Final Project

The Outliers

2025-04-16

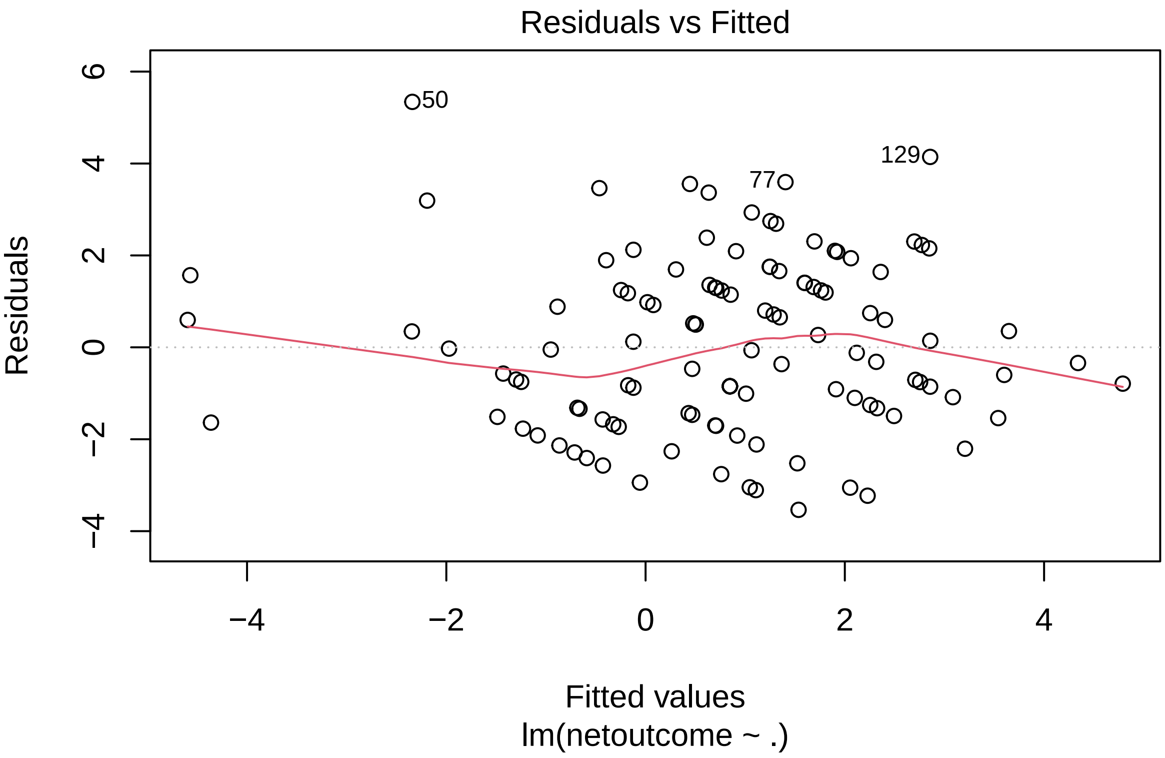
Data Loading and Cleaning

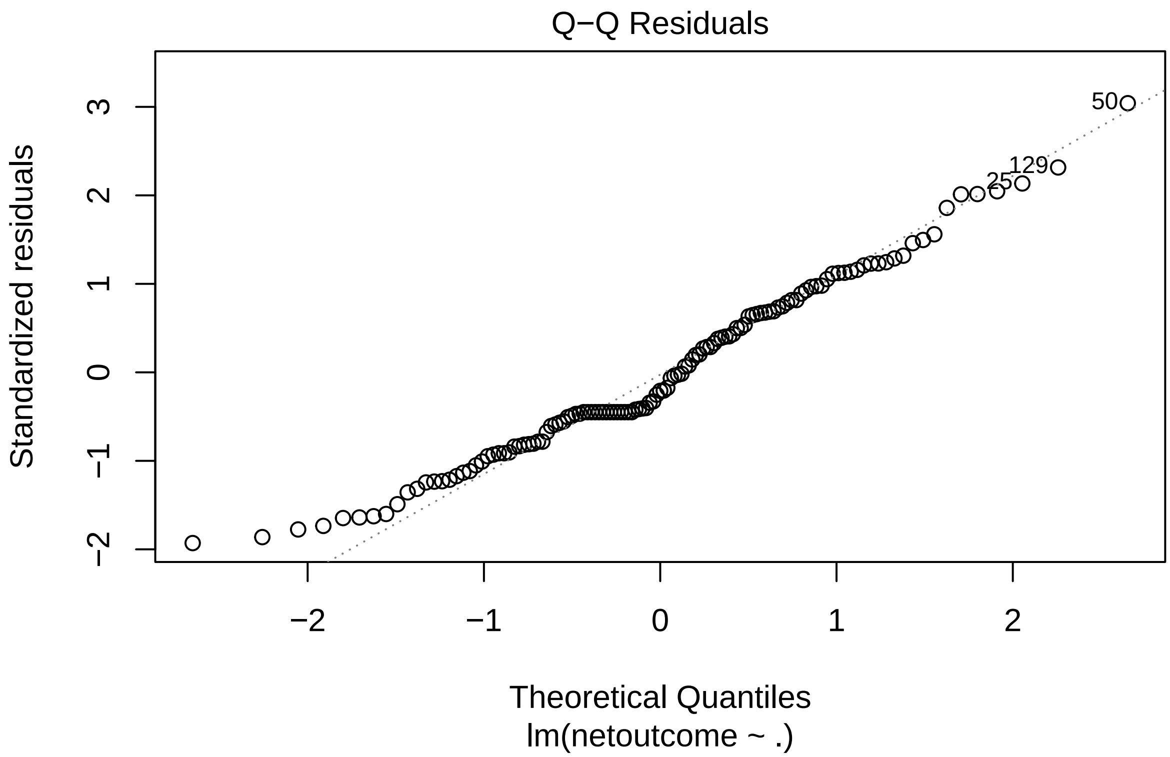
```r
raw <- read.csv(csv_path, header=TRUE)
raw$Hand
data.rp <- raw[raw$Hand == "R" & raw$Grip=="P",]

#Remove irrelevant information and sparse predictors
rp.lessCols <- data.rp[, !(names(data.rp) %in% c("Hand", "Grip", "Date", "Name", "Club.Team", "Height",
#Note that we no longer needed Hand and Grip since we already filtered for all RP


#rename predictors to a standard format
names(rp.lessCols)[names(rp.lessCols) == "D.CounterA"] <- "CounterA.D"
names(rp.lessCols)[names(rp.lessCols) == "TR.CounterA"] <- "CounterA.TR"
names(rp.lessCols)[names(rp.lessCols) == "P.Riposte..D..TS"] <- "RiposteDef.TS"
names(rp.lessCols)[names(rp.lessCols) == "D.P.Riposte"] <- "RiposteDef.D"
names(rp.lessCols)[names(rp.lessCols) == "TR.P.Riposte.D."] <- "RiposteDef.TR"
names(rp.lessCols)[names(rp.lessCols) == "P.Riposte..A..TS"] <- "RiposteAtt.TS"
names(rp.lessCols)[names(rp.lessCols) == "D.P.Riposte..A."] <- "RiposteAtt.D"
names(rp.lessCols)[names(rp.lessCols) == "TR.P.Riposte..A."] <- "RiposteAtt.TR"
names(rp.lessCols)[names(rp.lessCols) == "Caught..H..TR"] <- "CaughtH.TR"
names(rp.lessCols)[names(rp.lessCols) == "Caught..B..TR"] <- "CaughtB.TR"
names(rp.lessCols)[names(rp.lessCols) == "Caught..T..TR"] <- "CaughtT.TR"
names(rp.lessCols)[names(rp.lessCols) == "X6.2..D..TS"] <- "62Def.TS"
names(rp.lessCols)[names(rp.lessCols) == "TR.6.2..D."] <- "62Def.TR"
names(rp.lessCols)[names(rp.lessCols) == "X6.2..A..TS"] <- "62Att.TS"
names(rp.lessCols)[names(rp.lessCols) == "D.6.2.A."] <- "62Att.D"
names(rp.lessCols)[names(rp.lessCols) == "TR"] <- "62Att.TR"
names(rp.lessCols)[names(rp.lessCols) == "Lunge..T..TS"] <- "LungeT.TS"
names(rp.lessCols)[names(rp.lessCols) == "D.Lunge.T."] <- "LungeT.D"
names(rp.lessCols)[names(rp.lessCols) == "TR.Lunge..T."] <- "LungeT.TR"
names(rp.lessCols)[names(rp.lessCols) == "Lunge.L..TS"] <- "LungeL.TS"
names(rp.lessCols)[names(rp.lessCols) == "TR.1"] <- "LungeL.TR"
names(rp.lessCols)[names(rp.lessCols) == "Lunge..B..TS"] <- "LungeB.TS"
names(rp.lessCols)[names(rp.lessCols) == "D.Lunge.B."] <- "LungeB.D"
names(rp.lessCols)[names(rp.lessCols) == "TR.2"] <- "LungeB.TR"
names(rp.lessCols)[names(rp.lessCols) == "Jump.Lunge.TS"] <- "JumpLunge.TS"
names(rp.lessCols)[names(rp.lessCols) == "D.Jump.Lunge"] <- "JumpLunge.D"
names(rp.lessCols)[names(rp.lessCols) == "TR.Jump.Lunge"] <- "JumpLunge.TR"
names(rp.lessCols)[names(rp.lessCols) == "Adv..Lunge.TS"] <- "AdvLunge.TS"
names(rp.lessCols)[names(rp.lessCols) == "D.Adv..Lunge"] <- "AdvLunge.D"
names(rp.lessCols)[names(rp.lessCols) == "TR.Adv..Lunge"] <- "AdvLunge.TR"
names(rp.lessCols)[names(rp.lessCols) == "D.Fleche"] <- "Fleche.D"
names(rp.lessCols)[names(rp.lessCols) == "TR.Fleche"] <- "Fleche.TR"
names(rp.lessCols)[names(rp.lessCols) == "Prep.to.guard.TS"] <- "PrepGuard.TS"
names(rp.lessCols)[names(rp.lessCols) == "TR.Prep.to.guard"] <- "PrepGuard.TR"
names(rp.lessCols)[names(rp.lessCols) == "D.Squat"] <- "Squat.D"
names(rp.lessCols)[names(rp.lessCols) == "TR.Squat"] <- "Squat.TR"

#NA = 0
rp.lessCols[is.na(rp.lessCols)] <- 0

cols <- names(rp.lessCols)

# Remove suffix like ".TS", ".TR", ".D"
cleanedActions <- gsub("\\.(TS|TR|D)$", "", cols)          # remove suffix
# Get unique base action names
baseActions <- unique(cleanedActions)
baseActions

df <- rp.lessCols

for (a in baseActions) {
  ts <- paste0(a, ".TS")
  tr <- paste0(a, ".TR")
  d  <- paste0(a, ".D")

  present <- c(ts, tr, d) %in% names(df)
  if (any(present)) {
    ts_val <- if (ts %in% names(df)) as.numeric(df[[ts]]) else 0
    tr_val <- if (tr %in% names(df)) as.numeric(df[[tr]]) else 0
    d_val  <- if (d  %in% names(df)) as.numeric(df[[d]])  else 0

    df[[paste0(a, "_Exec")]] <- ts_val + tr_val + d_val
  }
}


# Net outcome = total TS - total TR
ts_total <- rowSums(df[, grep("\\.TS$", names(df))], na.rm = TRUE)
tr_total <- rowSums(df[, grep("\\.TR$", names(df))], na.rm = TRUE)
df$netoutcome <- ts_total - tr_total

clean <- df[,40:56]
```

