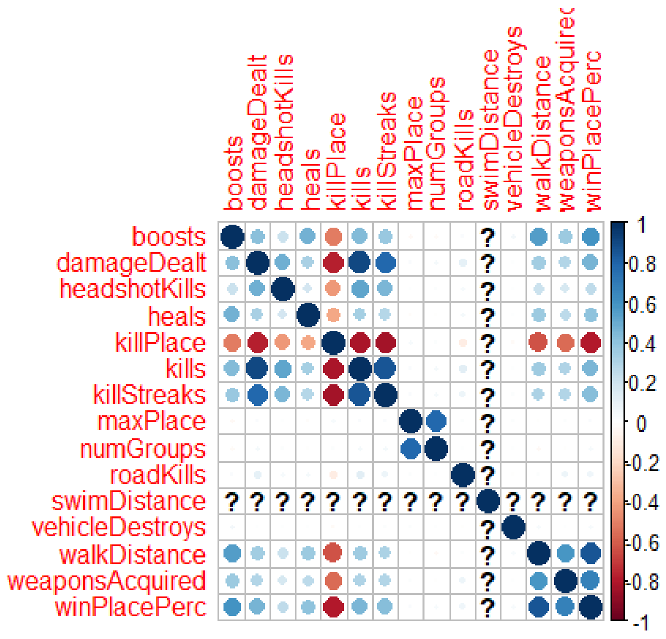
From the result of the detection, we can conclude that there is no NA value.

2.2 Data Description

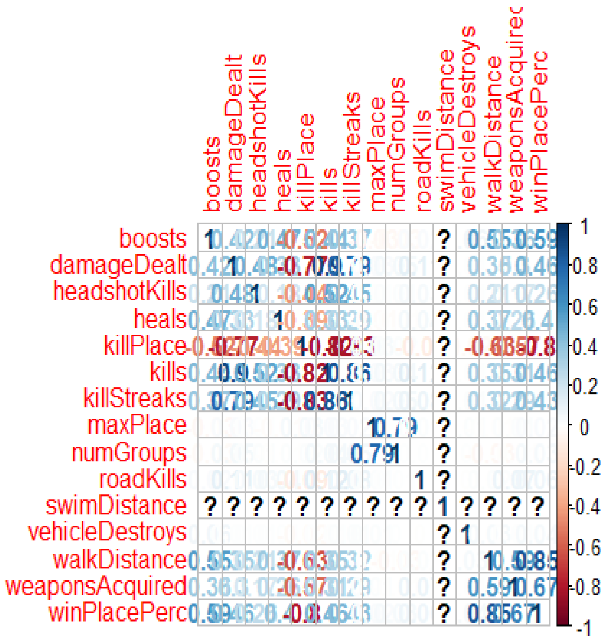
2.2.1 Tentative Exploration

```r
summary(data)

##      boosts          damageDealt     headshotKills        heals

##  Min.   : 0.0000   Min.   :  0.0    Min.   :0.0000   Min.   : 0.0000

##  1st Qu.: 0.0000   1st Qu.:  0.0    1st Qu.:0.0000   1st Qu.: 0.0000

##  Median : 0.0000   Median : 35.7    Median :0.0000   Median : 0.0000

##  Mean   : 0.6574   Mean   : 71.3    Mean   :0.1103   Mean   : 0.5584

##  3rd Qu.: 1.0000   3rd Qu.:100.0    3rd Qu.:0.0000   3rd Qu.: 0.0000

##  Max.   :11.0000   Max.   :415.8    Max.   :3.0000   Max.   :11.0000

##    killPlace          kills          killStreaks       maxPlace
##  Min.   :  2.00   Min.   :0.0000   Min.   :0.000   Min.   : 81.00
##  1st Qu.: 35.00   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.: 94.00
##  Median : 59.00   Median :0.0000   Median :0.000   Median : 96.00
##  Mean   : 56.05   Mean   :0.5089   Mean   :0.342   Mean   : 95.51
##  3rd Qu.: 78.00   3rd Qu.:1.0000   3rd Qu.:1.000   3rd Qu.: 97.00
##  Max.   :100.00   Max.   :5.0000   Max.   :3.000   Max.   :100.00
##    numGroups       roadKills          swimDistance vehicleDestroys
##  Min.   :80.00   Min.   :0.000000    Min.   :0     Min.   :0.000000
##  1st Qu.:90.00   1st Qu.:0.000000    1st Qu.:0     1st Qu.:0.000000
##  Median :93.00   Median :0.000000    Median :0     Median :0.000000
##  Mean   :92.31   Mean   :0.004742   Mean   :0      Mean   :0.001778
##  3rd Qu.:95.00   3rd Qu.:0.000000   3rd Qu.:0      3rd Qu.:0.000000
##  Max.   :99.00   Max.   :3.000000   Max.   :0      Max.   :1.000000
##   walkDistance     weaponsAcquired   winPlacePerc
##  Min.   :   0.00   Min.   : 0.000   Min.   :0.0000
##  1st Qu.:  94.34   1st Qu.: 2.000   1st Qu.:0.1771
##  Median : 392.55   Median : 3.000   Median :0.3804
##  Mean   : 730.13   Mean   : 3.342   Mean   :0.4088
##  3rd Qu.:1190.25   3rd Qu.: 5.000   3rd Qu.:0.6146
##  Max.   :3014.00   Max.   :14.000   Max.   :1.0000
```

```r
data.cor1 <- cor(data[c(1:15)])
corrplot(corr = data.cor1, method = "circle")
```



```r
corrplot(corr = data.cor1, method = "number")
```

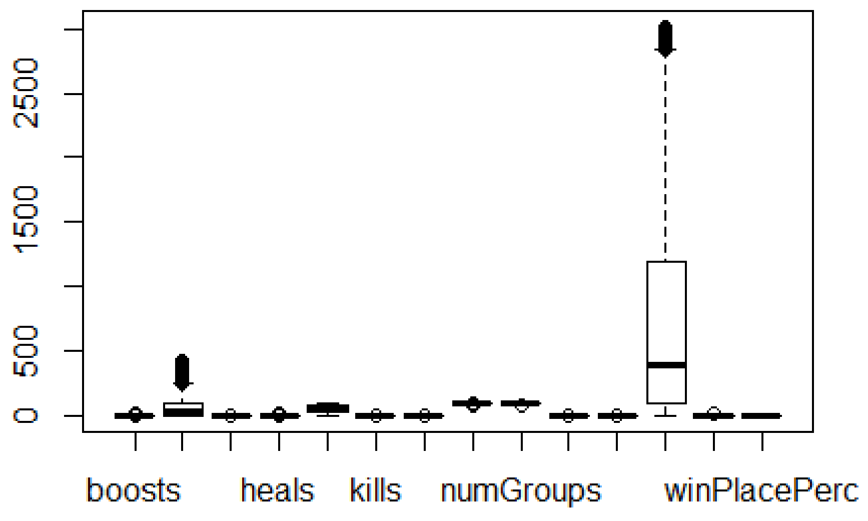


Unfortunately, there are question marks in the correlation plot. So there must be something wrong with our raw data.

After returing back to our data, we find that the column of **swimDistance** is composed of only 0 in our sample. So we decide to delete this variable, and then we check the boxplot and correlation plot again.

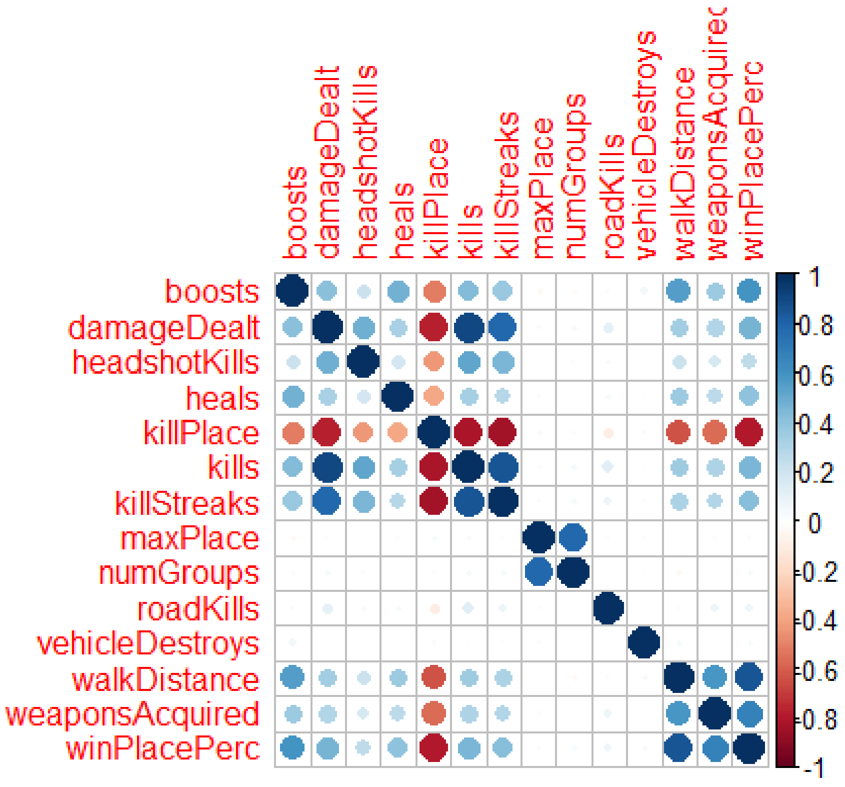
2.2.2 Boxplot Diagram

```
data=data[-c(11)]  # delete swimDistance column
boxplot(data)
```

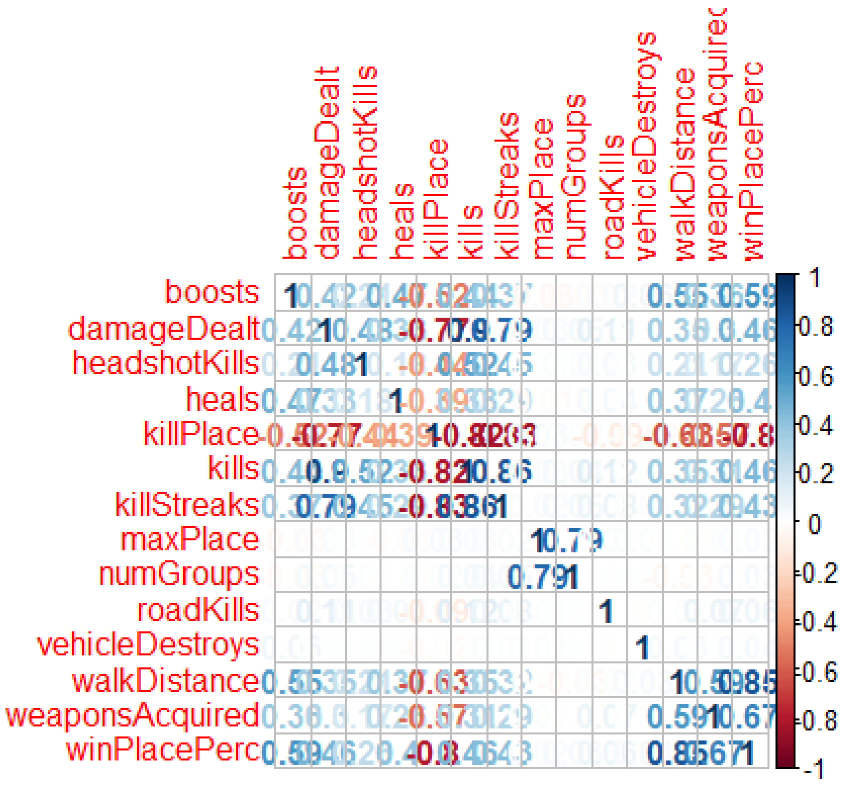


2.2.3 Correlation Diagram

```r
data.cor2 <- cor(data[c(1:14)])
corrplot(corr = data.cor2, method ="circle")
```



```r
corrplot(corr = data.cor2, method ="number")
```

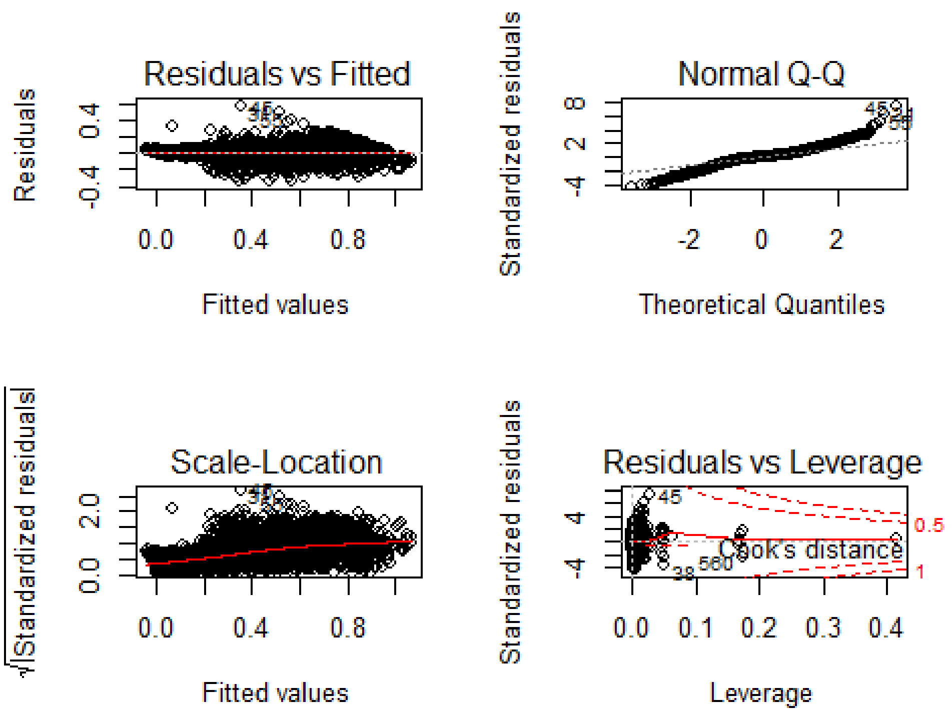


3 Model Building and Selection

3.1 LS

First, we do the linear regression using all the variables.

```
lm.fit1=lm(winPlacePerc~.,data = data)
summary(lm.fit1)

##
## Call:
## lm(formula = winPlacePerc ~ ., data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.32648 -0.03009  0.00360  0.03344  0.55674
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      3.131e-01  4.995e-02   6.268 4.13e-10 ***
## boosts           2.333e-02  1.397e-03  16.697  < 2e-16 ***
## damageDealt      6.331e-05  3.468e-05   1.826   0.0680 .
## headshotKills    1.293e-03  4.557e-03   0.284   0.7767
## heals            2.016e-03  1.114e-03   1.809   0.0705 .
## killPlace       -1.088e-02  1.598e-04 -68.050  < 2e-16 ***
## kills           -7.588e-02  4.614e-03 -16.445  < 2e-16 ***
## killStreaks     -2.196e-01  6.523e-03 -33.673  < 2e-16 ***
## maxPlace         5.111e-04  8.374e-04   0.610   0.5416
## numGroups        7.069e-03  6.272e-04  11.271  < 2e-16 ***
## roadKills        3.024e-02  1.684e-02   1.795   0.0727 .
## vehicleDestroys  2.857e-02  3.187e-02   0.896   0.3702
## walkDistance     1.085e-04  2.801e-06  38.731  < 2e-16 ***
## weaponsAcquired  5.124e-03  7.770e-04   6.595 4.93e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07783 on 3360 degrees of freedom
## Multiple R-squared:  0.9181, Adjusted R-squared:  0.9177
## F-statistic:  2895 on 13 and 3360 DF,  p-value: < 2.2e-16

par(mfrow=c(2,2))
plot(lm.fit1)
```



```r
par(mfrow=c(1,1))
```

From the summary of the lm.fit1 model, we can see that quite a few coefficients are not significant.

