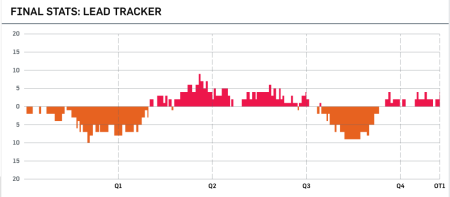
For each NBA game, nba.com has a really nice graphic which tracks the point differential between the two teams throughout the game. Here is the lead tracker graphic for the game between the LA Clippers and the Phoenix Suns on 10 Dec 2018:



Taken from <https://www.nba.com/games/20181210/LACPHX#/matchup>

I thought it would be cool to try recreating this graphic with R. You might ask: why try to replicate something that exists already? If we are able to pull out the data underlying this graphic, we could do much more than just replicate what is already out there; we have the power to make other visualizations which could be more informative or powerful. (For example, how does this chart look like for all games that the Golden State Warriors played in? Or, how does the chart look like for each quarter of the game?)

The full R code for this post can be found [here](https://github.com/kjytay/misc/blob/master/blog/2018-12-12_nba_play_by_play.R). For a self-contained script that accepts a game ID parameter and produces the lead tracker graphic, click [here](https://github.com/kjytay/misc/blob/master/blog/nba_play_by_play_single_game.R).

First, we load the packages that we will use:

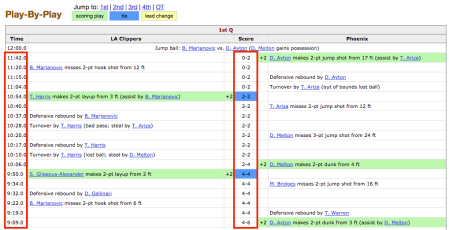
library(lubridate)

library(rvest)

library(stringr)

library(tidyverse)

We can get play-by-play data from [Basketball-Reference.com](https://www.basketball-reference.com/) ([here](https://www.basketball-reference.com/boxscores/pbp/201812100PHO.html) is the link for the LAC @ PHX game on 2018-12-10). Here is a snippet of the play-by-play table on that webpage, we would like to extract the columns in red:



Play-by-play data from basketball-reference.com.

The code below extracts the webpage, then pulls out rows from the play-by-play table:

# get webpage

url <- paste0("https://www.basketball-reference.com/boxscores/pbp/", current\_id,

".html")

webpage <- read\_html(url)

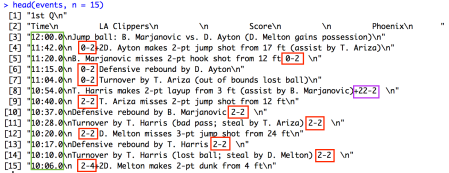
# pull out the events from the play-by-play table

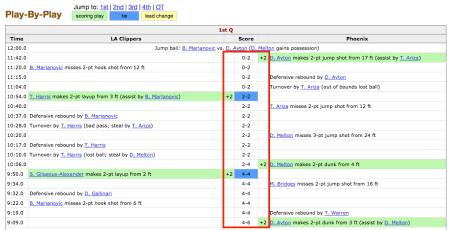
events <- webpage %>%

html\_nodes("#pbp") %>%

html\_nodes("tr") %>%

html\_text()

events is a character vector that looks like this:  
  
We would really like to pull out the data in the boxes above. Timings are easy enough to pull out with regular expressions (e.g. start of the string: at least 1 digit, then :, then at least one digit, then ., then at least one digit). Pulling out the score is a bit trickier: we can’t just use the regular expression denoting a dash – with a number on each side. An example of why that doesn’t work is in the purple box above. Whenever a team scores, basketball-reference.com puts a “+2” or “+3” on the left or right of the score, depending on which team scored. In events, these 3 columns get smushed together into one string. If the team on the left scores, pulling out number-dash-number will give the wrong value (e.g. the purple box above would give 22-2 instead of 2-2).



To avoid this issue, we extract the “+”s that may appear on either side of the score. In fact, this has an added advantage: we only need to extract a score if it is different from the previous timestamp. As such, we only have to keep the scores which have a “+” on either side of it. We then post-process the scores.