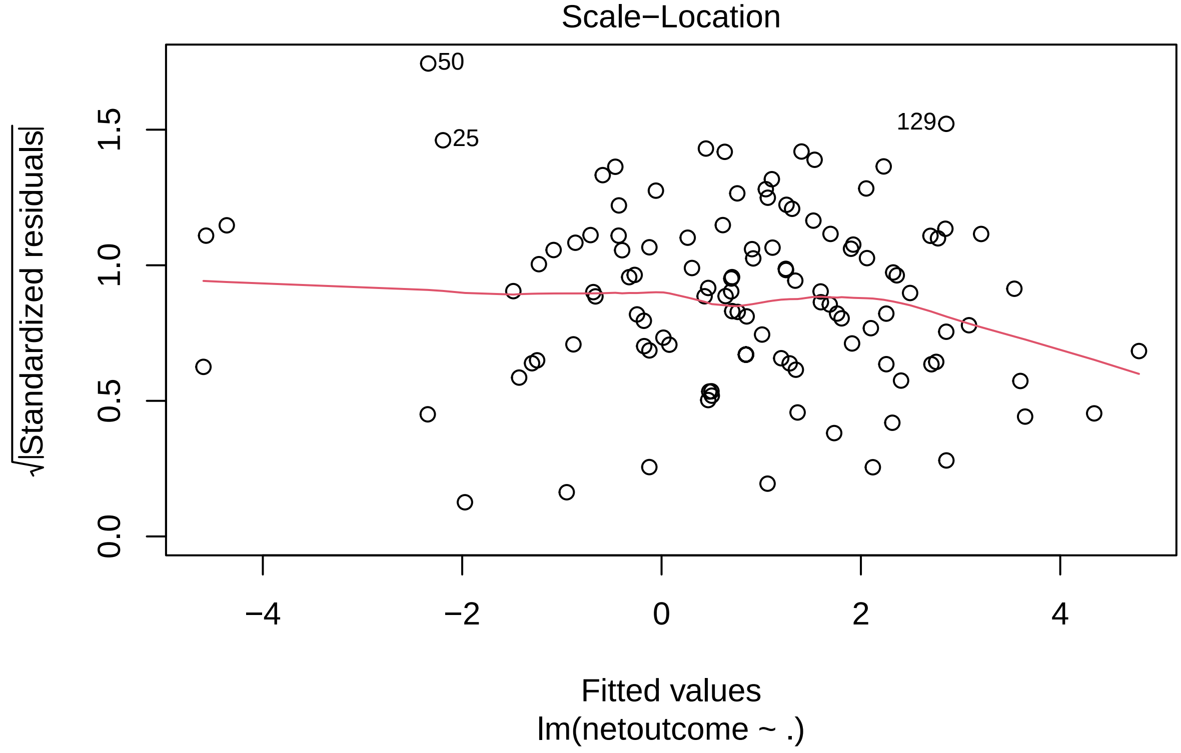
```
##   [1] "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "L" "R" "R"
##  [19] "R" "R" "R" "L" "L" "L" "R" "R" "R" "R" "R" "R" "L" "R" "R" "L" "R" "R"
##  [37] "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "L"
##  [55] "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R" "L" "R" "L" "R"
##  [73] "R" "R" "R" "R" "R" "R" "R" "R" "L" "R" "L" "R" "R" "R" "R" "R" "R" "R"
##  [91] "R" "R" "R" "R" "R" "R" "R" "R" ""  ""  ""  ""  "R" "L" "R" "R" "R" "" 
## [109] "R" "R" "R" "L" "R" "R" "R" "R" "L" ""  ""  ""  "L" "R" "L" "R" ""  "R"
## [127] "R" "R" "R" "R" "L" "R" "R" "R" "R" "R" "L" "R" ""  ""  "R" "R" "R" "R"
## [145] "R" "R" "R" "R" "R" "R" "R" "L" "R" "R" "R" "R" "R" "R" "R" "R" "R" "R"
## [163] "L" "R" "R" "L" "R" "R" "L" "R" "R" "L" "R" "R" "R" "R" "R" "R" "R" "R"
## [181] "L" "R" "R" ""
##  [1] "CounterA"   "RiposteDef" "RiposteAtt" "CaughtH"    "CaughtB"
##  [6] "CaughtT"    "62Def"      "62Att"      "LungeT"     "LungeL"
## [11] "LungeB"     "JumpLunge"  "AdvLunge"   "Fleche"     "PrepGuard"
## [16] "Squat"
```

