Choosing Optimal Multiple Linear Regression Models Using Best Subsets in R

James Gao

10/22/2017

Introduction

Throughout the course, we've looked at manipulating NBA data in the form of data frames and looking at many different factors to draw conclusions between different variables. However, we've been focusing on learning the fundamentals, and we haven't had many chances to use what we've learned to perform statistical analyses. While we've created graphs to describe relationships between different variables, we haven't created functions to automate certain tasks, which is what R is commonly used for. In this post, I'll be creating functions to predict certain variables in the NBA. By the end of the post, I'll have made functions that use a method called best subsets in order to find the best multiple regression model to predict certain variables. I'll be using data that we've been using throughout the course from the NBA 2016-2017 Regular Season. The purpose of this post is to show that using R, we can automate certain tasks that would take much longer in an application like Minitab.

Organization

This post will begin by preparing the NBA data for analysis. Then, we'll write functions to automate the process of choosing which variables to include in our model. Then we'll use the functions created throughout the post to create multiple regression models.

Preparing the Data

The data I'll be using in this post will be from HW03, found in the /data folder. The data is from the NBA 2016-2017 Regular Season. If you recall from HW03, we started with two datasets named stats and roster. Similar to in HW03, I'll be merging the two data frames, and creating another data frame that groups the statistics by team. However, while in HW03 we excluded many variables in the teams data frame, I'll be including more of the excluded statistics. Also, the teams data frame will include the mean of the variables, while in HW03 we took the sum. Additionally, I'll be adding some variables from http://www.basketball-reference.com to the nba2017-teams.csv: wins, losses, and win percentage.

Importing the Data

```
#Loading libraries
library(dplyr)
library(knitr)
library(kableExtra)
```

```
roster <- read.csv(file = "data/nba2017-roster.csv")
stats <- read.csv(file = "data/nba2017-stats.csv")
```

Merging the Two Data Sets

First, let's merge the two data sets like we did in HW03.

```
stats <- mutate(stats, missed_fg = points3_atts - points3_made + points2_atts - points2_made)
stats <- mutate(stats, missed_ft = points1_atts - points1_made)
stats <- mutate(stats, points = points3_made * 3 + points2_made * 2 + points1_made)
stats <- mutate(stats, rebounds = off_rebounds + def_rebounds)
stats <- mutate(stats, efficiency =
                  (points + rebounds + assists + steals +
                     blocks - missed_fg - missed_ft - turnovers) / games_played)
player <- left_join(roster, stats)
player$salary <- round(player$salary / 1000000, 2)
```

We now have *player*, a data frame with 34 variables.

```
str(player)
```

```
## 'data.frame':    441 obs. of  34 variables:
##  $ player          : Factor w/ 441 levels "A.J. Hammons",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ team            : Factor w/ 30 levels "ATL","BOS","BRK",..: 7 12 22 18 2 12 25 13 24 29 ...
##  $ position        : Factor w/ 5 levels "C","PF","PG",..: 1 3 4 2 1 1 4 4 1 5 ...
##  $ height          : int  84 72 81 82 82 82 81 78 80 78 ...
##  $ weight          : int  260 161 220 237 245 289 220 220 260 214 ...
##  $ age             : int  24 32 21 25 30 32 26 34 24 25 ...
##  $ experience      : int  0 8 2 2 9 12 6 7 1 5 ...
##  $ salary          : num  0.65 2.7 4.35 2.02 26.54 ...
##  $ games_played    : int  22 65 80 18 68 66 61 30 47 42 ...
##  $ minutes         : int  163 894 2298 135 2193 931 1773 308 708 653 ...
##  $ field_goals_made: int  17 121 393 23 379 235 183 30 138 99 ...
##  $ field_goals_atts: int  42 300 865 54 801 471 466 80 267 248 ...
##  $ field_goals_perc: num  0.405 0.403 0.454 0.426 0.473 0.499 0.393 0.375 0.517 0.399 ...
##  $ points3_made    : int  5 48 77 3 86 0 70 14 0 25 ...
##  $ points3_atts    : int  10 128 267 15 242 1 212 44 1 76 ...
##  $ points3_perc    : num  0.5 0.375 0.288 0.2 0.355 0 0.33 0.318 0 0.329 ...
##  $ points2_made    : int  12 73 316 20 293 235 113 16 138 74 ...
##  $ points2_atts    : int  32 172 598 39 559 470 254 36 266 172 ...
##  $ points2_perc    : num  0.375 0.424 0.528 0.513 0.524 0.5 0.445 0.444 0.519 0.43 ...
##  $ points1_made    : int  9 32 156 14 108 65 96 12 70 60 ...
##  $ points1_atts    : int  20 40 217 19 135 85 136 16 112 78 ...
##  $ points1_perc    : num  0.45 0.8 0.719 0.737 0.8 0.765 0.706 0.75 0.625 0.769 ...
##  $ off_rebounds    : int  8 18 116 9 95 75 77 3 94 17 ...
##  $ def_rebounds    : int  28 51 289 24 369 203 374 21 198 103 ...
##  $ assists         : int  4 125 150 7 337 57 99 11 23 30 ...
##  $ steals          : int  1 25 64 8 52 19 60 3 27 18 ...
##  $ blocks          : int  13 9 40 7 87 16 44 0 32 5 ...
##  $ turnovers       : int  10 66 89 8 116 33 94 7 37 35 ...
##  $ fouls           : int  21 93 172 32 138 125 102 35 125 50 ...
##  $ missed_fg       : int  25 179 472 31 422 236 283 50 129 149 ...
##  $ missed_ft       : int  11 8 61 5 27 20 40 4 42 18 ...
##  $ points          : num  48 322 1019 63 952 ...
##  $ rebounds        : int  36 69 405 33 464 278 451 24 292 120 ...
##  $ efficiency      : num  2.55 4.57 13.2 4.11 19.51 ...
```