From the plot of the residuals, it is obvious that the normality of residuals is not good to assure our model.

Furthermore, combined with our correlation diagram, we can see that some predictors are highly correlated, so we suspect that there is serious multicollinearity problem. So we go on to verify the multicollinearity problem before conducting further linear regression analysis.

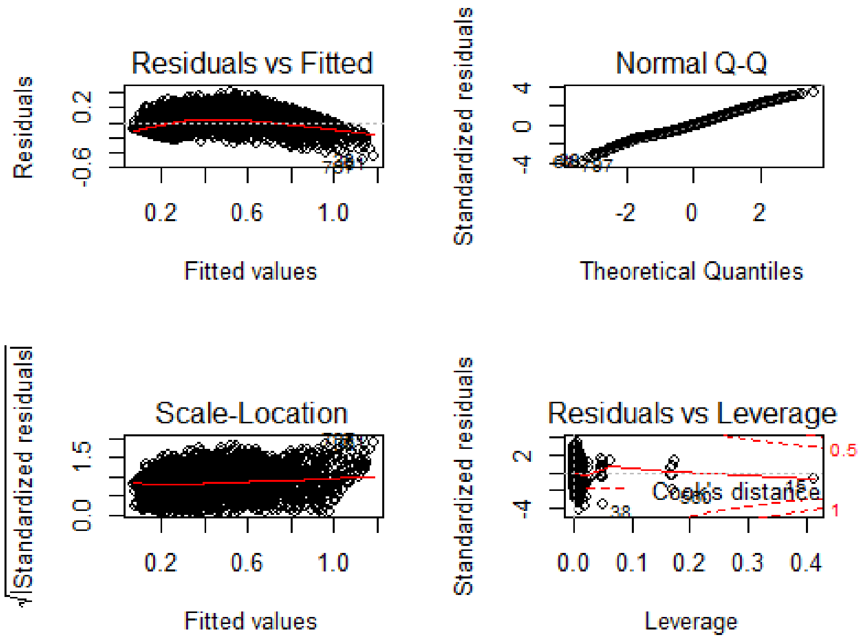
3.1.1 Multicollinearity Detection and Analysis

```r
vif(lm.fit1)

##          boosts     damageDealt   headshotKills           heals
##        1.759913        5.526565        1.381786        1.345420
##       killPlace           kills     killStreaks        maxPlace
##        9.303072        8.587941        5.769179        2.636388
##       numGroups       roadKills vehicleDestroys    walkDistance
##        2.633396        1.026993        1.004514        2.687854
## weaponsAcquired
##        1.881032
```

Based on the principle of value 4, we can see that there are some predictors which can be interpreted by other predictors to a large extent. So we decide to delete two predictors — **kill Place** and **kills**, because they have the highest vif value. What's more, intuitively, they are both correlated with killStreaks, which "Max number of enemy players killed in a short amount of time."

After deleting these two predictors, we do the linear regression again.

```
lm.fit2=lm(winPlacePerc~.-killPlace-kills,data = data)
summary(lm.fit2)

##
## Call:
## lm(formula = winPlacePerc ~ . - killPlace - kills, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48911 -0.08937 -0.00548  0.08056  0.39250
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -8.656e-02  7.648e-02  -1.132  0.25783
## boosts            2.314e-02  2.142e-03  10.803  < 2e-16 ***
## damageDealt       2.749e-04  3.937e-05   6.982 3.48e-12 ***
## headshotKills    -4.772e-03  6.905e-03  -0.691  0.48956
## heals             5.129e-03  1.717e-03   2.988  0.00283 **
## killStreaks       2.704e-02  6.933e-03   3.900 9.80e-05 ***
## maxPlace         -3.635e-03  1.288e-03  -2.822  0.00480 **
## numGroups         6.019e-03  9.667e-04   6.227 5.35e-10 ***
## roadKills         5.090e-02  2.587e-02   1.968  0.04919 *
## vehicleDestroys   5.856e-02  4.915e-02   1.191  0.23359
## walkDistance      2.074e-04  3.694e-06  56.147  < 2e-16 ***
## weaponsAcquired   2.653e-02  1.096e-03  24.213  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.12 on 3362 degrees of freedom
## Multiple R-squared:  0.8049, Adjusted R-squared:  0.8043
## F-statistic:  1261 on 11 and 3362 DF,  p-value: < 2.2e-16

par(mfrow=c(2,2))
plot(lm.fit2)
```



```r
par(mfrow=c(1,1))
```

Then we do the multicollinearity testing again.

```r
vif(lm.fit2)
```

```
##          boosts      damageDealt    headshotKills            heals
##        1.738499         2.993433         1.333095         1.342214
##     killStreaks         maxPlace        numGroups        roadKills
##        2.739349         2.620701         2.629676         1.017941
## vehicleDestroys     walkDistance weaponsAcquired
##        1.004137         1.964768         1.572167
```

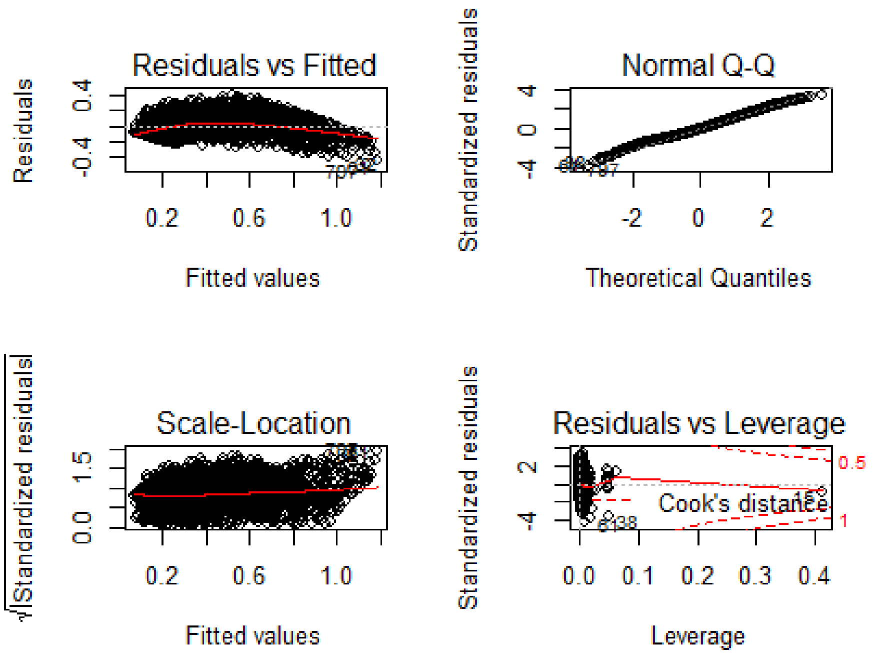
Until now, we can't see any multicollinearity anymore. However, some coefficients of predictors, such as **headshotKills** and **vehicleDestroys**, perhaps they need to be deleted, not taken into consideration.

To further simplify the model, we use the stepwise method with "both direction" to our lm.fit2 model, which may be helpful with our non-significant problem.

3.1.2 Stepwise Method

```r
# stepwise Method
stepAIC(lm.fit2, direction = "both")
```

```
## Start:  AIC=-14292.8
## winPlacePerc ~ (boosts + damageDealt + headshotKills + heals +
##     killPlace + kills + killStreaks + maxPlace + numGroups +
##     roadKills + vehicleDestroys + walkDistance + weaponsAcquired) -
##     killPlace - kills
##
##                   Df Sum of Sq    RSS    AIC
## - headshotKills    1     0.007 48.459 -14294
## - vehicleDestroys  1     0.020 48.473 -14293
## <none>                         48.452 -14293
## - roadKills        1     0.056 48.508 -14291
## - maxPlace         1     0.115 48.567 -14287
## - heals            1     0.129 48.581 -14286
## - killStreaks      1     0.219 48.672 -14280
## - numGroups        1     0.559 49.011 -14256
## - damageDealt      1     0.703 49.155 -14246
## - boosts           1     1.682 50.134 -14180
## - weaponsAcquired  1     8.449 56.902 -13752
## - walkDistance     1    45.434 93.886 -12063
##
## Step:  AIC=-14294.32
## winPlacePerc ~ boosts + damageDealt + heals + killStreaks + maxPlace
   + numGroups + roadKills + vehicleDestroys + walkDistance + weaponsAc
   quired
##
##                   Df Sum of Sq    RSS    AIC
## - vehicleDestroys  1     0.020 48.480 -14295
## <none>                         48.459 -14294
## + headshotKills    1     0.007 48.452 -14293
## - roadKills        1     0.057 48.516 -14292
## - maxPlace         1     0.115 48.575 -14288
## - heals            1     0.128 48.587 -14287
## - killStreaks      1     0.213 48.672 -14282
## - numGroups        1     0.560 49.020 -14258
## - damageDealt      1     0.707 49.167 -14247
## - boosts           1     1.687 50.146 -14181
## - weaponsAcquired  1     8.450 56.909 -13754
## - walkDistance     1    45.457 93.916 -12064
##
## Step:  AIC=-14294.9
## winPlacePerc ~ boosts + damageDealt + heals + killStreaks + maxPlace
   + numGroups + roadKills + walkDistance + weaponsAcquired
##
##                   Df Sum of Sq    RSS    AIC
## <none>                         48.480 -14295
## + vehicleDestroys  1     0.020 48.459 -14294
## + headshotKills    1     0.007 48.473 -14293
## - roadKills        1     0.057 48.536 -14293
## - maxPlace         1     0.114 48.594 -14289
## - heals            1     0.126 48.606 -14288
## - killStreaks      1     0.214 48.694 -14282
## - numGroups        1     0.559 49.039 -14258
## - damageDealt      1     0.704 49.184 -14248
## - boosts           1     1.711 50.191 -14180
## - weaponsAcquired  1     8.441 56.920 -13755
## - walkDistance     1    45.481 93.961 -12064

##
## Call:
## lm(formula = winPlacePerc ~ boosts + damageDealt + heals + killStrea
ks +
##     maxPlace + numGroups + roadKills + walkDistance + weaponsAcquire
d,
##     data = data)
##
## Coefficients:
##     (Intercept)           boosts      damageDealt            heals
##      -0.0876587        0.0233019        0.0002681        0.0050732
##     killStreaks         maxPlace        numGroups        roadKills
##       0.0265124       -0.0036231        0.0060220        0.0512205
##    walkDistance  weaponsAcquired
##       0.0002073        0.0265129

# with the following model selected by stepwise method.
## lm.fit3=lm(formula = winPlacePerc ~ boosts + damageDealt + heals +

   kill + Streaks + maxPlace + numGroups + roadKills + walkDistance +
   weaponsAcquired, data = data)
## summary(lm.fit3)

## Call:
## lm(formula = winPlacePerc ~ boosts + damageDealt + heals + killStrea
   ks +  maxPlace + numGroups + roadKills + walkDistance + weaponsAcqui
   red, data = data)

##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48832 -0.08960 -0.00569  0.08086  0.39184
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -8.766e-02  7.648e-02  -1.146 0.251779
## boosts           2.330e-02  2.138e-03  10.897  < 2e-16 ***
## damageDealt      2.681e-04  3.836e-05   6.989 3.33e-12 ***
## heals            5.073e-03  1.716e-03   2.956 0.003136 **
## killStreaks      2.651e-02  6.879e-03   3.854 0.000118 ***
## maxPlace        -3.623e-03  1.288e-03  -2.814 0.004926 **
## numGroups        6.022e-03  9.666e-04   6.230 5.23e-10 ***
## roadKills        5.122e-02  2.586e-02   1.980 0.047731 *
## walkDistance     2.073e-04  3.691e-06  56.177  < 2e-16 ***
## weaponsAcquired  2.651e-02  1.096e-03  24.201  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.12 on 3364 degrees of freedom
## Multiple R-squared:  0.8048, Adjusted R-squared:  0.8043
## F-statistic:  1541 on 9 and 3364 DF,  p-value: < 2.2e-16

par(mfrow=c(2,2))
plot(lm.fit3)
```



```
par(mfrow=c(1,1))
anova(lm.fit3,lm.fit2)

## Analysis of Variance Table
##
## Model 1: winPlacePerc ~ boosts + damageDealt + heals + killStreaks +
 maxPlace +
##     numGroups + roadKills + walkDistance + weaponsAcquired
## Model 2: winPlacePerc ~ (boosts + damageDealt + headshotKills + heal
s +
##     killPlace + kills + killStreaks + maxPlace + numGroups +
##     roadKills + vehicleDestroys + walkDistance + weaponsAcquired) -
##     killPlace - kills
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1   3364 48.480
## 2   3362 48.452  2    0.0273 0.9471  0.388

AIC(lm.fit3,lm.fit2)
```

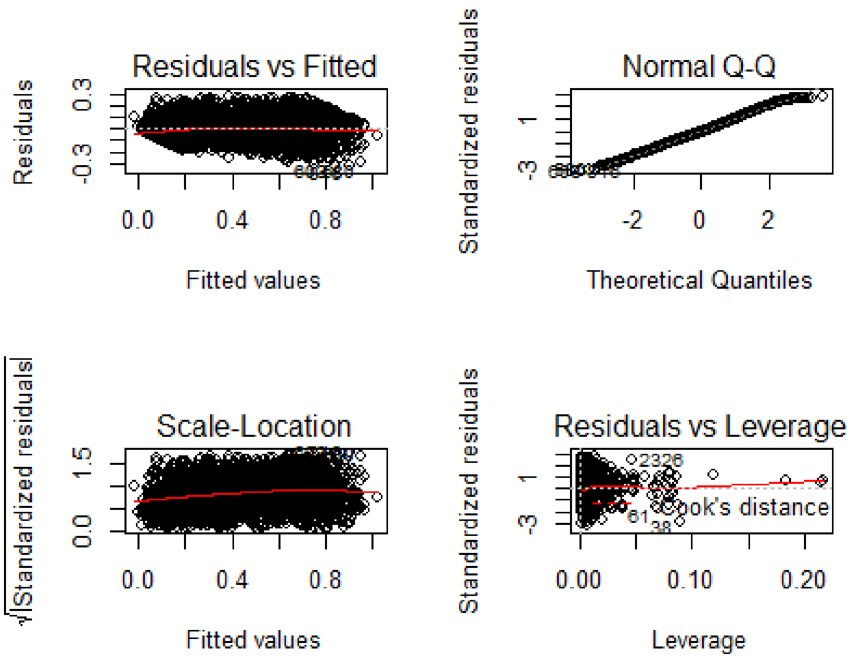
```
##          df     AIC
## lm.fit3 11 -4717.9
## lm.fit2 13 -4715.8
```

At the final step of anova and AIC analysis, we find that we indeed obtain a more concise model, with no decrease in adjusted R^2.

