Basketball End-Game Analysis

Carl Larson

5/22/2018

Outline

Problem statement:

Whether you're a coach or a fan, when watching basketball, the question naturally arises in many games, "Is it garbage time yet?"

Further, how can we derive a percent confidence for this?

This is a binary classification problem - is it garbage time or not?

The risk of star players being needlessly injured (or even fatigued) is a big one, and teams are looking to minimize that risk, while still maximizing the likelihood of a victory.

Conversely, Las Vegas oddsmakers are hard-pressed to quantify the mathematical end-game scenarios of basketball the way those same scenarios have been mapped out in chess and poker, where we often see dynamic odds for each player winning a hand.

Further, it's interesting to fans to note whether a game is still competitive or mathematically out of reach. Fans' time is valuable too, and people would be interested to know if they can turn off a game and move on with their lives, or if they need to keep watching.

Data Story: Why this matters

According to the NBA's estimate, the legal potential market size for basketball gambling could be close to $200 billion [Bonesteel, 2018], but it's a vastly under-quantified arena:

"We might just base our opinion on what we saw in the first half and what we expect in the second half." - Jay Kornegay, executive vice president of race and sports at Superbook Westgate Las Vegas, on setting odds during a game. While football odds are easy at halftime, Kornegay explains, "The NBA schedule might have four games at 4:05 p.m., then you get four halftimes around the same time, and the 5 p.m. or 5:30 games, plus college basketball games. It can really complicate things. You don't get to discuss it, you just have to get numbers up." [Everson, 2015]

Between gigantic gambling markets and record-setting player salaries, the problem statement at hand is becoming a very valuable question to ask.

The story below casts a spotlight on the risks a coach runs when he can't accurately quantify his team's victory likelihood, or thus predict whether his team can coast to a win with bench players or whether they need their star player in for the final minutes.

Derrick Rose grew up in Chicago, and after being drafted with the number 1 pick in the 2008 NBA draft by the Chicago Bulls, he instantly became the hometown favorite. It's rare in sports that players represent their hometown team, and with a fresh $190 million endorsement deal with Adidas, Rose was on a track of stardom parallel to the great Michael Jordan.

Rose went on to win Rookie-of-the-Year in 2009, and then the league Most Valuable Player (M.V.P.) award in 2011.

On opening day of the playoffs in 2012, Derrick Rose's Chicago Bulls were the number-1 ranked team in the Eastern Conference, even ranked ahead of the defending champion Miami Heat led by Lebron James.

Hosting the 8th seed Philadelphia 76ers in game 1 at Chicago's United Center, Rose and the Bulls found themselves up by 83-66 with just over 10 minutes remaining to play.

The Bulls' coach Tom Thibodeau chose to keep (at that point, the somewhat injury-prone) Rose in the game all through the 4th quarter, until they were in the final 80 seconds, with the score at 87-99, when Rose landed wrong on a jump and fell to the floor writhing in pain, as Rose's teammates had to foul to stop play a dozen seconds later.

The Bulls would go on to win game 1 with Rose carried off the court, but as Rose sat out the remainder on the bench injured, the *76ers* would then go on to make a historically rare comeback and win that series; an 8-seed beating a 1 seed has happened only about a few times in NBA history.

Rose's torn ACL kept him out for the entire following season. The Bulls team began to falter as key player contracts expired, and the team fell apart. The Bulls have not just slumped, but they have become one of the worst-performing franchises in the NBA today [Friedell, 2017], marking a cliff-like drop-off of success after Rose's injury in 2012.

The day of his injury, Rose was in the 2nd year of a jaw-dropping $92 million 5-year contract with the Bulls.

By the 2018 off-season, Rose had just finished a $419k 1-year contract with the Minnesota Timberwolves. Needless to say, these avoidable injuries are bad for players pockets, as well as teams, and fans.

Coaches are juggling a huge amount of dynamic information in the basketball end-game, and it's easy for emotion to get in the way of good coaching decisions. But if the "Moneyball" mindset has taught us anything, it's that actual data insights can improve executive level sports decision-making that previously relied on intuition or experience.

This project aims to mitigate needless injury to relatively high-value basketball players, representing potentially dozens or hundreds of millions of dollars in the air per year.

Indeed, NBA players have become the highest-paid professional athletes in sports today [Badenhausen, 2016], and there has been relatively scant attention paid to end-game NBA "Garbage Time"" scenarios.

Below are two definitions for some key terms that will be used in this paper (please note, a glossary is included at the end of this paper; these are just two especially vital terms).

Definition: Garbage Time

NBA Hall-of-Fame announcer, Chick Hearn, is famous for coining the phrase, "Garbage Time" [Carlson, 2002], which means the time at the end of a basketball game when the score difference is too large for the game to be competitive. Having a stronger definition for Garbage Time is key for basketball statistics.

Once Garbage Time is declared, it can't be taken back (like the garbage truck taking your rubbish at the curb, there is no undo button in declaring 'garbage time').

Definition: Gameflow Table:

The “Gameflow Chart” is basically an expanded total score but shown through time, over the whole game. It’s a line chart with time on the x-axis, and score on the y (optionally the score of both teams can be depicted, or the score difference).

“In contrast to the usual score by quarters (or by half in college play), the graph easily allows complete identification of items of interest, such as largest leads, lead changes, come-from-behind activity, and percentage of game time for which a particular team of interest held a lead.” [Westfall, 1990].

So rather than just reporting score at the end of each quarter like a basic box score, the Gameflow Chart shows the score in real time though the game as teams make their runs and go on slumps. It tells a detailed story of the game.

Given the Minutes Remaining, and the score, it’s possible to use this classifier to help head coaches optimize declaring Garbage Time as early as possible to *minimize injury risk*, while still *maximizing* likelihood of winning the game.

Mitigating injury risk means finding when it is safe to take star players out of the game to prevent injuries or fatigue.

Below is a histogram depicting the total number of players in columns (or buckets) $5 million wide.

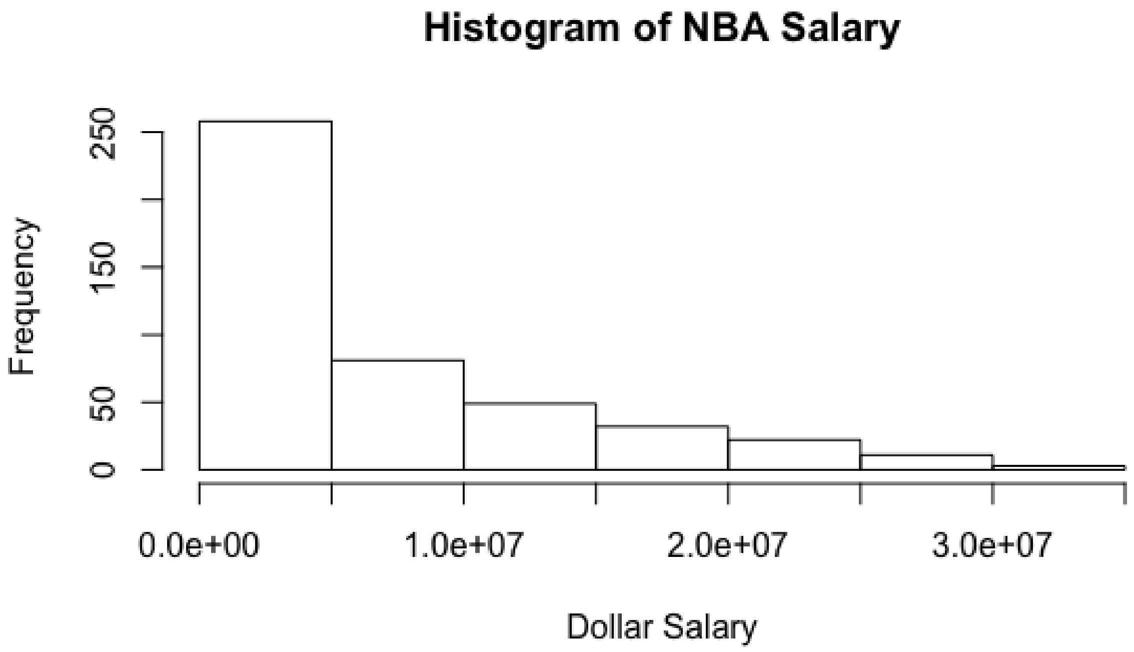


Figure 1: Heaviliy skewed-right

Below we see a glimpse of NBA salaries broken down by age as a scatter plot. source.

This view is helpful, because it spreads out the data over a time-related x-axis so we can really get into the details, while still maintaining a scatter-plot format, which provides an ideally high density, precision, and robustness of information.

Basketball being a timed sport, the teams can only score at a certain rate, which this project inspects in detail in the Regression Addendum. Thus there is an envelope where the game goes from being classified as

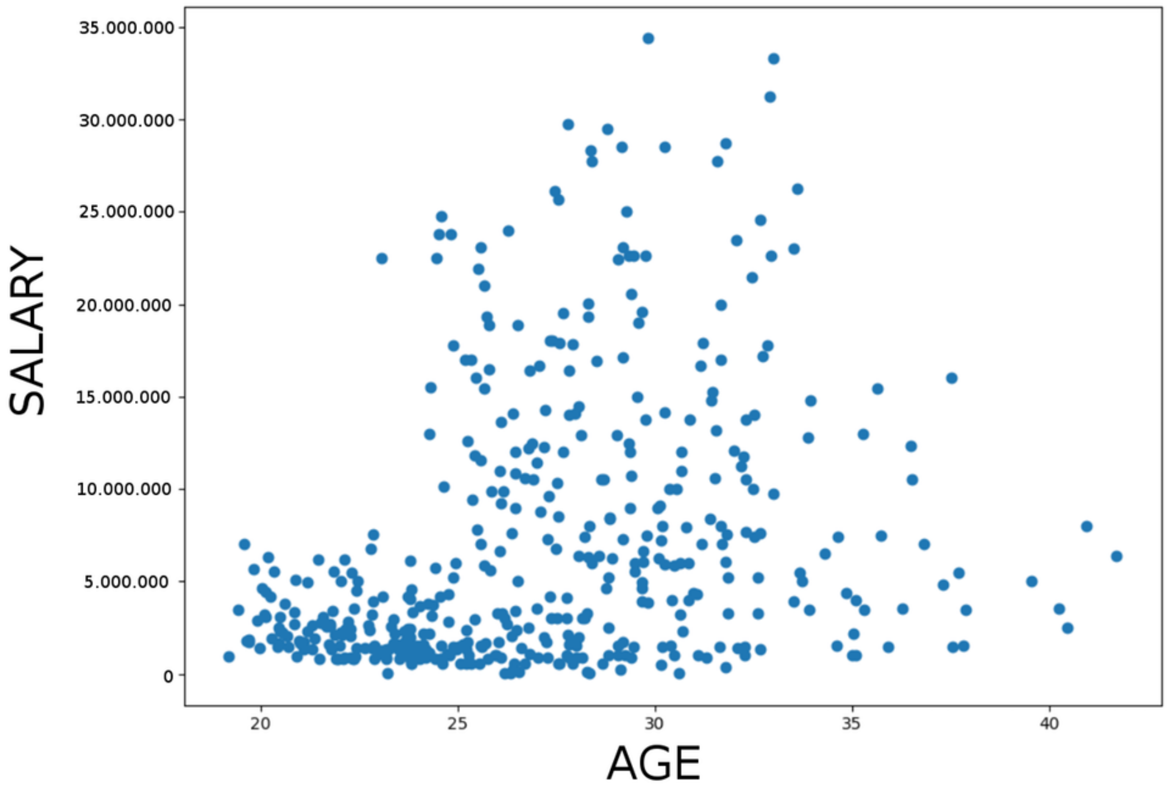


Figure 2: Like all salary data this is right-skewed, or skewed-out, away from the graph-origin

competitive, to out-of-reach for the trailing team simply due to the speed of time, and the speed of basketball.