Grouping By Team

Now, let's also group the data frame by team.

```
teams <- select(player, team, height, weight,
                age, games_played, minutes, experience,
                salary, field_goals_perc, points3 = points3_made,
                points2 = points2_made, free_throws = points1_made,
                points, off_rebounds, def_rebounds, rebounds, assists,
                steals, blocks, turnovers, fouls, efficiency)
teams$salary <- round(teams$salary / 1000000, digits = 2)
teams <- aggregate(teams[, attributes(teams)$names[-1]],
                   by = list(as.character(teams$team)), FUN = mean)
attributes(teams)$names[1] <- 'team'
```

Adding Wins, Losses, and Win Percentage

Using data from http://www.basketball-reference.com, I created a csv file with the number of wins, losses, and win percentage for each team in the 2016-2017 regular season.

Importing the data:

```
standings <- read.csv(file = "data/nba2017-standings.csv", stringsAsFactors = FALSE)
standings
```

```
##    team wins losses win_percentage
## 1   ATL   43     39         0.5244
## 2   BOS   53     29         0.6463
## 3   BRK   20     62         0.2439
## 4   CHI   41     41         0.5000
## 5   CHO   36     46         0.4390
## 6   CLE   51     31         0.6220
## 7   DAL   33     49         0.4024
## 8   DEN   40     42         0.4878
## 9   DET   37     45         0.4512
## 10  GSW   67     15         0.8171
## 11  HOU   55     27         0.6707
## 12  IND   42     40         0.5122
## 13  LAC   51     31         0.6220
## 14  LAL   26     56         0.3171
## 15  MEM   43     39         0.5244
## 16  MIA   41     41         0.5000
## 17  MIL   42     40         0.5122
## 18  MIN   31     51         0.3780
## 19  NOP   34     48         0.4146
## 20  NYK   31     51         0.3780
## 21  OKC   47     35         0.5732
## 22  ORL   29     53         0.3537
## 23  PHI   28     54         0.3415
## 24  PHO   24     58         0.2927
## 25  POR   41     41         0.5000
## 26  SAC   32     50         0.3902
## 27  SAS   61     21         0.7439
## 28  TOR   51     31         0.6220
## 29  UTA   51     31         0.6220
## 30  WAS   49     33         0.5976
```

Now, let's add these three stats to our *teams* data frame.

```
teams <- left_join(standings, teams)
```

Removing Non-Quantitative Variables

For the sake of this post, we'll only be looking at how quantitative variables predict other quantitative variables. For that reason, I'll be removing the categorical variables from our *player* data frame.

```
player$team = NULL
player$position = NULL
player$player = NULL
```

With our data frames set up, we can begin analyzing our data and creating models.

Automating the Process of Creating a Model

Going Step by Step

First, we have to decide which steps we need to take to find a model to predict a variable. Our goal is that given a certain variable (such as salary), we want to be able to find the multiple linear regression that best predicts that variable.