However, from the residuals plot, it seems that there should be a quadratic term in the model. Nevertheless, we don't know which predictor's quadratic term should be included. So we decide to add all the quadratic terms, then use stepwise and all-subset method to simplify our model.

3.1.3 Quadratic terms

```
# we add all the quadratic terms
lm.fit4=lm(formula = winPlacePerc ~ boosts + damageDealt + heals + kill
Streaks +
    maxPlace + numGroups + roadKills + walkDistance + weaponsAcquired +
 I(boosts^2) + I(damageDealt^2) + I(heals^2)+
    I(killStreaks^2) + I(maxPlace^2) + I(numGroups^2) + I(roadKills^2)
+ I(walkDistance^2) + I(weaponsAcquired^2),
    data = data)
summary(lm.fit4)

##
## Call:
## lm(formula = winPlacePerc ~ boosts + damageDealt + heals + killStrea
ks +
##     maxPlace + numGroups + roadKills + walkDistance + weaponsAcquire
d +
##     I(boosts^2) + I(damageDealt^2) + I(heals^2) + I(killStreaks^2) +

##     I(maxPlace^2) + I(numGroups^2) + I(roadKills^2) + I(walkDistance
^2) +
##     I(weaponsAcquired^2), data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.31703 -0.06627 -0.00474  0.06660  0.27875
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)          -5.973e+00  1.295e+00  -4.613 4.11e-06 ***
## boosts                3.598e-02  3.647e-03   9.865  < 2e-16 ***
## damageDealt           3.505e-04  7.308e-05   4.796 1.69e-06 ***
## heals                 2.338e-03  3.486e-03   0.671   0.5025
## killStreaks           8.611e-03  1.269e-02   0.679   0.4974
## maxPlace              2.451e-01  3.370e-02   7.275 4.29e-13 ***
## numGroups           -1.229e-01  2.222e-02  -5.528 3.48e-08 ***
## roadKills            4.260e-02  4.568e-02   0.933   0.3511    
## walkDistance         4.294e-04  9.235e-06  46.497  < 2e-16 ***
## weaponsAcquired      4.704e-02  2.558e-03  18.389  < 2e-16 ***
## I(boosts^2)         -1.540e-03  6.009e-04  -2.563   0.0104 *  
## I(damageDealt^2)    -4.326e-07  1.954e-07  -2.214   0.0269 *  
## I(heals^2)           1.853e-04  4.723e-04   0.392   0.6949    
## I(killStreaks^2)    -3.830e-04  8.561e-03  -0.045   0.9643    
## I(maxPlace^2)       -1.325e-03  1.793e-04  -7.391 1.83e-13 ***
## I(numGroups^2)       7.094e-04  1.223e-04   5.803 7.12e-09 ***
## I(roadKills^2)       6.769e-04  2.205e-02   0.031   0.9755    
## I(walkDistance^2)   -9.080e-08  3.347e-09 -27.131  < 2e-16 ***
## I(weaponsAcquired^2) -3.377e-03  2.447e-04 -13.801  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1012 on 3355 degrees of freedom
## Multiple R-squared:  0.8617, Adjusted R-squared:  0.8609
## F-statistic:  1161 on 18 and 3355 DF,  p-value: < 2.2e-16

par(mfrow=c(2,2))
plot(lm.fit4)
```



```
par(mfrow=c(1,1))
```

Obviously, taking the quadratic terms into account, residuals are quite nice, but some coefficients are not significant.

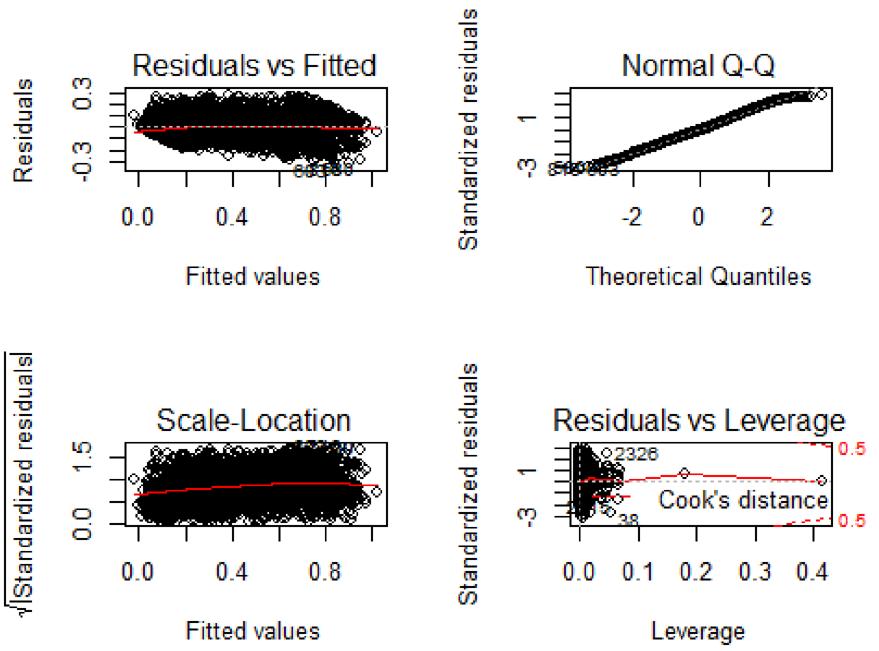
```
residplot <- function(fit, nbreaks=10) {
               z <- rstandard(fit)
               hist(z, breaks=nbreaks, freq=FALSE,
                    xlab="Standardized Residual",
                    main="Distribution of Errors")
               rug(jitter(z), col="brown")
               curve(dnorm(x, mean=mean(z), sd=sd(z)),
                     add=TRUE, col="blue", lwd=2)
               lines(density(z)$x, density(z)$y,
                     col="red", lwd=2, lty=2)
               legend("topright",
                      legend = c( "Normal Curve", "Kernel Density Curve
"),
                      lty=1:2, col=c("blue","red"), cex=.7)
        }
```

Stepwise Method

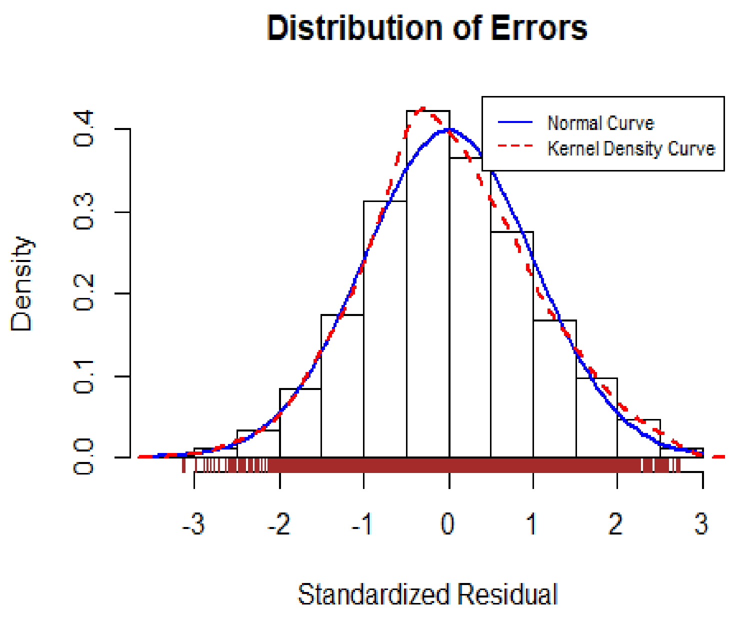
```
#stepwise method
stepAIC(lm.fit4)

## Start:  AIC=-15438.83
## winPlacePerc ~ boosts + damageDealt + heals + killStreaks + maxPlace
 +
##     numGroups + roadKills + walkDistance + weaponsAcquired +
##     I(boosts^2) + I(damageDealt^2) + I(heals^2) + I(killStreaks^2) +

##     I(maxPlace^2) + I(numGroups^2) + I(roadKills^2) + I(walkDistance
^2) +
##     I(weaponsAcquired^2)
##
##                          Df Sum of Sq    RSS    AIC
## - I(roadKills^2)          1    0.0000 34.356 -15441
## - I(killStreaks^2)        1    0.0000 34.356 -15441
## - I(heals^2)              1    0.0016 34.357 -15441
## - heals                   1    0.0046 34.360 -15440
## - killStreaks             1    0.0047 34.360 -15440
## - roadKills               1    0.0089 34.365 -15440
## <none>                              34.356 -15439
## - I(damageDealt^2)        1    0.0502 34.406 -15436
## - I(boosts^2)             1    0.0673 34.423 -15434
## - damageDealt             1    0.2356 34.591 -15418
## - numGroups               1    0.3130 34.669 -15410
## - I(numGroups^2)          1    0.3448 34.700 -15407
## - maxPlace                1    0.5420 34.898 -15388
## - I(maxPlace^2)           1    0.5594 34.915 -15386
## - boosts                  1    0.9965 35.352 -15344
## - I(weaponsAcquired^2)  1    1.9504 36.306 -15254
## - weaponsAcquired       1    3.4627 37.818 -15117
## - I(walkDistance^2)     1    7.5375 41.893 -14772
## - walkDistance          1   22.1387 56.494 -13763
##
## Step:  AIC=-15440.83
## winPlacePerc ~ boosts + damageDealt + heals + killStreaks + maxPlace
 +
##     numGroups + roadKills + walkDistance + weaponsAcquired +
##     I(boosts^2) + I(damageDealt^2) + I(heals^2) + I(killStreaks^2) +

##     I(maxPlace^2) + I(numGroups^2) + I(walkDistance^2) + I(weaponsAc
quired^2)
##
##                        Df Sum of Sq    RSS    AIC
## - I(killStreaks^2)      1    0.0000 34.356 -15443
## - I(heals^2)            1    0.0016 34.357 -15443
## - heals                 1    0.0046 34.360 -15442
## - killStreaks           1    0.0047 34.360 -15442
## <none>                              34.356 -15441
## - roadKills             1    0.0411 34.397 -15439
## - I(damageDealt^2)      1    0.0503 34.406 -15438
## - I(boosts^2)           1    0.0673 34.423 -15436
## - damageDealt           1    0.2359 34.592 -15420
## - numGroups             1    0.3130 34.669 -15412
## - I(numGroups^2)        1    0.3448 34.700 -15409
## - maxPlace              1    0.5420 34.898 -15390
## - I(maxPlace^2)         1    0.5594 34.915 -15388
## - boosts                1    0.9966 35.352 -15346
## - I(weaponsAcquired^2)  1    1.9506 36.306 -15256
## - weaponsAcquired       1    3.4632 37.819 -15119
## - I(walkDistance^2)     1    7.5419 41.898 -14773
## - walkDistance          1   22.1468 56.502 -13764
##
## Step:  AIC=-15442.83
## winPlacePerc ~ boosts + damageDealt + heals + killStreaks + maxPlace
 +
##     numGroups + roadKills + walkDistance + weaponsAcquired +
##     I(boosts^2) + I(damageDealt^2) + I(heals^2) + I(maxPlace^2) +
##     I(numGroups^2) + I(walkDistance^2) + I(weaponsAcquired^2)
##
##                        Df Sum of Sq    RSS    AIC
## - I(heals^2)            1    0.0016 34.357 -15445
## - heals                 1    0.0046 34.360 -15444
## - killStreaks           1    0.0165 34.372 -15443
## <none>                              34.356 -15443
## - roadKills             1    0.0412 34.397 -15441
## - I(damageDealt^2)      1    0.0552 34.411 -15439
## - I(boosts^2)           1    0.0673 34.423 -15438
## - damageDealt           1    0.2553 34.611 -15420
## - numGroups             1    0.3130 34.669 -15414
## - I(numGroups^2)        1    0.3448 34.701 -15411
## - maxPlace              1    0.5420 34.898 -15392
## - I(maxPlace^2)         1    0.5595 34.915 -15390
## - boosts                1    0.9975 35.353 -15348
## - I(weaponsAcquired^2)  1    1.9506 36.306 -15258
## - weaponsAcquired       1    3.4633 37.819 -15121
## - I(walkDistance^2)     1    7.5486 41.904 -14775
## - walkDistance          1   22.1701 56.526 -13765
##
## Step:  AIC=-15444.68
## winPlacePerc ~ boosts + damageDealt + heals + killStreaks + maxPlace
##     + numGroups + roadKills + walkDistance + weaponsAcquired +
##     I(boosts^2) + I(damageDealt^2) + I(maxPlace^2) + I(numGroups^2)
+
##     I(walkDistance^2) + I(weaponsAcquired^2)
##
##                        Df Sum of Sq    RSS    AIC
## - killStreaks           1    0.0163 34.374 -15445
## <none>                              34.357 -15445
## - roadKills             1    0.0416 34.399 -15443
## - I(damageDealt^2)      1    0.0551 34.412 -15441
## - heals                 1    0.0614 34.419 -15441
## - I(boosts^2)           1    0.0657 34.423 -15440
## - damageDealt           1    0.2543 34.612 -15422
## - numGroups             1    0.3135 34.671 -15416
## - I(numGroups^2)        1    0.3454 34.703 -15413
## - maxPlace              1    0.5435 34.901 -15394
## - I(maxPlace^2)         1    0.5610 34.918 -15392
## - boosts                1    1.0237 35.381 -15348
## - I(weaponsAcquired^2)  1    1.9494 36.307 -15260
## - weaponsAcquired       1    3.4630 37.820 -15123
## - I(walkDistance^2)     1    7.5507 41.908 -14776
## - walkDistance          1   22.2072 56.564 -13764
##
## Step:  AIC=-15445.08
## winPlacePerc ~ boosts + damageDealt + heals + maxPlace + numGroups +

##     roadKills + walkDistance + weaponsAcquired + I(boosts^2) +
##     I(damageDealt^2) + I(maxPlace^2) + I(numGroups^2) + I(walkDistan
##     ce^2) + I(weaponsAcquired^2)
##
##                        Df Sum of Sq    RSS    AIC
## <none>                              34.374 -15445
## - roadKills             1    0.0423 34.416 -15443
## - heals                 1    0.0623 34.436 -15441
## - I(boosts^2)           1    0.0682 34.442 -15440
## - I(damageDealt^2)      1    0.0985 34.472 -15437
## - numGroups              1    0.3148 34.688 -15416
## - I(numGroups^2)         1    0.3469 34.720 -15413
## - maxPlace               1    0.5439 34.917 -15394
## - I(maxPlace^2)          1    0.5614 34.935 -15392
## - damageDealt            1    0.6273 35.001 -15386
## - boosts                 1    1.0464 35.420 -15346
## - I(weaponsAcquired^2)   1    1.9658 36.339 -15259
## - weaponsAcquired        1    3.4967 37.870 -15120
## - I(walkDistance^2)      1    7.5430 41.917 -14778
## - walkDistance           1   22.1940 56.568 -13766

##
## Call:
## lm(formula = winPlacePerc ~ boosts + damageDealt + heals + maxPlace
+
##     numGroups + roadKills + walkDistance + weaponsAcquired +
##     I(boosts^2) + I(damageDealt^2) + I(maxPlace^2) + I(numGroups^2)
##     +I(walkDistance^2) + I(weaponsAcquired^2), data = data)
##
## Coefficients:
##          (Intercept)                boosts           damageDealt
##           -5.974e+00             3.600e-02             4.098e-04
##                heals              maxPlace             numGroups
##            3.608e-03             2.455e-01            -1.232e-01
##            roadKills          walkDistance       weaponsAcquired
##            4.443e-02             4.290e-04             4.716e-02
##          I(boosts^2)      I(damageDealt^2)         I(maxPlace^2)
##           -1.531e-03            -5.306e-07            -1.327e-03
##       I(numGroups^2)     I(walkDistance^2)  I(weaponsAcquired^2)
##            7.114e-04            -9.070e-08            -3.385e-03

# Stepwise with the following result
lm.fit5=lm(formula = winPlacePerc ~ boosts + damageDealt + heals + maxP
    lace +numGroups + roadKills + walkDistance + weaponsAcquired +
    I(boosts^2) + I(damageDealt^2) + I(maxPlace^2) + I(numGroups^2) +
    I(walkDistance^2) + I(weaponsAcquired^2), data = data)
summary(lm.fit5)

##
## Call:
## lm(formula = winPlacePerc ~ boosts + damageDealt + heals + maxPlace
+
##     numGroups + roadKills + walkDistance + weaponsAcquired +
##     I(boosts^2) + I(damageDealt^2) + I(maxPlace^2) + I(numGroups^2)
##     +I(walkDistance^2) + I(weaponsAcquired^2), data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.31659 -0.06590 -0.00521  0.06690  0.27698
##
```

```
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)    
## (Intercept)         -5.974e+00  1.294e+00  -4.617 4.04e-06 ***
## boosts               3.600e-02  3.560e-03  10.112  < 2e-16 ***
## damageDealt          4.098e-04  5.233e-05   7.830 6.51e-15 ***
## heals                3.608e-03  1.462e-03   2.468  0.01364 *  
## maxPlace             2.455e-01  3.367e-02   7.290 3.84e-13 ***
## numGroups           -1.232e-01  2.221e-02  -5.547 3.14e-08 ***
## roadKills            4.443e-02  2.185e-02   2.033  0.04212 *  
## walkDistance         4.290e-04  9.213e-06  46.570  < 2e-16 ***
## weaponsAcquired      4.716e-02  2.551e-03  18.485  < 2e-16 ***
## I(boosts^2)         -1.531e-03  5.933e-04  -2.581  0.00988 ** 
## I(damageDealt^2)    -5.306e-07  1.710e-07  -3.103  0.00193 ** 
## I(maxPlace^2)       -1.327e-03  1.792e-04  -7.407 1.63e-13 ***
## I(numGroups^2)       7.114e-04  1.222e-04   5.822 6.35e-09 ***
## I(walkDistance^2)   -9.070e-08  3.341e-09 -27.150  < 2e-16 ***
## I(weaponsAcquired^2) -3.385e-03  2.442e-04 -13.860  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.1012 on 3359 degrees of freedom
## Multiple R-squared:  0.8616, Adjusted R-squared:  0.861 
## F-statistic:  1494 on 14 and 3359 DF,  p-value: < 2.2e-16
```

