NBA Scoring Analysis

Aidan Berry

November 12, 2018

Overall Question

Can we accurately predict the number of points a player will score over the course of a season using their demographics, attributes, position, team, and performance data?

Step 1: Data Preprocessing and EDA

Here I create the normalize function that is used to standardize the data on a uniform scale as well as the RMSE function which calculates the Root Mean Square Error for a regression model and will be used to test the accuracy of the models created. I also create a function for one hot encoding categorical variables in the dataset.

```
normalize <- function(x) {
  num <- x - mean(x)
  denom <- sd(x)
  return (num/denom)
}

rmse <- function(y,yhat) {
  num <- sum((y - yhat)^2)
  denom <- length(y)
  return(sqrt(num/denom))
}

onehotencoder <- function(df_orig) {
  df<-cbind(df_orig)
  df_clmtyp<-data.frame(clmtyp=sapply(df,class))
  df_col_typ<-data.frame(clmnm=colnames(df),clmtyp=df_clmtyp$clmtyp)
  for (rownm in 1:nrow(df_col_typ)) {
    if (df_col_typ[rownm,"clmtyp"]=="factor") {
      clmn_obj<-df[toString(df_col_typ[rownm,"clmnm"])]
      dummy_matx<-data.frame(model.matrix( ~.-1, data = clmn_obj))
      dummy_matx<-dummy_matx[,c(1,3:ncol(dummy_matx))]
      df[toString(df_col_typ[rownm,"clmnm"])]<-NULL
      df<-cbind(df,dummy_matx)
      df[toString(df_col_typ[rownm,"clmnm"])]<-NULL
    }  }
  return(df)
  }
```

I now read in the dataset from an exported SQL query that joins fields from 3 of the tables. This dataset still needs to be adjusted however, to get the final cleaned dataset for analysis. We will use dplyr for this.

```
dataset <-  read.csv('data/regressorPreCleanedData.csv')
allStar <- read.csv('data/nba_all_star_games.csv')
```

The only column with null values is the college column with 2163 null values. To impute the null values for the college variable, I set all of the null values to the string "No College", which will act as a new category in this column. To do this, I had to first change the type of the College column to character, impute the missing values, and then convert the datatype back to factor (categorical). A player not going to college is still valuable data to include in the model because there could be a correlation between the players that didn't go to college and points scored in a season.

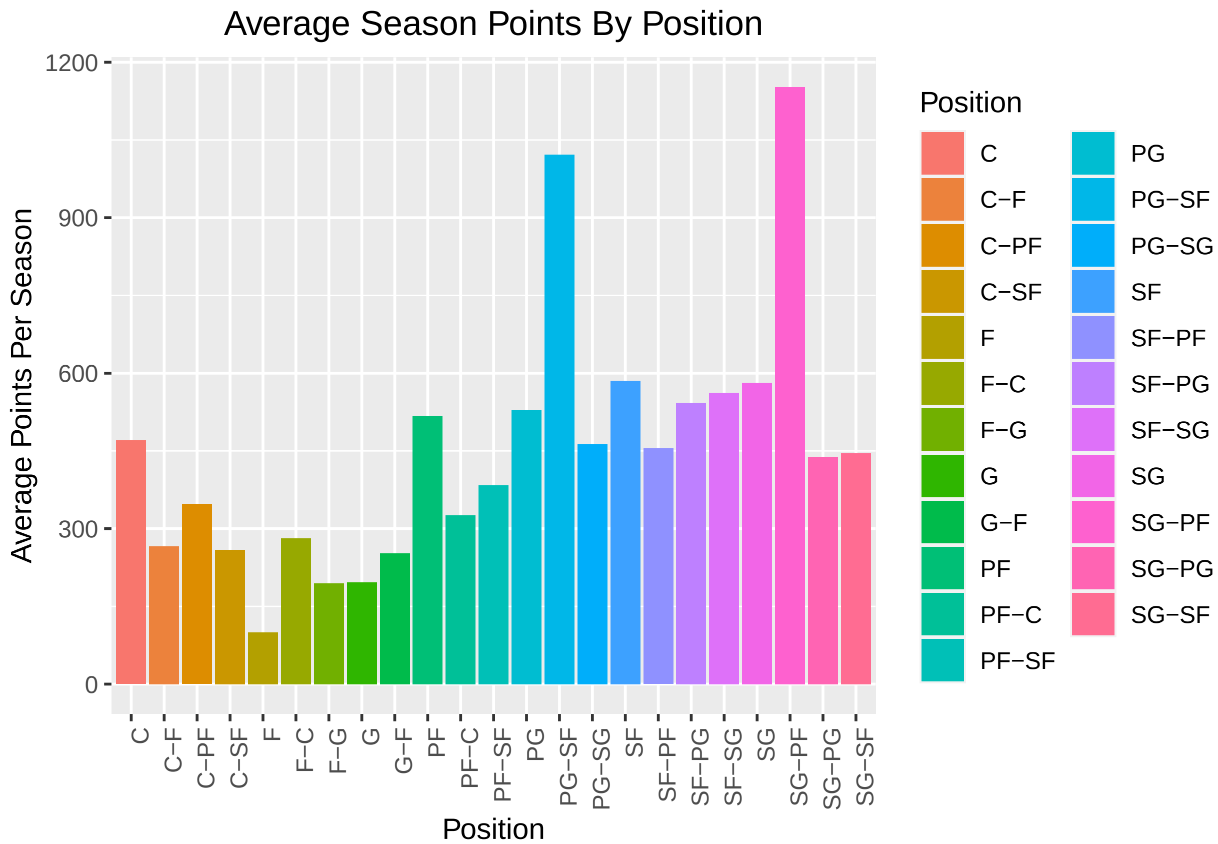
```
  dataset$College <- as.character(dataset$College)
  dataset$College[dataset$College==""] <- "No College"
  dataset$College <- as.factor(dataset$College)
```

Here we need to determine if each player in the dataset was an All Star at any point in their career. To do this I created a vector that contains whether each player in the dataset was contained in the allStar dataset by signifying true or false. I then used the vectorized functionality of R to go through the dataset and change the All_Star variable to 1 if the respective value in the allstarstatus array is true.

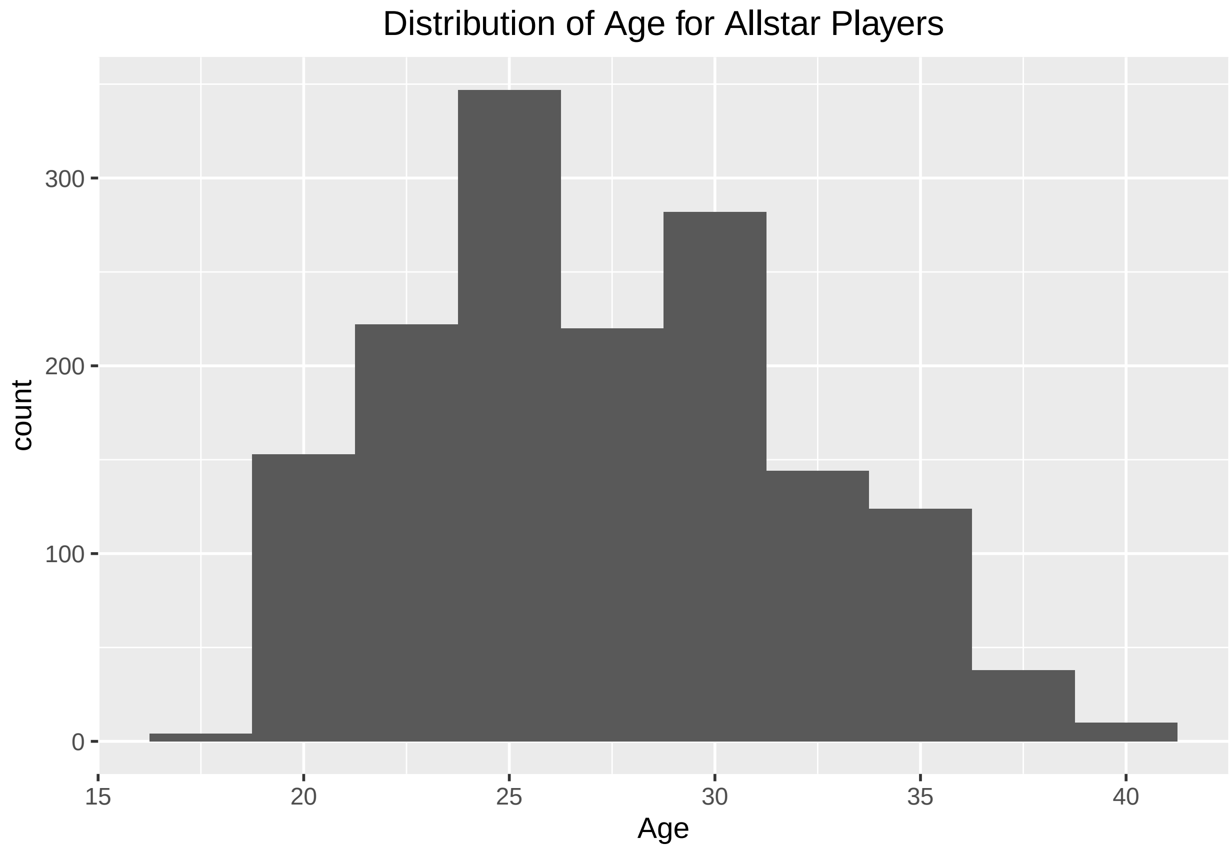
```
  allstarstatus <- dataset$Player_Name %in% allStar$player
  dataset$All_Star[allstarstatus==TRUE] <- 1
```

This is now the final version of the dataset that will be used. A description of each variable is described below.