Finding Correlations Between Variables

Let's start by trying to create a model to predict a player's salary. We'll start by talking about how to write code that quickly computes correlations between many variables at once. Let's compare the correlation between each variable and salary. As an example, we'll find the correlation between experience and salary. We use the function cor():

```
cor(player$salary, player$experience)
```

```
## [1] 0.4511793
```

The R-squared value shows that there is a decent correlation between salary and experience, and that we might want to consider having experience in our multiple regression model. Let's look at a few more:

Salary vs Age:

```
cor(player$salary, player$age)
```

```
## [1] 0.3138329
```

Salary vs Efficiency:

```
cor(player$salary, player$efficiency)
```

```
## [1] 0.6556547
```

Already, we can see how manually finding the correlation of all other 30 variables to salary would be very tiresome. Thankfully, we have for loops

to do this faster:

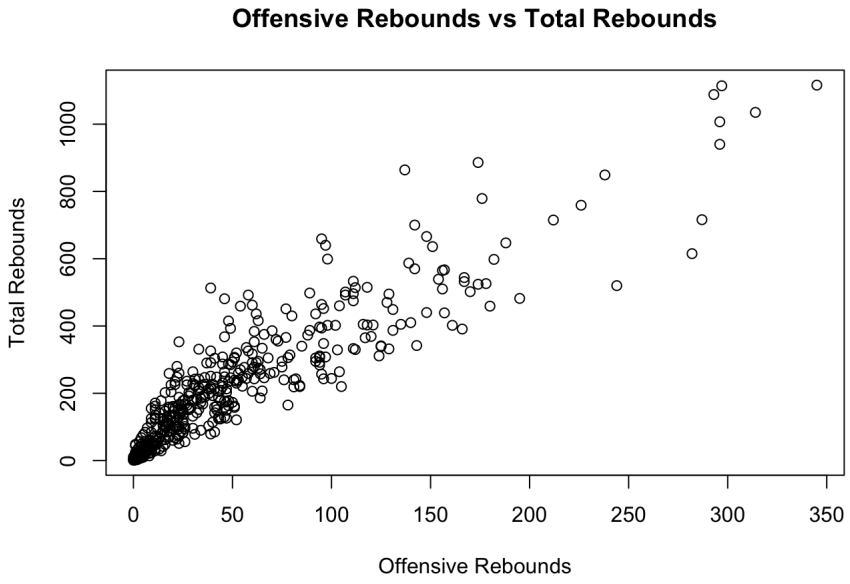
```
#Create a character vector named columns that has all of the variables names except salary
columns <- colnames(player)[-5]
#Create a vector to store the r-squared values of each variable vs salary
correlations <- rep(0, length(columns) - 2)
for (x in 1:length(columns)){
  correlations[x] = cor(player[, 'salary'], player[, columns[x]], use = "pairwise.complete.obs")
}
#Creates a data frame sorted in descending order by r-squared
arrange(data.frame(columns, correlations), desc(correlations))
```

```
##             columns correlations
## 1        efficiency   0.65565473
## 2            points   0.63672961
## 3  field_goals_made   0.63139638
## 4  field_goals_atts   0.62608188
## 5      points1_atts   0.62176753
## 6      points2_atts   0.61950078
## 7      points1_made   0.61059587
## 8      points2_made   0.60967427
## 9         missed_fg   0.60851558
## 10     def_rebounds   0.57029924
## 11        turnovers   0.56938425
## 12          minutes   0.56642142
## 13         rebounds   0.53157150
## 14        missed_ft   0.52534344
## 15          assists   0.48755345
## 16           steals   0.47485642
## 17       experience   0.45117931
## 18     points3_atts   0.41579712
## 19     points3_made   0.41158589
## 20            fouls   0.40910409
## 21     off_rebounds   0.36592031
## 22     games_played   0.36465468
## 23           blocks   0.32916299
## 24              age   0.31383287
## 25     points1_perc   0.18626882
## 26           weight   0.15210915
## 27 field_goals_perc   0.13022583
## 28     points2_perc   0.08466072
## 29     points3_perc   0.07957870
## 30           height   0.06568800
```

Now we have a table of the correlations of every variable in our player data frame compared to salary. However, now we need to make a decision: which variables do we use in our model? Do we choose the top two variables with the highest correlation? Top three? There's a lot of flaws with just arbitrarily choosing variables.