Aside from any considerations about skill or player chemistry, this analysis takes a scoreboard-only approach in simply communicating that mathematical envelope where *time itself* becomes the impossible obstacle, regardless of any player past performance (which is never a perfect indicator for future performance anyway).

```
# This code snippet sets up the below code snippets for success.

require(shiny)
```

```
## Loading required package: shiny
```

```
require(data.table)
```

```
## Loading required package: data.table

## Warning: package 'data.table' was built under R version 3.4.2
```

```
require(dplyr)
```

```
## Loading required package: dplyr

## Warning: package 'dplyr' was built under R version 3.4.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:data.table':
##
##     between, first, last

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

Explanation of User-Input Variables

Score difference and Minutes remaining, above, are variables only to frame the y-axis and can't be too wide or it can overwhelm a laptop-grade computer. Minutes-Left should be between 3 and 8, and Score Difference should be between 4 and 25. But ultimately, part of the fun is creating your own example numbers to see how it runs differently, and how the eventual contour map is affected by say, changing the framing score difference used. The contour map is written to display uncertainty as "water," and certainty as "high ground," so in a sense, keeping everything else constant and rendering contour maps for decreasing score differences is like watching a topography flood with water from space.

These variables have been set to workable defaults, and hashed-out are lines of code which would allow users to be able to enter this data from their console. Ultimately, this snippet was functionally obsoleted by the UI inputs of the Hooply app itself, but before Hooply was made, these were those inputs.

"MinutesLeft"

A placeholder to frame the time axis of the ultimate chart.

"ConfThresh"

Standing for confidence threshold, which is a linear bias adjustment. Setting this variable to 80% is more conservative (less risky of player health, and more risking of the game outcome, i.e. pre-season); 99.9% is more aggressive or risky of player health (i.e. crucial playoff game).

GameLength

Simply regular game length in minutes, though it should be functionally irrelevant whether 40 or 48 or whatever since this is a purely end-game analysis, just needing consistency throughout.

Project Goal: Inputs and Outputs

The goal of this project from the inception was about quantifying and mapping the space between very competitive games and wide blowouts where one team is the clear winner. Between those two extremes exists the garbage time envelope, which this project intends to map and easily convey for users.

The goal of this document itself is to also show how this project come together, as well as provide the key R code snippets that went directly into building this application.

App User-Inputs:

The front end inputs for Hooply from the user are as follows

1.) minutes remaining 2.) seconds remaining 3.) team A score 4.) team B score 5.) confidence threshold

App Outputs

- % of games where the leading team won ("win chance")
- chart showing user location on the contour plot of time-remaining vs score-difference (as x and y axes) yielding win chance on the z-axis
- Yes-or-no classifier directly answering the problem statement

Data Collection vs Generation

After experiencing frustration seeking large quantities of game-flow tables that can be analyzed in R using data mining, the decision was made for this project to use simulated basketball data based on a model rather than mined data, which creates the need to run model validation checks. Data mining is one obstacle for this type of analysis.

The Garbage Time App: Hooply

For our users, a data product-prototype was developed and named "Hooply."

Using reactive Shiny R, which allows users to key in their specific time-remaining, and score for each team, Hooply delivers a responsive recommendation on whether the game is in garbage time, and also shows a contour chart detailing the other "nearby" scenarios on the x-y map of time and score-difference combinations.

How This App Was Made

This app was made to create a data science recommender product for basketball coaches and fans, so they could know whether the game was mathematically decided or not, given the average rates at which teams score.

Below are the functions that were built to be able to create this app. This broke down into the following steps:

- 1.) The Gameflow-table maker function
- 2.) Function to read many gameflow tables, and extract and clean the data
- 3.) Function to graph and display the results for users

Gameflow Table Generator

Because game-flow charts are difficult to come by in large quantities, this project features a Gameflow Table generator function in r, which generates a game-flow chart from scratch, using R's rnorm, sample, runif, which, and cumsum functions.

This first function takes 3 user-defined variables, probA, probB, and n. These variables come with defaults that end up producing sensible data, but it's left open for users to tinker with. The default for "n" creates a normal distribution that allows games to have a sensible length; "n" below is a place-holder for the number of possessions in the game, or chances for either team to score.

First the need was to quantify the rates at which teams score, and to this end we have the "GameflowMaker4()" R-function which was built using normal distributions for team scoring probabilities through the game. So for each individual game flow table, there was a unique set of values of probability of scoring for each team. This ensures that each simulated game is truly unique, controlled by many different virtual "dice" thrown, which together form the randomized "skill levels" of each team in each simulated game. The nature of the normal distribution emerging from many input distributions is behind the Central Limit Theorem and how standards of deviation are added together. This property binds the range of potential outcomes to normal distributions which keeps simulated values realistic, yet always random.

What Is GameflowMaker4

GameflowMaker4 is a function that creates a basketball game-flow table from random variables, including rnorm, sample, and runif.

5 optional arguments are included, including GL for "Game Length," in minutes; this allows users to change the length of the game to match their level of basketball. Pro-basketball games are 48 minutes, and college games are 40 minutes, for example. This allows coaches to better-tailor this function to their specific length of game, but is mainly cosmetic since this is an end-game analysis.

The function uses "abs()", because if the user changes probA or probB such that any of the below values become negative, it will break the code. Please note probA and probB are pre-assigned to be values that will virtually never yield zero or negative results, based on the means and standards of deviation.

Minimum delta t

The issue of having a minimum increment of time between scores was key. So listA serves this purpose, by storing a sequence of the minimum distance between each action. Then the later rnorm() function is added to 'listA,' randomizing the output, and preserving the original distances (t) between each value.

Conceptually speaking, this prevents having too many scores too quickly due to random chance, which would eventually allow a small percentage of teams to come back in an unrealistically aggressive way. This fact of basketball physics is represented as a minimum delta t, representing the minimum amount of time that has to elapse before the next team scores.

nrorm() Usage

Supported by the 1976 Mertens and Zamir paper on game theory between two competitors in zero sum environments, normal distributions were used as the foundation of the GameflowMaker4 simulator. This makes intuitive sense according to the Central Limit Theorem [Mertens and Zamir, 1976].

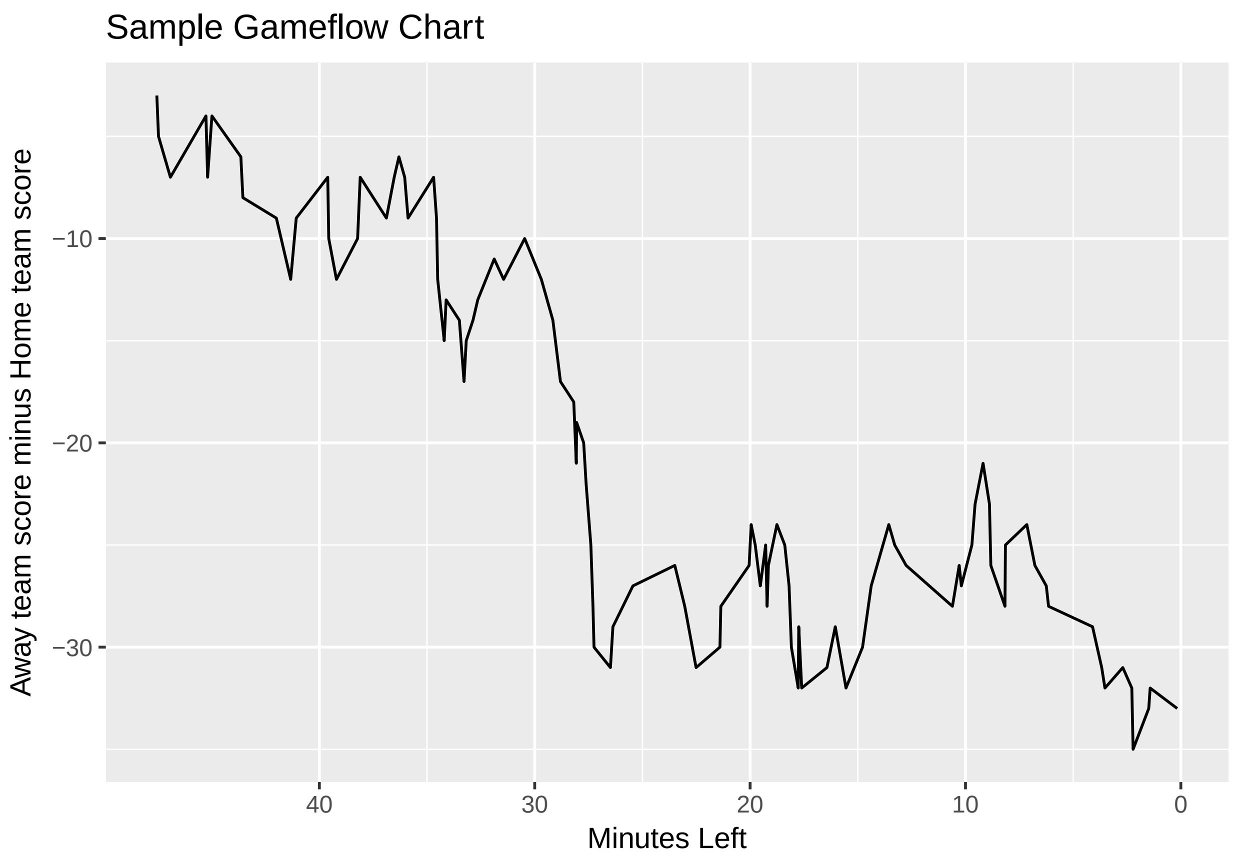
With R's rnorm() function, the first argument is the number of variables to yield, the second is the average, or mean, and the third argument is the standard of deviation. A relevant normal curve is created and 'n' variables are returned from that curve. It is a very useful function in the R language when dealing with random variables that correspond to comparisons between people, such as IQ or SAT tests, personality tests, or in this case basketball ability of a team. rnorm() ensures we will have unique teams, with unique skill levels, playing each game. This guards against algorithmic bias, because we create different values of probA and probB for each chart.