- Points: Total number of points the player scored during that season
- Team: 3 letter abbreviation for the team that the player is on
- height: Height of the player in cm
- weight: Weight of the player in kg
- Position: The abbreviation for the position the player plays on the court
- Age: The age of the player in years
- All_Star: 1 if the player was ever on an all-star team, 0 if not




```
## Warning: Use of `allStarAge$Age` is discouraged. Use `Age` instead.
```



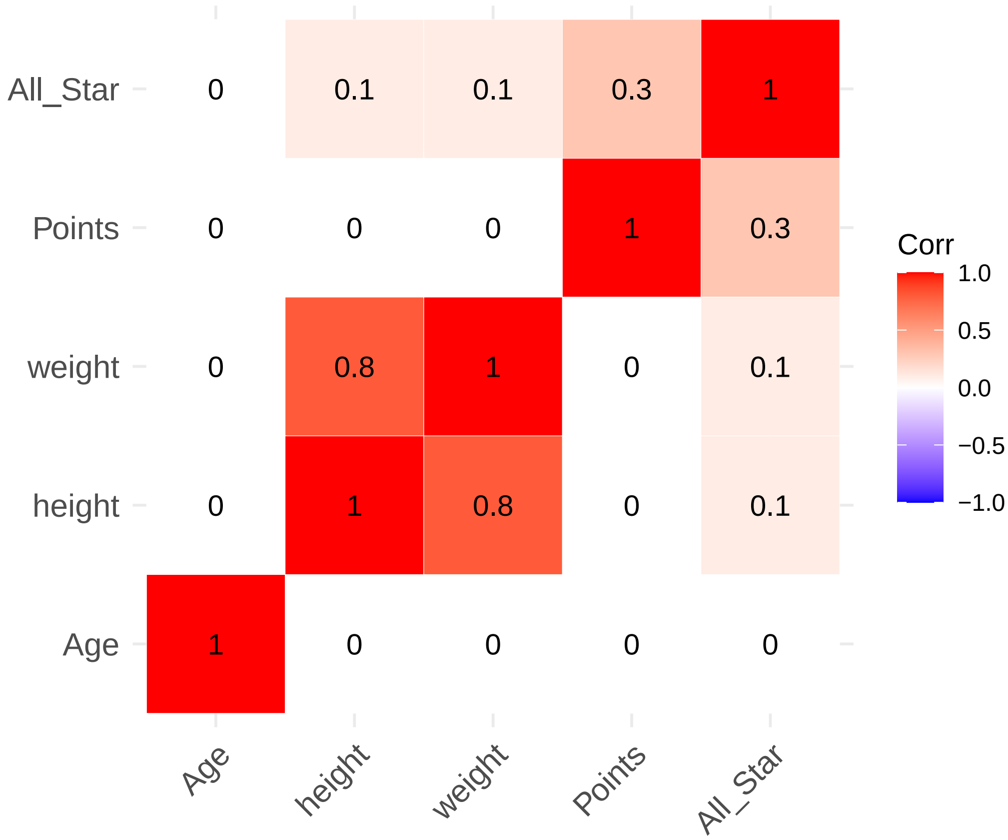
It appears that the SG-PF position and PG-SF positions score the most points on average per season. This makes sense because they are attacking positions that have a lot of time on the ball and are taking the most shots.

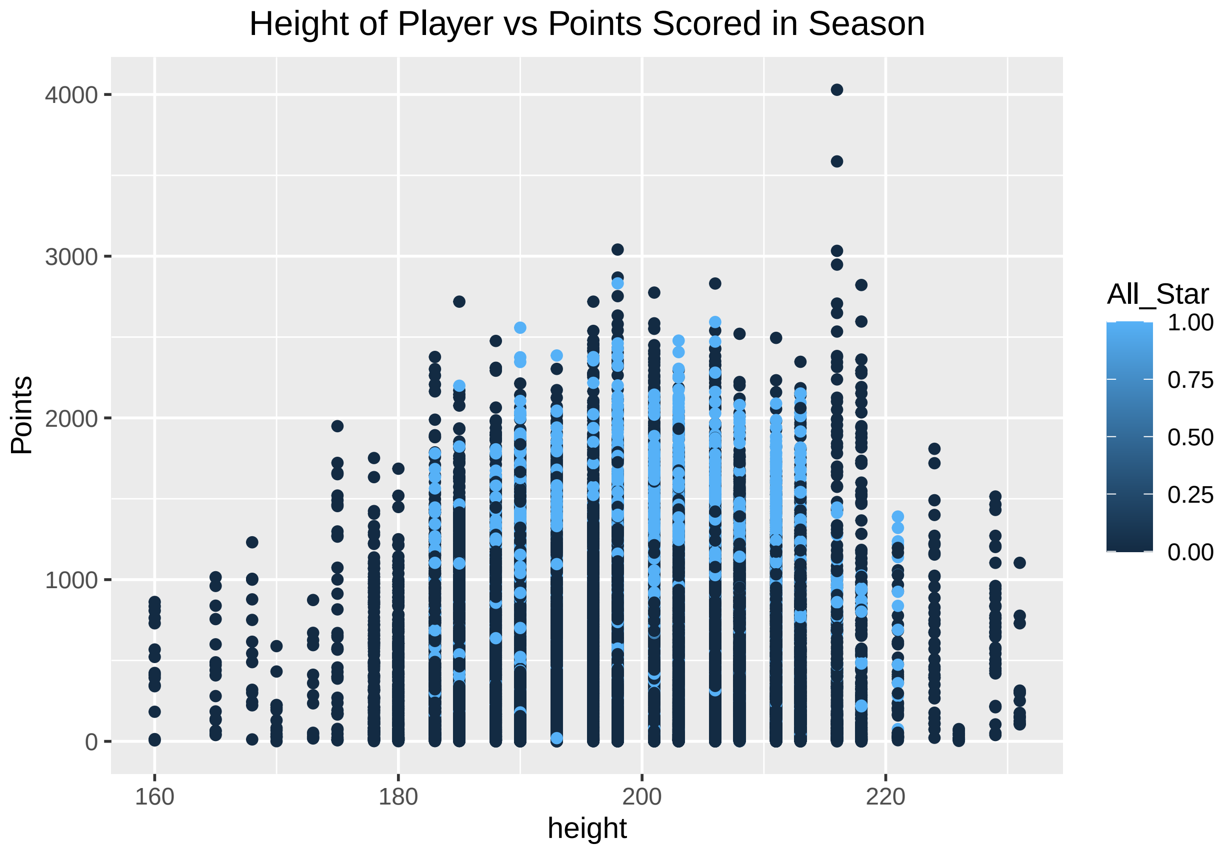
To get the final dataset that is going to be used in the models, all of the variables must be numeric, which means we have to encode the 2 categorical variables and drop unnecessary ones. Here I dropped Player_Name and College, because the name doesn't provide any relevant data and there are too many different factors for college that if it was oneHotEncoded, there would be a lot of noise and too many dimensions in the dataset for the models to make any sense of it.