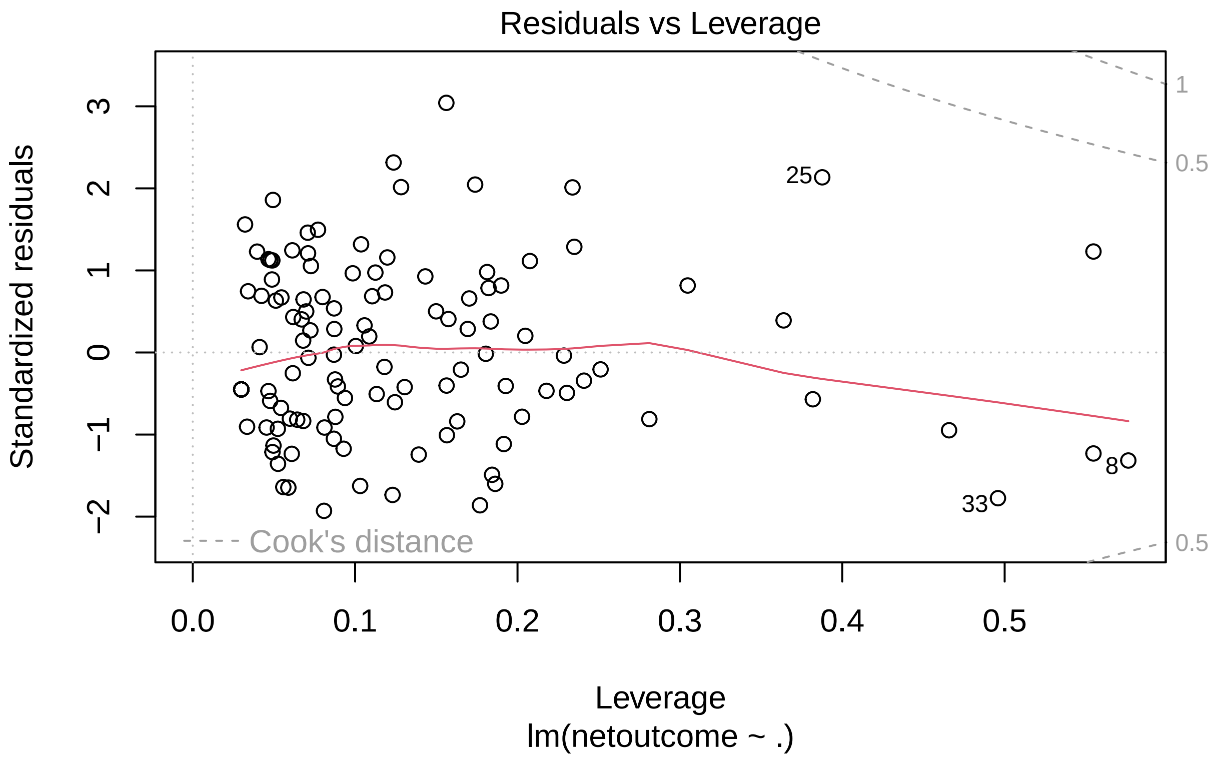
#Assumptions

```
library(usdm)
vifstep(clean[,-17], th=10)
clean <- clean[, -8]

lm.obj <- lm(netoutcome~., data = clean)
summary(lm.obj)
plot(lm.obj)
```




Q−Q Residuals
Standardized residuals
−2
−1
0
1
2
3
−2
−1
0
1
2
Theoretical Quantiles
lm(netoutcome ~ .)
50
129
25