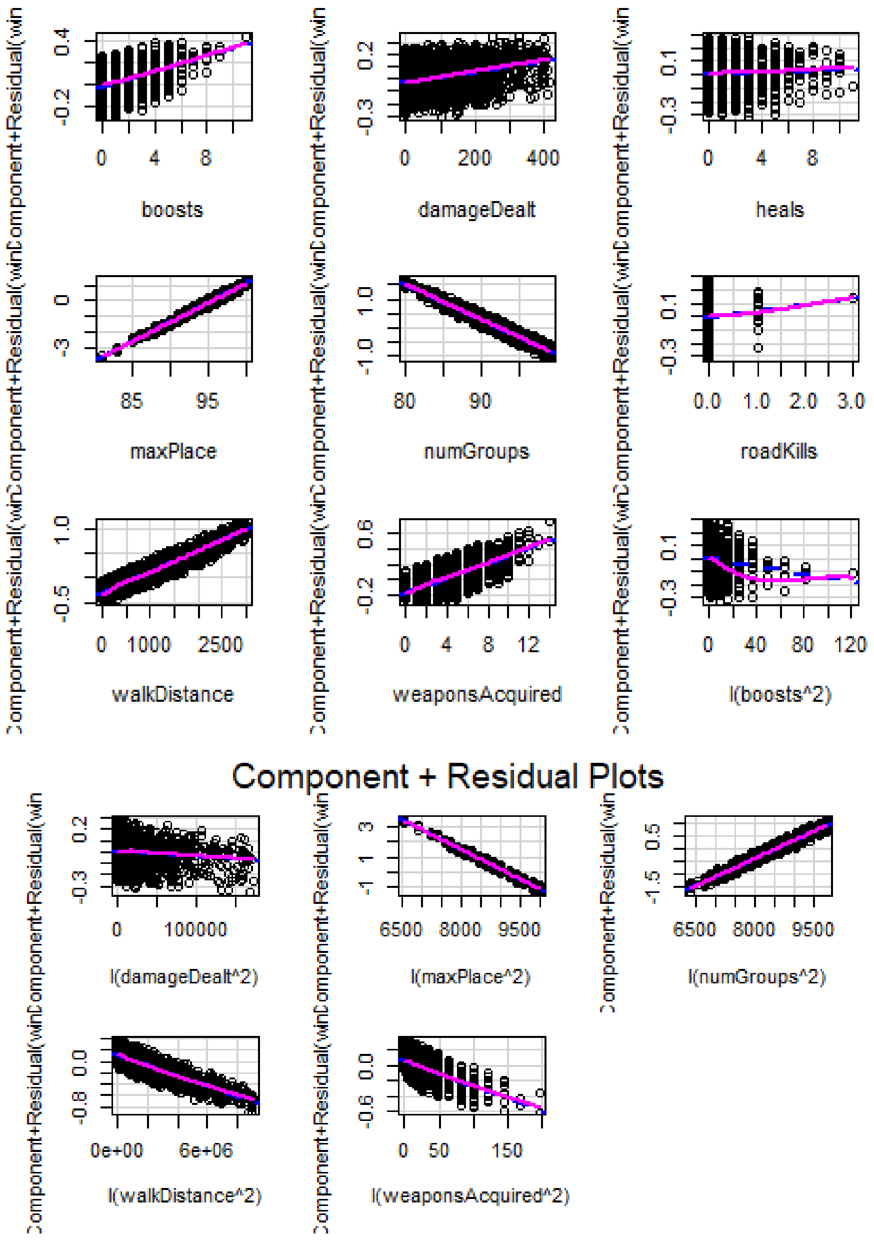
```r
par(mfrow=c(2,2))
plot(lm.fit5)
```



```r
par(mfrow=c(1,1))
# Model diagnostic
residplot(lm.fit5)
```



```r
ks.test(rstandard(lm.fit5),"pnorm",mean =0 ,sd =1)

##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rstandard(lm.fit5)
## D = 0.02349, p-value = 0.04831
## alternative hypothesis: two-sided

durbinWatsonTest(lm.fit5)

##  lag Autocorrelation D-W Statistic p-value
##    1      0.04134265      1.916476   0.012
##  Alternative hypothesis: rho != 0

ncvTest(lm.fit5)

## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 160.1532, Df = 1, p = < 2.22e-16

crPlots(lm.fit5)
```

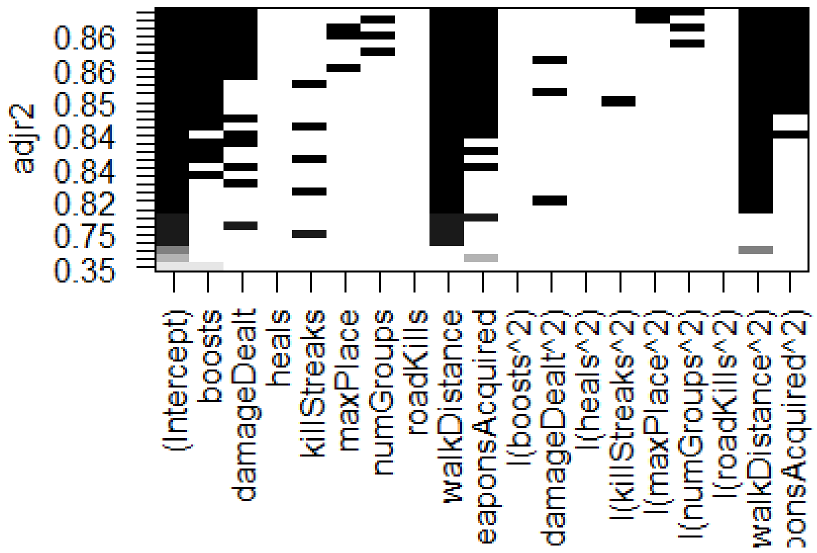


After stepwise process, we can see that only a few coefficients are not significant. When we do the model diagnose, the results are not so good but not so bad as well.

For the test of **normality**, the p-value of ks.test is 0.04831, very close to 0.05. However, from the histograms of residuals, it shows that the difference is little. For the test of **independence**, the p-value is 0.01. For the performance of **linearity**, the model fits very well. For the test of **homoscedasticity**, the result shows bad condition.

All-Subset Method

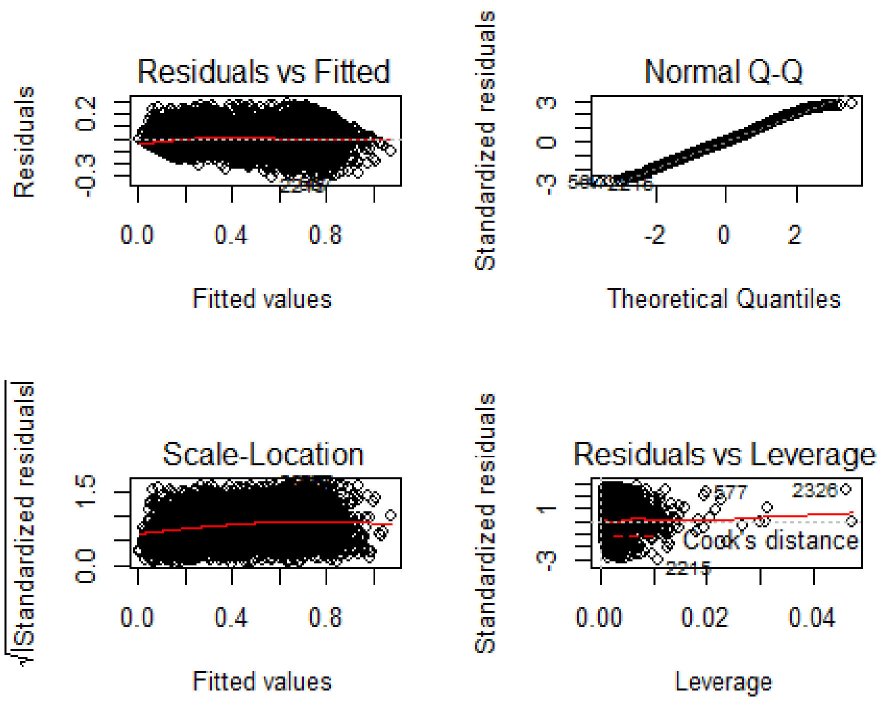
```
# all-subset method
leaps <- regsubsets(winPlacePerc ~ boosts + damageDealt + heals + killS
treaks +
    maxPlace + numGroups + roadKills + walkDistance + weaponsAcquired +
 I(boosts^2) + I(damageDealt^2) +
    I(heals^2) + I(killStreaks^2) +
    I(maxPlace^2) + I(numGroups^2) + I(roadKills^2) + I(walkDistance^2)
 + I(weaponsAcquired^2),
    data = data, nbest = 4)
plot(leaps, scale="adjr2")
```



```
# With the following result
lm.fit6=lm(formula = winPlacePerc ~ boosts + damageDealt + walkDistance
 + weaponsAcquired +
    I(maxPlace^2) + I(numGroups^2) + I(walkDistance^2) + I(weaponsAcqui
red^2),
    data = data)
summary(lm.fit6)

##
## Call:
## lm(formula = winPlacePerc ~ boosts + damageDealt + walkDistance +
##     weaponsAcquired + I(maxPlace^2) + I(numGroups^2) + I(walkDistanc
e^2) +
##     I(weaponsAcquired^2), data = data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.305815 -0.065869 -0.006049  0.066034  0.277371
##
```

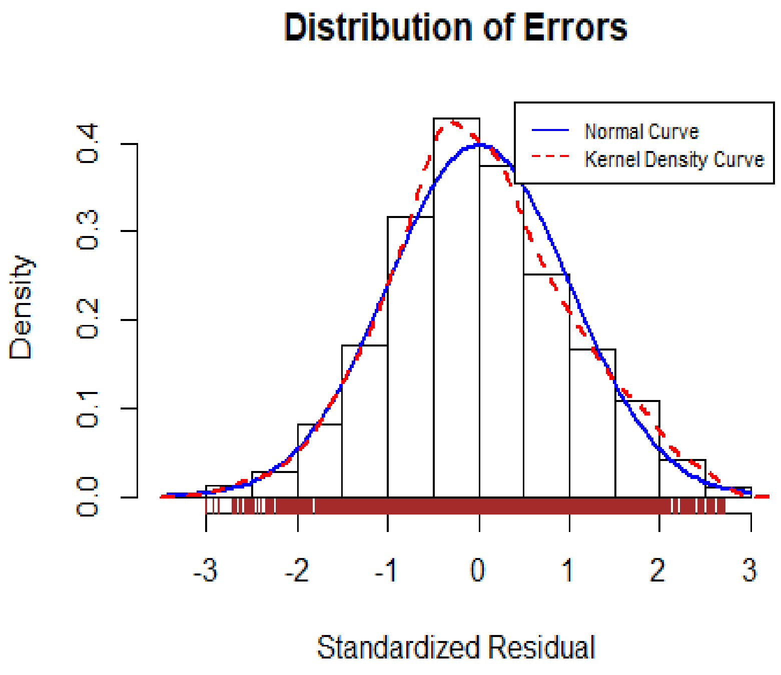
```
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -2.472e-02  3.331e-02  -0.742 0.458171
## boosts                 2.922e-02  1.741e-03  16.782  < 2e-16 ***
## damageDealt            2.828e-04  2.205e-05  12.828  < 2e-16 ***
## walkDistance           4.343e-04  9.174e-06  47.346  < 2e-16 ***
## weaponsAcquired        4.821e-02  2.561e-03  18.829  < 2e-16 ***
## I(maxPlace^2)         -2.250e-05  5.844e-06  -3.849 0.000121 ***
## I(numGroups^2)         3.344e-05  4.537e-06   7.371 2.11e-13 ***
## I(walkDistance^2)     -9.245e-08  3.341e-09 -27.673  < 2e-16 ***
## I(weaponsAcquired^2) -3.460e-03  2.452e-04 -14.114  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1023 on 3365 degrees of freedom
## Multiple R-squared:  0.8581, Adjusted R-squared:  0.8578
## F-statistic:  2544 on 8 and 3365 DF,  p-value: < 2.2e-16
```

```r
par(mfrow=c(2,2))
plot(lm.fit6)
```