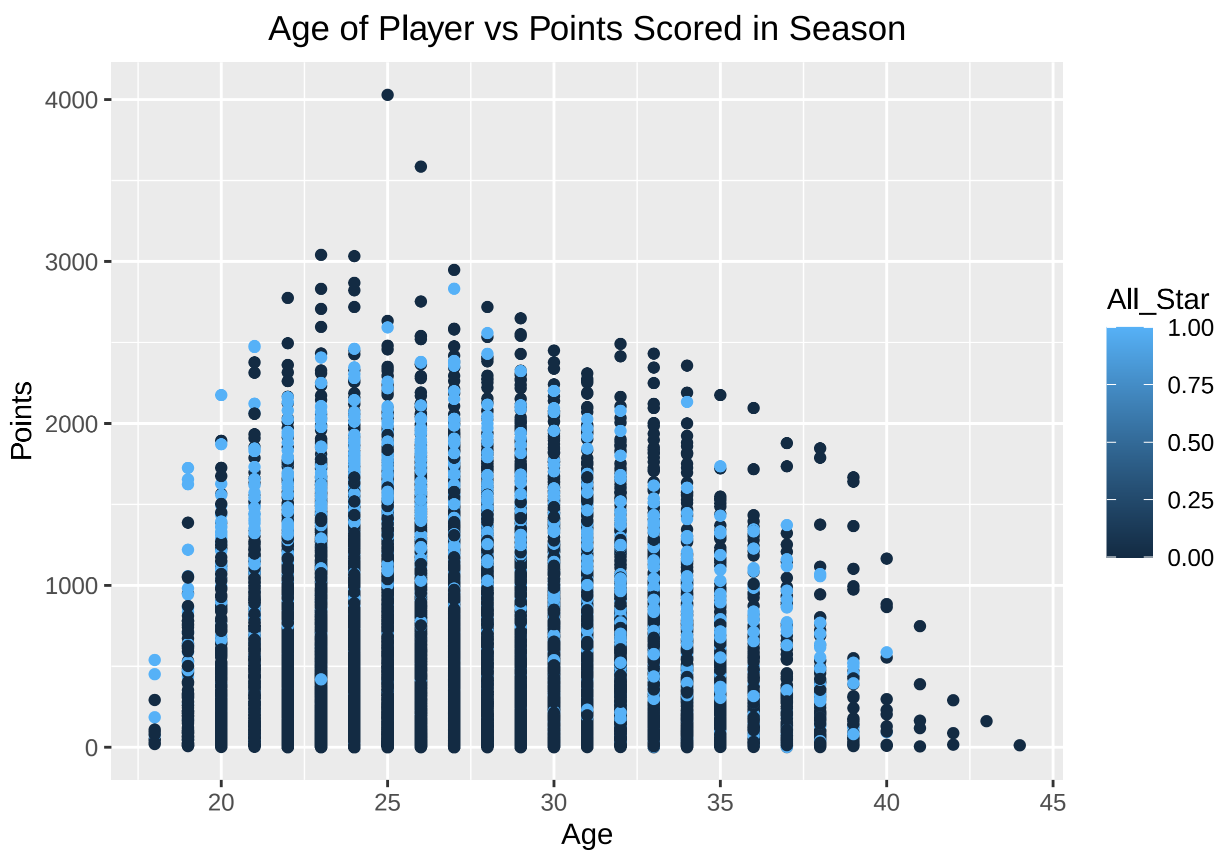
```
dataset <- dataset[-c(1,7)]
encodedDataset <- onehotencoder(dataset)
```

Here I visualized the relationship between the variables using a correlation plot. To build this I dropped the Team and Position variables because there is too many levels to make sense of in the chart, and it makes the chart easier to read without them. From this we can see there is a very strong relationship between height and weight and a small correlation between all_star status and Points scored, which both make sense in this case.

```
## Warning: package 'ggcorrplot' was built under R version 3.6.3
```


All_Star
Points
weight
height
Age
Age
height
weight
Points
All_Star
0
0.1
0.1
0.3
1
0
0
0
1
0.3
0
0.8
1
0
0.1
0
1
0.8
0
0.1
1
0
0
0
0
Corr
1.0
0.5
0.0
−0.5
−1.0


Height of Player vs Points Scored in Season
Points
4000
3000
2000
1000
0
160
180
200
220
height
All_Star
1.00
0.75
0.50
0.25
0.00



From these plots we can see that most of the all-stars are in the upper-middle height range from about 185 cm - 220 cm. The low end and very high end of the height range have extremely low numbers of all-stars. The number of points scored follows a similar pattern, with the most average season points being scored by players in the upper-middle height range, with dips in points near both height extremes.

The age of players also plays an important role in the number of points scored by players as well as their all-star status. The age range of 21-39 seems to produce the most all-stars. The optimal player age for scoring the most average season points appears to be around the 23-29 age range. The points scored starts low and peaks in this 23-29 range, and then slowly decreases from there on out as the player gets older.

Step 2: Building the Models

In order to build the model, we must first split the data into a training set and a test set. I use the split method from the caTools library to do this in one line with a train/test ratio of 75/25 since we have a pretty good sized dataset and want the test set to be as large as possible for validation. I then also created a scaled version of the train and test sets that are used in some models that use a euclidean distance algorithm, this way all the data is on the same scale.

```
set.seed(123)
split <- sample.split(encodedDataset$Points, SplitRatio = 0.75)
training_set <- subset(encodedDataset, split == TRUE)
test_set <- subset(encodedDataset, split == FALSE)
scaled_training_set =training_set
scaled_test_set=test_set
scaled_training_set[2:4] = scale(training_set[2:4])
scaled_test_set[2:4] = scale(test_set[2:4])
```