```r
# A crude but effective basketball game simulator

GameflowMaker4 <- function(
  # Creates a matrix serving as a gameflow table (simulation) of a basketball game
  #
  # Args:
  #   probA: the chance team A (away team) will score
  #   probB: the chance team B (home team) will score
  #   n: the number of possessions in the game
  #   dt: a minimum amount of time between made baskets
  #   GL: a time variable for game length in minutes
  # Returns:
  #   Time, ScoreA, and ScoreB, the score of each team at each time
  probA = c(abs(rnorm(1, 0.65, 0.005)),
            abs(rnorm(1, 0.09, 0.002)),
            abs(rnorm(1, 0.17, 0.005)),
            abs(rnorm(1, 0.09, 0.005))),
  probB = c(abs(rnorm(1, 0.64, 0.005)),
            abs(rnorm(1, 0.10, 0.003)),
            abs(rnorm(1, 0.17, 0.005)),
            abs(rnorm(1, 0.09, 0.005))),
  n = as.integer(abs(rnorm(1, 218, 22))),   #ad hoc game-length index variable
  dt = 0.005,   #min time between scores to allow for physics
  GL = 48){   #game length in minutes; less important in end-game analysis
  sta <- sample(0:3, n, replace=TRUE, prob=probA)
  stb <- sample(0:3, n, replace=TRUE, prob=probB)
  w <- which(sta > 0 & stb > 0)
  sta[w] <- 0
  stb[w] <- 0
  v = which(sta == 0 & stb == 0)
  sta <- sta[-c(v)]
  stb <- stb[-c(v)]
  m <- length(sta)
  listA <- as.vector(seq(from=0, to=((m * dt)-dt), by=dt))
  listB <- as.vector(sort(runif(m, 0, c(GL-listA[m]))))
  MinLeft <- listA + listB
  MinLeft <- rev(MinLeft)
  scoreA <- cumsum(sta)
  scoreB <- cumsum(stb)
  df = data.table(MinLeft, scoreA, scoreB)
  if(df$scoreA[nrow(df)] == df$scoreB[nrow(df)])
    df <- df[-c(nrow(df))]
  return(df)
}
```

Above the function is outlined, and below is a sample call of the function with a graph using ggplot.

```
# Runs and graphs the function above
df <- GameflowMaker4()
ggplot(df, aes(x=df$MinLeft, y=(df$scoreA-df$scoreB))) +
  geom_line() +
  labs(y="Away team score minus Home team score", x="Minutes Left") +
  ggtitle("Sample Gameflow Chart") +
  scale_x_reverse()
```



This function, Gameflowmaker4() essentially was the heart of the Hooply app; with each "heartbeat" a simulation was performed, where at time "x" and score difference "y", representing the current time and score difference respectively, some "m" number of games are simulated, say a hundred or a thousand, or even 10,000 as was the case for this project. The default "m" value below is set to 50 to keep computation loads small by default, but bumping "m" up by one or two zeros does not hang the program, it just takes an extra 5 or 10 minutes to finish rendering.

Running the above code snippet shows how a game-flow chart works for this simulation, and is necessary to establish this function or later usage in this R Markdown document. The inputs are various possible user-defined arguments that control normal distributions that combine to form random variables.

The output of this function is a single matrix with 3 columns and with a row for each unique made basket.

As you can see above, this creates a typical game-flow chart, with a few mentor-stipulations for this project, which were:

- to delete from the matrix all rows where both teams did not score
- to (of course) delete any rows where both teams did score
- to add a minimum amount of time (dt as discussed above) that had to elapse to allow for the ball to get up the floor physically

Next we need to loop with this function to create a new function that simulates 300 relevant games for the user.

Exploratory Visualizations

The first graph used to explore this data (via simple functions like GFMapper and ContMapSlicer) happened to be a style of presentation already published by Franklin Kenter in a paper ESPN would later call "hacking basketball."

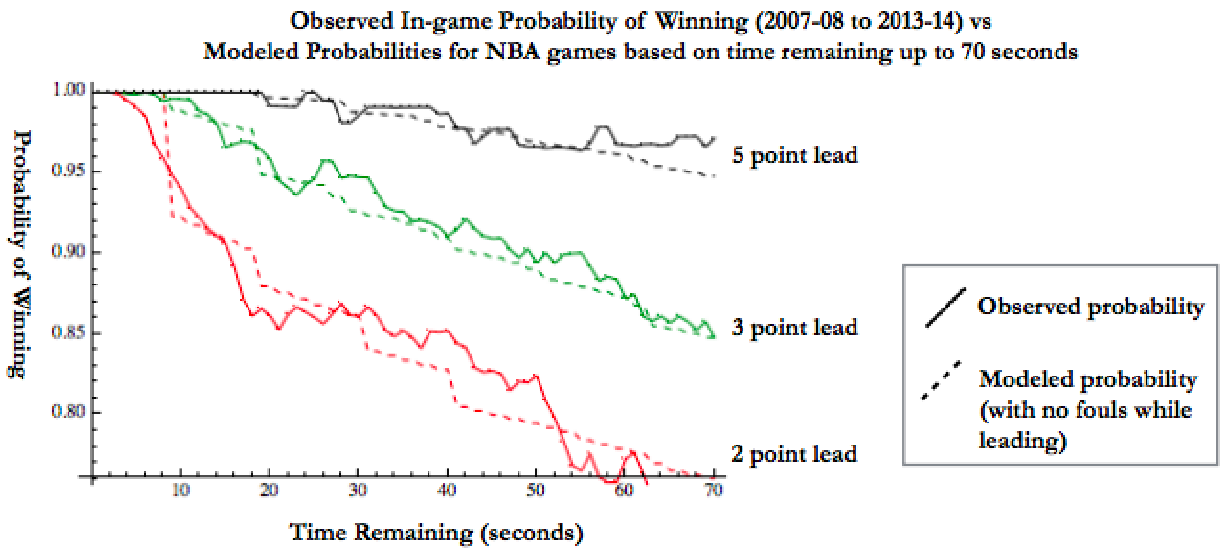


Figure 3: Kenter's Choice of Axes on Figure 1 of his paper frame the graph as points difference and time

Above we have a brief review of how this data can be sliced.

With Goel's data, he breaks down "Moneyball" author Michael Lewis's approach to this problem, in offering a rule of thumb that if a team is up by as many minutes as there are left, there is an 80% chance of winning for the leading team (Lewis, 2004). Goel finds that rule breaks down under 6 minutes, and offers a different rule of 2*sqrt('Minutes Left') as the conversion to score, still holding 80% confidence constant.

Goel's analysis doesn't depart from this 80% metric, and of course Hooply is there to explore metrics above 80% or to a user's discretion real-time.

Looping Gameflowmaker4

```
# Since this function isolates a scenario, we need to pick a score difference to use as an example.
# h of 30 means the 30th-to-last hoop made in the game representing the general time of our scenario.

algo2app <- function(h = 30, p = 9, n = 90){
  # Runs and records results of Gameflowmaker4 for a specific score-difference and index
  #
  # Args:
  #   h: the index number, counting down made hoops (the h-th to-last made basket)
  #   p: "points" for short, p is the absolute value of ScoreA - ScoreB
  #   n: number of iterations
  #
  # Returns:
  #   Time, ScoreA, and ScoreB, the score of each team at said time
  mec <- matrix(data = 0, nrow = 1, ncol = 5)
  for(i in 1:n){
    df <- GameflowMaker4()
    q <- (nrow(df) - h)
    mec[1,4] <-  mec[1,4] + as.numeric(df$MinLeft[q])
```

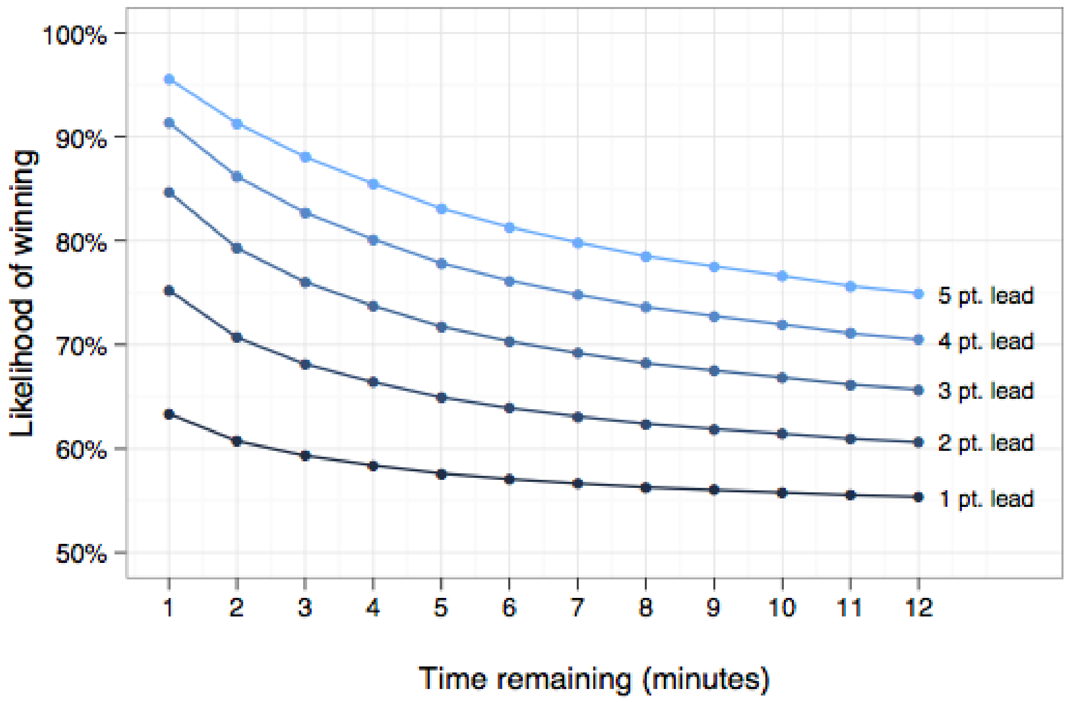


Figure 4: Goel's Contour Plot