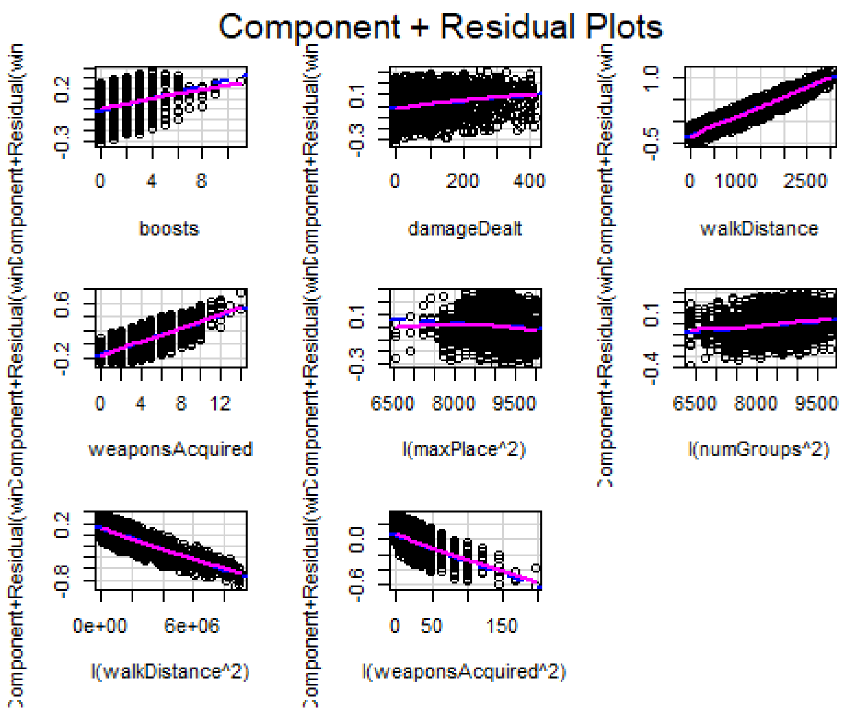


```r
par(mfrow=c(1,1))
```

```r
# Model diagnostic
residplot(lm.fit6)
```



```r
ks.test(rstandard(lm.fit6), "pnorm", mean = 0, sd = 1)

##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rstandard(lm.fit6)
## D = 0.02556, p-value = 0.02435
## alternative hypothesis: two-sided

durbinWatsonTest(lm.fit6)

##  lag Autocorrelation D-W Statistic p-value
##    1      0.03649446      1.925838   0.038
##  Alternative hypothesis: rho != 0

ncvTest(lm.fit6)

## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 173.9242, Df = 1, p = < 2.22e-16

crPlots(lm.fit6)
```



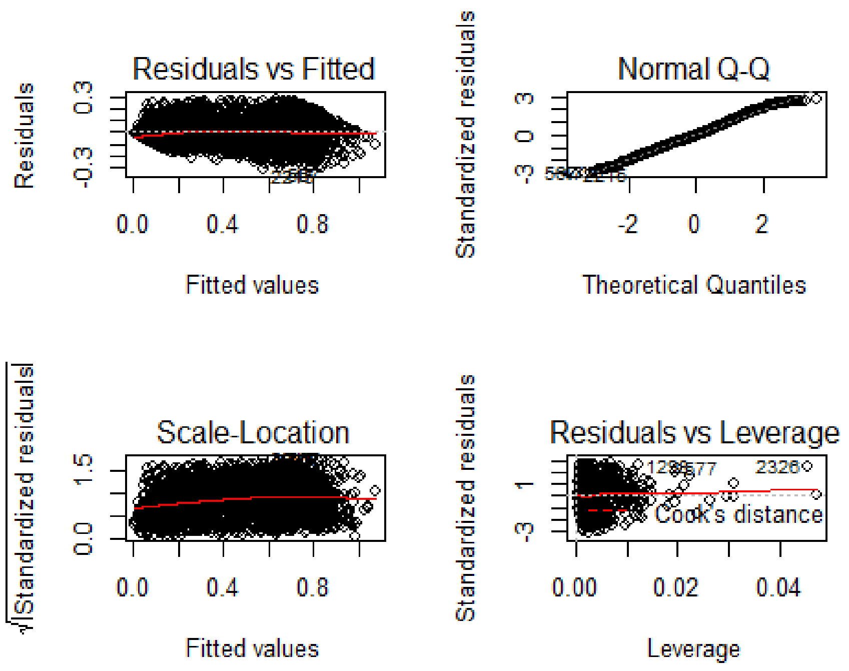
Compared with the result of stepwise regression, there is a noticeable advantage — the model is much more concise with similar R^2. And the result of model diagnostic is similar to that of stepwise regression.

As a consequence of the non-significant of the intercept term, we try to delete it and see the result.

Intercept Deleting

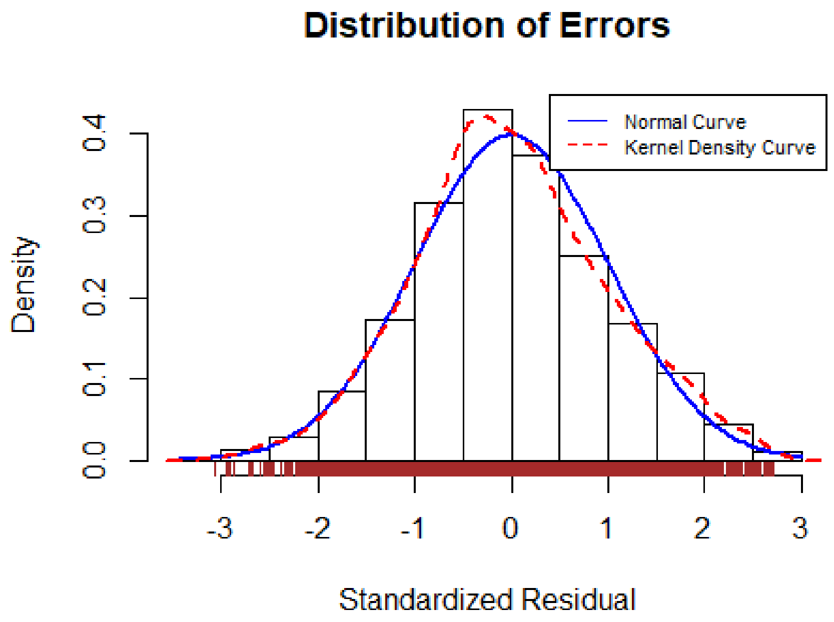
```
# delete the intercept term
lm.fit7=lm(formula = winPlacePerc ~ boosts + damageDealt + walkDista
        nce + weaponsAcquired + I(maxPlace^2) + I(numGroups^2) +
           I(walkDistance^2) + I(weaponsAcquired^2) - 1, data = data)
summary(lm.fit7)

##
## Call:
## lm(formula = winPlacePerc ~ boosts + damageDealt + walkDistance +
##     weaponsAcquired + I(maxPlace^2) + I(numGroups^2) + I(walkDistanc
e^2) +
##     I(weaponsAcquired^2) - 1, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.31275 -0.06591 -0.00600  0.06655  0.27689
##
```

```
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)    
## boosts                  2.918e-02  1.740e-03  16.767  < 2e-16 ***
## damageDealt             2.831e-04  2.204e-05  12.848  < 2e-16 ***
## walkDistance            4.345e-04  9.171e-06  47.380  < 2e-16 ***
## weaponsAcquired         4.804e-02  2.550e-03  18.842  < 2e-16 ***
## I(maxPlace^2)          -2.548e-05  4.240e-06  -6.010 2.06e-09 ***
## I(numGroups^2)          3.378e-05  4.514e-06   7.483 9.19e-14 ***
## I(walkDistance^2)      -9.250e-08  3.340e-09 -27.697  < 2e-16 ***
## I(weaponsAcquired^2) -3.445e-03  2.443e-04 -14.103  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1023 on 3366 degrees of freedom
## Multiple R-squared:  0.9566, Adjusted R-squared:  0.9565
## F-statistic:  9274 on 8 and 3366 DF,  p-value: < 2.2e-16

par(mfrow=c(2,2))
plot(lm.fit7)
```

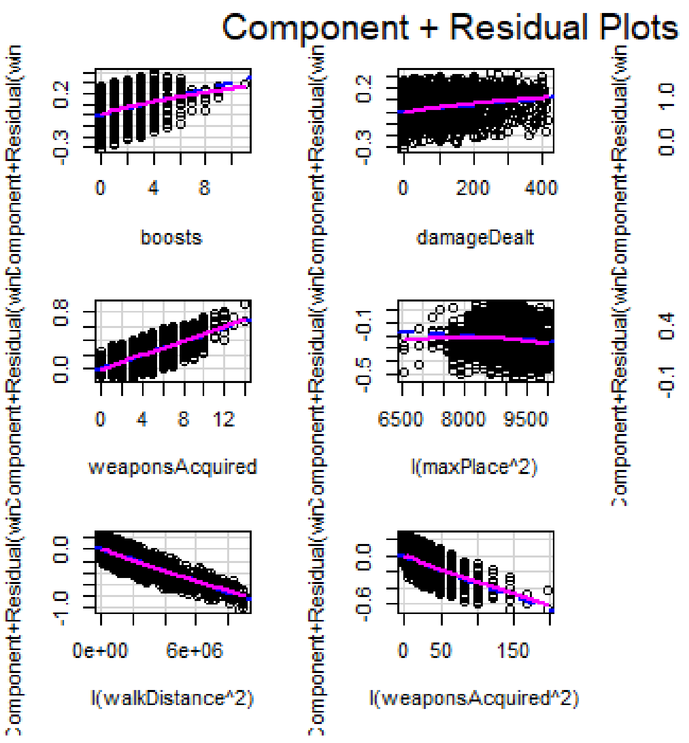


```
par(mfrow=c(1,1))

# Model diagnostic
ks.test(rstandard(lm.fit7),"pnorm",mean =0 ,sd =1)

##
##  One-sample Kolmogorov-Smirnov test
##
```

```
## data:  rstandard(lm.fit7)
## D = 0.026823, p-value = 0.01558
## alternative hypothesis: two-sided

residplot(lm.fit7)
```



```
durbinWatsonTest(lm.fit7)

##  lag Autocorrelation D-W Statistic p-value
##    1      0.03679238      1.925234   0.038
##  Alternative hypothesis: rho != 0

ncvTest(lm.fit7)

## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 174.4194, Df = 1, p = < 2.22e-16

crPlots(lm.fit7)
```



```
# test of the necessary to delete the intercept term
anova(lm.fit7,lm.fit6)

## Analysis of Variance Table
##
## Model 1: winPlacePerc ~ boosts + damageDealt + walkDistance + weapon
sAcquired +
##     I(maxPlace^2) + I(numGroups^2) + I(walkDistance^2) + I(weaponsAc
quired^2) -
##     1
## Model 2: winPlacePerc ~ boosts + damageDealt + walkDistance + weapon
sAcquired +
##     I(maxPlace^2) + I(numGroups^2) + I(walkDistance^2) + I(weaponsAc
quired^2)
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1   3366 35.243
## 2   3365 35.238  1 0.0057646 0.5505 0.4582

AIC(lm.fit7,lm.fit6)

##         df       AIC
## lm.fit7  9 -5797.754
## lm.fit6 10 -5796.306
```

After deleting the intercept term, the resulting R^2 shows evident improvement—from 0.86 to 0.95, with little change in model diagnostic part. Considering the anova and AIC analysiis between lm.fit7 and lm.fit6, the result suggests us ignoring the intercept term. According to Occam's Razor, "Do not multiply entities beyond necessity," so lm.fit7 is chosen as our final model in the linear regression part.

So why can we choose to delete the intercept term?

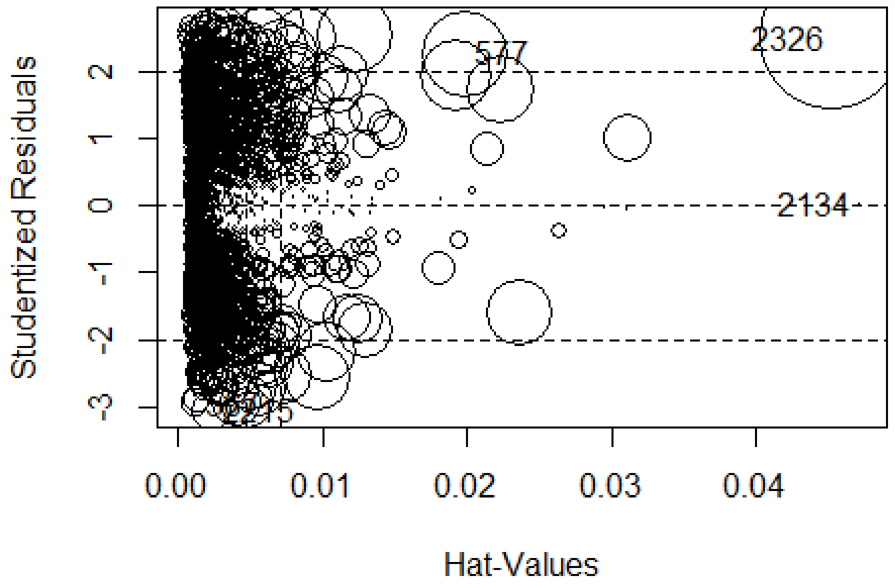
I think it can be interpreted intuitively. If all predictors are 0, meaning no kills, no damage, no weapons, no walkDistance — he was killed immediately when he entered the game. As a consequence, he was the first to be weeded, the winPlacePerc is 0.

For the result of model diagnostic, some tests are not so good. From my perspective, too many observations to fit, and it is ordinary to have some disadvantages of the residuals, because of only a few predictors.

Next we will do the outlier and influence point detection.

3.1.4 Outlier and Influence Points Analysis

```
outlierTest(lm.fit7)

## No Studentized residuals with Bonferroni p < 0.05
## Largest |rstudent|:
##       rstudent unadjusted p-value Bonferroni p
## 2215 -3.063882          0.0022022           NA

influencePlot(lm.fit7)
```



```
##           StudRes         Hat        CookD
## 567  -2.960135307 0.001254630 1.372757e-03
## 577   2.266684713 0.019916655 1.303503e-02
## 2134  0.009515927 0.047256121 5.615949e-07
## 2215 -3.063881943 0.002391609 2.806101e-03
## 2326  2.492985980 0.045390702 3.688235e-02
```

The test of unusual points shows that there is no large outlier, but with influence points. Here we take NO.2326 and NO.2215 as example.