Multiple Linear Regression

The first model I tried to build for this dataset is the multiple linear regression model. I created the regressor and then prediced the values of the test set using this regressor. The data doesn't need to be scaled for linear regression, so I just used the original training and test sets.

```
set.seed(123)
regressor_MLR = lm(formula = Points ~ .,data = training_set)
summary(regressor_MLR)
```

```
##
## Call:
## lm(formula = Points ~ ., data = training_set)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1141.3  -360.5  -121.5   262.7  3443.1
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -560.7151   185.3858  -3.025 0.002493 **
## height           4.7129     0.8087   5.828 5.72e-09 ***
## weight          -1.9143     0.5551  -3.449 0.000565 ***
## Age              1.6177     0.9530   1.698 0.089609 .
## All_Star       533.5736    14.7691  36.128  < 2e-16 ***
## TeamAND         38.8764   148.5392   0.262 0.793537
## TeamBAL        179.3321    49.9466   3.590 0.000331 ***
## TeamBLB        -46.2017    65.0888  -0.710 0.477822
## TeamBOS         49.1382    26.7228   1.839 0.065959 .
## TeamBRK       -180.7105    60.9671  -2.964 0.003040 **
## TeamBUF         83.8548    53.0676   1.580 0.114090
## TeamCAP        -35.6046   161.5784  -0.220 0.825597
## TeamCHA       -118.1194    46.9135  -2.518 0.011817 *
## TeamCHH        -42.4861    42.4075  -1.002 0.316429
## TeamCHI        -12.3060    28.5094  -0.432 0.666002
## TeamCHO        -82.6903    76.0732  -1.087 0.277059
## TeamCHP        231.1336   161.5727   1.431 0.152584
## TeamCHS         35.8977   148.7117   0.241 0.809255
## TeamCHZ        100.6865   130.4806   0.772 0.440327
## TeamCIN        251.1213    43.5967   5.760 8.55e-09 ***
## TeamCLE        -39.0598    28.5239  -1.369 0.170901
## TeamDAL        -76.0493    30.4069  -2.501 0.012391 *
## TeamDEN         25.9999    29.7790   0.873 0.382622
## TeamDET         33.3179    27.3382   1.219 0.222962
## TeamDNN        111.1302   143.3652   0.775 0.438258
## TeamFTW          8.5226    59.3995   0.143 0.885913
## TeamGSW         -6.9987    28.6881  -0.244 0.807266
## TeamHOU          8.6936    28.8574   0.301 0.763219
## TeamIND          1.3888    30.1715   0.046 0.963286
## TeamINO         22.1035    81.8182   0.270 0.787045
## TeamKCK        110.7016    48.6490   2.276 0.022887 *
## TeamKCO        135.3886    84.9204   1.594 0.110886
## TeamLAC        -91.0691    31.0097  -2.937 0.003320 **
## TeamLAL        104.9419    28.0566   3.740 0.000184 ***
```

```
## TeamMEM       -143.0968    38.8434  -3.684 0.000230 ***
## TeamMIA        -86.4114    32.8131  -2.633 0.008460 **
## TeamMIL          8.7003    28.3994   0.306 0.759338
## TeamMIN        -67.9701    33.8010  -2.011 0.044352 *
## TeamMLH       -151.3257    67.5873  -2.239 0.025170 *
## TeamMNL        100.7422    50.7817   1.984 0.047290 *
## TeamNJN        -72.3264    30.2280  -2.393 0.016735 *
## TeamNOH       -138.0742    48.6892  -2.836 0.004576 **
## TeamNOJ         42.2836    68.4937   0.617 0.537022
## TeamNOK       -143.7665   102.3421  -1.405 0.160109
## TeamNOP       -157.5617    63.4175  -2.485 0.012982 *
## TeamNYK         23.7473    26.6015   0.893 0.372026
## TeamNYN       -135.5619   134.9178  -1.005 0.315019
## TeamOKC        -55.6858    49.9507  -1.115 0.264945
## TeamORL        -45.8033    33.5532  -1.365 0.172241
## TeamPHI         17.7571    27.9855   0.635 0.525754
## TeamPHO         26.6412    28.5616   0.933 0.350955
## TeamPHW        135.4549    49.2234   2.752 0.005932 **
## TeamPOR         19.6365    29.0036   0.677 0.498392
## TeamROC         55.0857    63.5101   0.867 0.385760
## TeamSAC        -45.1278    31.6841  -1.424 0.154376
## TeamSAS        -40.8435    29.5156  -1.384 0.166439
## TeamSDC         69.3029    63.4290   1.093 0.274582
## TeamSDR        268.6761    79.6228   3.374 0.000741 ***
## TeamSEA         72.0820    30.5049   2.363 0.018140 *
## TeamSFW        242.1999    54.5720   4.438 9.13e-06 ***
## TeamSHE         70.9726   144.6030   0.491 0.623567
## TeamSTB        183.0678   185.9473   0.985 0.324876
## TeamSTL        215.4633    47.8741   4.501 6.82e-06 ***
## TeamSYR        154.8073    48.6664   3.181 0.001470 **
## TeamTOR       -137.3903    34.2033  -4.017 5.92e-05 ***
## TeamTOT       -106.9614    23.9308  -4.470 7.88e-06 ***
## TeamTRI        -78.0608    91.0928  -0.857 0.391492
## TeamUTA          4.4904    30.7473   0.146 0.883890
## TeamVAN       -120.3713    61.6956  -1.951 0.051067 .
## TeamWAS       -135.5263    36.2507  -3.739 0.000186 ***
## TeamWAT       -130.5845   142.6786  -0.915 0.360080
## TeamWSB         33.8533    35.7881   0.946 0.344193
## TeamWSC        -39.1471   109.0927  -0.359 0.719717
## PositionC      190.1067   113.2352   1.679 0.093196 .
## PositionC.PF   172.9163   153.0826   1.130 0.258676
## PositionC.SF   105.0891   358.5156   0.293 0.769432
## PositionF     -149.2094   124.6326  -1.197 0.231247
## PositionF.C     65.4375   130.4134   0.502 0.615836
## PositionF.G    -72.9566   136.0309  -0.536 0.591742
## PositionG      -60.0018   123.4561  -0.486 0.626961
## PositionG.F     50.6090   130.7761   0.387 0.698769
## PositionPF     250.9197   113.1604   2.217 0.026610 *
## PositionPF.C   119.4372   153.0693   0.780 0.435236
## PositionPF.SF  285.0312   162.8290   1.750 0.080050 .
## PositionPG     298.1968   113.7526   2.621 0.008763 **
## PositionPG.SF  864.6555   494.1498   1.750 0.080173 .
## PositionPG.SG  287.8840   154.9380   1.858 0.063177 .
## PositionSF     323.3949   113.1982   2.857 0.004283 **
```

```
## PositionSF.PF  291.4697   165.4397   1.762 0.078122 .
## PositionSF.PG  413.3349   494.1680   0.836 0.402927
## PositionSF.SG  348.6578   154.7068   2.254 0.024229 *
## PositionSG     340.4957   113.3569   3.004 0.002670 **
## PositionSG.PF  850.2253   300.1392   2.833 0.004620 **
## PositionSG.PG  280.0928   153.4021   1.826 0.067886 .
## PositionSG.SF  380.1852   171.6105   2.215 0.026746 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 480.9 on 17811 degrees of freedom
## Multiple R-squared:  0.1049, Adjusted R-squared:  0.1002
## F-statistic: 22.21 on 94 and 17811 DF,  p-value: < 2.2e-16
```

```r
y_pred_MLR = predict(regressor_MLR, newdata = test_set)
rmse_MLR <- rmse(test_set$Points, y_pred_MLR)
```

```r
# look at the vif values of the linear regression to see if there is multicollinearity
vif(regressor_MLR)
```

```
##        height        weight           Age      All_Star       TeamAND
##      4.441047      3.452289      1.029005      1.051111      1.048846
##       TeamBAL       TeamBLB       TeamBOS       TeamBRK       TeamBUF
##      1.189974      1.186453      2.213280      1.120705      1.162782
##       TeamCAP       TeamCHA       TeamCHH       TeamCHI       TeamCHO
##      1.015533      1.218847      1.278948      1.929691      1.073474
##       TeamCHP       TeamCHS       TeamCHZ       TeamCIN       TeamCLE
##      1.015462      1.051283      1.029872      1.262893      1.925068
##       TeamDAL       TeamDEN       TeamDET       TeamDNN       TeamFTW
##      1.742508      1.787650      2.093246      1.065813      1.154612
##       TeamGSW       TeamHOU       TeamIND       TeamINO       TeamKCK
##      1.903915      1.885870      1.760479      1.097652      1.199668
##       TeamKCO       TeamLAC       TeamLAL       TeamMEM       TeamMIA
##      1.058232      1.689593      1.989768      1.354046      1.575320
##       TeamMIL       TeamMIN       TeamMLH       TeamMNL       TeamNJN
##      1.940953      1.524088      1.102711      1.197055      1.748335
##       TeamNOH       TeamNOJ       TeamNOK       TeamNOP       TeamNYK
##      1.201655      1.092162      1.040355      1.109038      2.240857
##       TeamNYN       TeamOKC       TeamORL       TeamPHI       TeamPHO
##      1.022515      1.190171      1.534692      1.995475      1.920235
##       TeamPHW       TeamPOR       TeamROC       TeamSAC       TeamSAS
##      1.207487      1.870825      1.112280      1.639491      1.823838
##       TeamSDC       TeamSDR       TeamSEA       TeamSFW       TeamSHE
##      1.109442      1.066833      1.726987      1.153182      1.084298
##       TeamSTB       TeamSTL       TeamSYR       TeamTOR       TeamTOT
##      1.046193      1.210641      1.210635      1.506848      3.487678
##       TeamTRI       TeamUTA       TeamVAN       TeamWAS       TeamWAT
##      1.146160      1.707866      1.114984      1.428388      1.055628
##       TeamWSB       TeamWSC     PositionC    PositionC.PF  PositionC.SF
##      1.440569      1.079456    155.077705      2.226609      1.111478
##     PositionF    PositionF.C    PositionF.G     PositionG    PositionG.F
##      5.416048      3.886304      3.273156      6.032254      3.907948
##    PositionPF  PositionPF.C PositionPF.SF    PositionPG PositionPG.SF
```

```
##    160.516627      2.226223      1.947167    153.283677      1.055836
## PositionPG.SG    PositionSF PositionSF.PF PositionSF.PG PositionSF.SG
##      2.177355    152.550948      1.891971      1.055914      2.170860
##    PositionSG PositionSG.PF PositionSG.PG PositionSG.SF
##    157.273811      1.168416      2.235914      1.781475
```

```
# print the variables that have a vif value higher than 10 (multicollinearity problem)
which(vif(regressor_MLR)>10)
```

```
##  PositionC PositionPF PositionPG PositionSF PositionSG
##         73         81         84         87         91
```

```
set.seed(123)
regressor_MLR2 = lm(formula = Points ~ . - PositionPF,data = training_set)
summary(regressor_MLR2)
```

```
##
## Call:
## lm(formula = Points ~ . - PositionPF, data = training_set)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1141.8  -361.0  -121.4   263.0  3447.3
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -308.1241   146.2743  -2.106 0.035177 *
## height           4.6835     0.8087   5.791 7.10e-09 ***
## weight          -1.8855     0.5550  -3.397 0.000682 ***
## Age              1.6329     0.9531   1.713 0.086666 .
## All_Star       533.6402    14.7707  36.128  < 2e-16 ***
## TeamAND         35.3355   148.5470   0.238 0.811981
## TeamBAL        179.2347    49.9521   3.588 0.000334 ***
## TeamBLB        -52.2397    65.0390  -0.803 0.421866
## TeamBOS         47.6815    26.7176   1.785 0.074336 .
## TeamBRK       -180.7713    60.9738  -2.965 0.003033 **
## TeamBUF         83.9146    53.0734   1.581 0.113872
## TeamCAP        -35.6714   161.5961  -0.221 0.825294
## TeamCHA       -118.2015    46.9187  -2.519 0.011768 *
## TeamCHH        -42.5327    42.4122  -1.003 0.315951
## TeamCHI        -12.2725    28.5125  -0.430 0.666892
## TeamCHO        -82.7304    76.0816  -1.087 0.276879
## TeamCHP        231.1976   161.5905   1.431 0.152516
## TeamCHS        -11.3770   147.1916  -0.077 0.938391
## TeamCHZ         99.1505   130.4931   0.760 0.447376
## TeamCIN        250.9706    43.6014   5.756 8.75e-09 ***
## TeamCLE        -39.5539    28.5262  -1.387 0.165587
## TeamDAL        -76.0976    30.4102  -2.502 0.012345 *
## TeamDEN         25.9840    29.7822   0.872 0.382965
## TeamDET         33.3188    27.3412   1.219 0.223001
## TeamDNN        107.6356   143.3722   0.751 0.452818
## TeamFTW          4.0403    59.3717   0.068 0.945746
## TeamGSW         -6.9963    28.6913  -0.244 0.807351
```

```