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
−4
−2
0
2
4
Fitted values
lm(netoutcome ~ .)
50
25
129



```
## 1 variables from the 16 input variables have collinearity problem:
##
## 62Att_Exec
##
## After excluding the collinear variables, the linear correlation coefficients ranges between:
## min correlation ( LungeB_Exec ~ LungeT_Exec ):  -0.001032925
## max correlation ( LungeB_Exec ~ CaughtT_Exec ):  0.2540481
##
## ---------- VIFs of the remained variables --------
##          Variables      VIF
## 1    CounterA_Exec 1.208533
## 2  RiposteDef_Exec 1.140937
## 3  RiposteAtt_Exec 1.130921
## 4     CaughtH_Exec 1.150833
## 5     CaughtB_Exec 1.105136
## 6     CaughtT_Exec 1.134497
## 7       62Def_Exec 1.086469
## 8      LungeT_Exec 1.146990
## 9      LungeL_Exec 1.130484
## 10     LungeB_Exec 1.158701
## 11  JumpLunge_Exec 1.114313
## 12   AdvLunge_Exec 1.065461
## 13     Fleche_Exec 1.127257
## 14  PrepGuard_Exec 1.175412
## 15      Squat_Exec 1.141173
##
## Call:
## lm(formula = netoutcome ~ ., data = clean)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max 
## -3.5357 -1.3154 -0.3649  1.3011  5.3409 
## 
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)    
## (Intercept)      0.846781   0.330387   2.563  0.01174 *  
## CounterA_Exec   -0.147020   0.149082  -0.986  0.32624    
## RiposteDef_Exec -0.341445   0.120803  -2.826  0.00560 ** 
## RiposteAtt_Exec -0.211339   0.263942  -0.801  0.42505    
## CaughtH_Exec    -0.823917   0.749355  -1.100  0.27397    
## CaughtB_Exec    -1.798803   0.388154  -4.634 9.98e-06 ***
## CaughtT_Exec    -3.647527   0.733132  -4.975 2.45e-06 ***
## `62Def_Exec`    -1.818564   1.420235  -1.280  0.20310    
## LungeT_Exec      0.194180   0.226023   0.859  0.39216    
## LungeL_Exec     -0.006858   0.403191  -0.017  0.98646    
## LungeB_Exec      1.145577   0.283075   4.047 9.73e-05 ***
## JumpLunge_Exec   0.653551   0.242782   2.692  0.00823 ** 
## AdvLunge_Exec    0.439822   0.164164   2.679  0.00852 ** 
## Fleche_Exec     -0.046415   0.124839  -0.372  0.71077    
## PrepGuard_Exec   0.355500   0.328454   1.082  0.28149    
## Squat_Exec      -0.239310   0.312293  -0.766  0.44515    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 1.911 on 109 degrees of freedom
## Multiple R-squared:  0.4421, Adjusted R-squared:  0.3653 
## F-statistic: 5.758 on 15 and 109 DF,  p-value: 1.418e-08
```

```
set.seed(1)
#AIC
cat("\n--- AIC (step) Model ---\n")
step(lm.obj, direction="both",trace=0, steps=1000, k=2)

AIC_step <- lm(formula = netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec +
    LungeB_Exec + JumpLunge_Exec + AdvLunge_Exec, data = clean)
aic_formula <- netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec +
    LungeB_Exec + JumpLunge_Exec + AdvLunge_Exec

n <- nrow(clean)

#BIC
cat("-------BIC (step) Model-------")
step(lm.obj, direction="both",trace=0, steps=1000, k=log(n))
BIC_step <- lm(formula = netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec + LungeB_Exec + Jump
BICS_formula <- netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec + LungeB_Exec + JumpLunge_Exe

#All-Subset
library(leaps)

mm <- model.matrix(netoutcome ~ ., data = clean)
# drop the "(Intercept)" column
xMat <- mm[, -1]

# Fit all subset models
```

```r
regfit <- regsubsets(netoutcome ~ ., data = clean, nvmax = 16)  #16 total predictors
reg_summary <- summary(regfit)

# Find best size for each model
best_adjr2_size <- which.max(reg_summary$adjr2)
best_cp_size <- which.min(reg_summary$cp)
best_bic_size <- which.min(reg_summary$bic)
cat("adjusted R² model size:", best_adjr2_size, "\n")
cat("mallows Cp model size:", best_cp_size, "\n")
cat("Best BIC model size:", best_bic_size, "\n")

# Extract var names for best model
adjr2_vars <- names(coef(regfit, best_adjr2_size))[-1]  # remove intercept
cp_vars <- names(coef(regfit, best_cp_size))[-1]
bic_vars <- names(coef(regfit, best_bic_size))[-1]  # remove intercept

# create formulas
adjr2_formula <- netoutcome ~ RiposteDef_Exec + CaughtH_Exec + CaughtB_Exec + CaughtT_Exec + `62Def_Exe
cp_formula <- netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec + LungeB_Exec + JumpLunge_Exec
bic_formula <- netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec + LungeB_Exec + JumpLunge_Exe

# fit models and summarize
adjr2_model <- lm(adjr2_formula, data = clean)
cp_model <- lm(cp_formula, data = clean)
bic_model <- lm(bic_formula, data = clean)

cat("\n--- BIC Model ---\n")
print(summary(bic_model))
cat("BIC:", BIC(bic_model), "\n")
cat("\n--- Adjusted R^2 Model ---\n")
print(summary(adjr2_model))
cat("\n--- Mallows CP Model ---\n")
print(summary(cp_model))
```

```
##
## --- AIC (step) Model ---
##
## Call:
## lm(formula = netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec +
##     LungeB_Exec + JumpLunge_Exec + AdvLunge_Exec, data = clean)
##
## Coefficients:
##     (Intercept)  RiposteDef_Exec     CaughtB_Exec     CaughtT_Exec
##          0.6711          -0.3488          -1.8583          -3.4056
##     LungeB_Exec   JumpLunge_Exec    AdvLunge_Exec
##          1.2122           0.7274           0.3856
##
## -------BIC (step) Model-------
## Call:
## lm(formula = netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec +
##     LungeB_Exec + JumpLunge_Exec + AdvLunge_Exec, data = clean)
##
## Coefficients:
##     (Intercept)  RiposteDef_Exec     CaughtB_Exec     CaughtT_Exec
##          0.6711          -0.3488          -1.8583          -3.4056
##     LungeB_Exec   JumpLunge_Exec    AdvLunge_Exec
##          1.2122           0.7274           0.3856
##
## adjusted R² model size: 8
## mallows Cp model size: 6
## Best BIC model size: 6
##
## --- BIC Model ---
##
## Call:
## lm(formula = bic_formula, data = clean)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8040 -1.0954 -0.3519  1.1167  5.3559
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.6711     0.2538   2.644  0.00930 **
## RiposteDef_Exec  -0.3488     0.1128  -3.092  0.00248 **
## CaughtB_Exec     -1.8583     0.3676  -5.055 1.59e-06 ***
## CaughtT_Exec     -3.4056     0.7076  -4.813 4.45e-06 ***
## LungeB_Exec       1.2122     0.2708   4.477 1.76e-05 ***
## JumpLunge_Exec    0.7274     0.2289   3.178  0.00190 **
## AdvLunge_Exec     0.3856     0.1590   2.425  0.01684 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.892 on 118 degrees of freedom
## Multiple R-squared:  0.4084, Adjusted R-squared:  0.3784
## F-statistic: 13.58 on 6 and 118 DF,  p-value: 1.131e-11
##
## BIC: 545.5279
##
## --- Adjusted R^2 Model ---
##
## Call:
## lm(formula = adjr2_formula, data = clean)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4471 -1.1408 -0.3638  1.2675  5.3362
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.7552     0.2572   2.936  0.00401 **
## RiposteDef_Exec  -0.3450     0.1129  -3.055  0.00279 **
## CaughtH_Exec     -0.9264     0.7114  -1.302  0.19543
## CaughtB_Exec     -1.7902     0.3773  -4.744 6.02e-06 ***
## CaughtT_Exec     -3.4382     0.7046  -4.879 3.42e-06 ***
## `62Def_Exec`     -1.6753     1.3904  -1.205  0.23068
## LungeB_Exec       1.1416     0.2727   4.187 5.54e-05 ***
## JumpLunge_Exec    0.6919     0.2287   3.025  0.00307 **
## AdvLunge_Exec     0.4206     0.1607   2.616  0.01007 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.883 on 116 degrees of freedom
## Multiple R-squared:  0.4238, Adjusted R-squared:  0.3841
## F-statistic: 10.66 on 8 and 116 DF,  p-value: 3.816e-11
##
##
## --- Mallows CP Model ---
##
## Call:
## lm(formula = cp_formula, data = clean)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8040 -1.0954 -0.3519  1.1167  5.3559
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.6711     0.2538   2.644  0.00930 **
## RiposteDef_Exec  -0.3488     0.1128  -3.092  0.00248 **
## CaughtB_Exec     -1.8583     0.3676  -5.055 1.59e-06 ***
## CaughtT_Exec     -3.4056     0.7076  -4.813 4.45e-06 ***
## LungeB_Exec       1.2122     0.2708   4.477 1.76e-05 ***
## JumpLunge_Exec    0.7274     0.2289   3.178  0.00190 **
## AdvLunge_Exec     0.3856     0.1590   2.425  0.01684 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.892 on 118 degrees of freedom
## Multiple R-squared:  0.4084, Adjusted R-squared:  0.3784
## F-statistic: 13.58 on 6 and 118 DF,  p-value: 1.131e-11
```

Additional 2-time Step Regression

```r
#Two-Way Interaction
model_bic <- step(lm.obj, direction="both",trace=0, steps=1000, k=log(n))
model_full <- lm(netoutcome~(RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec + LungeB_Exec + JumpLunge_Exe

step(model_bic, direction="both", scope=list(lower=model_bic, upper=model_full), trace=0, steps=1000, k
step(model_bic, direction="both", scope=list(lower=model_bic, upper=model_full), trace=0, steps=1000, k

model_step <- lm(netoutcome~RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec + LungeB_Exec + JumpLunge_Exec
BIC2_formula <- netoutcome~RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec + LungeB_Exec + JumpLunge_Exec
```

```
##
## Call:
## lm(formula = netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec +
##     LungeB_Exec + JumpLunge_Exec + AdvLunge_Exec + CaughtT_Exec:LungeB_Exec +
##     RiposteDef_Exec:AdvLunge_Exec, data = clean)
##
## Coefficients:
```

```
##                   (Intercept)                RiposteDef_Exec
##                        0.4494                        -0.2403
##                  CaughtB_Exec                   CaughtT_Exec
##                       -1.6588                        -1.8979
##                   LungeB_Exec                 JumpLunge_Exec
##                        1.4590                         0.7450
##                 AdvLunge_Exec       CaughtT_Exec:LungeB_Exec
##                        0.5767                        -1.0237
## RiposteDef_Exec:AdvLunge_Exec
##                       -0.1792
##
##
## Call:
## lm(formula = netoutcome ~ RiposteDef_Exec + CaughtB_Exec + CaughtT_Exec +
##     LungeB_Exec + JumpLunge_Exec + AdvLunge_Exec, data = clean)
##
## Coefficients:
##     (Intercept)  RiposteDef_Exec     CaughtB_Exec     CaughtT_Exec
##          0.6711          -0.3488          -1.8583          -3.4056
##     LungeB_Exec   JumpLunge_Exec    AdvLunge_Exec
##          1.2122           0.7274           0.3856
```