After checking the data, we find that, NO.2326, his walkDistance doesn't coincidence with its weapons. Intuitively, the more distance you walk, the more weapons you obtain, which means positive correlation. But compard with the competitors of the same level, he attained many weapons with only a few walkDistance, and that's why the observation was screened out.

As for NO.2215, all of his indicators are good, but his score-place is very bad. Perhaps many of his competitors are excellent, and that's why the observation was screened out. We did't delete any observation and then fit the model again, because the action of deleting observations should be careful. Our model is pretty good enough with large R^2 and few influence points. We should use the model to match the data, instead of doing the opposite.

3.1.5 Validation Set Testing

We have another 997 observations to construct a test set, let's use lm.fit7 to predict in the test set and calculate the R^2.

```
setwd("D:/Collection_NUT/SUSTech_31/R/HW/Project/data")  # delete
test_data <- read_excel("test_data.xlsx")
test_data <- as.data.frame(test_data)
test_pre <- predict(lm.fit7, newdata=test_data[-14])  # predict percent
age
mean_test=mean(test_data$winPlacePerc)
SSR_test=sum((test_pre-mean_test)^2)
SST_test=sum((test_data$winPlacePerc-mean_test)^2)
paste("The R^2 of the linear model in the test set is", SSR_test/SST_te
st )

## [1] "The R^2 of the linear model in the test set is 0.92247875566839
3"
```

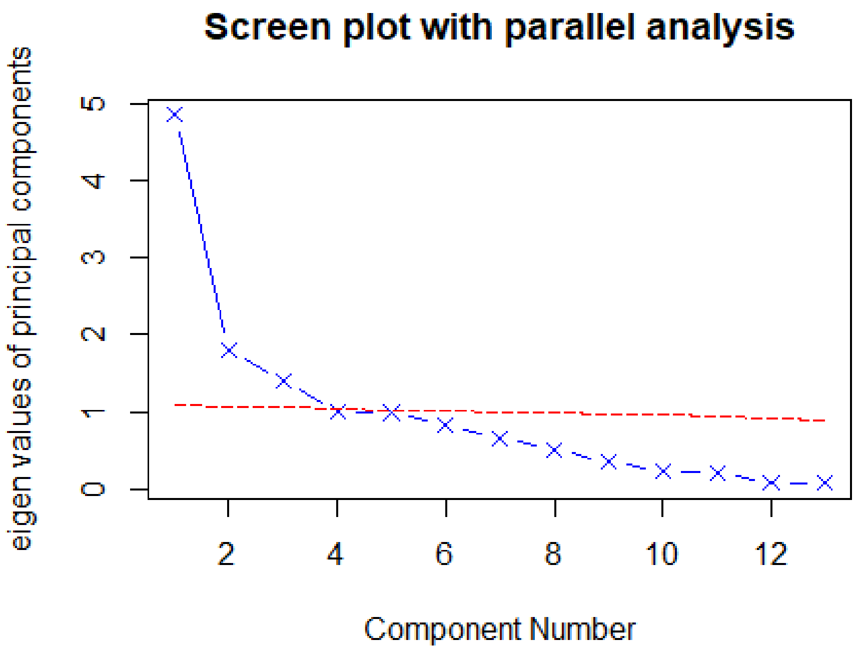
As we can see, in test data set, R^2 is above than 0.922, which shows the basic linear model prediction seems good enough, let's see what will happen in other models.

3.2 PCA

In the process of establishing multiple linear regression model, we often choose **least square method** (OLS) to solve the regression coefficient. However, when multicollinearity exists, OLS estimation is not ideal, and even symbols that are inconsistent with the actual situation appear, so that it is difficult to give a realistic explanation to the established regression equation, resulting in chaotic regression results. To this end, we use **Principal Component Regression** (PCR), a multivariate Regression analysis method, which aims to solve the problem of multicollinearity among independent variables.

First, let's figure out how many principal components we need. By **parallel analysis**, using the function **fa.parallel()**, we can see that choosing four principal components can hold most of the information.

```
#data import
setwd("D:/Collection_NUT/SUSTech_31/R/HW/Project/data")  # delete
data = read_excel("train_data.xlsx")
data = data[-11]
data_test = read_excel("test_data.xlsx")
#PCA regression
fa.parallel(scale(data[,1:13]), fa="pc", n.iter=100, show.legend=FALSE,
 main="Screen plot with parallel analysis")
```



```
## Parallel analysis suggests that the number of factors =  NA  and the
 number of components =  3

my_data <- princomp(scale(data[,1:13]), data=data, cor=T)
```

```
## Warning: In princomp.default(scale(data[, 1:13]), data = data, cor =
 T) :
##  extra argument 'data' will be disregarded

my_data_test <- princomp(scale(data_test[,1:13]), data=data_test, cor=
T)

## Warning: In princomp.default(scale(data_test[, 1:13]), data = data_t
est,
##     cor = T) :
##  extra argument 'data' will be disregarded

# my_data <- princomp(~boosts + damageDealt + headshotKills + heals + k
illPlace + kills + killStreaks + maxPlace + numGroups + roadKills + veh
icleDestroys + walkDistance + weaponsAcquired, data=data, cor=T)
#biplot(my_data,choices = 1:2)
summary(my_data,loadings = TRUE)

## Importance of components:
##                           Comp.1    Comp.2    Comp.3    Comp.4     C
omp.5
## Standard deviation     2.2012252 1.3436129 1.1813749 1.0021779 0.995
03244
## Proportion of Variance 0.3727225 0.1388689 0.1073574 0.0772585 0.076
16074
## Cumulative Proportion  0.3727225 0.5115914 0.6189488 0.6962073 0.772
36807
##                            Comp.6     Comp.7     Comp.8     Comp.9
## Standard deviation     0.91131610 0.81532889 0.71824182 0.59348270
## Proportion of Variance 0.06388439 0.05113548 0.03968241 0.02709398
## Cumulative Proportion  0.83625246 0.88738793 0.92707034 0.95416432
##                           Comp.10    Comp.11     Comp.12     Comp.13
## Standard deviation     0.47520903 0.46109111 0.285830933 0.275201544
## Proportion of Variance 0.01737105 0.01635423 0.006284563 0.005825838
## Cumulative Proportion  0.97153537 0.98788960 0.994174162 1.000000000
##
## Loadings:
##                 Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Com
p.8
## boosts           0.291         0.342                0.332         0.
686
## damageDealt      0.390        -0.276                       0.176

## headshotKills    0.253        -0.297                      -0.902

## heals            0.233         0.293                0.729 -0.103 -0.
554
## killPlace       -0.426                              0.177 -0.149

## kills            0.404        -0.290                       0.157
## killStreaks      0.383        -0.291                      0.265

## maxPlace               -0.690  0.157

## numGroups              -0.694  0.127

## roadKills                            -0.539 -0.819

## vehicleDestroys                       0.819 -0.560

## walkDistance     0.291         0.477               -0.221         0.
211
## weaponsAcquired  0.255         0.425 -0.145         -0.512        -0.
386
##                 Comp.9 Comp.10 Comp.11 Comp.12 Comp.13
## boosts           0.438
## damageDealt            -0.658  -0.168   0.516
## headshotKills
## heals
## killPlace        0.147 -0.258          -0.116  -0.810
## kills                  -0.222          -0.819
## killStreaks             0.604   0.161   0.204  -0.500
## maxPlace               -0.158   0.686
## numGroups               0.178  -0.684
## roadKills
## vehicleDestroys
## walkDistance    -0.702                         -0.254
## weaponsAcquired  0.531                         -0.143
```

Output from the above results, we can see that the model on the basis of the original data generated four principal components, the four principal components explained 70% of the original data variance. This also means that after dealing with the dimension reduction, the original data has little loss of data information.

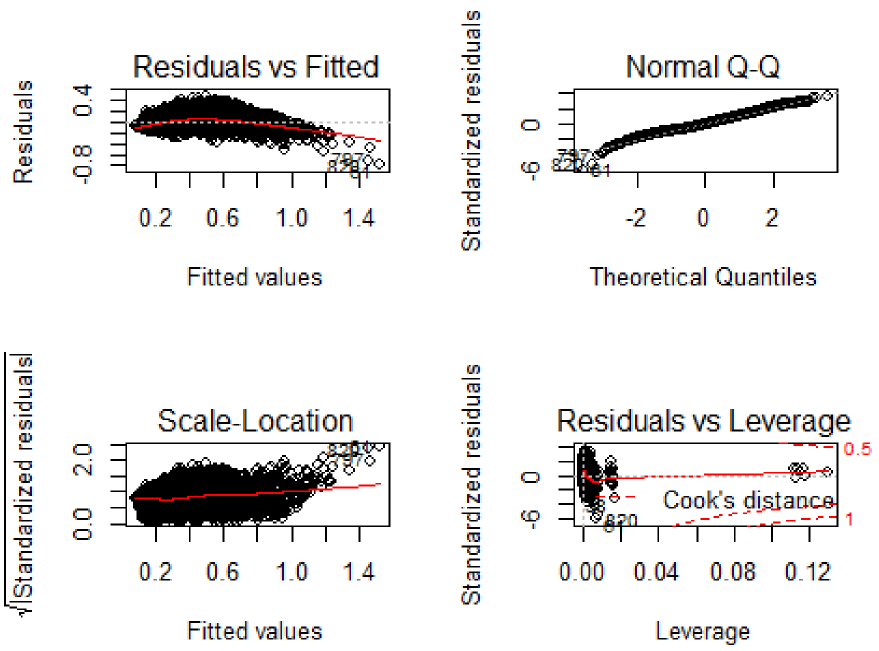
```
# PCR model used for prediction
my_data_pre <- predict(my_data)
my_data_pre_test <- predict(my_data_test)
# Add three columns to the original dataset a1,a2,a3 & a4
data$a1 <- my_data_pre[,1]
data$a2 <- my_data_pre[,2]
data$a3 <- my_data_pre[,3]
data$a4 <- my_data_pre[,4]
fit1 <- lm(winPlacePerc~a1+a2+a3+a4,data = data)
summary(fit1)

##
## Call:
## lm(formula = winPlacePerc ~ a1 + a2 + a3 + a4, data = data)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.77400 -0.09247 -0.01185  0.08797  0.45163
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.408760   0.002244 182.129  < 2e-16 ***
## a1           0.090960   0.001020  89.213  < 2e-16 ***
## a2           0.013121   0.001670   7.855 5.32e-15 ***
## a3           0.107304   0.001900  56.482  < 2e-16 ***
## a4          -0.012961   0.002239  -5.787 7.81e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1304 on 3369 degrees of freedom
## Multiple R-squared:  0.7695, Adjusted R-squared:  0.7692
## F-statistic:  2811 on 4 and 3369 DF,  p-value: < 2.2e-16
```

```
data_test$a1 <- my_data_pre_test[,1]
data_test$a2 <- my_data_pre_test[,2]
data_test$a3 <- my_data_pre_test[,3]
data_test$a4 <- my_data_pre_test[,4]
fit2 <- lm(winPlacePerc~a1+a2+a3+a4,data = data_test)
summary(fit2)
```

```
##
## Call:
## lm(formula = winPlacePerc ~ a1 + a2 + a3 + a4, data = data_test)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.77458 -0.10877 -0.01442  0.09337  0.57242
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.494362   0.004820 102.555  < 2e-16 ***
## a1           0.102123   0.002132  47.890  < 2e-16 ***
## a2           0.017332   0.003519   4.925 9.87e-07 ***
## a3           0.094194   0.004205  22.400  < 2e-16 ***
## a4          -0.017892   0.004782  -3.742 0.000193 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1522 on 992 degrees of freedom
## Multiple R-squared:  0.7407, Adjusted R-squared:  0.7396
## F-statistic: 708.4 on 4 and 992 DF,  p-value: < 2.2e-16
```

```
par(mfrow=c(2,2))
plot(fit1)
```



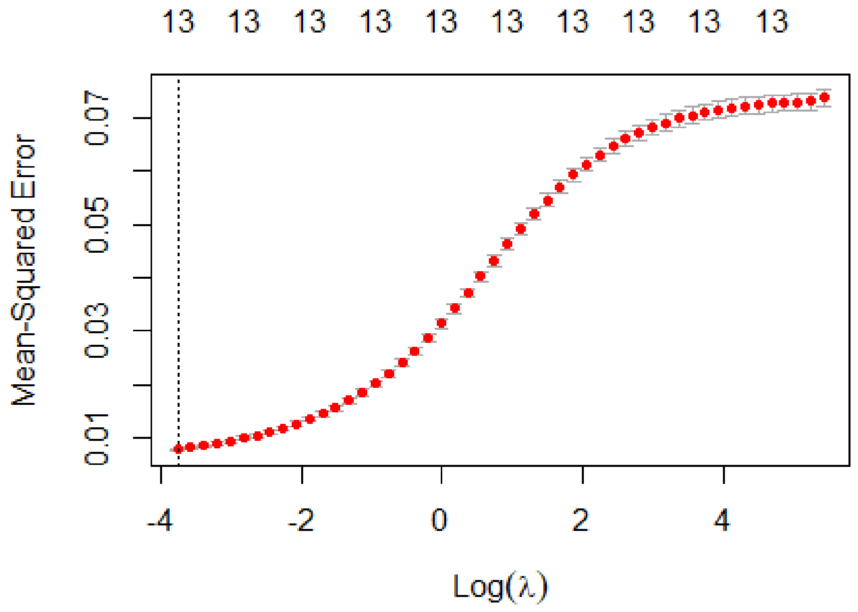
```
par(mfrow=c(1,1))
```

It can be seen from the above table that the four principal components of the newly established model have significant influence on the model. Therefore, in this case, the principal component of the original data was reduced in dimension, and the four principal components were selected to re-establish the regression model, which not only excluded multicollinearity, but also effectively explained the relationship between independent variables and dependent variables.

Next, we use another two regression methods: **Ridge regression** and **Lasso regression** and compare the results of them with that of **PCR**.