## TeamHOU          8.7004    28.8605   0.301 0.763065
## TeamIND          1.3785    30.1748   0.046 0.963562
## TeamINO         13.9230    81.7440   0.170 0.864757
## TeamKCK        110.7879    48.6543   2.277 0.022795 *
## TeamKCO        135.4859    84.9297   1.595 0.110669
## TeamLAC        -91.1186    31.0131  -2.938 0.003307 **
## TeamLAL        104.9077    28.0597   3.739 0.000186 ***
## TeamMEM       -143.1616    38.8476  -3.685 0.000229 ***
## TeamMIA        -86.4824    32.8167  -2.635 0.008413 **
## TeamMIL          8.6805    28.4025   0.306 0.759896
## TeamMIN        -68.0146    33.8048  -2.012 0.044237 *
## TeamMLH       -156.3346    67.5570  -2.314 0.020673 *
## TeamMNL        100.0844    50.7864   1.971 0.048775 *
## TeamNJN        -72.2624    30.2313  -2.390 0.016844 *
## TeamNOH       -138.2465    48.6945  -2.839 0.004530 **
## TeamNOJ         42.3290    68.5013   0.618 0.536630
## TeamNOK       -143.6439   102.3533  -1.403 0.160511
## TeamNOP       -157.6888    63.4244  -2.486 0.012919 *
## TeamNYK         22.6383    26.5997   0.851 0.394740
## TeamNYN       -135.5162   134.9326  -1.004 0.315236
## TeamOKC        -55.8222    49.9562  -1.117 0.263828
## TeamORL        -45.8205    33.5569  -1.365 0.172128
## TeamPHI         17.7432    27.9886   0.634 0.526125
## TeamPHO         26.6670    28.5647   0.934 0.350542
## TeamPHW        130.7063    49.1822   2.658 0.007877 **
## TeamPOR         19.6401    29.0067   0.677 0.498359
## TeamROC         54.7507    63.5169   0.862 0.388707
## TeamSAC        -45.1658    31.6876  -1.425 0.154075
## TeamSAS        -40.8132    29.5188  -1.383 0.166800
## TeamSDC         69.3626    63.4360   1.093 0.274221
## TeamSDR        268.6499    79.6316   3.374 0.000743 ***
## TeamSEA         72.0682    30.5082   2.362 0.018175 *
## TeamSFW        242.0277    54.5780   4.435 9.28e-06 ***
## TeamSHE         66.2804   144.6035   0.458 0.646700
## TeamSTB        144.5473   185.1544   0.781 0.434998
## TeamSTL        215.3001    47.8793   4.497 6.94e-06 ***
## TeamSYR        154.3701    48.6714   3.172 0.001518 **
## TeamTOR       -137.4567    34.2070  -4.018 5.88e-05 ***
## TeamTOT       -107.0432    23.9334  -4.473 7.78e-06 ***
## TeamTRI        -96.9329    90.7043  -1.069 0.285233
## TeamUTA          4.4370    30.7506   0.144 0.885274
## TeamVAN       -120.3916    61.7024  -1.951 0.051053 .
## TeamWAS       -135.5996    36.2546  -3.740 0.000184 ***
## TeamWAT       -134.1504   142.6852  -0.940 0.347136
## TeamWSB         33.8972    35.7920   0.947 0.343621
## TeamWSC        -42.1110   109.0965  -0.386 0.699503
## PositionC      -59.5600    12.0223  -4.954 7.33e-07 ***
## PositionC.PF   -76.9951   103.6053  -0.743 0.457396
## PositionC.SF  -144.5671   340.4148  -0.425 0.671076
## PositionF     -395.7511    56.3173  -7.027 2.18e-12 ***
## PositionF.C   -179.1648    69.5710  -2.575 0.010024 *
## PositionF.G   -319.6323    78.2940  -4.082 4.48e-05 ***
## PositionG     -305.4176    54.7028  -5.583 2.40e-08 ***
## PositionG.F   -192.1964    71.5055  -2.688 0.007198 **
```

```
## PositionPF.C  -130.4076   103.6184  -1.259 0.208214
## PositionPF.SF   35.3004   117.6099   0.300 0.764067
## PositionPG      48.5597    16.2789   2.983 0.002858 **
## PositionPG.SF  614.9081   481.1954   1.278 0.201310
## PositionPG.SG   38.0787   106.3790   0.358 0.720382
## PositionSF      73.8061    12.0024   6.149 7.95e-10 ***
## PositionSF.PF   41.6152   121.1475   0.344 0.731220
## PositionSF.PG  163.6593   481.2216   0.340 0.733792
## PositionSF.SG   98.8833   106.0566   0.932 0.351161
## PositionSG      90.9355    13.5252   6.723 1.83e-11 ***
## PositionSG.PF  600.4719   278.2338   2.158 0.030929 *
## PositionSG.PG   30.2566   104.1137   0.291 0.771352
## PositionSG.SF  130.3670   129.4633   1.007 0.313958
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 480.9 on 17812 degrees of freedom
## Multiple R-squared:  0.1047, Adjusted R-squared:    0.1
## F-statistic: 22.39 on 93 and 17812 DF,  p-value: < 2.2e-16
```

```r
y_pred_MLR2 = predict(regressor_MLR2, newdata = test_set[-1])
rmse_MLR2 <- rmse(test_set$Points, y_pred_MLR2)


# this fixed the multicollinearity problem but the regression fit is still not great
which(vif(regressor_MLR2)>10)
```

```
## named integer(0)
```

```r
# perform stepwise selection to pick the variables in the model systematically
set.seed(123)
train_control = trainControl(method="cv", number = 10)
step_wise_model = train(Points~., data = training_set, method = "leapSeq",
                        tuneGrid=data.frame(nvmax=1:95), trControl = train_control)
```

```
## Warning in leaps.setup(x, y, wt = weights, nbest = nbest, nvmax = nvmax, : 1
## linear dependencies found

## Reordering variables and trying again:

## Warning: predictions failed for Fold02: nvmax=95 Error in method$predict(modelFit = modelFit, newdata
##   Some values of 'nvmax' are not in the model sequence.

## Warning: 1  linear dependencies found

## Reordering variables and trying again:

## Warning: predictions failed for Fold10: nvmax=95 Error in method$predict(modelFit = modelFit, newdata
##   Some values of 'nvmax' are not in the model sequence.

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```r
# the best model contained 81 variables
step_wise_model$results
```

```
##    nvmax     RMSE   Rsquared      MAE   RMSESD  RsquaredSD    MAESD
## 1      1 492.5847 0.06352857 389.0463 7.143910 0.008904095 4.942410
## 2      2 491.7208 0.06672220 388.3222 7.088769 0.008543061 4.956679
## 3      3 490.5960 0.07087496 386.6737 6.897200 0.007146835 4.917954
## 4      4 489.8972 0.07354328 385.7532 7.120677 0.008061656 5.409516
## 5      5 489.2403 0.07597686 384.9611 7.134440 0.007892107 5.490038
## 6      6 489.3207 0.07561339 385.2398 7.148749 0.007645635 5.457132
## 7      7 489.7376 0.07415941 385.9948 7.389400 0.011941553 5.576446
## 8      8 488.6554 0.07817350 384.8143 7.353211 0.008669677 5.389487
## 9      9 488.6990 0.07819111 385.2012 7.052943 0.007229905 5.218492
## 10    10 488.3381 0.07931297 384.6365 6.711564 0.007332499 5.281028
## 11    11 488.6221 0.07826411 385.2850 7.843621 0.008342485 6.526024
## 12    12 488.9506 0.07711223 385.5851 7.048728 0.008386094 5.084662
## 13    13 488.4990 0.07887248 385.1671 7.464287 0.007045598 5.810736
## 14    14 487.5272 0.08245827 384.3869 7.494005 0.009102786 5.859294
## 15    15 487.7242 0.08162322 384.5587 7.328101 0.008619866 5.467925
## 16    16 487.6946 0.08191508 384.7984 6.928362 0.006077059 4.934461
## 17    17 486.9496 0.08457120 384.0155 7.006284 0.007433052 5.423698
## 18    18 487.7161 0.08181544 384.8429 8.437272 0.011026762 6.823368
## 19    19 487.4296 0.08290342 384.6637 5.922558 0.006481139 4.469669
## 20    20 486.5401 0.08613758 383.6505 7.146028 0.008222849 5.569326
## 21    21 486.9716 0.08439405 383.9966 6.151242 0.007707214 5.079312
## 22    22 486.9930 0.08443266 384.2062 8.339564 0.011847740 6.569507
## 23    23 486.5475 0.08627297 383.9411 6.810181 0.005690683 5.029694
## 24    24 486.4397 0.08663027 383.7897 7.232517 0.006554616 5.859820
## 25    25 486.5721 0.08604435 383.8303 8.517333 0.012729805 6.739643
## 26    26 486.0650 0.08797212 383.2594 7.094368 0.007212542 5.516875
## 27    27 485.8098 0.08894053 383.0747 7.102851 0.007767159 5.541129
## 28    28 486.8167 0.08512197 383.9216 7.286400 0.012673942 5.444084
## 29    29 486.0663 0.08790965 383.2648 7.034939 0.009690190 5.585154
## 30    30 485.3210 0.09073860 382.5827 6.935527 0.007592797 5.449009
## 31    31 485.1965 0.09118475 382.4607 6.908309 0.007337291 5.466141
## 32    32 485.1236 0.09146449 382.4005 6.835345 0.007503115 5.456363
## 33    33 485.3814 0.09040336 382.4678 6.244451 0.008714691 5.316571
## 34    34 485.7592 0.08918899 382.9053 8.587590 0.013811108 6.761931
## 35    35 484.9241 0.09224176 382.3539 6.774730 0.008118308 5.440576
## 36    36 485.5814 0.08987890 382.9712 7.212023 0.007422832 5.995426
## 37    37 486.0300 0.08808845 383.2211 6.837801 0.010997274 5.192937
## 38    38 484.9160 0.09230702 382.4265 6.825933 0.007955897 5.377885
## 39    39 484.9383 0.09224119 382.3979 6.845493 0.007904905 5.454267
## 40    40 485.8630 0.08878243 383.5378 6.872692 0.011517395 5.523493
## 41    41 485.9552 0.08844932 383.4655 7.191351 0.009082226 6.269037
## 42    42 485.2149 0.09115320 382.6757 6.963175 0.012253751 6.061513
## 43    43 486.5938 0.08609878 383.9024 7.355962 0.011816991 5.926987
## 44    44 486.0500 0.08788994 383.2899 6.729550 0.014460579 6.331175
## 45    45 485.2134 0.09127787 382.7016 7.323259 0.007956270 6.085131
## 46    46 484.5167 0.09383109 382.1166 6.942287 0.008546818 5.593084
## 47    47 484.5038 0.09388230 382.1055 6.990469 0.008560861 5.558944
## 48    48 485.0651 0.09173109 382.5320 7.219896 0.012584797 6.109789
## 49    49 485.0128 0.09189682 382.6478 7.080085 0.011020556 5.889433
```

```
## 50    50 484.3602 0.09439160 382.0224 6.910623 0.008034671 5.447121
## 51    51 486.1394 0.08778257 383.4617 7.319588 0.010242351 5.798907
## 52    52 484.9639 0.09212601 382.3885 7.245413 0.012563254 6.156724
## 53    53 485.4492 0.09022959 382.8500 7.734203 0.013992719 6.413011
## 54    54 484.3753 0.09436407 381.9395 6.939935 0.008327146 5.507337
## 55    55 485.6243 0.08969806 383.4108 8.302065 0.012885651 6.213404
## 56    56 485.7829 0.08907215 383.0307 7.249422 0.013704622 5.698266
## 57    57 485.0275 0.09184288 382.3846 6.224439 0.010343737 5.270547
## 58    58 485.1327 0.09150509 382.6997 7.200257 0.011377723 6.001603
## 59    59 484.5094 0.09388720 382.0328 6.983424 0.008495922 5.488997
## 60    60 484.4635 0.09404178 382.0172 6.989106 0.008418094 5.543709
## 61    61 485.8284 0.08901398 383.1295 7.245686 0.009261405 5.346138
## 62    62 484.4185 0.09419559 381.9301 6.843253 0.007934679 5.453687
## 63    63 485.7051 0.08952791 383.1450 8.262594 0.013823307 6.065548
## 64    64 484.8459 0.09266517 382.6583 6.912692 0.007861401 5.114431
## 65    65 484.4504 0.09410302 381.9950 6.989763 0.008148227 5.528943
## 66    66 484.8090 0.09282277 382.2367 7.142013 0.006959975 5.733611
## 67    67 484.7525 0.09291981 382.3435 7.055800 0.009557842 5.813275
## 68    68 484.8394 0.09268183 382.6223 6.865309 0.007648348 5.070152
## 69    69 484.4164 0.09424034 381.9232 6.929491 0.008309412 5.464251
## 70    70 484.4425 0.09414726 381.9606 6.955284 0.008267171 5.503048
## 71    71 484.4412 0.09415282 381.9429 6.925623 0.008284369 5.484394
## 72    72 485.0356 0.09199839 382.4423 6.949698 0.010073786 5.003972
## 73    73 484.6429 0.09343302 382.1847 6.978077 0.007138629 5.633889
## 74    74 484.8997 0.09238260 382.3506 7.244394 0.010455659 5.947993
## 75    75 484.8476 0.09261320 382.3473 6.529400 0.007125989 5.394460
## 76    76 484.6673 0.09330889 382.0695 6.914579 0.009546669 5.697471
## 77    77 485.6000 0.08975523 383.0005 6.823683 0.008540653 4.974890
## 78    78 484.6858 0.09326734 382.1116 6.964984 0.009516074 5.746210
## 79    79 484.4007 0.09429229 381.8762 6.907100 0.008032206 5.516110
## 80    80 484.6085 0.09352508 381.9307 6.821025 0.008089574 5.459625
## 81    81 484.2568 0.09478787 381.8475 6.902679 0.007690307 5.499354
## 82    82 484.4098 0.09425764 381.8861 6.903003 0.007888842 5.546070
## 83    83 484.2700 0.09473658 381.8822 6.883396 0.007640213 5.491572
## 84    84 484.4261 0.09420021 381.9084 6.864125 0.007925460 5.520485
## 85    85 484.3632 0.09441468 381.8959 6.912037 0.007895170 5.541044
## 86    86 484.4332 0.09418195 381.9478 6.908887 0.008101133 5.564170
## 87    87 484.4371 0.09415600 381.9377 6.884567 0.007823598 5.501084
## 88    88 484.4495 0.09412242 381.9593 6.911859 0.007963677 5.559307
## 89    89 484.4277 0.09419129 381.9343 6.853479 0.007932078 5.490798
## 90    90 484.4206 0.09421170 381.9414 6.891539 0.007845893 5.487461
## 91    91 484.3937 0.09431767 381.9294 6.905049 0.007951948 5.492286
## 92    92 484.4090 0.09426673 381.9354 6.908262 0.008042619 5.573192
## 93    93 484.4101 0.09426115 381.9070 6.946429 0.008019806 5.544403
## 94    94 484.3897 0.09433461 381.9036 6.906962 0.007942480 5.519409
## 95    95 484.3897 0.09433461 381.9036 6.906962 0.007942480 5.519409
```

```
step_wise_model$bestTune
```

```
##    nvmax
## 81    81
```