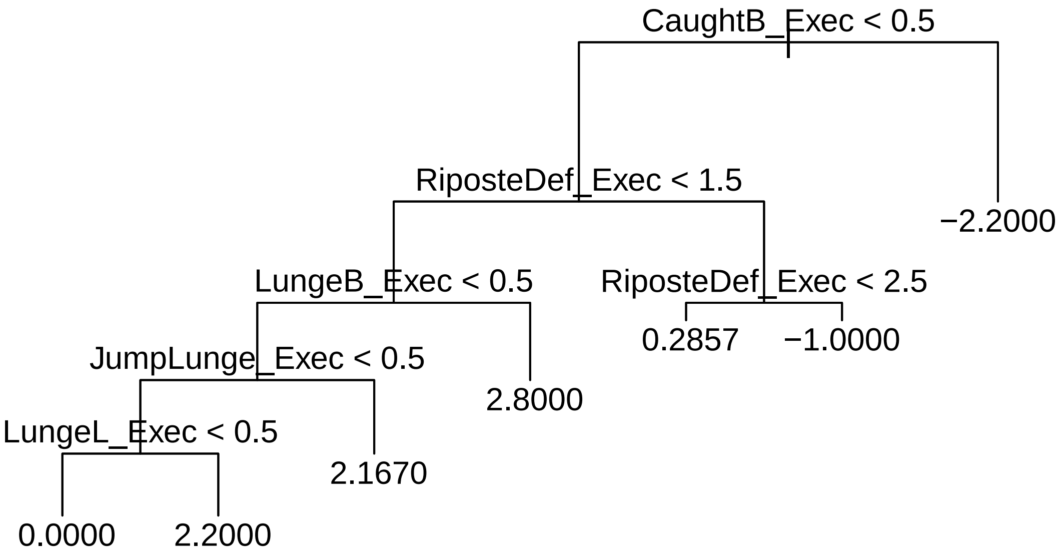
Perform Cross Validations

```
# Leave one out
n <- nrow(clean)
library(asbio)
aic_loocv <- press(AIC_step)/n
bicS_loocv <- press(BIC_step)/n
bic_loocv <- press(bic_model)/n
adj_loocv <- press(adjr2_model)/n
cp_loocv <- press(cp_model)/n
bicTwo_loocv <- press(model_step)/n
mlr_loocv <- press(lm.obj)/n

cat("AIC (step) LOOCV score: ", aic_loocv, "\n")
cat("BIC (step) LOOCV score: ", bicS_loocv, "\n")
cat("BIC LOOCV score: ", bic_loocv, "\n")
cat("AdjR2 LOOCV score: ", adj_loocv, "\n")
cat("CP LOOCV score: ", cp_loocv, "\n")
cat("BIC (2-time step) LOOCV score: ", bicTwo_loocv, "\n")
cat("Full LOOCV score: ", mlr_loocv, "\n")
cat("-------", "-------\n")

#K-Fold
set.seed(1)
mydata <- clean #load your data set
n <- nrow(clean) #sample size
K <- 5 #5-fold CV as an example
n.fold <- floor(n/K) #size of each fold, rounded down to the nearest integer
n.shuffle <- sample(1:n, n, replace=FALSE) #shuffle the n indexes
index.fold <- list()
for(i in 1:K) {
  if(i<K) {
    index.fold[[i]] <- n.shuffle[((i-1)*n.fold+1):(i*n.fold)]
  } else {
    index.fold[[i]] <- n.shuffle[((K-1)*n.fold+1):n]
  }
}

CV.scoreBIC <- 0
CV.scoreAdjR <- 0
CV.scoreCP <- 0
CV.scoreAIC <- 0
CV.scoreBICS <- 0
CV.scoreBIC2 <- 0
CV.scoreFull <- 0
for(i in 1:K) {
  #fit the full model based on the data excluding the ith fold
  BICfit <- lm(bic_formula, data=mydata[-index.fold[[i]],])
  AdjRfit <- lm(adjr2_formula, data=mydata[-index.fold[[i]],])
  CpMallowfit <- lm(cp_formula, data=mydata[-index.fold[[i]],])
  AICfit <- lm(aic_formula, data=mydata[-index.fold[[i]],])
  BICSfit <- lm(BICS_formula, data=mydata[-index.fold[[i]],])
  BIC2fit <- lm(BIC2_formula, data=mydata[-index.fold[[i]],])
  fullfit <- lm(netoutcome~., data=mydata[-index.fold[[i]],])

  #make prediction on each observation in the ith fold
  predBIC <- predict(BICfit,mydata[index.fold[[i]],])
  predAdjR <- predict(AdjRfit,mydata[index.fold[[i]],])
  predCP <- predict(CpMallowfit,mydata[index.fold[[i]],])
  predAIC <- predict(AICfit,mydata[index.fold[[i]],])
  predBICS <- predict(BICSfit,mydata[index.fold[[i]],])
  predBIC2 <- predict(BIC2fit,mydata[index.fold[[i]],])
  predFull <- predict(fullfit,mydata[index.fold[[i]],])

  #compute average squared error for the ith fold
  CV.scoreBIC <- CV.scoreBIC+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predBIC)^2)
  CV.scoreAdjR <- CV.scoreAdjR+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predAdjR)^2)
  CV.scoreCP <- CV.scoreCP+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predCP)^2)
  CV.scoreAIC <- CV.scoreAIC+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predAIC)^2)
  CV.scoreBICS <- CV.scoreBICS+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predBICS)^2)
  CV.scoreBIC2 <- CV.scoreBIC2+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predBIC2)^2)
  CV.scoreFull <- CV.scoreFull+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predFull)^2)
}
cat("AIC (step) 5-fold score: ", CV.scoreAIC, "\n")
cat("BIC (step) 5-fold score: ", CV.scoreBICS, "\n")
cat("BIC 5-fold score: ", CV.scoreBIC, "\n")
cat("AdjR 5-fold score: ", CV.scoreAdjR, "\n")
cat("CP 5-fold score: ", CV.scoreCP, "\n")
cat("BIC (2-time step) 5-fold score: ", CV.scoreBIC2, "\n")
cat("Full 5-fold score: ", CV.scoreFull, "\n")
```

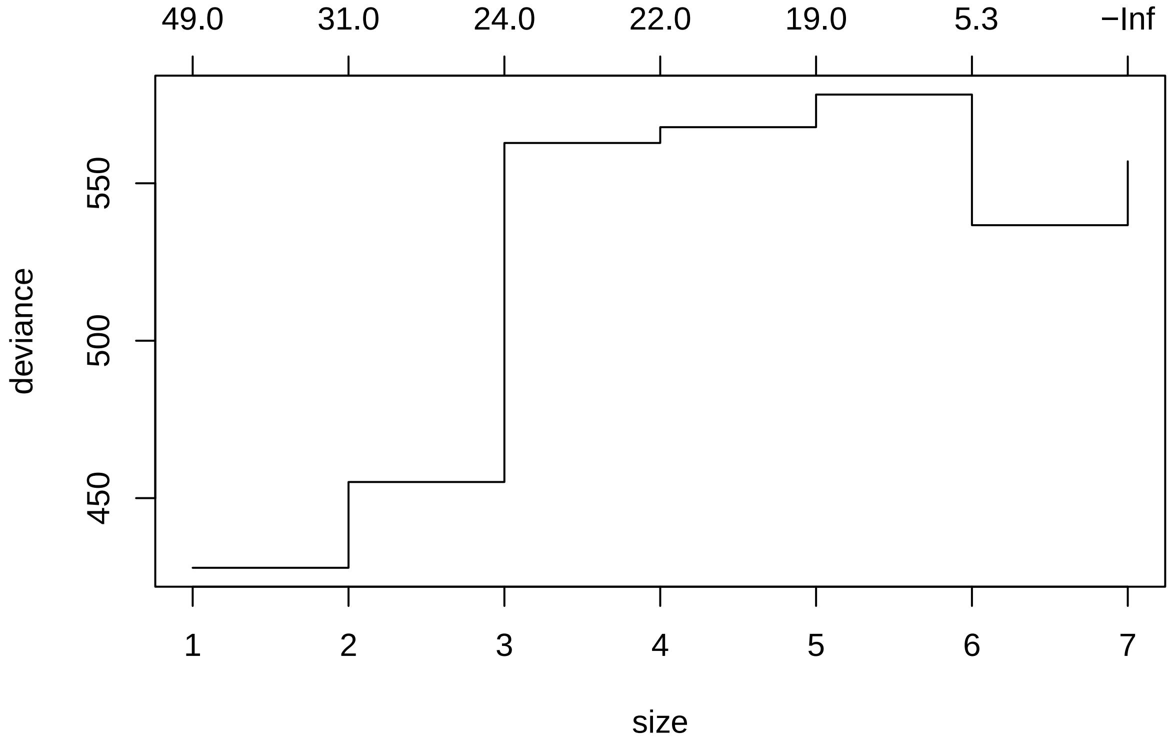
```
## AIC (step) LOOCV score:  3.962502
## BIC (step) LOOCV score:  3.962502
## BIC LOOCV score:  3.962502
## AdjR2 LOOCV score:  4.106143
## CP LOOCV score:  3.962502
## BIC (2-time step) LOOCV score:  3.962502
## Full LOOCV score:  4.59915
## -------- --------
## AIC (step) 5-fold score:  3.890077
## BIC (step) 5-fold score:  3.890077
## BIC 5-fold score:  3.890077
## AdjR 5-fold score:  4.116329
## CP 5-fold score:  3.890077
## BIC (2-time step) 5-fold score:  3.890077
## Full 5-fold score:  5.164047
```

