Data Story - Intro Capstone Project Milestone

Carl Larson

2/1/2018

Data Story Milestone - Springboard Intro Data Science Course

Rather than taking pictures of charts as shown in the previous report, the initial project proposal, here we will open up the data with R to see the story of our game of focus- the beginning of the demise of the Chicago Bulls' from top-seed in the east in 2012 to one of the worst teams in the NBA, by current standings today in 2018.

This game showcased a typical coaching error (how typical, exactly, would be a great topic for a future project), but what was atypical about their "Game 1" on April 28, 2012, was how costly that error was. To see this story, let's open up R and get the data.

Analyzing The Gameflow Table

Our data is basically pretty simple, a Gameflow table which contains a time variable, as well as scores for each team at that time. It contains more data than that, but for this analysis that is all we need.

To wit, I know I have a gameflow chart at the below filepath, but other people following along will need to change the below filepath to something that works for their computer, after downloading the data.

```
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:stats':
##
##     filter, lag
```

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

```
require(tidyr)
```

```
## Loading required package: tidyr
```

```
## Warning: package 'tidyr' was built under R version 3.4.2
```

```
require(ggplot2)
```

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

```
require(reshape2)
```

```
## Loading required package: reshape2
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
df <- read.csv("/Users/EagleFace/Downloads/Gameflow of Bulls76ersApr282012.csv", header = FALSE)
head(df)
```

```
##   V1 V2    V3 V4    V5             V6     V7             V8
## 1  0  0 11:38  1 76ers    Elton Brand DefReb April 28, 2012
## 2  3  0 11:26  1 76ers   Jrue Holiday  Make3
## 3  3  0 11:26  1 76ers Andre Iguodala Assist
## 4  3  2 11:07  1 Bulls      Luol Deng  Make2
## 5  3  2 11:07  1 Bulls  Carlos Boozer Assist
## 6  3  2 10:44  1 76ers    Lavoy Allen  miss2
```

```
tail(df)
```

```
##     V1  V2   V3 V4 V5 V6 V7 V8
## 436 89 101 0:37  4
## 437 89 101 0:36  4
## 438 89 103 0:21  4
## 439 89 103 0:21  4
## 440 91 103 0:09  4
## 441 91 103 0:09  4
```

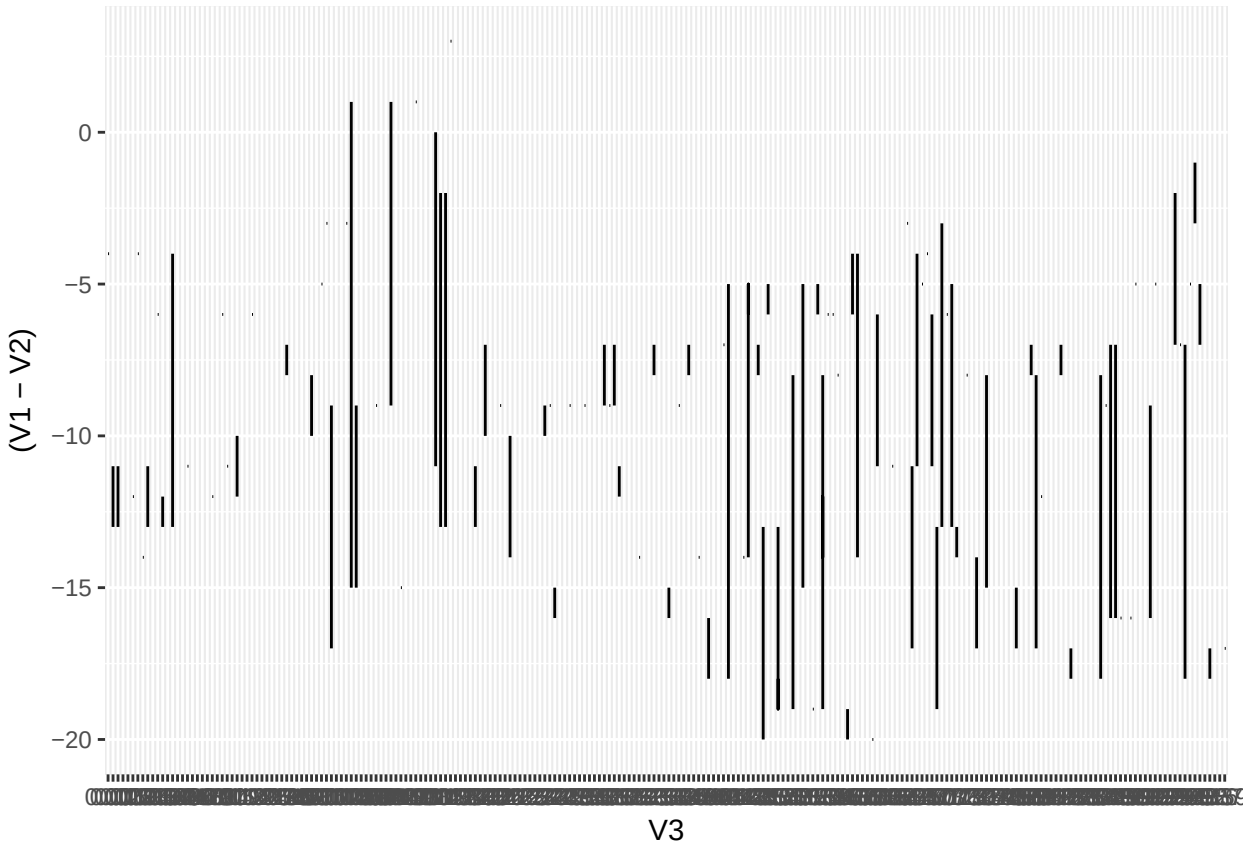
So here we have our first glimpse of the gameflow table for this particular game of interest. The head(df) function shows us why we have the header argument set to "FALSE."