For one, we need to remove variables that are heavily correlated with each other. This is called multicollinearity. When two of the predictor variables are heavily correlated, they are basically measuring the same variance as the other. For example, imagine that the three variables with the highest correlation with salary are offensive rebounds, defensive rebounds, and total rebounds. Creating a model with offensive rebounds, defensive rebounds, and total rebounds would be a terrible idea, regardless of how high the correlations are. Even if the correlation for all three variables were above 0.9, it'd still be a bad idea to make a model with those three variables because total rebounds explains much of the same variation in salary as offensive rebounds and defensive rebounds. Here's a scatterplot of offensive rebounds vs total rebounds to illustrate how heavily correlated they are:

```
plot(player$off_rebounds, player$rebounds,
     main = 'Offensive Rebounds vs Total Rebounds',
     ylab = 'Total Rebounds', xlab = 'Offensive Rebounds')
```



You can see from the scatterplot that they have a strong positive correlation. In fact, let's calculate it:

```
cor(player$off_rebounds, player$rebounds)
```

```
## [1] 0.9083841
```

Let's also look at offensive rebounds vs defensive rebounds:

```
cor(player$off_rebounds, player$def_rebounds)
```

```
## [1] 0.8273467
```

And defensive rebounds vs total rebounds:

```
cor(player$rebounds, player$def_rebounds)
```

```
## [1] 0.9864125
```

They are all heavily correlated with one another. As a result, you'd never want to use two of these variables at the same time in your model.

Criteria to Find the Best Model to Predict a Variable

We need criteria to rate a model. Let's say we decide to use linear regression to predict salary based on experience:

```
model <- lm(formula = player$salary ~ player$efficiency)
summary(model)
```

```
##
## Call:
## lm(formula = player$salary ~ player$efficiency)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.5925  -3.0107  -0.3732   2.5876  18.9270
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -0.70820    0.44675  -1.585    0.114
## player$efficiency  0.68020    0.03739  18.194   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.968 on 439 degrees of freedom
## Multiple R-squared:  0.4299, Adjusted R-squared:  0.4286
## F-statistic:   331 on 1 and 439 DF,  p-value: < 2.2e-16
```

If you look at the output, you'll see the R-squared value, but you'll also see something called the Adjusted R-squared. We'll be using Adjusted R-Squared to rate our models. Adjusted R-squared is similar to R-squared, but it includes a method of checking whether adding more variables into your model is actually helping your model. The distinction between R-squared and adjusted R-squared is that the R-squared will always increase when you add more variables into your model. However, adjusted R-squared determines whether the variables in your model actually improves the model more than would be expected by chance. Adding variables will actually sometimes decrease your adjusted R-squared even though your R-squared increases. The adjusted R-squared helps account for the multicollinearity that I talked about earlier.

Best Subset Regression

Now that we've decided how we'll be rating our models, we can start finding the best models to predict salary using a method known as best subset regression. In this method, we run every possible multiple regression to see which model is the best. For example, if we were only considering three variables: age, experience, and weight, we would run 8 regressions with the following regressions:

- No predictor
- Age
- Experience
- Weight
- Age and Experience
- Experience and Weight
- Age and Weight
- Age, Experience, and Weight

Then, we would consider the adjusted R-squared for each model and decide on the best one.

Using for loops and recursion, we can create a way to automate the process of calculating best subsets. First, I'm going to create a character vector named *predictors* containing age, weight, and experience. Then, I'm going to make a recursive function named *subset* that will return a list every combination of boolean vectors.

```
predictors <- c('age', 'weight', 'experience')
#subset is a recursive function that returns a list
#of every combination of boolean vectors of length x
subset <- function(x){
  if (x == 1){
    return (list(TRUE, FALSE))
  }
  else{
    z = NULL
    for (y in subset(x - 1)){
      z <- c(list(c(TRUE, y), c(FALSE, y)), z)
    }
    return (z)
  }
}
subsets <- subset(length(predictors))
subsets
```

```
## [[1]]
## [1]  TRUE FALSE  TRUE
##
## [[2]]
## [1] FALSE FALSE  TRUE
##
## [[3]]
## [1] TRUE TRUE TRUE
##
## [[4]]
## [1] FALSE  TRUE  TRUE
##
## [[5]]
## [1]  TRUE FALSE FALSE
##
## [[6]]
## [1] FALSE FALSE FALSE
##
## [[7]]
## [1]  TRUE  TRUE FALSE
##
## [[8]]
## [1] FALSE  TRUE FALSE
```