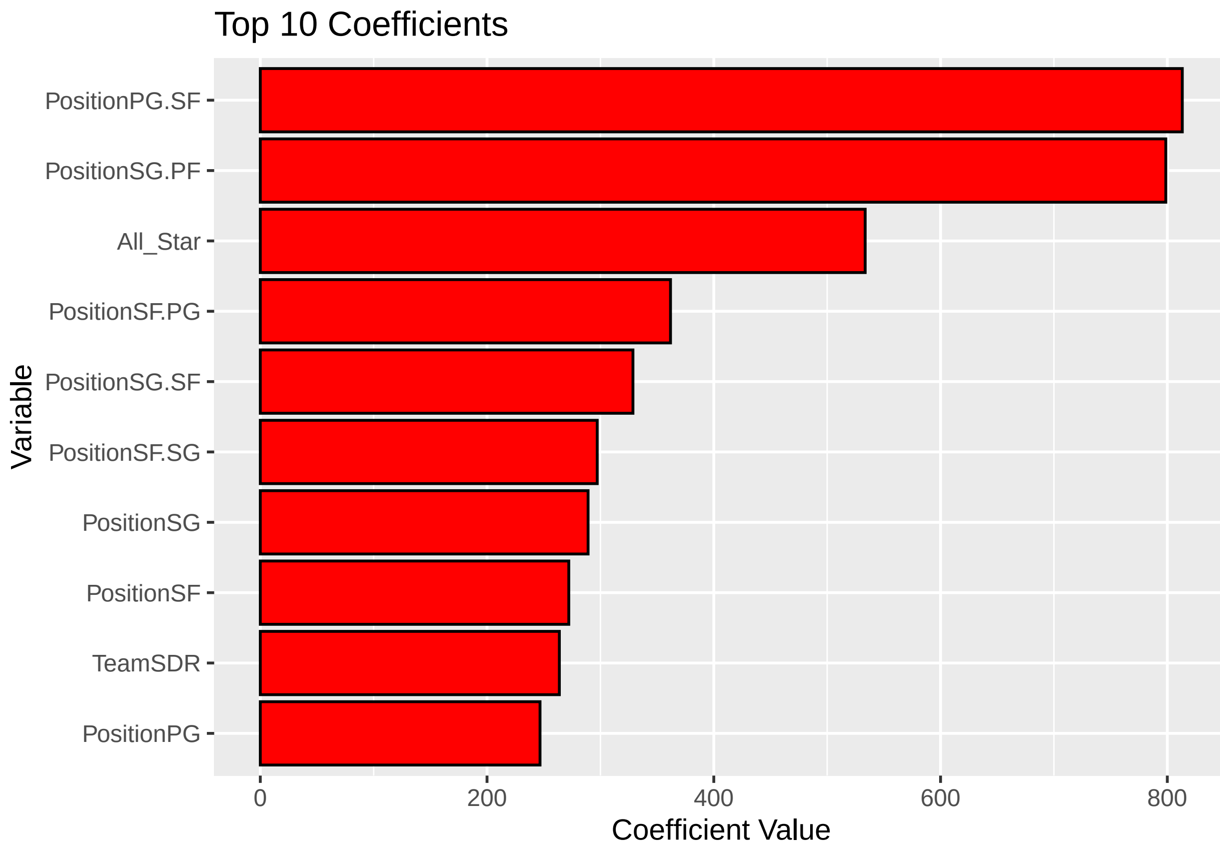
```r
# look at the coefficients of the best tuned model with 81 variables
coef(step_wise_model$finalModel, 81)
```

```
##   (Intercept)        height        weight           Age      All_Star
##   -501.387499      4.697110     -1.915173      1.621616    533.514839
##       TeamBAL       TeamBLB       TeamBOS       TeamBRK       TeamBUF
##    174.625306    -51.523309     44.017985   -185.554636     78.940618
##       TeamCHA       TeamCHH       TeamCHI       TeamCHO       TeamCHP
##   -122.983238    -47.391332    -17.167235    -87.535478    226.214180
##       TeamCHZ       TeamCIN       TeamCLE       TeamDAL       TeamDEN
##     96.712173    246.265185    -43.996695    -80.903992     21.118654
##       TeamDET       TeamDNN       TeamGSW       TeamKCK       TeamKCO
##     28.426896    109.293039    -11.883710    105.811260    130.470640
##       TeamLAC       TeamLAL       TeamMEM       TeamMIA       TeamMIN
##    -95.928525    100.069874   -147.950763    -91.276915    -72.826848
##       TeamMLH       TeamMNL       TeamNJN       TeamNOH       TeamNOJ
##   -157.021109     95.741644    -77.200656   -142.930054     37.375230
##       TeamNOK       TeamNOP       TeamNYK       TeamNYN       TeamOKC
##   -148.648454   -162.411909     18.667647   -140.493239    -60.527435
##       TeamORL       TeamPHI       TeamPHO       TeamPHW       TeamPOR
##    -50.667157     12.880604     21.792300    129.835745     14.777697
##       TeamROC       TeamSAC       TeamSAS       TeamSDC       TeamSDR
##     50.348134    -49.998015    -45.710062     64.413153    263.759778
##       TeamSEA       TeamSFW       TeamSHE       TeamSTB       TeamSTL
##     67.201101    237.287996     66.570249    170.419331    210.479324
##       TeamSYR       TeamTOR       TeamTOT       TeamTRI       TeamVAN
##    150.016038   -142.253415   -111.669845    -83.949338   -125.248867
##       TeamWAS       TeamWAT       TeamWSB     PositionC   PositionC.PF
##   -140.386283   -134.946396     28.967624    138.960940    121.550697
##     PositionF   PositionF.G     PositionG    PositionPF   PositionPF.C
##   -201.786045   -123.718153   -109.744151    199.717298     68.096217
## PositionPF.SF    PositionPG PositionPG.SF PositionPG.SG    PositionSF
##    233.600467    246.719202    813.215296    236.273385    272.090001
## PositionSF.PF PositionSF.PG PositionSF.SG    PositionSG PositionSG.PF
##    240.055875    361.796939    297.171836    289.108783    798.662326
## PositionSG.PG PositionSG.SF
##    228.439915    328.690076
```

```r
# predict on the test set
y_pred_MLR3 = predict.train(step_wise_model, test_set[-1], type="raw")
# compute RMSE
rmse_MLR3 = rmse(test_set$Points, y_pred_MLR3)

# plot the 10 highest coefficient magnitude variables
index_of_top_10 = c(which(abs(coef(step_wise_model$finalModel, 81))>246.3))[-1]
var_names = c('PositionPG.SF', 'PositionSG.PF', 'All_Star', 'PositionSF.PG', 'PositionSG.SF',
              'PositionSF.SG', 'PositionSG', 'PositionSF', 'TeamSDR', 'PositionPG')
coef_data = data.frame(coef(step_wise_model$finalModel, 81)[index_of_top_10])
colnames(coef_data) = c("Coef")
coef_data = coef_data %>% arrange(desc(Coef)) %>% filter(Coef>246.3)
coef_data = data.frame(cbind(var_names, coef_data))
colnames(coef_data)= c('Variable', 'Coefficient')
```

```r
ggplot(coef_data, aes(x=Coefficient, y = reorder(Variable, Coefficient))) +
  geom_bar(stat = 'identity', color = "black", fill = "red") +
  labs(title = "Top 10 Coefficients", y = "Variable",
       x = "Coefficient Value")
```