What Important Fields And Information Does The Data Set Have?

With this first glimpse of our gameflow table, we can see that the important fields are going to be points for each team, and the time variable V3, which resets to 12 exactly three times a game. V4 is the current quarter of the game. V5, V6, and V7 all show the player who owns the action being reported on the gameflow table, and shows the team they represent. V8 is a meta-variable which is the date of the game.

From here, we can easily use the capacity of R to plot the chart.

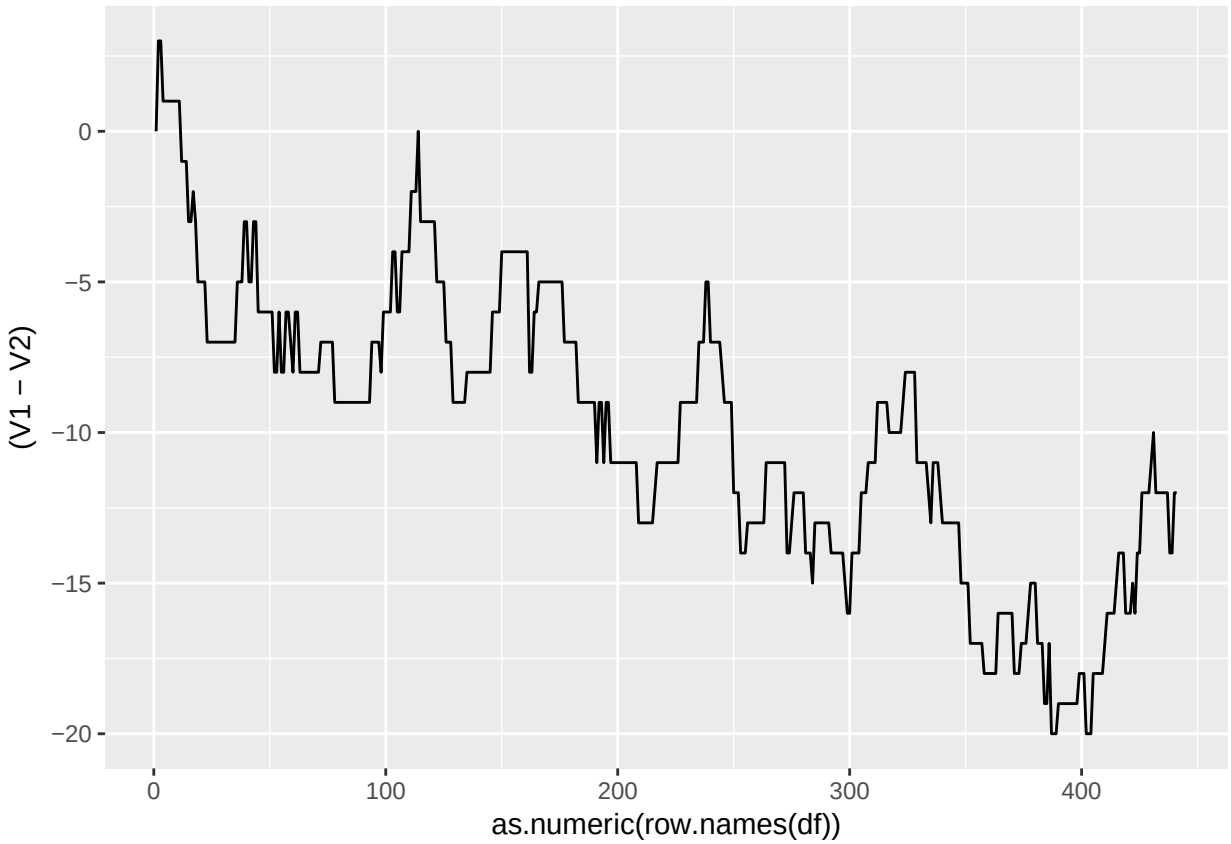
```
ggplot(df, aes(x=V3, y=(V1-V2)))+
  geom_line()
```



As you can see, the data needs to be cleaned before it can make sense. Ggplot is being confused by the fact that there are multiple values for the same time, as the clock in basketball does reset 3 times a game, at the end of each quarter except the 4th. So we need to clean the data by creating a uniform time variable that never repeats values.