```
  mec[1,5] <- mec[1,5] + as.numeric(nrow(df))
  if(((df$scoreA[q]-p) > df$scoreB[q]) && (df$scoreA[nrow(df)] >   df$scoreB[nrow(df)])) {
    mec[1,1] <- mec[1,1] + 1
    next}  #team A led at q and eventually won
    else if(((df$scoreB[q]-p) > df$scoreA[q]) && (df$scoreB[nrow(df)] > df$scoreA[nrow(df)])){
      mec[1,1] <- mec[1,1] + 1
      next}  #team B led at q and eventually won
    else if(((df$scoreA[q]-p) > df$scoreB[q]) && (df$scoreB[nrow(df)] > (df$scoreA[nrow(df)]))){
      mec[1,2] <- mec[1,2] + 1
      next}  #team A led at q but team B came back and won
    else if(((df$scoreB[q]-p) > df$scoreA[q]) && (df$scoreA[nrow(df)] > df$scoreB[nrow(df)])){
      mec[1,2] <- mec[1,2] + 1
      next  #team B led at q but team A came back and won
  } else {
      mec[1,3] <- mec[1,3] + 1 #functional "q-ties" irrelevant
      next}
  if(i==n)
    break}
mec2 <- matrix(data=0, nrow = 1, ncol = 4) #for output metadata
mec2[1,2] <- (mec[1,1]/(mec[1,1]+mec[1,2])) #cannot include q-ties (or use "m") in this denominator
mec2[1,1] <- h
mec2[1,3] <- (mec[1,4] / n)
mec2[1,4] <- (mec[1,5] / n)
colnames(mec2) <- c("Game_Action_index", "Leader_win_chance:", "Avg_sample_MinsLeft", "avg_Game_Actio
return(mec2)
```

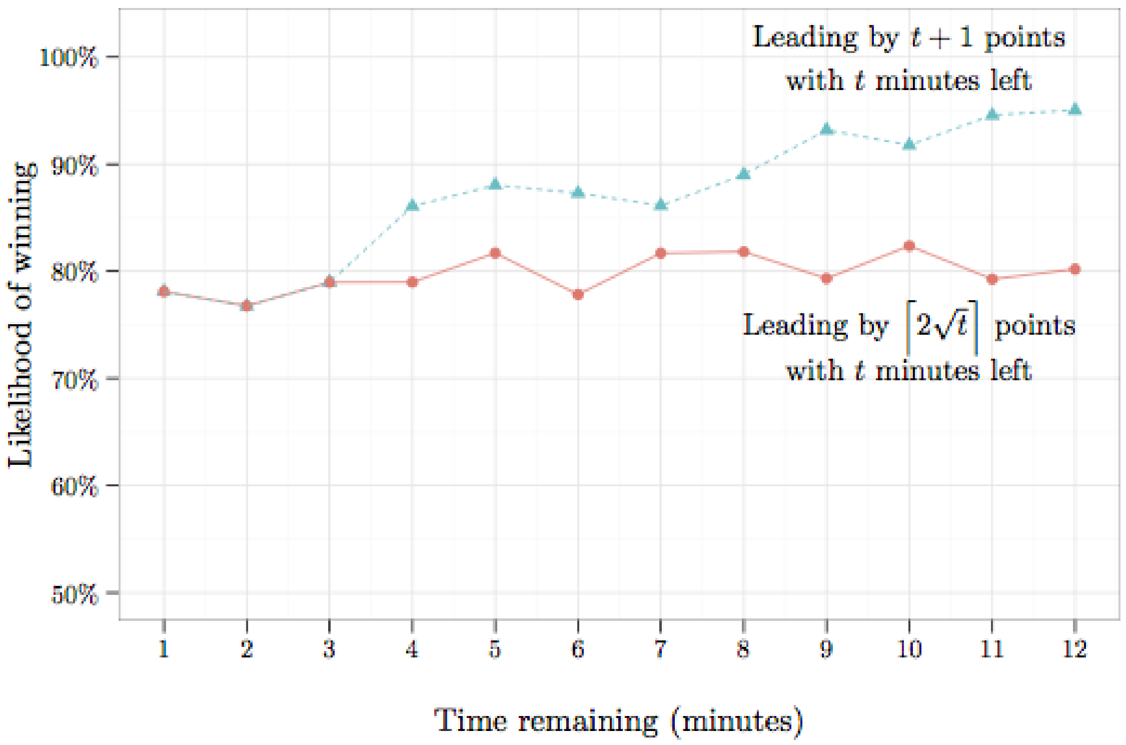


Figure 5: Goel's Formula vs Michael Lewis's Reported Rule of Thumb

```
}
```

This function aggregates key data from "m" -many simulated basketball games. Below we see a time-slice at the 30th-to-last made basket by either team, where given the pre-set score difference of 9 in this case, games are first grouped into scenarios where a team was or was not in the lead by 9 or more points at the 30th-to-last made hoop. Then inside the scenarios where a team *was* in the lead by 9 or more at this time, we measure how often there was a comeback made or not, and re

```r
# Below is a couple of sample runs of this function as an example.
algo2app()
```

```
##      Game_Action_index Leader_win_chance: Avg_sample_MinsLeft
## [1,]                30          0.9818182            14.54098
##      avg_Game_Actions
## [1,]         102.5444
```

```r
algo2app(h = 40, p=2, n=50)
```

```
##      Game_Action_index Leader_win_chance: Avg_sample_MinsLeft
## [1,]                40          0.8888889            20.34982
##      avg_Game_Actions
## [1,]            96.66
```

The above function "algo2app" was one of the first functions used in this project to usefully reduce the dimensions of potentially thousands or millions of game-flow chart-style basketball game simulations into something that can be interpreted in a graph.

Below a for-loop is explored to print out "algo2app" over multiple basketball end-game scenarios. This was still required to complete a "slice" for a Goel style graph where probability of the leading team winning is on the y-axis, and time is on the x-axis, with score difference represented as a line on the graph.

```r
GFmapper <- function(k = 35, p = 9, n = 90){
  # runs the previous function for the index "k" game-action down to 1,
  # creating cleaned data for the next step
  #
  # Args:
  #   k: the index-key of focus of the matrix and eventual graph
  #   p: points difference
  #   n: number of iterations to run
  #
  # Returns:
  #   A matrix with k-many columns outputting leading team win chance
  matx <- matrix(data = 0, nrow = k, ncol = 4)
  h <- k
  for(i in 1:k){
    h <- h
    p <- p
    n <- n
    dfJ <- as.matrix(algo2app(h=h, p=p, n=n))
    matx[i,1] <- dfJ[1,1]
    matx[i,2] <- dfJ[1,2]
    matx[i,3] <- dfJ[1,3]
    matx[i,4] <- dfJ[1,4]
    h <- h - 1
    if(h == 0){
      break
      }else{
        i <- i+ 1
        next}}
  return(matx)
}
```

Depicting Multiple Basketball Games On One Plot

This is graphed below, essentially holding score difference as one constant line, and simulating "m" number of games from the algo2app function.

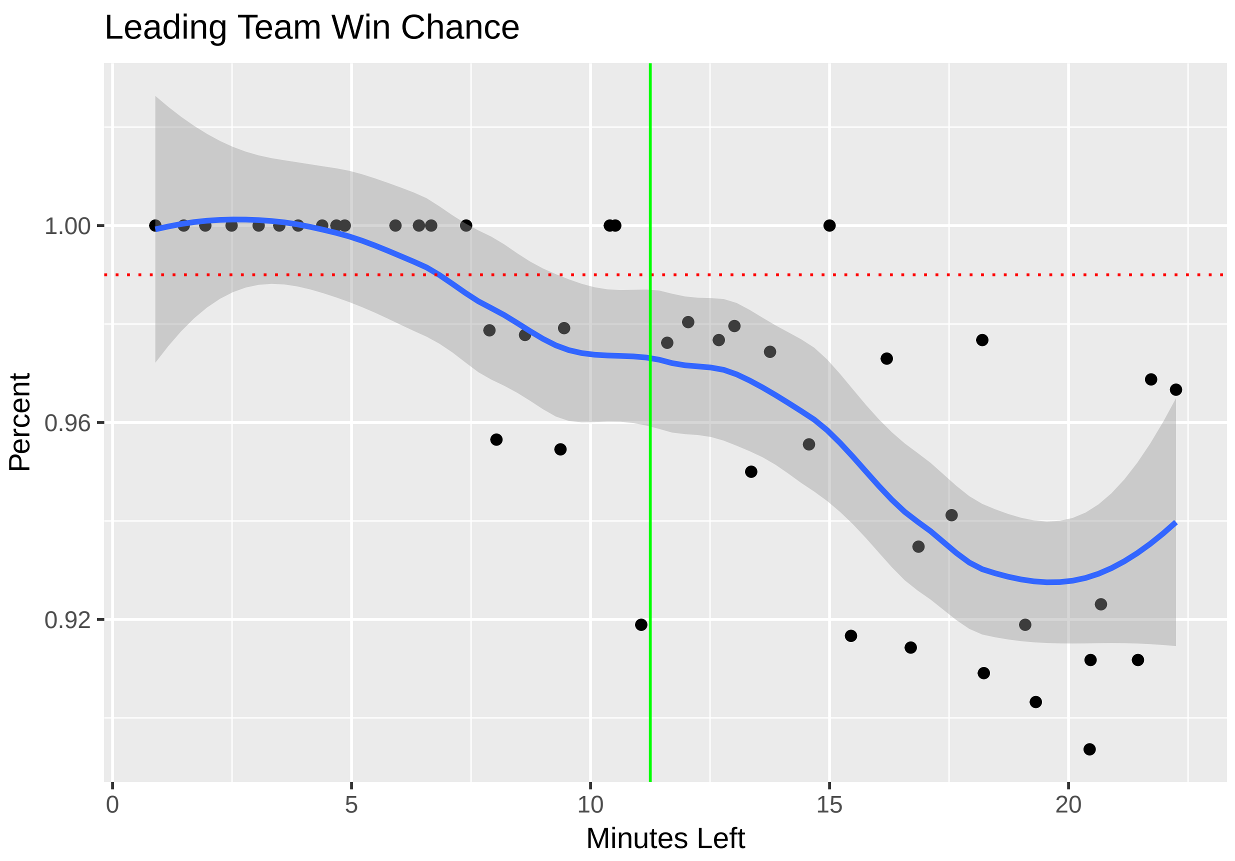
```
ContMapSlicer <- function(k = 35, p = 9, n = 90){
  # This function graphs the function above

  # Args: k, the index

  k <- k
  p <- p
  n <- n
df <- as.data.frame(GFmapper(k = k, p = p, n = n))
colnames(df) <- c("Game_Action_index", "Leader_win_chance:", "Avg_sample_MinsLeft", "avg_Game_Actions")
#plot(df[1:k,3], df[1:k,2], type="b", main="Leading Team Win Chance",
#     xlab = "Minutes Left", ylab="Percent")
ggplot(df, aes(x=df[1:k,3], y=df[1:k,2])) +
  geom_point() +
  geom_smooth(span = 0.6) +
  geom_hline(yintercept = 0.99, linetype="dotted", color="red") +
  geom_vline(xintercept = k/4, color="green") +
  labs(y="Percent", x="Minutes Left") +
  ggtitle("Leading Team Win Chance")
}

ContMapSlicer(k=45, p=13)  #including a sample call of this function
```

```
## `geom_smooth()` using method = 'loess'
```



Graphically this is one slice, or a horizontal cross-section, of the eventual contour map we will plot.

The green line represents the user's actual reported minutes left, and the red dotted line is the 99.5% cutoff for sensible confidence that "garbage time" has begun.

Further, these graphs above are depicting discrete data (score difference) as the graph output (or contour lines). Discrete data makes a much better Y-axis, so this paper will set out to eventually do that and leave the continuous variable, percent win chance, as the graph output, which as a continuous variable, can be much more naturally color coded and mapped out than step-wise score differences.

Model Validation: Overview

So now that we have a model, we can do some basic empirical model validation. Dr. Sharad Goel of Stanford performed an analysis using data collected from 7,000 games over the past decade, to answer these same questions, in his write up, "How Close Is Close?"

"Basketball games can be reasonably well modeled as a series of independent one-minute rounds consisting of approximately one possession for each team. In each interval, the score differential between the teams changes by about two points, with each team roughly equally likely to win that round. In statistical terms, the intervals are approximately mean 0 with standard deviation 2. The central limit theorem then shows that with t minutes remaining, the score gap changes approximately according to a normal random variable..."

Goel's reference to the central limit theorem supports the reasoning from the 1976 Mertens and Zamir paper detailing how zero-sum two-player game outcomes are subject to normal distributions.