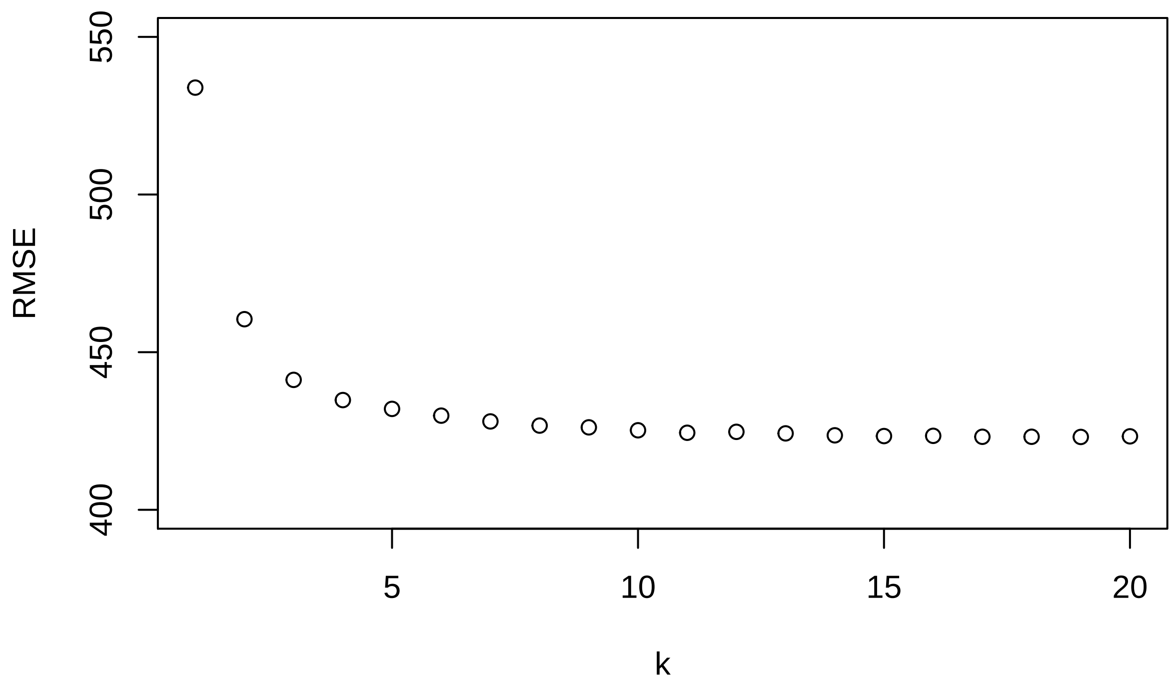


This plot shows the top 10 variables of the multiple linear regression model that have the highest magnitude coefficient. The Point Guard - Shooting Forward and Shooting Guard - Power Forward positions as well as whether or not the player is an all-star contribute the highest increases in points per season, holding all other variables constant. This is fairly intuitive because those 2 positions are the ones that are taking the most shots at the net in game and have the most scoring opportunities relative to other positions. Being an all-star means that the player is more talented than the average NBA player, so these all-star players by default would be expected to be scoring more points per season than a normal player. It could also be the other way around, where all-star players are picked as all-stars because of their high scoring nature. Regardless, an all-star that is either a PG-SF or SG-PF position would have the highest season scoring potential based on this regression model.

KNN Regression

The next model I am going to test out for this dataset in prediciting the number of points scored in a season by a player is KNN regression. To do this I fitted the KNN model using the scaled training and test sets, since this algorithm utilizes euclidean distance metrics in its analysis. The one hyperparameter that you must tune for KNN models it he k value. To figure out the best k-value to use in the model to get the lowest RMSE, I loop through the k-values 1-20 and build a KNN model using each k-value. I then predict the test set points using this newly created model and plot the RMSE for each k-value. You then want to pick the k-value that minimizes the RMSE, which appears to be at $k = 19$ in this case.

```r
set.seed(123)
a <- NULL
for (i in 1:20) {
  model <- kknn(Points ~ .,scaled_training_set,scaled_test_set,k=i,kernel="rectangular")
  a[i] <- rmse(scaled_test_set$Points,predict(model))
}
plot(1:20,a,xlab="k",ylab="RMSE", ylim =c(400,550))
```




```r
modelFinal <- kknn(Points ~ .,scaled_training_set,scaled_test_set,k=19,kernel="rectangular")
rmse_KNN <- rmse(scaled_test_set$Points,predict(modelFinal))
```

Random Forest

The next model I tested out was a random forest ensemble regression model. A random forest is basically a "forest" of decision trees where it takes the average of the result returned from all of the decision trees. The advantage of the Random Tree Model is that it is more robust than a sinlge decision tree since it takes the average of many trees that are all built seperately, allowing it to predict new values more accurately. The disadvantage of the random forest is that you must choose the number of trees in the forest as a hyper parameter. You want to use enough trees to allow the model to make accurate predictions, but you also don't want the model to overfit to the data. Like a decision tree, random forests do not need normalized values, so I used the original training and test sets to build the model.