With these boolean vectors, we can use boolean subsetting on *predictors* to select every combination of *predictors* for a model:

```
var_subsets <- list()
#This takes the list of boolean vectors and uses them to subset our vector of predictors,
#returning a list of the possibilities for the predictors in our model.
for (x in subsets){
  var_subsets <- c(var_subsets, list(predictors[x]))
}
var_subsets
```

```
## [[1]]
## [1] "age"        "experience"
##
## [[2]]
## [1] "experience"
##
## [[3]]
## [1] "age"        "weight"     "experience"
##
## [[4]]
## [1] "weight"     "experience"
##
## [[5]]
## [1] "age"
##
## [[6]]
## character(0)
##
## [[7]]
## [1] "age"    "weight"
##
## [[8]]
## [1] "weight"
```

Now we have a list named *var_subsets*, which contains every single possible combination of the variables in *predictors*. The next step is to iterate through *var_subsets*, run regressions for each of those variables, and then observe their adjusted R-squared values.

```
#This code chunk runs a regression for each of the different combinations of
#predictors and records their r-squared and adjusted r-squared in a data frame.
adjusted_r <- rep(0, length(var_subsets))
variables <- rep(0, length(var_subsets))
num_variables <- rep(0, length(var_subsets))
for (x in 1:length(var_subsets)){
  if (length(var_subsets[[x]]) > 0){
    formula <- 'salary ~'
    v <- paste(var_subsets[[x]], collapse = '+')
    formula <- paste(formula, v)
    variables[x] <- v
    num_variables[x] <- length(var_subsets[[x]])
    regression <- lm(formula, player)
    adjusted_r[x] <- round(summary(regression)[9][[1]], 4)
  }
}
#Creates a data frame and sorts it in descending order by adjusted r-squared.
bestsubsets <- arrange(data.frame(adjusted_r, num_variables, variables), desc(adjusted_r))
bestsubsets %>%
  #Using the knitr::kable package to make the table look better
  kable("html") %>%
  kable_styling()
```

adjusted_r	num_variables	variables
0.2543	3	age+weight+experience
0.2458	2	age+experience
0.2125	2	weight+experience
0.2017	1	experience
0.1125	2	age+weight
0.0964	1	age
0.0209	1	weight
0.0000	0	0

I've displayed all of the different possible regressions sorted by adjusted R-squared. With this, we can see which models are best at predicting salary.

Automating the Process

Now I'll be taking what I did in the previous example and creating a function to do it automatically. The function will take in a data frame, the variable to be predicted, and a list of potential predictors.

We've already done the entire process above, so all we need to do is put the same code into a function, and have the code apply to the parameters of our function. Our function *subset_regression* will return a list of the 10 models with the highest adjusted R-squared values.

```
#This code is just a combination of all the code chunks in the section above
subset_regression <- function(data, predicted, predictors, num_models){
  subset <- function(x){
    if (x == 1){
      return (list(TRUE, FALSE))
    }
    else{
      z = NULL
      for (y in subset(x - 1)){
        z <- c(list(c(TRUE, y), c(FALSE, y)), z)
      }
      return (z)
    }
  }
  subsets <- subset(length(predictors))
  var_subsets <- list()
  for (x in subsets){
    var_subsets <- c(var_subsets, list(predictors[x]))
  }
  adjusted_r <- rep(0, length(var_subsets))
  variables <- rep(0, length(var_subsets))
  num_variables <- rep(0, length(var_subsets))
  for (x in 1:length(var_subsets)){
    if (length(var_subsets[[x]]) > 0){
      formula <- paste(predicted, '~')
      v <- paste(var_subsets[[x]], collapse = ' + ')
      formula <- paste(formula, v)
      variables[x] <- v
      num_variables[x] <- length(var_subsets[[x]])
      regression <- lm(formula, data)
      adjusted_r[x] <- round(summary(regression)[9][[1]], 4)
    }
  }
  bestsubsets <- arrange(data.frame(
    Adjusted_R_Squared = adjusted_r,
    Number_Of_Variables = num_variables, Variables = variables),
    desc(adjusted_r))[1:num_models, ]
  return (bestsubsets)
}
```

Let's test our function by checking if our output of our function matches what we got before:

```
subset_regression(player, 'salary', c('age', 'weight', 'experience'), 8) %>%
  #Using the knitr::kable package to make the table look better
  kable("html") %>%
  kable_styling()
```

Adjusted_R_Squared	Number_Of_Variables	Variables
0.2543	3	age + weight + experience
0.2458	2	age + experience
0.2125	2	weight + experience
0.2017	1	experience
0.1125	2	age + weight
0.0964	1	age
0.0209	1	weight
0.0000	0	0

Now that we've confirmed that our function works, we can use it to to try and predict any of the variables in our data frames.