```r
# Designed to print out home win percentage running m-many game simulations

HomeWinQApp <- function(m = 200){
  mec <- matrix(data = 0, nrow = 1, ncol = 2)
    for(i in 1:m){
    df <- GameflowMaker4()
      if((df$scoreA[nrow(df)]) > df$scoreB[nrow(df)]){
        mec[1,1] <- mec[1,1] + 1  #away team won
        next}
      else if(df$scoreA[nrow(df)] < df$scoreB[nrow(df)]){
        mec[1,2] <- mec[1,2] + 1  #home team won
      next
      } else if(i==m){break}}
    mec2 <- matrix(data=0, nrow = 1, ncol = 2)
    mec2[1,1] <- (mec[1,1]/m)  #away team record
    mec2[1,2] <- (mec[1,2]/m)  #home team record
    colnames(mec2) <- c("Away Team Record", "Home Team Record")
    return(mec2)
}

HomeWinQApp()  #A sample run is included
```

```
##      Away Team Record Home Team Record
## [1,]            0.415            0.585
```

Fortunately this produces a realistic result, with NBA home team record being 53% empirically most recently.

Model Validation: Comparing This To Goel's Analysis

Sharad Goel's write-up on garbage time in basketball provides a great foundation for how to judge a statistically rigorous basketball game simulator [Goel, 2012]. Goel's write-up, along with a few other metrics, will be used as model validation in this section.

"Basketball games can be reasonably well modeled as a series of independent one-minute rounds consisting of approximately one possession for each team. In each interval, the score differential between the teams changes by about two points, with each team roughly equally likely to win that round." Essentially this is exactly what this paper does, with the regression addendum elaborating.

With the custom function "finalScoreAverager," we can increase the number of Gameflow Tables that go into the figure. The "n" variable below controls the number of games to simulate to use to focus in on a true value for average away and home scores of teams.

```
# Creating a base function to use in a loop
finalScoreAverager2 <- function(n){
  df<-GameflowMaker4()
  AvgScore <- c(df$scoreA[nrow(df)], df$scoreB[nrow(df)])

  return(AvgScore)
}
finalScoreAverager2()
```

```
## [1] 122 116
```

Above we have a function that averages the scores of the home and away team. Below, this function is then looped many times to try to find a sharper picture of the true sample average.

```
# Game flow table score mass averager; works for m-values up past 10,000

GFTsma2 <- function(m = 50){
```

```r
  totalA <- 0
  totalB <- 0
  for(i in 1:m){
  tuple <- finalScoreAverager2()
  totalA <- totalA + tuple[1]
  totalB <- totalB + tuple[2]
  }
  totalA <- (totalA / m)
  totalB <- (totalB / m)
  return(c(totalA, totalB))
  }
GFTsma2(m=1000)  #Sample run included
```

```
## [1]  97.710 100.766
```

Context

Depending on the year, average scores for NBA games have fluctuated between 91 in 1998 and 118 in 1962. Values in this range are acceptable as realistic - Haberstroh has an excellent expansion on this (rather deep and nuanced) topic (that is beyond the scope of this paper).

With this, we can see that the scores correspond to within ranges for empirical NBA average scores, thus providing some additional model validation.

Comparing To Empirical Data

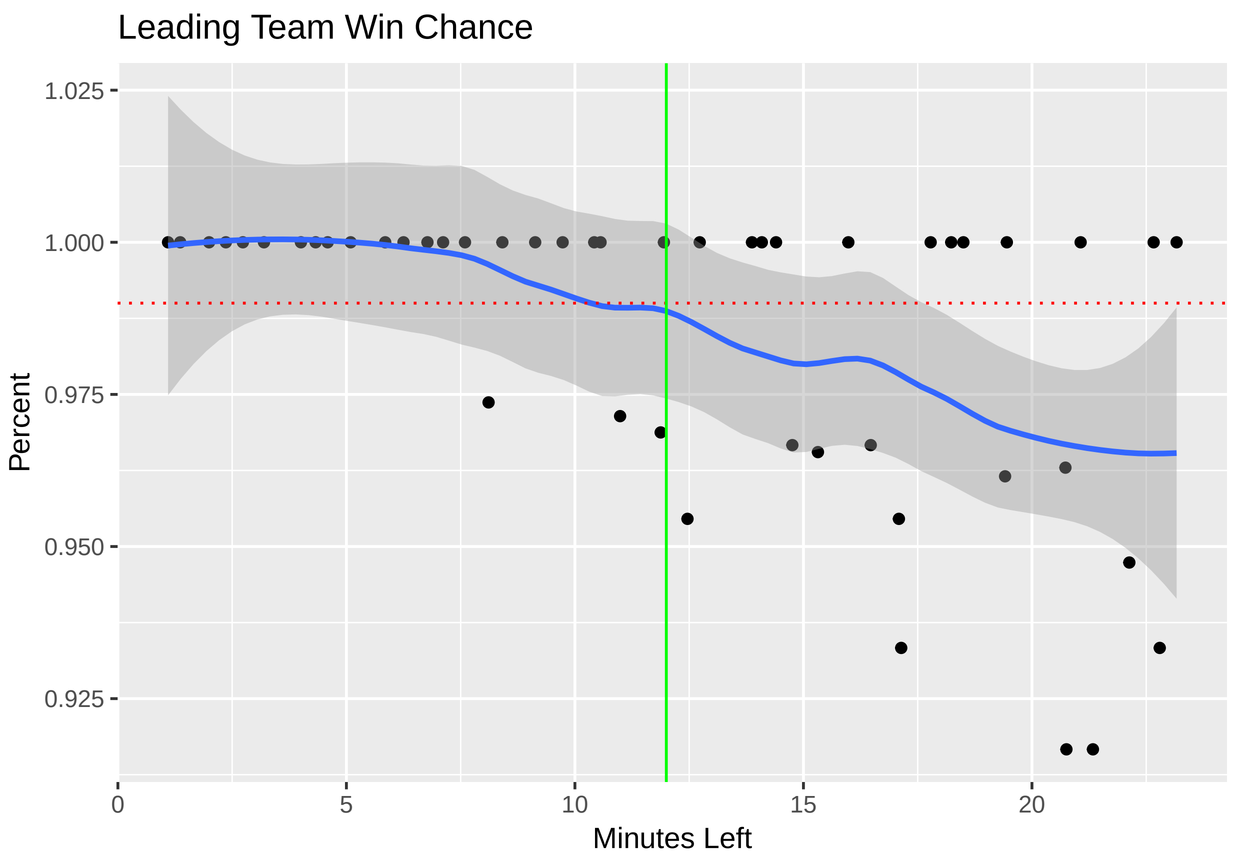
We can now revisit the ContMapSlicer() function from back around line 400 to see what our model shows with regards to Goel's main 3-dimensional landmark, which was:

- 12 minutes
- 7 points
- 80% confidence

This just requires a call to ContMapSlicer() having set scdiff to 7, we can find 12 minutes on the graph and see what percent it suggests.

```r
# The green line represent's the user's time remaining
ContMapSlicer(k = 48, p = 17)
```

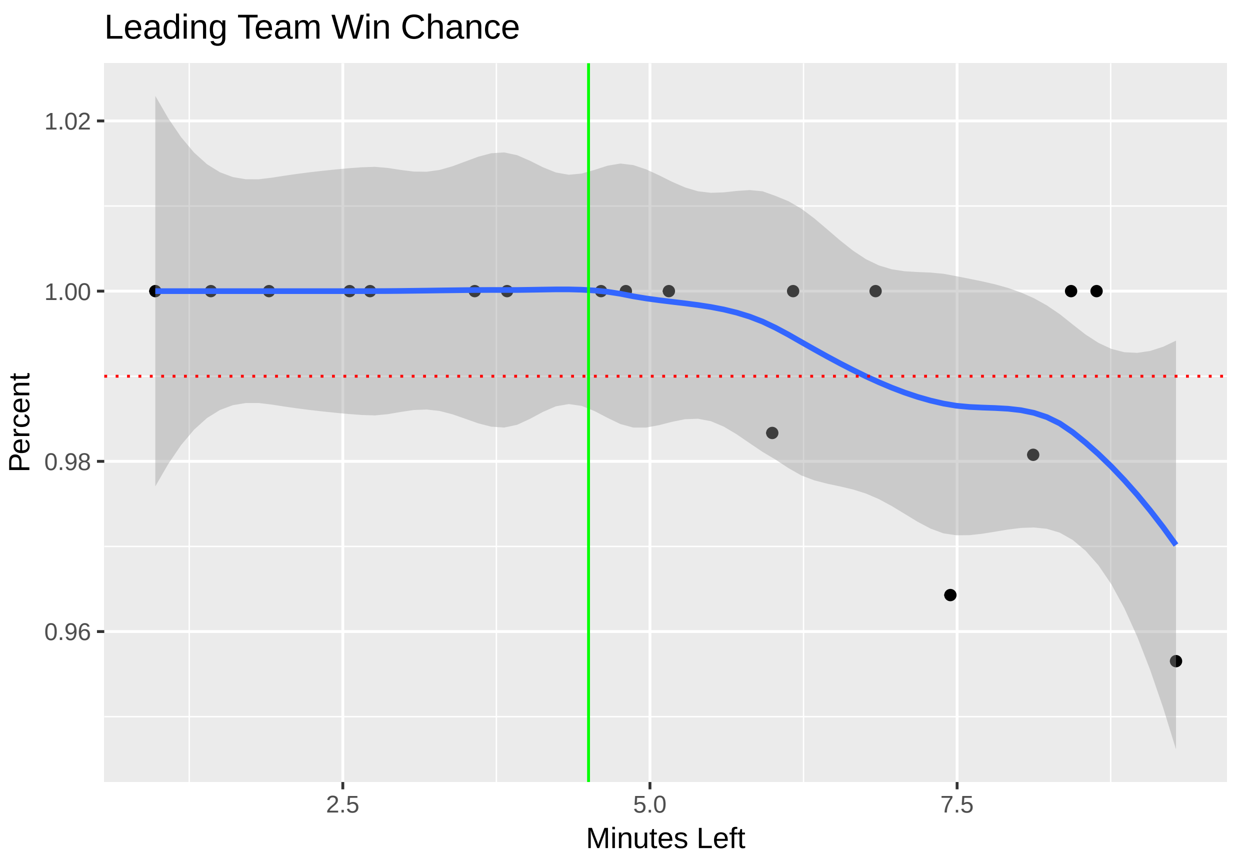
```
## `geom_smooth()` using method = 'loess'
```



```
# Resetting variables to properly frame the contour map.