3.3 Ridge

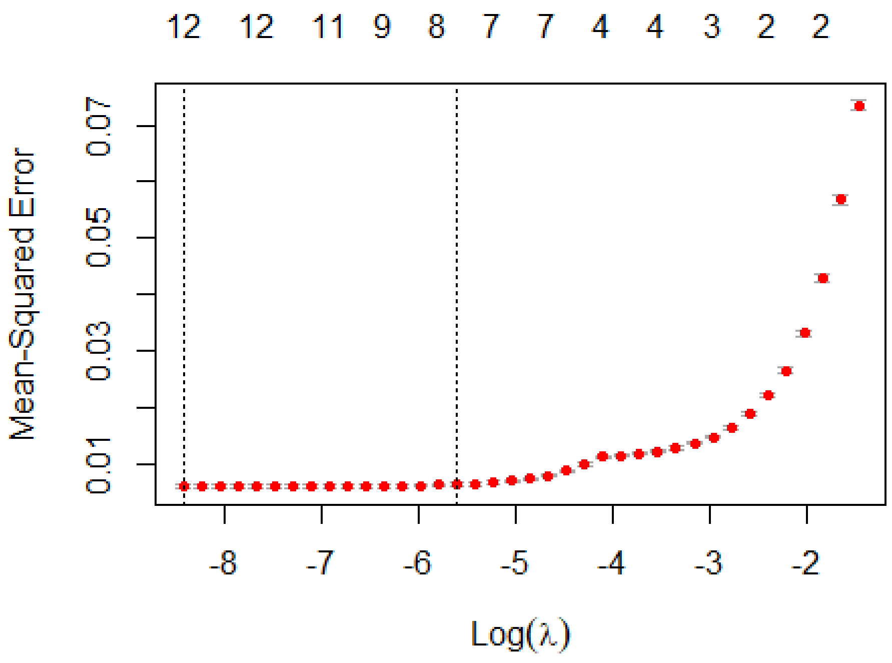
```
#ridge regression
ridg1 <- cv.glmnet(x=model.matrix(~.,as.data.frame(scale(data[,1:1
3]))),
                   y=data$winPlacePerc,
                   family = 'gaussian',
                   alpha = 0,
                   # nfolds = 10,
                   nlambda = 50)
plot(ridg1)
```



```
bestlam1=ridg1$lambda.min
bestlam2=ridg1$lambda.1se
pred.ridg <- predict(ridg1,s=c(bestlam1,bestlam2),
             newx=model.matrix(~.,as.data.frame(scale(data_test[,1:1
3]))))
mean((pred.ridg[,1]-data_test$winPlacePerc)^2)#whenlambda=lambda.min

## [1] 0.01937349

mean((pred.ridg[,2]-data_test$winPlacePerc)^2)#whenlambda=lambda.lse

## [1] 0.01937349

# get the 2 corresponging independent variables coefficients of 2 lambd
a values
predict(ridg1,s=c(bestlam1,bestlam2),
        newx=model.matrix(~.,as.data.frame(scale(data[,1:13]))),
        type="coefficients")

## 15 x 2 sparse Matrix of class "dgCMatrix"
##                            1             2
## (Intercept)      0.4087601956  0.4087601956
## (Intercept)      .             .
## boosts           0.0306295433  0.0306295433
## damageDealt      0.0035440104  0.0035440104
## headshotKills   -0.0011768925 -0.0011768925
## heals            0.0065156632  0.0065156632
## killPlace       -0.1596355969 -0.1596355969
## kills           -0.0305186906 -0.0305186906
## killStreaks     -0.0467874370 -0.0467874370
## maxPlace         0.0004765447  0.0004765447
## numGroups        0.0179301548  0.0179301548
## roadKills        0.0024836044  0.0024836044
## vehicleDestroys  0.0017947994  0.0017947994
## walkDistance     0.1049211124  0.1049211124
## weaponsAcquired  0.0359033595  0.0359033595
```

3.4 Lasso

```
#lasso regression
laso2 <- cv.glmnet(x=model.matrix(~.,as.data.frame(scale(data[,1:1
3]))),
                   y=data$winPlacePerc,
                   family = 'gaussian',
                   alpha = 1,
                   # nfolds = 10,
                   nlambda = 50)
plot(laso2)
```



```
bestlam1_la=laso2$lambda.min
bestlam2_la=laso2$lambda.1se
pred.laso <- predict(laso2,s=c(bestlam1_la,bestlam2_la),
              newx=model.matrix(~.,as.data.frame(scale(data_test[,1:1
3]))))
mean((pred.laso[,1]-data_test$winPlacePerc)^2)#whenlambda=lambda.min

## [1] 0.01750406

mean((pred.laso[,2]-data_test$winPlacePerc)^2)#whenlambda=lambda.lse

## [1] 0.0178244

# lambda.min 误差最小, lambda.lse 性能优良,自变量个数最少
```

From the output, we find that lambda.min has the least error and lambda.lse has good performance, and the number of independent variables is the least.

```
# get the 2 corresponging independent variables coefficients of 2 lambd
a values
predict(laso2,s=c(bestlam1_la,bestlam2_la),
        newx=model.matrix(~.,as.data.frame(scale(data[,1:13]))),
        type="coefficients")

## 15 x 2 sparse Matrix of class "dgCMatrix"
##                             1            2
## (Intercept)      0.408760196  0.4087601956
## (Intercept)      .            .
## boosts           0.029447763  0.0260320326
## damageDealt      0.003705046  .
## headshotKills    .            .
## heals            0.002728442  0.0004999435
## killPlace       -0.275135148 -0.2355683247
## kills           -0.061553789 -0.0418743373
## killStreaks     -0.106658497 -0.0861816280
## maxPlace         0.001067406  .
## numGroups        0.024355784  0.0195602439
## roadKills        0.002197587  .
## vehicleDestroys  0.001012945  .
## walkDistance     0.085914848  0.0954356294
## weaponsAcquired  0.012505105  0.0169711546
```

Finally, we compare the R^2 squares of three models.

```
# omparison between models

# rmse.laso <- min(sqrt(mean((pred.laso[,1]-data$winPlacePerc)^2)),
#                  sqrt(mean((pred.laso[,2]-data$winPlacePerc)^2)))
# rmse.laso
#
# rmse.ridg <- min(sqrt(mean((pred.ridg[,1]-data$winPlacePerc)^2)),
#                  sqrt(mean((pred.ridg[,2]-data$winPlacePerc)^2)))
```

```r
# rmse.ridg
#
# rmse.lm <- sqrt(mean(resid(fit)^2))
# rmse.lm
SST_rid <- sum((data_test$winPlacePerc-mean(data_test$winPlacePerc))^2)
SSE_rid <- sum((pred.ridg[,1]-data_test$winPlacePerc)^2)
R2_rid <- 1 - SSE_rid/SST_rid
R2_rid

## [1] 0.7820591

SST_las <- sum((data_test$winPlacePerc-mean(data_test$winPlacePerc))^2)
SSE_las <- sum((pred.laso[,1]-data_test$winPlacePerc)^2)
R2_las <- 1 - SSE_las/SST_las
R2_las

## [1] 0.8030892
```

3.5 KNN

3.5.1 Model Training

KNN is a discriminative algorithm since it models the conditional probability of a sample belonging to a given class. Frequently, KNN is used for classification. Nevertheless, it can be used for regression as well.

There is not a fixed N value which is used for choosing the nearest points. Because:

(1) Small values for K can be noisy and subject to the effects of outliers.

(2) Larger values of K will have smoother decision boundaries which mean lower variance but increased bias.

Conventionally, $\sqrt{n}$, where n is the total training observations. So here we train the model with N = 10, 20……100. respectively. And then choose the best N using R^2 and the decision rule.

Because each model needs about a few minutes to run, so here I only list the final result — N = 10 rather than run it again.

And the corresponding R^2

N	10	20	30	40	50

R^2	0.821	0.792	0.772	0.756	0.743
N	60	70	80	90	100
R^2	0.730	0.721	0.711	0.702	0.694

```
matrix=as.matrix(data[-14])
scale=scale(matrix)
mean=mean(data$winPlacePerc)

# KNN for N=10
predict1=matrix(0,nrow=3374,ncol=1)

for (j in 1:3374) {
difference=matrix(0,nrow=1,ncol=3374)
for (i in 1:3374) {
    difference[1,i]=(scale[i,1]-scale[j,1])^2+(scale[i,2]-scale[j,2])^2
+(scale[i,3]-scale[j,3])^2+(scale[i,4]-scale[j,4])^2+(scale[i,5]-scale
[j,5])^2+(scale[i,6]-scale[j,6])^2+(scale[i,7]-scale[j,7])^2+(scale[i,
8]-scale[j,8])^2+(scale[i,9]-scale[j,9])^2+(scale[i,10]-scale[j,10])^2+
(scale[i,11]-scale[j,11])^2+(scale[i,12]-scale[j,12])^2+(scale[i,13]-sc
ale[j,13])^2

   }
   orderdiff_scale=order(difference[1,],decreasing=FALSE)
   orderdiff1_scale=as.vector(orderdiff_scale[1:11])
   total=0
   for(k in orderdiff1_scale[-1]){
     total=total+as.numeric(data[k,14])
   }

     predict1[j,1]=total/10
 }


residuals1=data[,14]-predict1
SSR1=sum((predict1-mean)^2)
SST=sum((data[,14]-mean)^2)
MSE1=sum(residuals1^2)/3374
```

3.5.2 Model Testing

```
data_test <- read_excel("test_data.xlsx")
matrix_test=as.matrix(data_test[-14])
scale_test=scale(matrix_test)
```

```
mean_test=mean(data_test$winPlacePerc)
#
# Test set for N=10
predict_test=matrix(0,nrow=997,ncol=1)
for (j in 1:997) {
difference_test=matrix(0,nrow=1,ncol=3374)
for (i in 1:3374) {
    difference_test[1,i]=(scale[i,1]-scale_test[j,1])^2+(scale[i,2]-sca
le_test[j,2])^2+(scale[i,3]-scale_test[j,3])^2+(scale[i,4]-scale_test
[j,4])^2+(scale[i,5]-scale_test[j,5])^2+(scale[i,6]-scale_test[j,6])^2+
(scale[i,7]-scale_test[j,7])^2+(scale[i,8]-scale_test[j,8])^2+(scale[i,
9]-scale_test[j,9])^2+(scale[i,10]-scale_test[j,10])^2+(scale[i,11]-sca
le_test[j,11])^2+(scale[i,12]-scale_test[j,12])^2+(scale[i,13]-scale_te
st[j,13])^2

   }
   orderdiff_test=order(difference_test[1,],decreasing=FALSE)
   orderdiff1_test=as.vector(orderdiff_test[1:11])
  total=0
   for(k in orderdiff1_test[-1]){
     total=total+as.numeric(data[k,14])
   }

     predict_test[j,1]=total/10
 }

residuals_test_KNN=data_test[,14]-predict_test
SSR_test_KNN=sum((predict_test-mean_test)^2)
SST_test_KNN=sum((data_test[,14]-mean_test)^2)

paste("The R^2 of the KNN model in the test set is", SSR_test_KNN/SST_t
est_KNN )
```

The R^2 of the KNN model in the test set is 0.848

3.6 SVM

```
setwd("D:/Collection_NUT/SUSTech_31/R/HW/Project/data")  # delete
data = read_excel("train_data.xlsx")
data =  subset(data, select = -swimDistance )
```

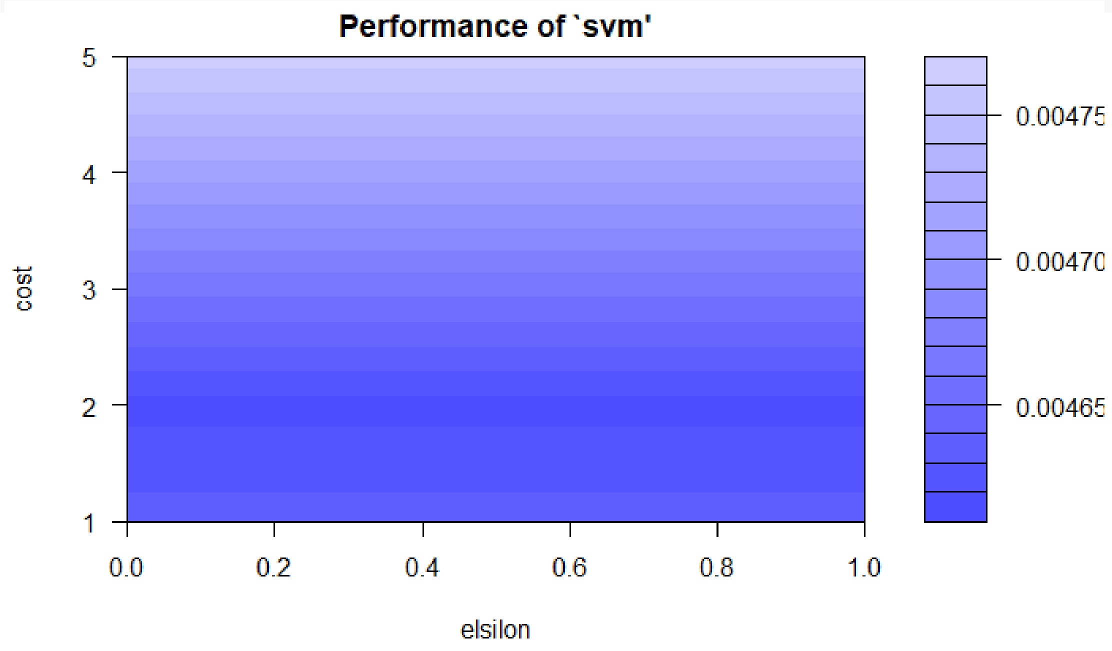
3.6.1 Model Training

```
# Regression with SVM
modelsvm = svm(winPlacePerc~.,data)

# Predict using SVM regression
predYsvm = predict(modelsvm, data)

# Calculate RMSE
RMSEsvm = rmse(predYsvm,data$winPlacePerc)
```

Tuning SVR model by varying values of maximum allowable error and cost parameter:

```
# Tune the SVM model
OptModelsvm=tune(svm, winPlacePerc~., data=data, ranges=list(elsilon=se
q(0,1,0.2), cost=1:5))

# Print optimum value of parameters
print(OptModelsvm)

# Plot the perfrormance of SVM Regression model
plot(OptModelsvm)

## Select the best model out of  trained models and compute RMSE

# Find out the best model
BstModel=OptModelsvm$best.model

# Predict Y using best model
PredYBst=predict(BstModel,data)

# Calculate RMSE of the best model
MSEBst=mse(PredYBst,data$winPlacePerc)
```

```
## Parameter tuning of ‘svm’:
## - sampling method: 10-fold cross validation
## - best parameters:
## - best performance: 0.004616673
```



```
## Cal.R2 = function(y,PredYBst){
  # R2
  SST = sum((y-mean(y))^2)
  SSE = sum((y-PredYBst)^2)

  SSR = SST - SSE
  R2 = SSR/SST
  return(R2)
}

## details about best model
BstModel

##
## Call:
## svm(formula = winPlacePerc ~ ., data = data, cost = 2, epsilon = 0.
1)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  2
##       gamma:  0.07692308
##     epsilon:  0.1
##
```

```
##
## Number of Support Vectors:  1663

# to train best model, simply call:
# svm(winPlacePerc~.,data=data, cost = 2, epsilon = 0.1)
```

After training there are 1663 support vectors out of 3374 data points.

```
print(MSEBst)

## [1] 0.002791205

print(Cal.R2(data$winPlacePerc,PredYBst))

## [1] 0.9620806
```

This model explained about 96.2% variance of data.

```
## Calculate parameters of the Best SVR model

# Find value of W
W = t(BstModel$coefs) %*% BstModel$SV

# Find value of b
b = BstModel$rho
```

3.6.2 Model Testing

```
# import test data
setwd("D:/Collection_NUT/SUSTech_31/R/HW/Project/data")  # delete
TstData = read_excel("test_data.xlsx")

y.test.truth = TstData$winPlacePerc
y.test.pred = predict(BstModel,TstData)

# R^2
Cal.R2(y.test.truth, y.test.pred)

## [1] 0.8969854
```

This means about 89.45% variability has been explained.