Creating Models

Now that we've made our functions for best subsets, let's start creating models using them.

Using Best Subsets to Predict Win Percentage

Let's try to create a model to predict a team's win percentage. Which variables should we consider as predictors? We could just check all the variables in our data frame, but now we need to start considering runtime. When we had just three predictors, we only had to run regressions for 8 scenarios. However, if we increase to four predictors, we'll find that there are 16 scenarios. In fact, the number of different possible models given n predictors is 2^n. Given the 24 variables in our data frame for teams, we would have to run 2^24 regressions. The amount of time and space required is too much for us to do, so we need to decrease the number of variables we're checking for. Instead of 24 variables, I'll be testing just 10, by choosing the 10 variables with the highest correlation with win percentage.

Earlier on, we created a data frame of the correlations between all variables and salary. Now let's create a function that does the same thing. Here's a function that takes in a list of variables and returns a certain number of variables with the highest correlation with a certain variable.

```
#predicted: variable to be predicted, ex: win percentage
#predictors: character vector of potential predictor variables
#num_predictors: number of variables to return
choose_variables <- function(data, predicted, predictors, num_predictors){
  correlations <- rep(0, length(predictors))
  for (x in 1:length(predictors)){
    correlations[x] = cor(data[, predicted], data[, predictors[x]], use = "pairwise.complete.obs")
  }
  return (slice(arrange(data.frame(predictors, correlations), desc(correlations)), 1:num_predictors))
}
```

With our new function *choose_variables*, we can find the top 10 variables with the highest correlation in respect to win percentage.

```
choose_variables(teams, 'win_percentage', colnames(teams)[c(-1, -4)], 10) %>%
  #Using the knitr::kable package to make the table look better
  kable("html") %>%
  kable_styling()
```

predictors	correlations
wins	1.0000000
points	0.5983109
points3	0.5833484
assists	0.5749866
experience	0.5295189
def_rebounds	0.5186870
age	0.4872973
field_goals_perc	0.4579624
games_played	0.4483168
steals	0.4473842

Looking at our output, it seems that we've forgotten to do one important thing. Wins has an r-squared of 1. Of course it will have an r-squared of 1, because number of wins and win percentage are directly related. We need to remove variables that are directly related to the variable that we are trying to predict.

```
top_ten <- choose_variables(teams, 'win_percentage', colnames(teams)[c(-1, -2, -4)], 10)
top_ten %>%
  #Using the knitr::kable package to make the table look better
  kable("html") %>%
  kable_styling()
```

predictors	correlations
points	0.5983109
points3	0.5833484
assists	0.5749866
experience	0.5295189
def_rebounds	0.5186870
age	0.4872973
field_goals_perc	0.4579624
games_played	0.4483168
steals	0.4473842
minutes	0.4330754

Now let's run our best subsets function on these 10 variables:

```
win_percentage_subsets <- subset_regression(teams, 'win_percentage', top_ten$predictors, 10)
win_percentage_subsets %>%
  #Using the knitr::kable package to make the table look better
  kable("html") %>%
  kable_styling()
```

Adjusted_R_Squared	Number_Of_Variables	Variables
0.5702	4	points3 + age + field_goals_perc + steals
0.5642	5	points3 + def_rebounds + age + field_goals_perc + steals
0.5630	5	points3 + age + field_goals_perc + games_played + steals
0.5603	5	points + points3 + age + field_goals_perc + steals
0.5597	4	points3 + experience + field_goals_perc + steals
0.5540	5	points3 + age + field_goals_perc + steals + minutes
0.5532	5	points3 + assists + age + field_goals_perc + steals
0.5527	5	points3 + experience + age + field_goals_perc + steals
0.5521	5	points3 + experience + def_rebounds + field_goals_perc + steals
0.5509	5	points3 + experience + field_goals_perc + games_played + steals