ContMapSlicer(k=18, p=10)
```

```
## `geom_smooth()` using method = 'loess'
```



This is interesting in that it shows the model is has higher confidence than Goel's data given his landmark of 80% confidence at 7 points lead with 12 minutes. The Hooply model reports confidence of roughly 94% in that particular scenario. Ultimately this could be compensated by the logic of the application itself, given that the threshold for a recommendation is still within our control, that allows users to tune their compensation for any perceived bias in the Hooply data.

In a sense, this model validation shows roughly a 14% bias (80% vs 94%). Will users be able to perceive a 14% bias? It's possible, over the long term. Ultimately, this does require more empirical data and model validation, but it's an interesting window to compare to Dr. Goel's data.

Creating The Contour Map

The contour map itself is a 3 dimensional graph where the "topology" is the percent likelihood of a team winning. Since percent likelihood can be so easily reduced to a single dimension of data, it makes a perfectly natural fit for the z-axis of the contour map shown below, where the x and y axis are framed to depict score-time scenarios that are generally relevant to the basketball end-game.

To do this, first a dimension-reducing intermediary function was created called "algo3app" that applied the Gameflowmaker4() function to a specific value of time and score, and this function is repeated below. Note, "algo2app" was built to the same purpose but output slightly different values than "algo3app."

For "algo3app" to work, however, there need to be "Minutes Left" and "Score" data for each team. So to make that work what's required are some default input values for the function - feel free to change those values and hit "Play" and see how they affect the output.

Applied Dimension Reduction

To reduce the dimension of a large amount of game-flow charts, this function is tailored to create the kind of output that will be required by the z-matrix of our contour plot:

```
#"h" for hoop, "p" for point difference, and m for number of runs per cell
algo3app <- function(h = 12, p = 13, m = 50){
  # Creates a matrix which loops Gameflowmaker4 and
  # outputs data tailored to plug directly into R's contour function.
  #
  # Args:
  #   MinzLeft: variable for user's minutes remaining
  #   scdiff: for the user's score difference
  #   m: the number of simulations per score-time scenario
  # Returns:
  #   Index, leading team win chance, and time
  mec <- matrix(data = 0, nrow = 1, ncol = 4)
  for(i in 1:m){
  df <- GameflowMaker4()
  q <- (nrow(df) - round(h / 0.4847 + 0.4733))
  mec[1,4] <-  mec[1,4] + as.numeric(df$MinLeft[q])

  if(((df$scoreA[q]-p) > df$scoreB[q]) && (df$scoreA[nrow(df)] > df$scoreB[nrow(df)])) {
  mec[1,1] <- mec[1,1] + 1
  next} #team A led at q and eventually won

  else if(((df$scoreB[q]-p) > df$scoreA[q]) && (df$scoreB[nrow(df)] > df$scoreA[nrow(df)])){
  mec[1,1] <- mec[1,1] + 1
  next} #team B led at q and eventually won

  else if(((df$scoreA[q]-p) > df$scoreB[q]) && (df$scoreB[nrow(df)] > (df$scoreA[nrow(df)]))){
  mec[1,2] <- mec[1,2] + 1
  next} #team A led at q but team B came back and won

  else if(((df$scoreB[q]-p) > df$scoreA[q]) && (df$scoreA[nrow(df)] > df$scoreB[nrow(df)])){
  mec[1,2] <- mec[1,2] + 1
  next} #team B led at q but team A came back and won

    else{mec[1,3] <- mec[1,3] + 1 #functional "q-ties" irrelevant
      next}
if(i==n)
  break}
  mec2 <- matrix(data=0, nrow = 1, ncol = 3)
  mec2[1,2] <- (mec[1,1]/(mec[1,1]+mec[1,2]))
  mec2[1,1] <- q
  mec2[1,3] <- (mec[1,4] / m)
  colnames(mec2) <- c("Game_Action_index", "Leader_win_chance:", "Avg_sample_MinsLeft")
  return(mec2)
}
#Below are 2 sample runs of this function as an example.
algo3app()
```

```
##      Game_Action_index Leader_win_chance: Avg_sample_MinsLeft
## [1,]                76           0.952381            12.48848
```

```
algo3app(h=2, p=9)
```

```
##      Game_Action_index Leader_win_chance: Avg_sample_MinsLeft
## [1,]               112                  1            3.093288
```

Note on Regression:

Rather than taking an index as a time metric like in algo2app, algo3app here actually takes the users' actual reported time remaining and automatically converts it (see regression addendum for details).

```
#to find the slope of minutes per hoop
k <- 35
df <- as.data.frame(GFmapper(k=k))
colnames(df) <- c("Game_Action_index", "Leader_win_chance:", "Avg_sample_MinsLeft", "avg_Game_Actions")

X <- df[1:k, 3] #Minutes left
Y <- df[1:k, 1] #output game action guess

LinReg <- lm(X ~ Y, data=df)

print(LinReg)
```

```
##
## Call:
## lm(formula = X ~ Y, data = df)
##
## Coefficients:
## (Intercept)            Y
##      0.4074       0.4907
```

```
#Please note running this snippet will yield slightly different coefficients every time,
#and it's just helpful to be consistent about it and less important the exact thousandths
r_coef_A <<- 0.4847
r_coef_B <<- 0.4733
```

Given the above, we can now easily call a function "gftopo4," which maps a contour map of game-flow outcomes as sliced by the function "algo3app," which itself is a dimension reduction of our original GameflowMaker4 core-simulation.

Mapping Comebacks / Comeback Likelihood

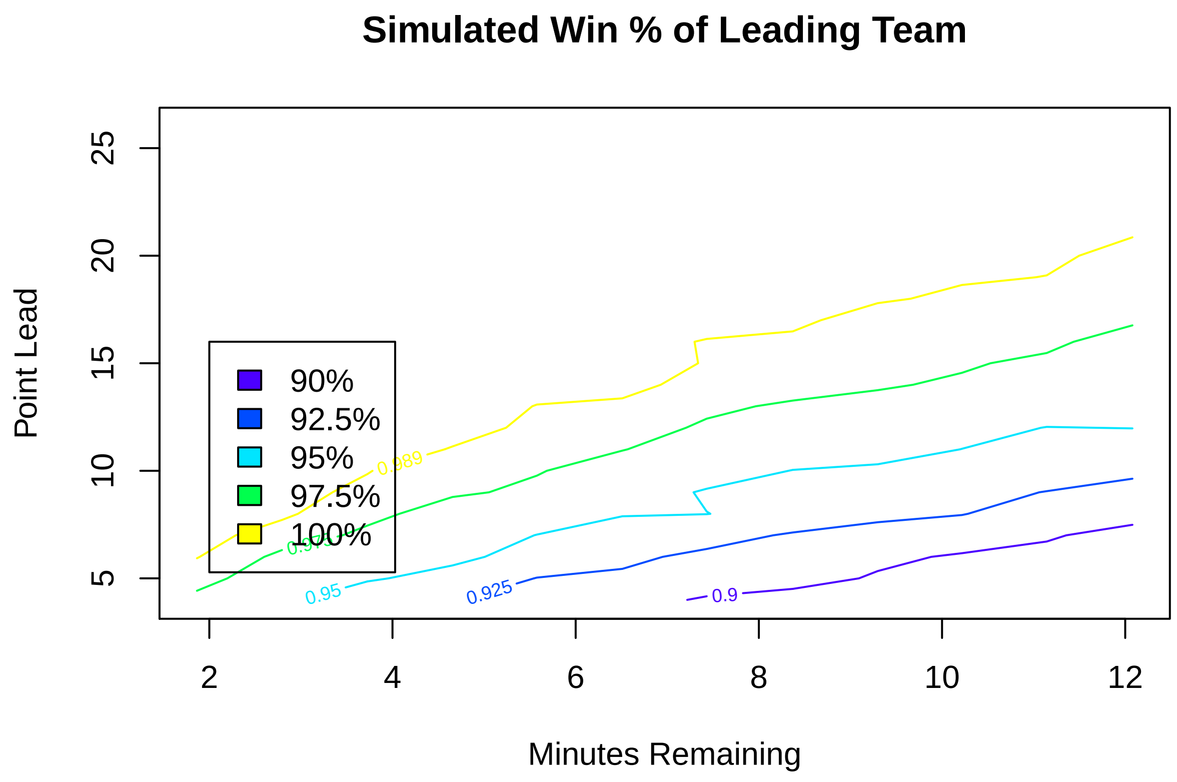
The graph below depicts the quotient of the number of times the leading team at the x,y scenario ended up winning the game.

Basically, given say, 10 minutes remaining, you can look up and see how much of a lead a team would need to start feeling confident about the outcome of the game. With 10 minutes remaining, a team would need to have over a 20 point lead to feel like it was garbage time.