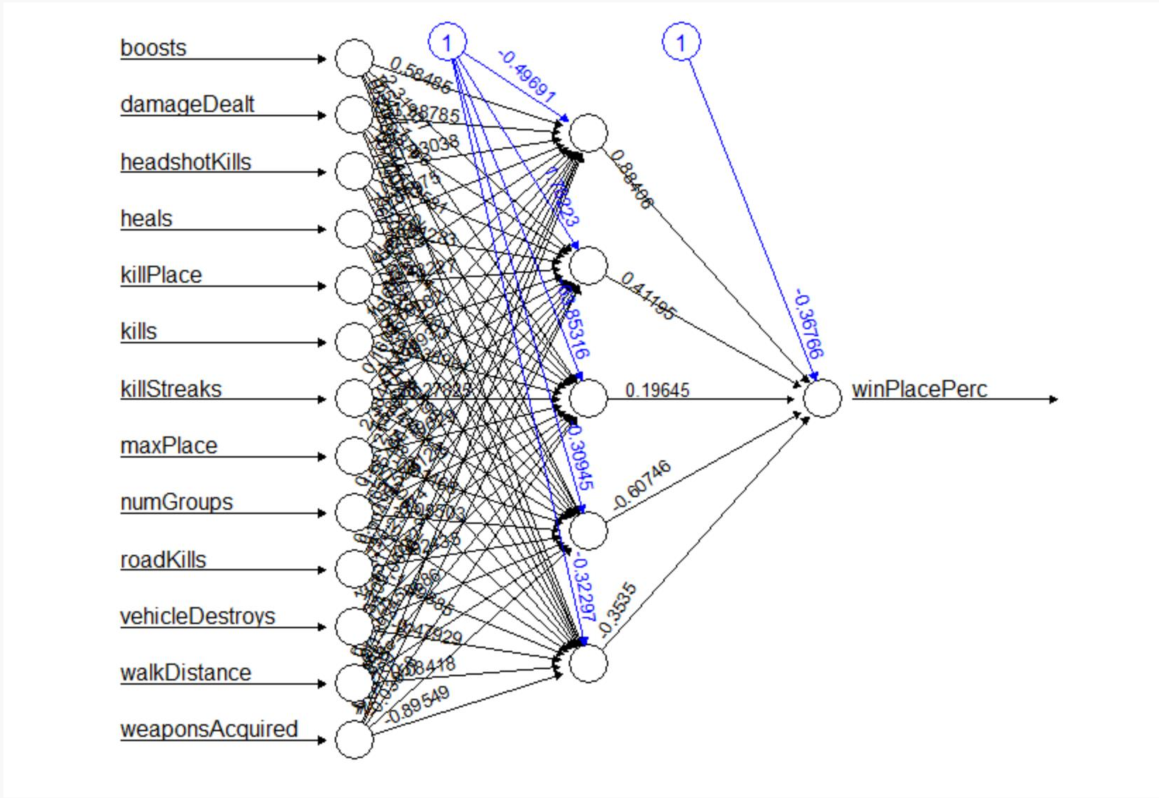
Let's see MSE:

```
mean((y.test.truth-y.test.pred)^2)

## [1] 0.00915731
```

3.7 Neural Network

3.7.1 Model Training

```r
setwd("D:/Collection_NUT/SUSTech_31/R/HW/Project/data")  # delete
data = read_excel("train_data.xlsx")
data = subset(data, select = -swimDistance )

# fit neural network
set.seed(2)
NN = neuralnet(winPlacePerc~., data, hidden = c(5) , linear.output = T
 )
#
# # plot neural network
plot(NN)
```



```r
PredYBst=predict(NN,data)

Cal.R2 = function(y,PredYBst){
  # R2
  SST = sum((y-mean(y))^2)
  SSE = sum((y-PredYBst)^2)
  SSR = SST - SSE
  R2 = SSR/SST
  return(R2)
}

Cal.R2(data$winPlacePerc,PredYBst)

## [1] 0.8867439
```

From the result, it can be seen that R^2 of the neural network model on the training set is 0.8867439.

3.7.2 Model Testing

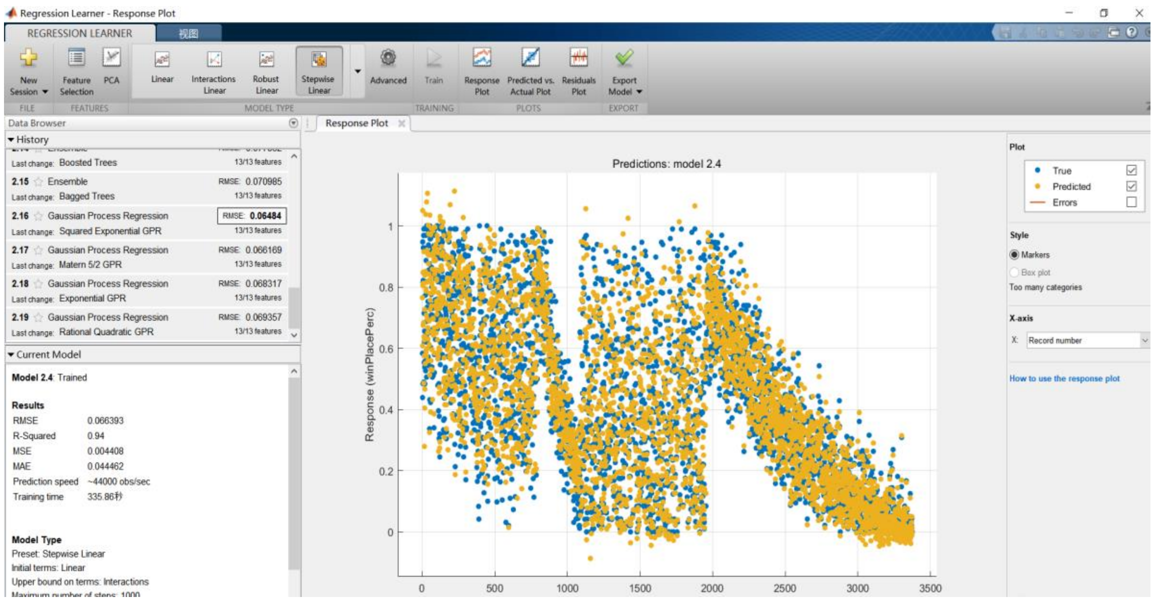
```
# import test data
setwd("D:/Collection_NUT/SUSTech_31/R/HW/Project/data")  # delete
TstData = read_excel("test_data.xlsx")

y.test.truth = TstData$winPlacePerc
y.test.pred = predict(NN,TstData)

Cal.R2(y.test.truth, y.test.pred)

## [1] 0.861229
```

From the result, it can be seen that R^2 of the neural network model on the training set is 0.861229, relatively stable.

3.8 Regression Learner

Besides the model we use above, there is a very interesting APP in MATLAB---regression learner, which can assist us run a few models frequently used and then list the best one.

The following is our output in MATLAB, showing that Gaussian Process Regression model is the best one among the built-in models. And the corresponding code for this model liess in the ending Appendix part.



History

Last change: Boosted Trees		13/13 features
2.15 Ensemble	RMSE: 0.070985	
Last change: Bagged Trees		13/13 features
2.16 Gaussian Process Regression	RMSE: **0.06484**	
Last change: Squared Exponential GPR		13/13 features
2.17 Gaussian Process Regression	RMSE: 0.066169	
Last change: Matern 5/2 GPR		13/13 features
2.18 Gaussian Process Regression	RMSE: 0.068317	
Last change: Exponential GPR		13/13 features
2.19 Gaussian Process Regression	RMSE: 0.069357	
Last change: Rational Quadratic GPR		13/13 features

Current Model

Model 2.4: Trained

Results

RMSE	0.066393
R-Squared	0.94
MSE	0.004408
MAE	0.044462
Prediction speed	~44000 obs/sec
Training time	335.86秒

Model Type

Preset: Stepwise Linear
Initial terms: Linear
Upper bound on terms: Interactions
Maximum number of steps: 1000

The R^2 for the Gaussian Process Regression model is 0.94 as shown in the figure. This model has the similar principle as KNN model, based on Bayes statistics, assuming the predicted variable as normal distribution and conducting the prediction based on the nearest points. Nevertheless, this model in our project is implemented in MATLAB, so we just mention the good fitting result here rather than further explore this model again.

4 Model Comparison

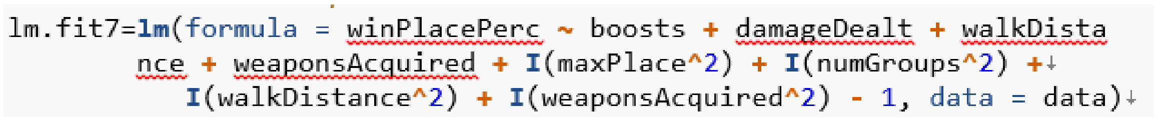
Model	LS	PCA	Lasso	Ridge	KNN	SVM	Neural Network
R^2 (training)	0.956	0.769	0.918	0.895	0.821	0.962	0.887
R^2 (testing)	0.992	0.789	0.797	0.774	0.848	0.894	0.861

The tables shows that the best R^2 in the training set belongs to SVM model, and the best R^2 in the validation set belongs to LS model (lm.fit7) in our analysis. The reason why R^2 of SVM model decreases much, I think SVM model overfits the data to some extent.

5 Conclusion

What should be noticed more is the LS model, with relatively stable character. It has only 8 predictors, compared with thousands of observations, more difficult to have the overfitting problem, thus having a better generalization ability.

Considering both the interpretability and the goodness on the validation set, we finally choose the lm.fit7 model for our prediction, which settles our first question. With respect to our second question, let's return to lm.fit7.



```
lm.fit7=lm(formula = winPlacePerc ~ boosts + damageDealt + walkDista
        nce + weaponsAcquired + I(maxPlace^2) + I(numGroups^2) +↓
           I(walkDistance^2) + I(weaponsAcquired^2) - 1, data = data)↓
```

The predictors of walkDistance and weaponsAcquired are significant in our model, because they both appear on the linear term and quadratic term. So for the second question---what' s the best strategy to win in PUBG? The answer is very simple.

Run more, keep calm and obtain weapons!

Even if you are a good player, do not beat other players unless you have to. Because f ighting can increase your chance of death much higher. Also, run towards the safe ar ea all the time and try to find the good gun accessories like expansion packs and high power lens which can increase player' s probability of survival.

Appendix (MATLAB®)

```
function [trainedModel, validationRMSE] = trainRegressionModel(training
Data)
% [trainedModel, validationRMSE] = trainRegressionModel(trainingData)
% returns a trained regression model and its RMSE. This code recreate
s the
% model trained in Regression Learner app. Use the generated code to
% automate training the same model with new data, or to learn how to
% programmatically train models.
%
%  Input:
%      trainingData: a table containing the same predictor and response
%       columns as imported into the app.
%
%  Output:
%      trainedModel: a struct containing the trained regression mode
l. The
%       struct contains various fields with information about the train
ed
%       model.
%
%      trainedModel.predictFcn: a function to make predictions on new d
ata.
%
```

```
%      validationRMSE: a double containing the RMSE. In the app, the
%       History list displays the RMSE for each model.
%
% Use the code to train the model with new data. To retrain your model,
% call the function from the command line with your original data or ne
w
% data as the input argument trainingData.
%
% For example, to retrain a regression model trained with the origina
l data
% set T, enter:
%   [trainedModel, validationRMSE] = trainRegressionModel(T)
%
% To make predictions with the returned &apos;trainedModel&apos; on ne
w data T2, use
%   yfit = trainedModel.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns a
s used
% during training. For details, enter:
%   trainedModel.HowToPredict

% Auto-generated by MATLAB on 2019-12-25 16:29:56

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {&apos;boosts&apos;, &apos;damageDealt&apos;, &apos;he
adshotKills&apos;, &apos;heals&apos;, &apos;killPlace&apos;, &apos;kill
s&apos;, &apos;killStreaks&apos;, &apos;maxPlace&apos;, &apos;numGroups
&apos;, &apos;roadKills&apos;, &apos;vehicleDestroys&apos;, &apos;walkD
```

```
istance&apos;, &apos;weaponsAcquired&apos;};
predictors = inputTable(:, predictorNames);
response = inputTable.winPlacePerc;
isCategoricalPredictor = [false, false, false, false, false, false, fal
se, false, false, false, false, false, false];

% Train a regression model
% This code specifies all the model options and trains the model.
regressionGP = fitrgp(...
    predictors, ...
    response, ...
    &apos;BasisFunction&apos;, &apos;constant&apos;, ...
    &apos;KernelFunction&apos;, &apos;squaredexponential&apos;, ...
    &apos;Standardize&apos;, true);

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
gpPredictFcn = @(x) predict(regressionGP, x);
trainedModel.predictFcn = @(x) gpPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedModel.RequiredVariables = {&apos;boosts&apos;, &apos;damageDealt
&apos;, &apos;headshotKills&apos;, &apos;heals&apos;, &apos;killPlace&a
pos;, &apos;kills&apos;, &apos;killStreaks&apos;, &apos;maxPlace&apo
s;, &apos;numGroups&apos;, &apos;roadKills&apos;, &apos;vehicleDestroys
&apos;, &apos;walkDistance&apos;, &apos;weaponsAcquired&apos;};
trainedModel.RegressionGP = regressionGP;
trainedModel.About = &apos;This struct is a trained model exported fro
m Regression Learner R2018b.&apos;;
trainedModel.HowToPredict = sprintf(&apos;To make predictions on a ne
w table, T, use: \n  yfit = c.predictFcn(T) \nreplacing &apos;&apos;c&a
pos;&apos; with the name of the variable that is this struct, e.g. &apo
```

```
s;&apos;trainedModel&apos;&apos;. \n \nThe table, T, must contain the v
ariables returned by: \n  c.RequiredVariables \nVariable formats (e.
g. matrix✌ector, datatype) must match the original training data. \n
Additional variables are ignored. \n \nFor more information, see <a hre
f="matlab:helpview(fullfile(docroot, &apos;&apos;stats&apos;&apos;, &ap
os;&apos;stats.map&apos;&apos;), &apos;&apos;appregression_exportmodelt
oworkspace&apos;&apos;)">How to predict using an exported model</a>.&ap
os;);

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {&apos;boosts&apos;, &apos;damageDealt&apos;, &apos;he
adshotKills&apos;, &apos;heals&apos;, &apos;killPlace&apos;, &apos;kill
s&apos;, &apos;killStreaks&apos;, &apos;maxPlace&apos;, &apos;numGroups
&apos;, &apos;roadKills&apos;, &apos;vehicleDestroys&apos;, &apos;walkD
istance&apos;, &apos;weaponsAcquired&apos;};
predictors = inputTable(:, predictorNames);
response = inputTable.winPlacePerc;
isCategoricalPredictor = [false, false, false, false, false, false, fal
se, false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedModel.RegressionGP, &apos;KFold&apo
s;, 5);

% Compute validation predictions
validationPredictions = kfoldPredict(partitionedModel);

% Compute validation RMSE
```

```
validationRMSE = sqrt(kfoldLoss(partitionedModel, &apos;LossFun&apo
s;, &apos;mse&apos;));
```