# get event times & scores

times <- str\_extract(events, "^\\d+:\\d+.\\d+")

scores <- str\_extract(events, "[\\+]\*\\d+-\\d+[\\+]\*")

scores <- ifelse(str\_detect(scores, "\\+"), scores, NA)

df <- data.frame(time = times, score = scores, stringsAsFactors = FALSE) %>%

na.omit()

# remove the +'s

parseScore <- function(x) {

if (startsWith(x, "+")) {

return(str\_sub(x, 3, str\_length(x)))

} else if (endsWith(x, "+")) {

return(str\_sub(x, 1, str\_length(x) - 1))

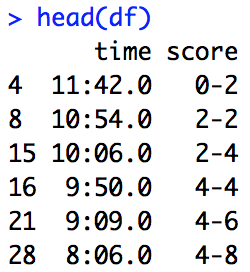
} else {

return(x)

}

}

df$score <- sapply(df$score, parseScore)



Next, we split the score into visitor and home score and compute the point differential (positive means the visitor team is winning):

# split score into visitor and home score, get home advantage

df <- df %>%

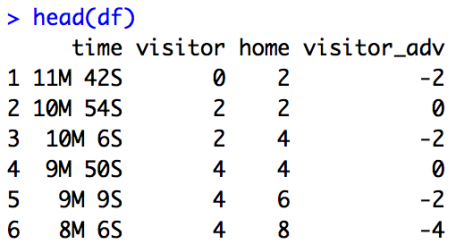
separate(score, into = c("visitor", "home"), sep = "-") %>%

mutate(visitor = as.numeric(visitor),

home = as.numeric(home),

time = ms(time)) %>%

mutate(visitor\_adv = visitor - home)



Next we need to process the timings. Each of the 4 quarters lasts for 12 minutes, while each overtime period (if any) lasts for 5 minutes. The time column shows the amount of time remaining in the current period. We will amend the times so that they show the time elapsed (in seconds) from the start of the game. This notion of time makes it easier for plotting, and works for any number of overtime periods as well.

# get period of play (e.g. Q1, Q2, ...)

df$period <- NA

period <- 0

prev\_time <- ms("0:00")

for (i in 1:nrow(df)) {

curr\_time <- df[i, "time"]

if (prev\_time < curr\_time) {

period <- period + 1

}

df[i, "period"] <- period

prev\_time <- curr\_time

}

# convert time such that it runs upwards. regular quarters are 12M long, OT

# periods are 5M long

df <- df %>% mutate(time = ifelse(period <= 4, as.duration(12 \* 60) - as.duration(time), as.duration(5 \* 60) - as.duration(time))) %>%

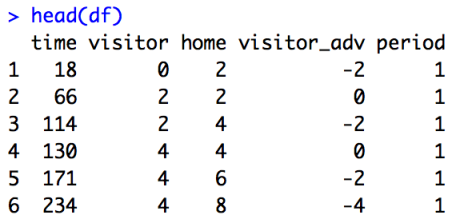
mutate(time = ifelse(period <= 4,

time + as.duration(12 \* 60 \* (period - 1)),

time + as.duration(12 \* 60 \* 4) +

as.duration(5 \* 60 \* (period - 5))

))



At this point, we have enough to make crude approximations of the lead tracker graphic:

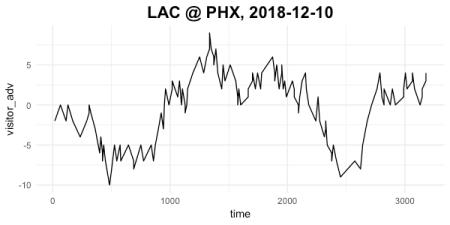
ggplot() +

geom\_line(data = df, aes(x = time, y = visitor\_adv)) +

labs(title = "LAC @ PHX, 2018-12-10") +

theme\_minimal() +

theme(plot.title = element\_text(size = rel(1.5), face = "bold", hjust = 0.5))



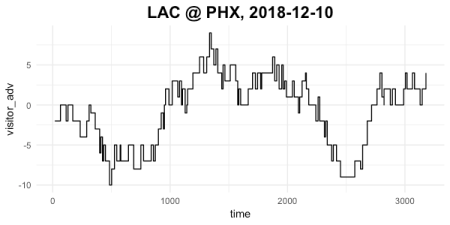
ggplot() +

geom\_step(data = df, aes(x = time, y = visitor\_adv)) +

labs(title = "LAC @ PHX, 2018-12-10") +

theme\_minimal() +

theme(plot.title = element\_text(size = rel(1.5), face = "bold", hjust = 0.5))



Getting the fill colors that NBA.com’s lead tracker has requires a bit more work. We need to split the visitor\_adv into two columns: the visitor’s lead (0 if they are behind) and the home’s lead (0 if they are behind). We can then draw the chart above and below the x-axis as two geom\_ribbons. (It’s a little more complicated than that, see [this StackOverflow question](https://stackoverflow.com/questions/21887088/generate-a-filled-geom-step) and [this gist](https://gist.github.com/Teebusch/db0ab76d31fd31a13ccf93afa7d77df5) for details.) Colors were obtained using [imagecolorpicker.com](https://imagecolorpicker.com/).