One clever way to show this is to take a peek at perhaps the "V0," or the row names. R automatically numbers your rows numbers and assigns those numbers as a name, which can be called by the row.names() function. These numbers come out as a string, so they need to be nested inside the as.numeric() function.

```
ggplot(df, aes(x = as.numeric(row.names(df)), y = (V1-V2))) +
  geom_line()
```



Though a bit crude, this is something that does make intuitive sense, as we are no longer repeating y-values overtop each other on the same x-axis- we have essentially quadrupled the x-axis and mapped the correct scores to the correct quarter.

This chart treats a basketball game like a timeless-type of turn-based game, as each observation represents a basketball-specific action type (and player, and team) that may or may not have affected the score, a sample of which we saw in the variable V7 which was returned above in the head(df) function. Some actions occur technically "simultaneously" in basketball just looking at the game-clock (such as a score and an assist, or a foul and multiple subsequently made free throws).

Other stretches of the game may include a lot of dribbling and passing, where much time may roll off the clock with no action reportable to a gameflow table. Thus, the above chart represents actions only, and is only loosely correlated with actual time left in the game.

For most purposes though, this chart is readable as a gameflow chart. It shows the 76ers going up early, and then quickly losing that lead to the top-seeded Bulls, who never really relinquished except for a bit at the end of the game, which was exactly what the Bulls' coach was referring to when he told reporters after the game,

"We don't work backwards like you guys. The score was going the other way," he said, referring to that late run that is clear as the Chicago Bulls started losing their biggest lead of the game.

It more than likely was just a regression to their mean lead of about 12 points in this game.

We also know that right at about 90 seconds remaining in the game, Chicago's star Derrick Rose blew out his ACL, ending his season and setting his career back by years, limiting his potential, and cutting his future salaries.

What Are This Data's Limitations?

Excellent as it is as a type of data, this type of data does have a main limitation. Right now it is difficult to access more gameflow chart data, and it's limited what can be done without being handy with perl script and XML files to do the data mining and obtain the gameflow chart.

Once we have the gameflow chart though, we can do a lot of really interesting math.

Gameflow charts apply to all timed sports, and this type of analysis technically should also apply to other sports just based on the fact that there is a score and a clock. Of course, the envelope-equation itself would be different and tailored to the clock and method of scoring points in each of those different sports.

Ultimately this data is perfect for these purposes and really the only tiny limitation is that we need to do some cleaning to get a real time variable, rather than using actions as our x-axis above.

What Kind Of Cleaning And Wrangling Are Needed?

We need to change the time variable into something that makes sense for the algorithm, decimalized total minutes remaining. Currently the time variable, V3, contains a colon between minutes and seconds. This colon needs to be removed, and we need separate columns for minutes and seconds, before combining them into the desired single column.

```
#Leaving out irrelevant columns as we export data into a
#new dataframe.
df <- df[,1:4]
gf <- separate(df, V3, c("MIN", "SEC"), sep = ":", remove = TRUE)


#Changing time categories to numeric so we can operate on them.
gf$MIN <- as.numeric(gf$MIN)
gf$SEC <- as.numeric(gf$SEC)

#Next we need to combine minutes and seconds into a variable
#which we can call "ft" for a formatted time variable.
ft <- numeric(length = nrow(df))

for(i in 1:(nrow(gf))){
  while(gf$V4[i] == 1){
    ft[i] <- (gf$MIN[i] + (gf$SEC[i]/60) + 36)
      (i=i+1)}

  while(gf$V4[i] == 2){
    ft[i] <- (gf$MIN[i] + (gf$SEC[i]/60) + 24)
      (i=i+1)}

  while(gf$V4[i] == 3){
    ft[i] <- (gf$MIN[i] + (gf$SEC[i]/60) + 12)
      (i=i+1)}

  while(gf$V4[i] == 4){
    ft[i] <- (gf$MIN[i] + (gf$SEC[i]/60))
    if(i==nrow(gf)){break}
    else{
      (i=i+1)}}}
```

This has decimalized the "minutes remaining" variable to include seconds data and minutes data.

```
#Here is a countown of the game's final actions in this format:

ft[430:nrow(gf)]
```

```
##  [1] 1.1500000 1.1500000 0.9833333 0.9833333 0.7333333 0.7166667 0.6166667
##  [8] 0.6000000 0.3500000 0.3500000 0.1500000 0.1500000
```