Tree-Based Methods

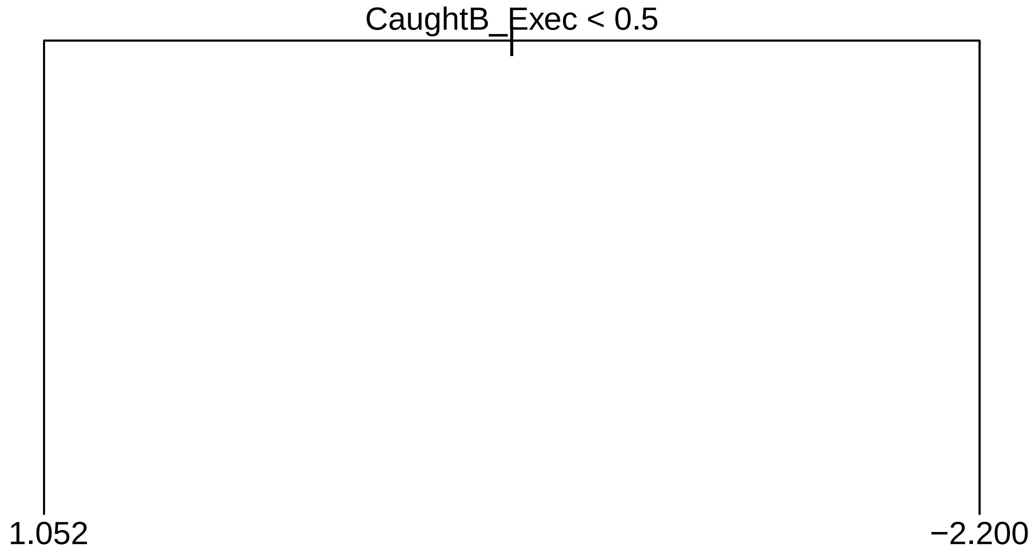
```
set.seed(1)
n <- nrow(clean)
train_idx <- sample(1:n, 63, replace=FALSE)
train_t <- clean[train_idx,]
test_t <- clean[-train_idx, ]

library(tree)
names(train_t) <- gsub("^62", "X62", names(train_t))
tree_train <- tree(train_t$netoutcome ~ ., data = train_t)
plot(tree_train)
text(tree_train)
```



```
result = cv.tree(tree_train,K=10,FUN=prune.tree)
plot(result)
```

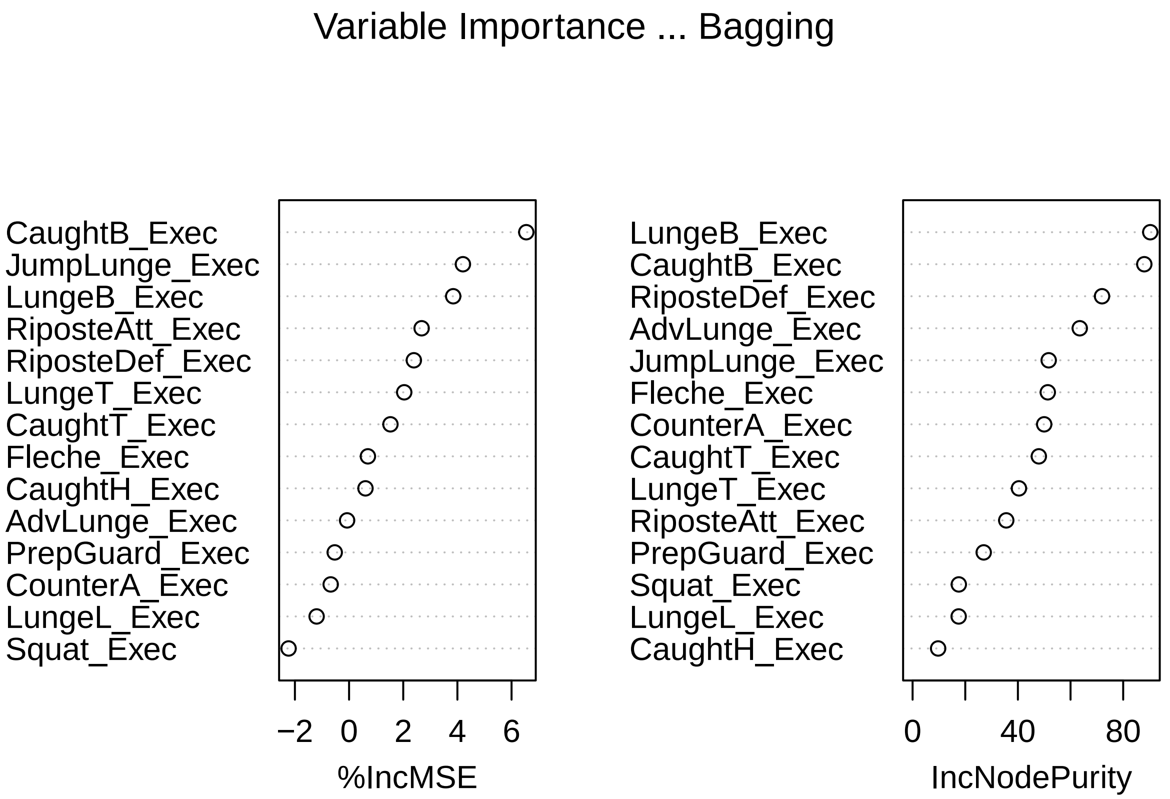


```
besttree <- prune.tree(tree_train, best = 2)
plot(besttree)
text(besttree)
```