df$visitor\_lead <- pmax(df$visitor\_adv, 0)

df$home\_lead <- pmin(df$visitor\_adv, 0)

df\_extraSteps <- df %>% mutate(visitor\_adv = lag(visitor\_adv),

visitor\_lead = lag(visitor\_lead),

home\_lead = lag(home\_lead))

df2 <- bind\_rows(df\_extraSteps, df) %>%

arrange(time)

ggplot() +

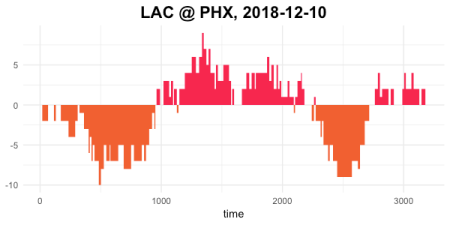
geom\_ribbon(data = df2, aes(x = time, ymin = 0, ymax = visitor\_lead), fill = "#F7174E") +

geom\_ribbon(data = df2, aes(x = time, ymin = home\_lead, ymax = 0), fill = "#F16031") +

labs(title = "LAC @ PHX, 2018-12-10") +

theme\_minimal() +

theme(plot.title = element\_text(size = rel(1.5), face = "bold", hjust = 0.5))



Almost there! The code below does some touch up to the figure, giving it the correct limits for the y-axis as well as vertical lines for the end of the periods.

# get score differential range (round to nearest 5)

ymax <- round(max(df$visitor\_adv) \* 2, digits = -1) / 2

ymin <- round(min(df$visitor\_adv) \* 2, digits = -1) / 2

# get period positions and labels

periods <- unique(df$period)

x\_value <- ifelse(periods <= 4, 12 \* 60 \* periods,

12 \* 60 \* 4 + 5 \* 60 \* (periods - 4))

x\_label <- ifelse(periods <= 4, paste0("Q", periods),

paste0("OT", periods - 4))

ggplot() +

geom\_ribbon(data = df2, aes(x = time, ymin = 0, ymax = visitor\_lead), fill = "#F7174E") +

geom\_ribbon(data = df2, aes(x = time, ymin = home\_lead, ymax = 0), fill = "#F16031") +

geom\_vline(aes(xintercept = x\_value), linetype = 2, col = "grey") +

scale\_y\_continuous(limits = c(ymin, ymax)) +

labs(title = "LAC @ PHX, 2018-12-10") +

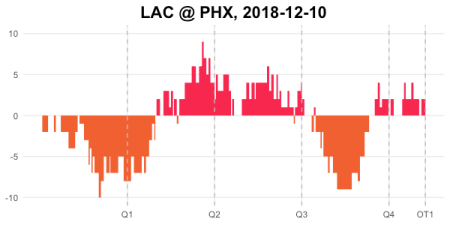
scale\_x\_continuous(breaks = x\_value, labels = x\_label) +

theme\_minimal() +

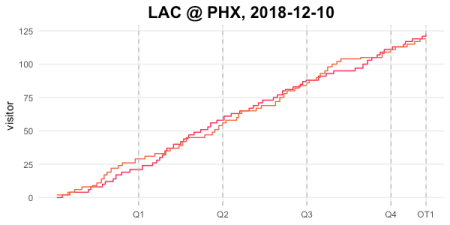
theme(plot.title = element\_text(size = rel(1.5), face = "bold", hjust = 0.5),

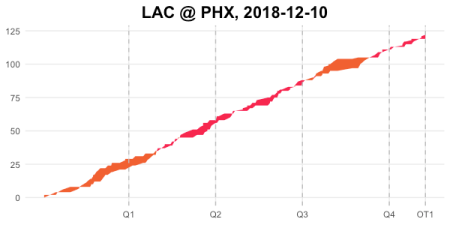
axis.title.x = element\_blank(), panel.grid.minor.x = element\_blank(),

panel.grid.minor.y = element\_blank())

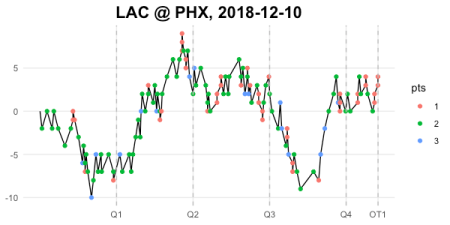


The figure above is what we set out to plot. However, since we have the underlying data, we can now make plots of the same data that may reveal other trends (code at the end of [this R file](https://github.com/kjytay/misc/blob/master/blog/2018-12-12_nba_play_by_play.R)). Here are the line and ribbon plots where we look at the absolute score rather than the point differential:





Here, we add points to the line plot to indicate whether a free throw, 2 pointer or 3 pointer was scored:



The possibilities are endless!