```
gftopo4 <- function(h=12,p=13, m=60){
  # Runs algo3app m-many times per score-time scenario, creating 3
  # matrices, one for the x, y, and z variable of the contour function.
  #
  # Args:
  #
  #   h: minutes remaining (frames x-axis)
  #
  #   p: team point difference (frames y-axis)
  #
  #   m: number of simulations to run per cell (frames 'z-axis', mitigates contour heteroskedasticity w
  #
  #
  # Returns:
  #   One contour plot mapping basketball end-game win probabilities.
  MinzLeft <- 4.5
  scdiff <- 10
  #x_framer <- round(1.67 * (MinzLeft / r_coef_A + r_coef_B))
  y_max <- (p *2)
  y_min <- round(p / 4)
  i <- 1 #x index set before looping
  #dfz <- matrix(data = 0.50, nrow = x_framer, ncol = (y_max - y_min)) #z matrix
  dfz <- matrix(data = 0.50, nrow = h, ncol = (y_max - y_min)) #z matrix
  xax <- matrix(data = 0, nrow = h, ncol = 1) # x axis spacing
  yax <- as.vector(seq(from = y_min, to = y_max)) #framing a y axis
  for(i in 0:h){
    xvr <- 0  #average calculator, x-average
    j <- 1 #y index set before looping
    for(j in 1:(y_max-y_min)){
      df2 <- matrix(data=0, nrow=1, ncol=3)
      df2 <- algo3app(h=i, p=j, m=m)
      xvr <- xvr + df2[1,3]
      dfz[i, j] <- df2[1,2]
      if(j < (y_max+1)){
        j <- j + 1
      } else {
          break
        }
    }
    xax[i] <- xvr / (y_max + 1 - y_min)
    if(i <= h){
      i <- i + 1
    } else {
      break}
  }
  contour(x = xax,
          y = seq(from = y_min+1, to = y_max, length.out = ncol(dfz)),
          z = dfz, levels = c(0.9, 0.925, 0.95, 0.975, 0.989),
          col = topo.colors(5),
          xlab = "Minutes Remaining", ylab = "Point Lead")
          title("Simulated Win % of Leading Team", font = 4)
          legend(x=2,y=16,legend=c("90%", "92.5%", "95%", "97.5%", "100%"),
          fill = topo.colors(5), col = topo.colors(5))
#  write.csv(dfz, "dfz4.csv")
#  write.csv(xax, "xax4.csv")
}
```

Note that the above function, gftopo4, is very computationally expensive, and for m-values over 500, requires a dozen or more minutes to run on a MacBook laptop, and for values over 12000 borders on crashing the computer.

```r
# A sample call of gftopo4 running m-many simulations per scenario-cell
gftopo4(m=9000)
```



This above call to gftopo4 is expensive to run, and it would take a typical laptop hours, but this is what was used to create the outputs used in the current version of the Hooply app. There are 23 columns and 13 rows in the z-matrix for Hooply, meaning that this ran roughly 2.6 million games, which is well more than the roughly 400,000 NBA games that have ever been played in history.

The app is hosted on the ShinyApps.io server via this link.

There is a backup version of the app hosted at this link as well.

The data that goes into the contour map was a computational bottleneck (in other words - not something to attempt on the user's machine). The above function was run with a "MinzLeft" of 3.1, a "scdiff" or score difference of 9, and then looped over an "m" value of 9,000 times. This signifies 9,000 basketball game-flow tables generated and analyzed for each of the 13 by 23 map (13 * 23 = 299) of relevant end-game score-time combinations. This was a double for-loop where the y-loop was nested inside the x-loop for the time-variable, or 23 different score scenarios for each semi-randomly selected "moment," or average time. Those specific values of score difference and time only frame the eventual graph, and aren't meant to refer to a specific user.

Product: Hooply- Basketball's 'Garbage-Time' App

The final step was to convert this presentation of data into an actionable recommender application that was easy to use.

Below is the current prototype for the Hooply web application, which can easily be run and hosted out of this R Markdown document as a viable Shiny Application. The data for the app was taken from the raw .csv outputs of the above "gftopo4" function, and read into the "matrix()" function below.

Additionally, the x-axis was reversed to create this app after some brief user testing revealed it to be more intuitive to keep time flowing left-to-right as shown in a typical game-flow chart.

```r
#
# This is a Shiny web application by Carl R. Larson. You can run the application by clicking
# the 'Run App' button above or by visiting it at bit.ly/hooply2.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#
# Back-end Inputs: the outputs of gftopo4()
# Front-end Inputs: Score difference of the 2 teams, scoreboard time,
# and confidence interval
# Outputs: the responsive web app Hooply in HTML5 via R Shiny
#
# Detail: After running the gftopo4 function or otherwise creating plausible
# values for x and z, we can now run the app here which uses
# those 2 matrices (1 of which is a vector, x) to build the contour plot
# that serves as the heart of Hooply's user-facing visual interface. The contour
# plot function requires x, y, and z arguments of acceptable dimension.
# We import the z-matrix which is the most complex by far, and the x-axis,
# which has 17 unique non-integer values. The y-axis is simply integers from 3 to 20, which
# is easier to re-create as 'seq(3, 20)' than saving and calling a whole extra csv file.
#

library(shiny)



# Defining UI
ui <- fluidPage(
  # App title
  titlePanel("Hooply"),

  headerPanel("The Basketball 'Garbage Time' App"),

  # Sidebar area
  sidebarPanel(

    h2("Is It Garbage Time Or Not?"),

    h4("Please Select Your Scenario:"),

    numericInput(inputId = "ScoreD",
                 label = "Score Difference",
                 value=13, min = 0, max = 100, step = 1),

    div(style="display: inline-block;vertical-align:top; width: 150px;",
        numericInput(inputId="WholeMinutesLeft",
                     label="Minutes", min = 0, max = 31, value = 11, step = 1)),

    div(style="display: inline-block;vertical-align:top; width: 150px;",
        numericInput(inputId= "SecondsLeft",
                     label = "Seconds:", min = 0, max = 5900, value = 12, step = 1)),
    numericInput(inputId = "ConfTh",
                 label = "Confidence Threshold:",
                 min = 0.8, max = 1, value = 0.99, step = 0.005)
  ),

  mainPanel(
    # Output: topographical style contour map of probability field
    plotOutput("contourP"),
    textOutput("rec"),
    tags$head(tags$style("#rec{color: black;
                         font-size: 18px;
                         font-style: italic;
                         }"
                      )
    )
    ,
    textOutput("num"),
    tags$head(tags$style("#num{color: black;
                         font-size: 36px;
                         font-style: bold;
                         }"
                      )
    ),
    verbatimTextOutput("blurb"),

    tags$style(type="text/css",
              ".shiny-output-error { visibility: hidden; }",
              ".shiny-output-error:before { visibility: hidden; }"  # this section mutes unwanted erro
    )
    ))

# Define server logic;
server <- function(input, output) {

  output$contourP <- renderPlot({
    dfz <<- as.matrix(read.csv('data/dfz4.csv'))
    xax <<- as.matrix(read.csv('data/xax4.csv'))
    yax <<- (seq(3,20))
    scdiff <- input$ScoreD
    MinzLeft <- -1*(as.numeric(input$WholeMinutesLeft) + (as.numeric(input$SecondsLeft) / 60))
    if (MinzLeft < 0) {
      contour(x = c(rev(xax[,-1])*-1, 0),
              y = yax,
              z = rbind(apply(dfz[,-1], 2, rev), 1),
              levels = c(0.9, 0.925, 0.95, 0.975, 0.989),
              labels = c("90%", "92.5%", "95%", "97.5%", "99%"),
              col = c('darkorchid2', 'royalblue3', 'turquoise4', 'green4', 'gold4', 'grey39'),
              xlab = "Minutes Remaining", ylab = "Point Lead",
              labcex = 3.5)
        abline(h = scdiff, v = MinzLeft, col = "red", lty=3, lwd=6)
        title(main = "Percent Win Chance For Leading Team")
    }
  })
```

```r
output$rec <- renderText({
  scdiff <<- input$ScoreD
  MinzLeft <<- as.numeric(input$WholeMinutesLeft) + (as.numeric(input$SecondsLeft) / 60)
  x_index <<- round(MinzLeft * r_coef_A + r_coef_B)
  y_index <<- as.numeric(scdiff - 4)
if (MinzLeft == 0) {
    print("Game over")}
  else if (MinzLeft != 0 && scdiff < 4.5) {
    print("Warning: The score is too close to make a prediction")
  }
  else if (scdiff > 23) {
    print("Warning: Score difference too large.")
  }
  else if (MinzLeft > 33) {
    print("Please enter 30 or fewer minutes.")  #this section handles error reporting
  }
  else if (input$ConfTh > (dfz[x_index, y_index])) {
    print("No, it is not garbage time - the leaders' % win chance is roughly:")
  } else if (scdiff>5) {
    print("Yes, it is garbage time - the leaders' % win chance is roughly:")
  }
})

output$num <- renderText({
  scdiff <- input$ScoreD
  MinzLeft <- as.numeric(input$WholeMinutesLeft) + (as.numeric(input$SecondsLeft) / 60)
  x_index <<- round(MinzLeft * 0.4847 + 0.4733)
  y_index <<- as.numeric(scdiff - 4)
  val <- ((dfz[x_index, y_index]) * 100)
  if (MinzLeft == 0) {
    print("Game is over.")}
  if (scdiff < 5.5 | scdiff > 23) {
    print("Please enter a score difference between 6 and 23.")
  } else if (MinzLeft != 0) {
    print(floor(val))
  }
})

output$blurb <- renderText({
  print("The red dotted lines intersect near the location of your scenario on the map.

        For preseason-type urgency, use a confidence threshold of 85%.

        For playoffs-type urgency, use a confidence threshold of 100%.

        Default of 99% will work for most users.

        Created using 2.4 million simulated basketball games.

        Hooply quantifies the end-game of basketball.

        Copyright 2018")
})
```

```
  }


# Run the app
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

A confidence interval variable is included, so coaches can edit their "risk," meaning they can put Confidence Interval closer to 90% to indicate a conservative approach towards star player fatigue or injury, showing more indifference to losing a game, perhaps like in the pre-season when games don't count.

A coach-user selecting a much higher confidence interval of perhaps 100% would be playing his star players as aggressively as possible, perhaps in a situation like the NBA finals where games matter the most.

Ultimately, it's not a necessary detail to edit, so the app's confidence threshold has been defaulted to 99.5%, which is plenty high, and still shows enough room on the map to help users make a meaningful decision.

Given the roughly 14% bias compared to Goel's data, it's worth noting that the default confidence interval for the ultimate recommender is still set at 99.5%, meaning this linear bias can be offset by up-adjusting our linear confidence threshold.

App Details

The y-range, score difference between team A and team B, is kept between 4 and 25. This is because with a score less than 4, the game is truly a close game, and there's no way to realistically predict garbage time. Scores that close just aren't relevant. Similarly, scores differences beyond 25 aren't relevant either, since everyone knows that is a blowout and the leading team is really going to win. If there's a 30 point lead, no one is going to be surprised to punch in the app and confirm that the leading team is actually going to win, that just isn't impressive at all.

Similarly, there is a minimum value for time remaining at 5 seconds. Time-values less than 4 seconds were actually breaking the app for some unknown reason, and the easiest fix was to cap the minimum time value at 5 seconds; if there's no time left in the game, there's no point in trying to make a "prediction."

To create Hooply, this code was run for "gftopo4(m=9000)" which yields slightly over 2.6 million games when multiplied out in the double for-loop over the 13 by 23 grid of the x and y axis. To run this many simulations, it takes roughly 20-30 minutes; so I have changed the code above to just show 90 simulations, which wouldn't take as long. Feel free to challenge your computer and take m up to 10,000 or more, and see how "in-focus" this grid can become.