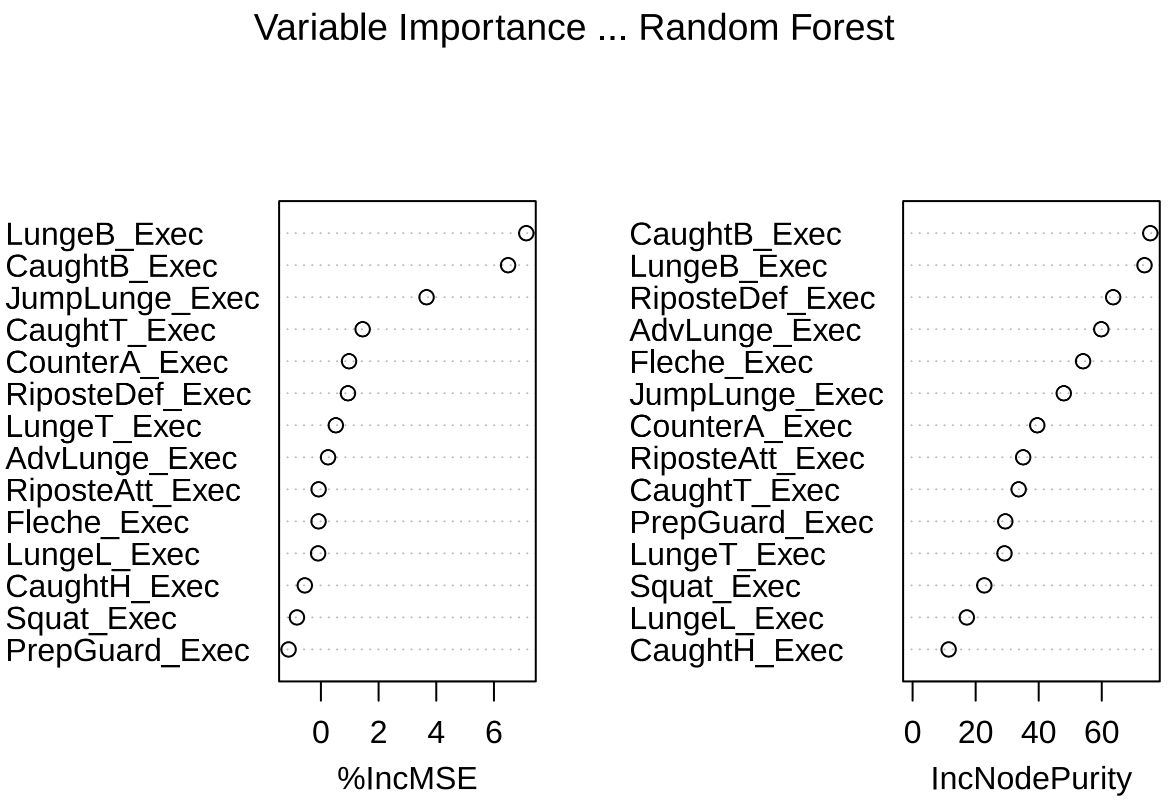


#Bagging and Random Forest with Cross Validation

```
#bagging and random forest
library(randomForest)
clean_tree <- clean[,-7]
bag_model <- randomForest(netoutcome~., data=clean_tree, ntree=100, mtry=15, importance=TRUE)
RF_model <- randomForest(netoutcome~., data=clean_tree, ntree=100, mtry=4, importance=TRUE)
ntree<- nrow(clean_tree)

varImpPlot(bag_model, main = "Variable Importance - Bagging")
```



```r
varImpPlot(RF_model, main = "Variable Importance - Random Forest")
```

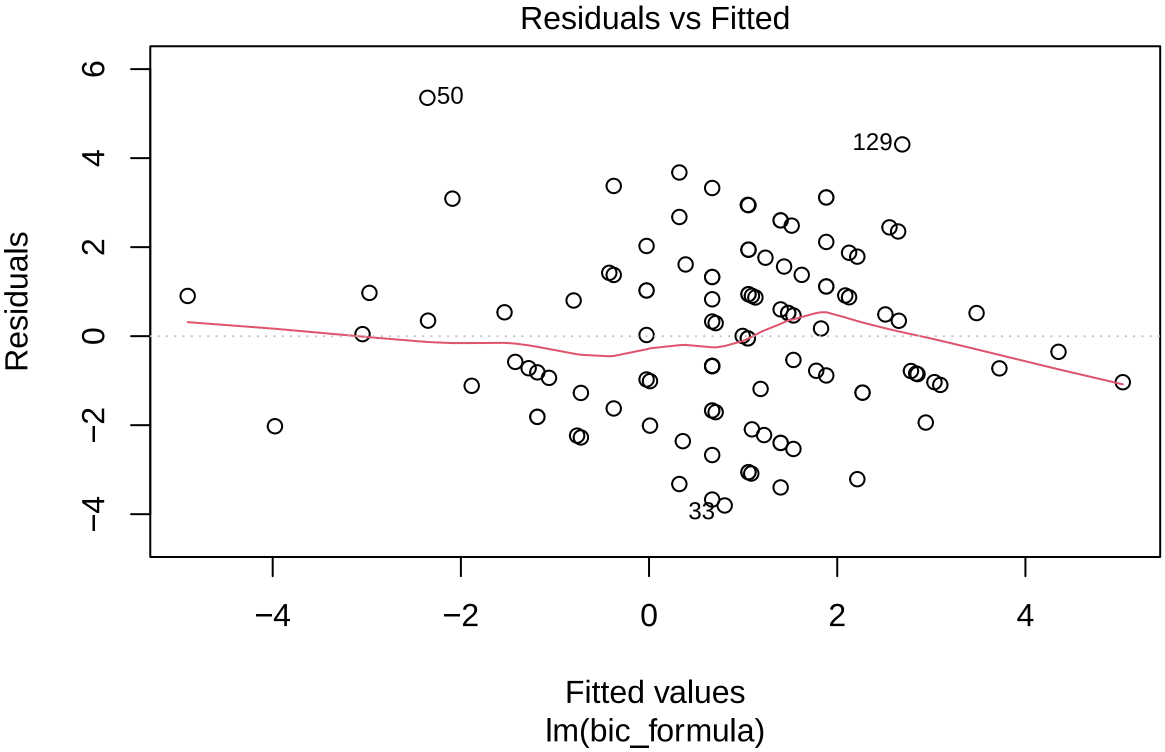


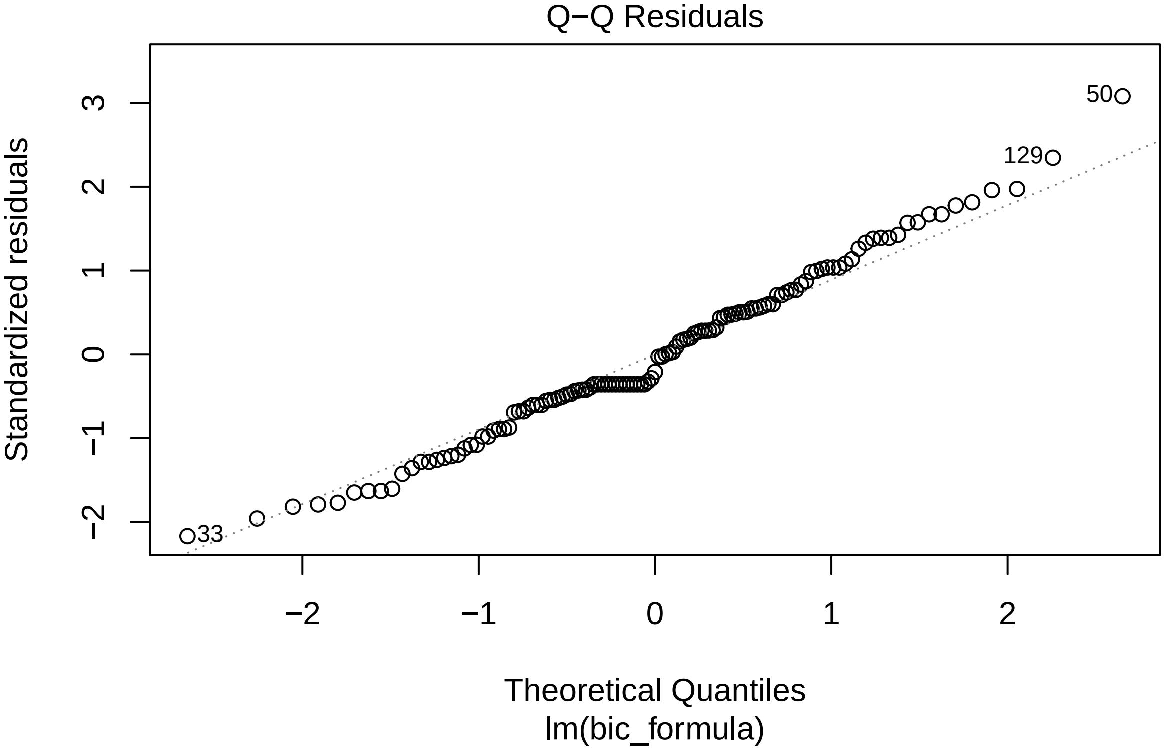
```r
#LOOCV not applicable

set.seed(1)
#K-Fold
mydata <- clean_tree #load your data set
n <- nrow(clean) #sample size
K <- 5 #5-fold CV as an example
n.fold <- floor(n/K) #size of each fold, rounded down to the nearest integer
n.shuffle <- sample(1:n, n, replace=FALSE) #shuffle the n indexes
index.fold <- list()
for(i in 1:K) {
  if(i<K) {
    index.fold[[i]] <- n.shuffle[((i-1)*n.fold+1):(i*n.fold)]
  } else {
    index.fold[[i]] <- n.shuffle[((K-1)*n.fold+1):n]
  }
}


CV.scoreBag <- 0
CV.scoreRF <- 0
CV.scoreTree <- 0
for(i in 1:K) {
  fit.bag <- randomForest(netoutcome~.,data=mydata[-index.fold[[i]],],ntree=100, mtry=15)
  fit.RF <- randomForest(netoutcome~., data=mydata[-index.fold[[i]],], ntree=100, mtry=sqrt(15))
  fit.tree <- tree(netoutcome ~ ., data=mydata[-index.fold[[i]],])
  predBag <- predict(fit.bag, mydata[index.fold[[i]],])
  predRF <- predict(fit.RF, mydata[index.fold[[i]],])
  predTree <- predict(fit.tree, mydata[index.fold[[i]],])
  CV.scoreBag <- CV.scoreBag+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predBag)^2)
  CV.scoreRF <- CV.scoreRF+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predRF)^2)
  CV.scoreTree <- CV.scoreTree+(1/n)*sum((clean$netoutcome[index.fold[[i]]]-predTree)^2)
}


cat("Bagging 5-fold score: ", CV.scoreBag, "\n")
cat("Random Forest 5-fold score: ", CV.scoreRF, "\n")
cat("Single Tree 5-fold score: ", CV.scoreTree)
```