Preliminary Exploration

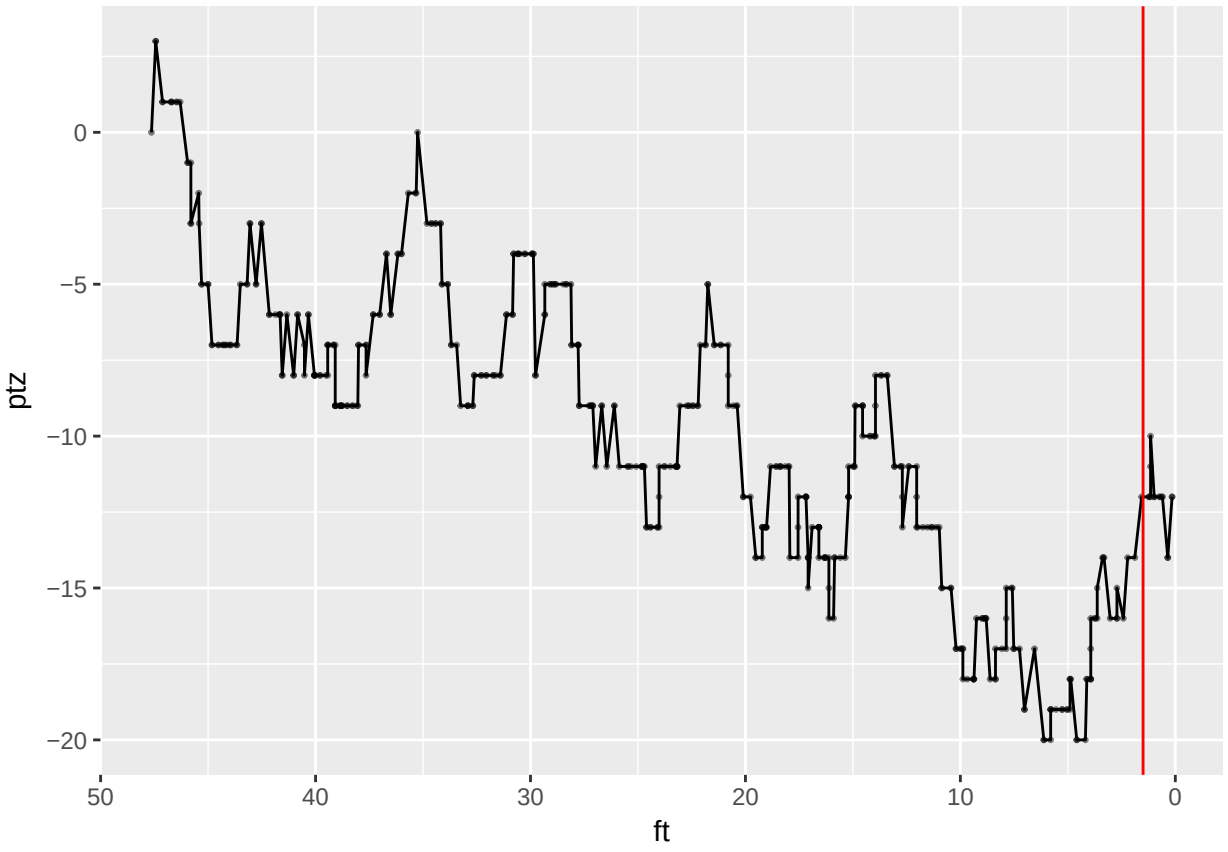
Let's draw a vertical line on this chart right at 1.5 minutes remaining, which is roughly when Chicago's star Derrick Rose blew out his knee in this game.

This should now be possible since we can map actual time on the x-axis rather than game actions.

```
#Making points difference vector for y axis, ok to be negative or positive.
ptz <- gf$V1 - gf$V2

#First putting our vectors of interest into a common dataframe, gdf
gdf <- data.frame(ft, ptz)


#This is a straightforward graph that we need to x-reverse to get
#it counting down to zero time remaining. We also want a vertical
#line where Chicago's star was injured with 1.5 minutes left.
ggplot(gdf, aes(x=ft, y=ptz)) +
  geom_point(size=0.5, alpha=0.5) +
  geom_line() +
  geom_vline(xintercept=1.5, color="red") +
  scale_x_reverse()
```



What Approach To Take Next?

One question we can ask ourselves is, when do we feel garbage time should have begun in this game? The top seed was beating the bottom seed and it was looking like a blowout. But how much of a blowout was it? Time can move quickly on people, and sometimes people need alarms - when should the alarm have gone off to the head coaches in this game that the star players were no longer needed, and the game was decided?

At this point, the project is clearly defined, and has strong scope, as well as many ideas for future projects.

Should garbage time have begun before or after the red line above?

Are players more prone to injury when they are tired at the end of a game?

When should garbage time begin?