Next, these above 3 matrices (dfz or the z-matrix, xax or the x axis labels, and yax or the y axis label) needed to be exported into the ultimate app.R file for the web app. This proved difficult because having any kind of file-path or file tree in your app is extremely tricky and was personally over my head for this project. It was initially my approach to try to link to files on my own machine in the app directory, but I still couldn't get it to work that way. Ultimately, I just had to hard-code the data into the app.R file itself. So rather than trying to unzip and link to other files, I just opened each .csv in notepad, and copied the comma-separated data, and pasted it into the stated data for the z-matrix.

Initially my thought was to graph out many game-flow charts on the same graph and then run analyses on that big set of graphed data. Though this is indeed possible, the resulting graphs ended up too cluttered, and I couldn't gain any insight. Overall, these types of graphs contain too much information, in a sense.

Ultimately, we are only concerned with the end-game, and the user's position in that end-game, as well as the eventual game outcome. With respect to the end-game analysis, what happened on the scoreboard to get you up to that point really has nothing to do with how the game will end.

Graphically depicting all the actual action itself for a large number of whole games is perhaps superficially interesting, but the real key is in reducing the dimension of that information, and focusing on the user's own end-game scenario.

Contour Map Matrix

Even though Goel's write up on basketball end-games was helpful and provided real-world empirical data, the

Contour maps are essentially a type of 3-dimensional graph, where the third dimension "z" is compressed into a line and made into "altitude," which creates the contours. This is ideal for our purposes as R's contour function draws piece-wise regression lines for whatever contour bands we want. For visual design purposes, this was chosen to be 5, so as not to overwhelm the viewer but still provide robust information.

So essentially this needed to be a matrix along with two vectors representing non-index values for the x and y axis, minutes left, and score difference, respectively. Together, these 3 matrices (two of which only have one column) go into the contour function and create the chart.

This block of code is what was run to create the matrices used in the final app. Note this code both produces the 3 needed matrices in .csv format, as well as provides an early snapshot glimpse of the contour map itself.

Initially my thought was to graph out many game-flow charts on the same graph and then run analyses on that big set of graphed data, and though this is indeed possible, the resulting graphs ended up too cluttered, and I couldn't gain any insight. Overall, these types of graphs contain too much information, in a sense. Ultimately, we are only concerned with the user's position, and with the ultimate result of the game. Graphically depicting the actual action itself for a large number of games is perhaps superficially interesting but the real key is in gleaning much smaller amounts of information, and keeping that information of the utmost relevance.

Ideas For Further Research

It's easy to envision trying to plot different "pet" algorithms on this chart, like Goel's 2*sqrt(Minutes-remaining), or some similar relationship. This makes an excellent initial topic for further exploration beyond this introductory data science course thesis project, the process of measuring the success of "pet" algorithms - Hooply works around this type of analysis by displaying multiple layers of potential "pet" algorithms next to each other, by slicing at specific confidence-contour intervals.

One future potential product to break out would be an expanded range for time values. Right now, Hooply is only for the 4th quarter. But what about the 3rd quarter? It would require more computations, and simulations, and over all, that would be one example of a possible product to release.

Another one would be to expand the data where there is less than one minute left, and slice (or index) the final minute a dozen times or so. It is somewhat computationally expensive to create the data that would go into a temporally zoomed-in or out version of Hooply and this is a potential higher quality product.

In some brief user testing, it's become apparent that some users are asking the thesis problem question with 15 seconds left, and want to know the nuance of that probability map, while other users are positing the thesis problem statement with 18 minutes left (in the 3rd quarter); and both of these are valid, but so far, Hooply has published somewhat of a compromise on the "view" setting, essentially trying to encapsulate the greatest number of users' potential preference, but due to the limits of computing power, and the graphical challenges of zooming a contour map out from one minute to 20 or more, Hooply has quite a difficult and interesting challenge here in displaying this data in a way that maximizes the number of satisfied users.

It's plausible that a logarithmic scale would better suit the display, and a logarithmic x-axis could even be another Hooply spin-off product idea - it's definitely the case that this illuminates a lot of different product ideas.

It would also be interesting to see this type of analysis applied to other timed sports, such as football, or hockey - one such data scientist who has pushed this field forward is Michael Beuoy, who published a live win probability calculator for the NBA in 2015, and has also done this for football and (interestingly enough, a non-timed sport,) tennis.

Data mining for this topic was an ongoing obstacle for this project, and I personally see the need to expand my programming skill-set to encompass advanced data scraping and mining so I can access the data at PopcornMachine.net.

Now that it has been defined more quantitatively, there are a number of different pressing questions at hand regarding "Garbage Time," and the effect it might have on a player to be taken out consistently during garbage time, or not.

I have my hunches Gregg Popovich, head coach of the San Antonio Spurs, for one, has his own mental version of this analysis, because he takes his star players out of games earlier than other coaches, I have noticed; Popovich is also the longest-tenured coach in the NBA, with 5 championships, his players have some of the longest careers in sports - could Popovich's respect for garbage time be playing a part in extending his players' careers? This is a much more advanced question and well suited for a future project, but no doubt it is a scientific question that could be answered with more skilled data mining from PopcornMachine.net and the style of analysis outlined in this project.

Conclusion

Basketball is a popular, big-money sport, and hopefully this project is helpful for people thinking about the closing minutes of games in a serious and statistically rigorous way.

Ultimately, the nuance of Hooply is in its scope - we are just here to rigorously quantify the end-game probabilities, and provide a framework for how other programmers can easily refine their own thinking into workable models.

Recommendations

Users who plug in their scenario to the Hooply app will see a classification - whether the game is in garbage time or not. The recommendation is clear after garbage time is called - without fear of missing out, fans can go home and try to beat the traffic, and coaches can substitute out their star players and mitigate a damaging injury.

Glossary:

Garbage Time

Originally coined by Los Angeles Lakers' famous announcer Chick Hearn, "Garbage Time" refers to the end of a basketball game when one team is in the lead by a comfortable enough scoring margin that it's customary for the coach to put in the substitutes or non-starters, in place of the starters, so that the starters can rest and avoid injury and fatigue.

"Overtime"

At the end of regulation time in basketball, 48 minutes, if the score is tied, there must be an overtime round of 5 minutes.

“Possession”

Each time a team’s player starts to control the basketball, it’s considered a possession. It’s essentially the number of die-throws in the basketball game allowed by both teams, where each team can score 0, 1, 2, or 3 points on each throw of the die with probabilities outlined by “probA” and “probB” in the Gameflowmaker4 function above.

“Basketball-action”

A basketball-action is an official action in basketball recordable as one of the official statistics in basketball. These can include baskets, assists, rebounds, turnovers, fouls, blocked shots, or a number of other recordable statistics that may or may not affect the score. In many game-flow tables and charts, each basketball-action is given one point along the x-axis which represent time. For this analysis, the computational task was streamlined by initially deleting non-scoring basketball actions from the game-flow tables analyzed. Ultimately, this simply meant fewer points along horizontal stretches of the line representing score through time, so it’s not much of a change aesthetically.

“Bench player”

As opposed to a starter, a bench player just sits on the team bench for the start of the game, and swaps in during action at the coach’s discretion. Managing which bench players and starters are in and out of the game at what times is one of the toughest dynamic challenges coaches face.

“N.B.A. or NBA”

This stands for the “National Basketball Association,” and is the leading professional basketball league in the United States and the world. As of 2018, every NBA team was worth over $1 billion according to Forbes Magazine.

“Starter”

A starter in any sport, including basketball, means a player who is considered “good” relative to their teammates. “Starters” are in the game when the game starts (at tip-off), and, typically when it finishes. Basketball is like hockey and football (and unlike baseball and soccer) in that players can re-enter the game after being substituted out by the coach.

Tip-off

Tip-off is the beginning of a basketball game, or an overtime period. Like the game-starting face-off in hockey, or kick-off in football, basketball’s tip off starts the game, and some other scenarios can lead to a tip-off during the game. In basketball, the team winning the game-opening tip off must give possession to the other team to start the second quarter and third quarter, but keeps possession to start the fourth.

Bibliography:

1. Mertens, J.F. & Zamir, S. Int J Game Theory (1976) 5: 187. https://doi.org/10.1007/BF01761601
2. Westfall, Peter H. "Graphical Presentation of a Basketball Game." The American Statistician, vol. 44, no. 4, 1990, pp. 305–307. JSTOR, JSTOR, www.jstor.org/stable/2684357.
3. Carlson, Michael. (2002, August 7) Chick Hearn. Retrieved from: https://www.theguardian.com/news/2002/aug/08/guardianobituaries1
4. Lewis, Michael. Moneyball. New York : W.W. Norton, 2004.
5. Goel, Sharad. "How Close Is Close?" Messy Matters, May 31, 2012. http://messymatters.com/moneyball/
6. Everson, Patrick. "Halftime Lines: How Books Set The Odds And How Sharps Bet Them." Covers, January 12, 2015. https://www.covers.com/editorial/Article/31e4a81b-b51e-e711-80cb-44a8423171c1/Halftime-lines-How-books-set-them-and-how-sharps-bet-them.
7. Haberstroh, Tom. (2015, January 28). Home-court advantage? Not so much. Retrieved from: http://www.espn.com/nba/story/_/id/12241619/home-court-advantage-decline.
8. Kenter, Franklin. "An Analysis of the Basketball Endgame : When to Foul When Trailing and Leading !" (2015).
9. Beuoy, Michael. (2015, April 19). Live Win Probabilities for the NBA Playoffs. Retrieved from: http://www.inpredictable.com/2015/04/live-win-probabilities-for-nba-playoffs.html
10. Groves, Roger. (2015, December 12). Why The NBA Will Eventually Overtake The NFL In Popularity And Why It Matters. Retrieved from: https://www.forbes.com/sites/rogergroves/2015/12/12/why-the-nba-will-eventually-overtake-the-nfl-in-popularity-and-why-it-matters/#4448dfd1e4c6.
11. Badenhausen, Kurt. (2016, December 15) The Average Player Salary And Highest-Paid In NBA, MLB, NHL, NFL And MLS. Retrieved from: https://www.forbes.com/sites/kurtbadenhausen/2016/12/15/average-player-salaries-in-major-american-sports-leagues/#1056de0e1050.
12. Minter, Adam. (2017, September 28). China Is Hoops Country. Retrieved from: https://www.bloomberg.com/view/articles/2017-09-28/basketball-not-soccer-is-china-s-game-of-choice.
13. Friedell, Nick. (2017, October 24) The Story Of The Chicago Bulls' Downfall. Retrieved from: http://www.espn.com/nba/story/_/id/21082183/story-chicago-bulls-downfall.
14. Bonesteel, Matt. (2018, January 25) "If sports gambling is legalized, the NBA wants in on the profits." Retrieved from: https://www.washingtonpost.com/news/early-lead/wp/2018/01/25/if-sports-gambling-is-legalized-the-nba-wants-in-on-the-profits/?noredirect=on&utm_term=.f3b1c064eb0a