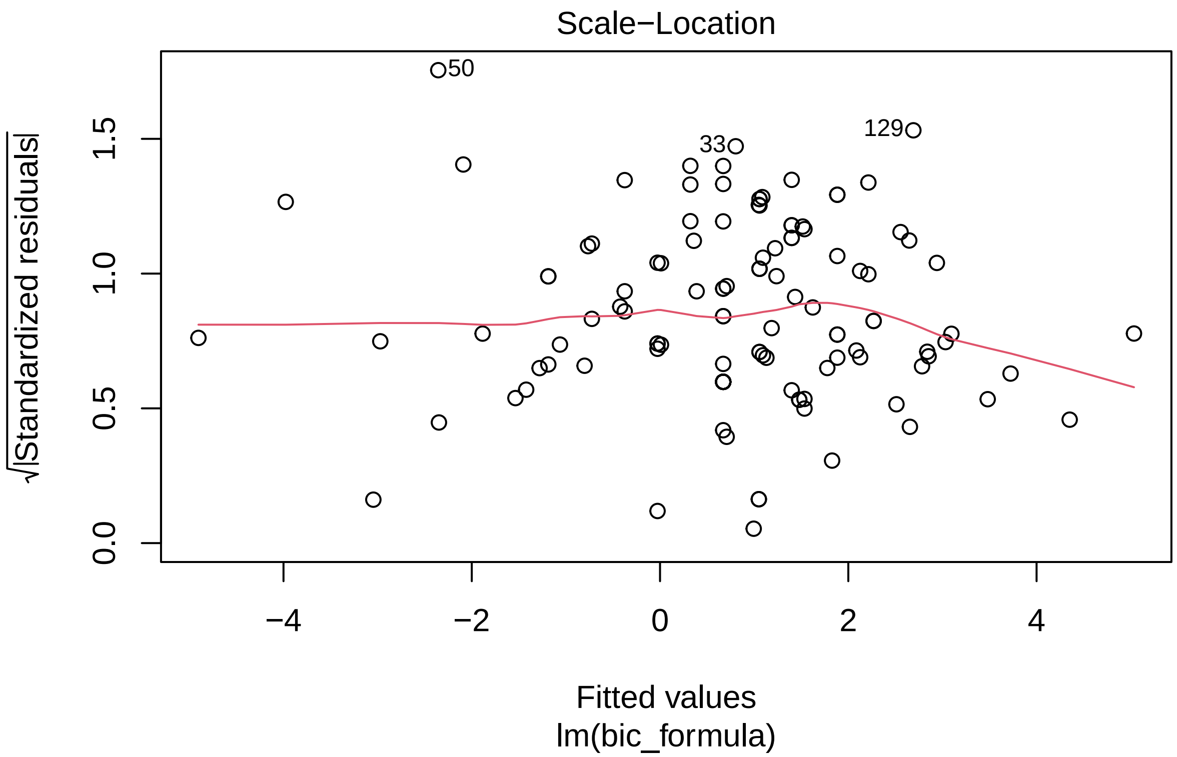
```
## Bagging 5-fold score:  5.19934
## Random Forest 5-fold score:  5.031806
## Single Tree 5-fold score:  6.178653
```

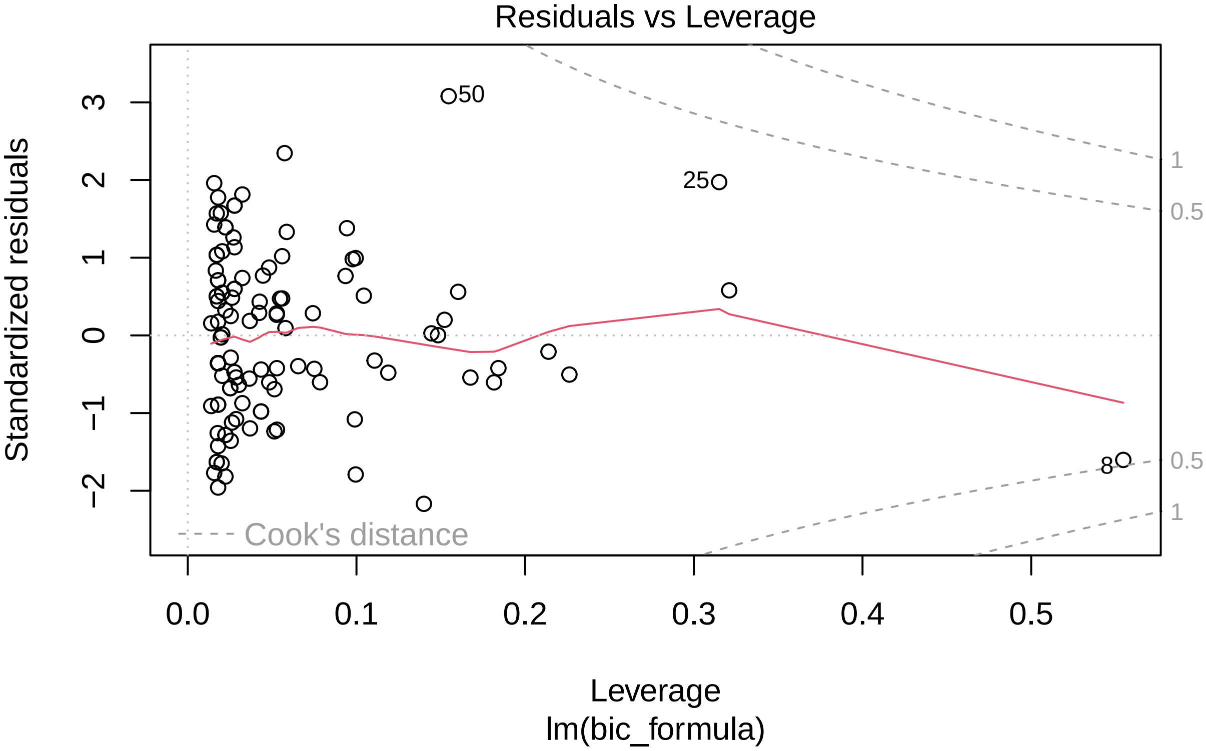
Diagnostics

```r
summary(bic_model)
plot(bic_model)
```


Residuals vs Fitted
Residuals
6
4
2
0
−2
−4
50
129
33
−4
−2
0
2
4
Fitted values
lm(bic_formula)


Q−Q Residuals
Standardized residuals
3
2
1
0
−1
−2
50
129
33
−2
−1
0
1
2
Theoretical Quantiles
lm(bic_formula)





```
##
## Call:
## lm(formula = bic_formula, data = clean)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.8040 -1.0954 -0.3519  1.1167  5.3559
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.6711     0.2538   2.644  0.00930 **
## RiposteDef_Exec  -0.3488     0.1128  -3.092  0.00248 **
## CaughtB_Exec     -1.8583     0.3676  -5.055 1.59e-06 ***
## CaughtT_Exec     -3.4056     0.7076  -4.813 4.45e-06 ***
## LungeB_Exec       1.2122     0.2708   4.477 1.76e-05 ***
## JumpLunge_Exec    0.7274     0.2289   3.178  0.00190 **
## AdvLunge_Exec     0.3856     0.1590   2.425  0.01684 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.892 on 118 degrees of freedom
## Multiple R-squared:  0.4084, Adjusted R-squared:  0.3784
## F-statistic: 13.58 on 6 and 118 DF,  p-value: 1.131e-11
```