It looks like our model with the highest adjusted r-squared included points from three pointers, age, field goal percentage, and number of steals. Let's look at our regression equation.

```
regression <- lm(win_percentage~points3+age+field_goals_perc+steals, teams)
regression
```

```
##
## Call:
## lm(formula = win_percentage ~ points3 + age + field_goals_perc +
##     steals, data = teams)
##
## Coefficients:
##      (Intercept)           points3               age  field_goals_perc
##        -1.538284          0.005036          0.031645          1.139094
##           steals
##         0.011140
```

So our regression equation is:

y = 0.005a + 0.032b + 1.139c + 0.011d - 1.538

Where:
- y : win_percentage
- a : points3
- b : age
- c : field_goals_perc
- d : steals

Predicting a Player's Salary

Now that we've finished making all the functions we need, we can go back to the question we started with: what variables best predict a player's salary? Before, we only looked at age, experience, and weight. Using these new functions, we can quickly find the best model among all variables:

```
top_ten <- choose_variables(player, 'salary', colnames(player)[-5], 10)
salary_subsets <- subset_regression(player, 'salary', top_ten$predictors, 10)
salary_subsets %>%
  #Using the knitr::kable package to make the table look better
  kable("html") %>%
  kable_styling()
```

Adjusted_R_Squared	Number_Of_Variables	Variables
0.4726	7	efficiency + field_goals_made + points1_atts + points2_atts + points2_made + missed_fg + def_rebounds
0.4726	8	efficiency + field_goals_made + field_goals_atts + points1_atts + points2_atts + points2_made + missed_fg + def_rebounds
0.4726	7	efficiency + field_goals_atts + points1_atts + points2_atts + points2_made + missed_fg + def_rebounds
0.4726	7	efficiency + field_goals_made + field_goals_atts + points1_atts + points2_atts + points2_made + def_rebounds
0.4723	8	efficiency + points + field_goals_made + points1_atts + points2_atts + points1_made + missed_fg + def_rebounds
0.4723	9	efficiency + points + field_goals_made + field_goals_atts + points1_atts + points2_atts +

Adjusted_R_Squared	Number_Of_Variables	Variables
0.4723	8	points1_made + missed_fg + def_rebounds
0.4723	8	efficiency + points + field_goals_atts + points1_atts + points2_atts + points1_made + missed_fg + def_rebounds
0.4723	8	efficiency + field_goals_made + points1_atts + points2_atts + points1_made + points2_made + missed_fg + def_rebounds
0.4723	9	efficiency + points + field_goals_made + points1_atts + points2_atts + points1_made + points2_made + missed_fg + def_rebounds
0.4723	8	efficiency + points + points1_atts + points2_atts + points1_made + points2_made + missed_fg + def_rebounds

Given our input, we see that the model with the highest adjusted R-squared is :

```
regression <- lm(salary~efficiency + field_goals_made + points1_atts +
                   points2_atts + points2_made + missed_fg + def_rebounds, player)
regression
```

```
##
## Call:
## lm(formula = salary ~ efficiency + field_goals_made + points1_atts +
##     points2_atts + points2_made + missed_fg + def_rebounds, data = player)
##
## Coefficients:
##      (Intercept)        efficiency  field_goals_made      points1_atts
##         -0.380065           0.383046           0.050805           0.007304
##     points2_atts      points2_made         missed_fg      def_rebounds
##         0.060121          -0.138457          -0.029909           0.009351
```

And our regression equation is:

y = 0.383a + 0.051b + 0.007c + 0.060d - 0.138e - 0.030f + 0.009g - 0.380

Where:
- y : salary
- a : efficiency
- b : field_goals_made
- c : points1_atts
- d : points2_atts
- e : points2_made
- f : missed_fg
- g : def_rebounds

Conclusion

Throughout this post, I've demonstrated how we can quickly write R code to automate tasks that would normally take much longer. While I only created functions for best subsets regression, there are many other methods of statistical analysis that can be easily computed using R. The take-home message is that R is a powerful and flexible computational tool that can save a lot of time while conducting statistical analysis.

References

- https://onlinecourses.science.psu.edu/stat501/node/330
- http://www.basketball-reference.com
- http://jadianes.me/best-subset-model-selection-with-R
- http://blog.minitab.com/blog/adventures-in-statistics-2/multiple-regession-analysis-use-adjusted-r-squared-and-predicted-r-squared-to-include-the-correct-number-of-variables
- http://web.csulb.edu/~msaintg/ppa696/696regmx.htm
- https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html
- https://www.r-bloggers.com/r-tutorial-series-multiple-linear-regression/