Once the model is built, I found the optimal hyperparameter for the number of trees in the forest by testing out different benchmark values as shown below.

- 25 trees: RMSE = 392.8573
- 50 trees: RMSE = 390.6675
- 75 trees: RMSE = 389.6562
- 100 trees: RMSE = 388.7601
- 300 trees: RMSE = 387.6793

From these findings, I chose to use 50 trees in the final model. After 50 trees, there is diminishing improvements in the RMSE, but the number of trees is still low enough to avoid overfitting to the data. 50 is the right balance between accurate predicitons (minimizing RMSE) and avoiding overfitting in my opinion.

```
set.seed(123)
RFModel <- randomForest(x = training_set[-1], y = training_set$Points,ntree =50)
y_pred_RF <- predict(RFModel, test_set)
rmse_RF <- rmse(test_set$Points, y_pred_RF)
```

SVR (Support Vector Regression)

Here we fit the SVR (Support vector regression) model to the dataset using the radial (non-linear) kernel and the eps-regression type as hyperparamters. The SVR algorithm is very effective with outliers and on non-linear problems, so I wanted to see how it compares to the other models built earlier on in the analysis. I also used the scaled training and test sets as the datasets because the SVR algorithm uses distance metrics in its computations, so the variables must be on the same scale. From the report of the RMSE, this model doesn't seem to be as effective as the previous models, so I am going to try a different kernel.

```
regressor_SVR = svm(formula = Points ~ .,data = scaled_training_set,type = 'eps-regression',
                    kernel = 'radial')
y_pred_SVR = predict(regressor_SVR, scaled_test_set)
rmse_NLSVR <- rmse(scaled_test_set$Points, y_pred_SVR)
```

Here I fit the same SVR model using a linear kernel to see if that improves the RMSE. Unfortunately the linear kernel did worse than the radial (non-linear) kernel. We can also try using the polynomial, gaussian, and sigmoid kernels to see if these can give us better results. The RMSE of these other kernel types are shown below.

- linear: 433.3105
- radial: 427
- polynomial: 433.6522
- sigmoid: 961.5413

```
regressor_SVR_Linear = svm(formula = Points ~ .,data = scaled_training_set,type = 'eps-regression',
                           kernel = 'linear')
y_pred_SVR_Linear = predict(regressor_SVR_Linear, scaled_test_set)
rmse_LSVR <- rmse(scaled_test_set$Points, y_pred_SVR_Linear)
```

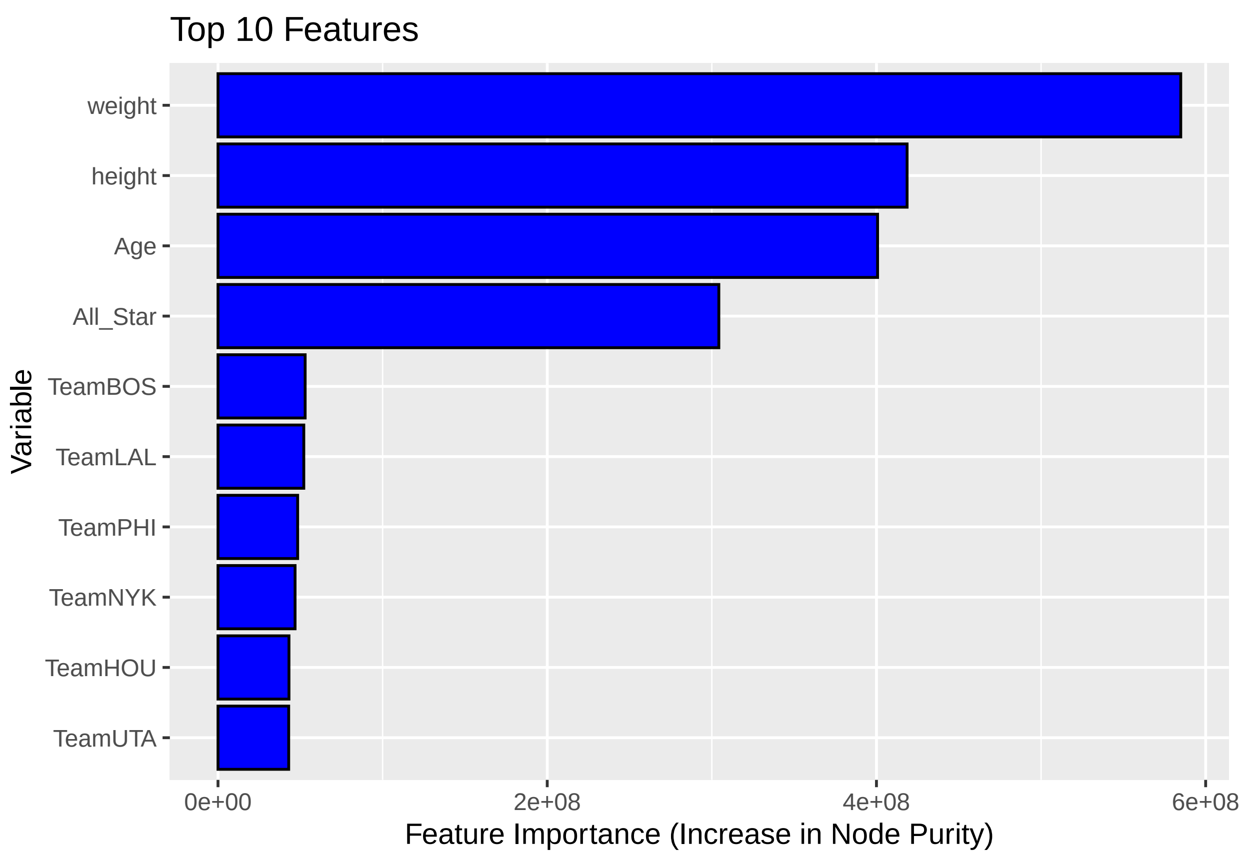
Step 3: Evaluating the Performance and Describing Findings

This chart shows the model name and respective RMSE value for each of the models that I built in this analysis. To choose the best model for use in predicting a player's points scored in a future season given their data, I would personally pick the Random Forest Regression model. This model got the smallest RMSE with only 390.6675 and is still not overfitted to the data because of the low number of trees used. Ensemble methods are very powerful for regression, and because of this I think the Random Forest model is the best overall candidate for the final prediction model. I am confident in this model's ability to be able to accurately forecast how many points an NBA player is expected to score during a season, as long as it has access to all the data inputted into the model.

```
##                        Model     RMSE
## 1 Multiple Linear Regression 426.4662
## 2             KNN Regression 423.1054
## 3              Random Forest 390.6675
## 4             Non-Linear SVR 427.3704
## 5                 Linear SVR 433.3105
```

```r
feature_importances = importance(RFModel)
var_names = colnames(training_set[-1])
feat_imp_plot_data = data.frame(var_names, feature_importances) %>% arrange(desc(IncNodePurity)) %>%
  filter(IncNodePurity>=42951190)

ggplot(feat_imp_plot_data, aes(x = IncNodePurity, y=reorder(var_names, IncNodePurity))) +
  geom_bar(stat = 'identity', color = "black", fill = "blue") +
  labs(title = "Top 10 Features", y = "Variable",
       x = "Feature Importance (Increase in Node Purity)")
```



This plot shows the feature importance values (in terms of increase in node purity) for the top 10 most important variables from the optimal model. It looks that weight, height, age, and all-star status are all very predictive of the amount of points a given player will score in a season. THe teams that the players are on are still predicitive of season points, but not nearly to the scale of the former variables mentioned. It seems that physical build and age of players is more important than I would have guessed and actually is more important than all-star status as to scoring ability of the player. This somewhat surprised me, as I would think that all-star players would be top performers and that they would be able to overpower this phyiscal traits, but it also could be a problem with imbalanced data. There are a lot less all-star players than non all-star players in the dataset, which could have caused this feature to become undervalued in the model for predicting season points. It makes sense that the team the player is on has a small role in the number of points a player scores. If a given player is on the court with extremely talented or untalented teammates, it will affect the players ability to get the ball in shooting positions and therefore affect the points scored throughout the season.