barry bonds' without a bat

the apostrophe is kinda important

a recreation of What if Barry Bonds had played without a baseball bat? | Chart Party

setup

so this is actually one of the more tedious parts. getting data from retrosheet was kinda difficult for me because ~~the tools documentation is crap~~ i am too stupid for command line programs. nevertheless, there's some tools out there like chadwick and smarter people than me.

i followed the instructions of some random blog post to get a workable play-by-play dataset of the 2004 season. the mentioned r script is in this repository, just call the function `parse.retrosheet2.pbp(2004)` from the R file and you're good to go. the csv of the 2004 season is in this repository as well.

```r
# read in play-by-play data.
# there's around 194k plays in there.
season.pbp <- read.csv("data/all2004.csv", header = FALSE)

# only keep those that belong to our friend barry
# and only those that concern him
# and only the pitch sequences and results to save some memory
barry <- season.pbp[season.pbp$V11=="bondb001" & season.pbp$V36 == TRUE,
                    c("V8", "V30")]

# a copy of barry, we'll need it later
barry_copy <- barry

# seems to be correct
print(nrow(barry))
```

```
## [1] 617
```

that's the 617 plate appearances.

now to the fun part.

we initialize some variables to keep track of our work so far.

```r
# number of walks so far
bb <- 0
# number of hbp so far
hbp <- 0
# number of plate appearances
pa <- nrow(barry)

# an obp function that we can call to keep track of barry
obp <- function() {
  (bb + hbp)/pa
}
```

```r
# set the random seed
# this will be important later
set.seed(1)
```

walks and hit by pitches

despite what the title says, i'll keep the structure of the video. beaned barry comes first, then intentional walks, then walks where barry swung.

hit by pitches

we don't really care about them. jon just counts them as a walk, so i will too. we can find them in our data as pitch sequences that end in "H".

```r
# get the indices of PA that ended in a HBP
# they stay as they are
hbp_ind <- grep("H$", barry[,1])

# there's nine of them
print(length(hbp_ind))
```

```
## [1] 9
```

```r
# we remove them from our dataset - they're taken care of
barry <- barry[-hbp_ind,]
```

let's take a look at our barry's obp now.

```r
# update hbp counter
hbp <- length(hbp_ind)

# calculate obp
obp()
```

```
## [1] 0.01458671
```

that's pretty bad. but hey, we got a lot of ground to cover still. we have 608 plate appearances left.

walks

oh boy, this is a "fun" one. as far as i've understood it, we have to analyze all walks that were not intentional. so, let's get the intentional ones out of the way first.

intentional walks

from what i could see in the video, the relevant pitch sequences end in the letter "I". let's get rid of them.

```r
# get the indices of PA that ended in an ibb
ibb_ind <- grep("I$", barry[,1])

# there's **120** of them
print(length(ibb_ind))
```

```
## [1] 120
```

```r
# jesus christ.
# we remove them from our dataset - they're taken care of.
barry <- barry[-ibb_ind,]
```

okay, let's add them to our walk counter:

```r
# update walk counter
bb <- bb + length(ibb_ind)

# show obp
print(obp())
```

```
## [1] 0.2090762
```

and now barry has a .209 obp. what a machine.

real walks

okay, now let's take a look at the remaining walks and put them in our barry bonds batless simulator. we write a function to simulate all pitches barry swung at according to our pitch sequence data. i think now is the time to explain what some of the letters in the pitch sequence mean.

letter	meaning	what to do with it
B	ball	ball
C	called strike	strike
F	foul	simulate
H	hit batter	walk
I	intentional ball	ball
K	strike (unknown type)	strike
L	foul bunt	simulate
M	missed bunt attempt	simulate
N	no pitch (on balks and interference calls)	walk
O	foul tip on bunt	simulate
P	pitchout	ball
Q	swinging on pitchout	ball
R	foul ball on pitchout	ball
S	swinging strike	simulate
T	foul tip	simulate
U	unknown or missed pitch	skip
V	called ball because pitcher went to his mouth	ball

a simulated ball has a 19.1% chance of being a strike. note that this can only *remove* walks, and we will never run out of pitches to simulate.

so the steps we have to take now are:

- get all the walks remaining in the data
- run them through our simulation function
- add the walks to our counter

a walk has an event descriptor that starts with "W".

```r
# extract walks from the event description
walk_ind <- grep("^W", barry$V30)

walks <- barry[walk_ind,]
```

```r
# there's 112 walks. i'm starting to understand how a .609 obp is possible
print(nrow(walks))
```

```
## [1] 112
```

```r
# the walk simulation function.
# return true if it's a walk, false otherwise.
# i know we could cut some corners here, but i don't want to right now.
walk_sim <- function(pitch_sequence) {
  # strike and ball counter
  balls <- 0
  strikes <- 0
  # iterate through the sequence
  for (i in strsplit(pitch_sequence,"")[[1]]) {
    # we got a simulated ball
    if (i %in% c("F","L","M","O","S","T")) {
      # generate random number between 1 and 1000
      r <- runif(n = 1, min = 1, max = 1000)
      # ball
      if (r < 192) {
        balls <- balls + 1
      } else {
        # strike
        strikes <- strikes + 1
      }
    } else if (i %in% c("C","K")) {
      # called strike, unknown strike
      strikes <- strikes + 1
    } else if (i %in% c("B", "I", "Q", "P", "Q", "R","V")) {
      # ball
      balls <- balls + 1
    } else if (i == "N") {
      return(TRUE)
    }
    # check if we're done
    if (balls == 4) {
      return(TRUE)
    } else if (strikes == 3) {
      return(FALSE)
    }
  }
}
```

turn on the walk machine

```r
# how many simulated walks do we get?
simulated_walks <- sum(sapply(walks$V8, walk_sim))

# add them to our walk counter
bb <- simulated_walks + bb

# how many walks do we have now?
print(bb)
```

```
## [1] 221
```

```r
# 222. that's only 4 off jon's number

# remove the walks from the plate appearances - they're taken care of
barry <- barry[-walk_ind,]
```

what about the on-base percentage though?

```r
print(obp())
```

```
## [1] 0.3727715
```

oh boy.

strikeouts

how often did barry bonds strike out in 2004?

```r
k_ind <- grep("^K", barry$V30)

print(length((k_ind)))
```

```
## [1] 41
```

just 41 times. alright, i'll be honest with you, i just wanted a sanity check here. but still, 41 strikeouts to 222 walks is insane. we again have to simulate. but this time, it can happen that we have to invent new pitches. in this case, the pitch is a strike 41.3% of the time.

```r
# the strikeout simulation function.
# return true if it's a walk, false otherwise.
# i know we could cut some corners here, but i don't want to right now.
k_sim <- function(pitch_sequence) {
  # strike and ball counter
  balls <- 0
  strikes <- 0
  # iterate through the sequence
  for (i in strsplit(pitch_sequence,"")[[1]]) {
    # we got a simulated ball
    if (i %in% c("F","L","M","O","S","T")) {
      # generate random number between 1 and 1000
      r <- runif(n = 1, min = 1, max = 1000)
      # ball
      if (r < 192) {
        balls <- balls + 1
      } else {
        # strike
        strikes <- strikes + 1
      }
    } else if (i %in% c("C","K")) {
      # called strike, unknown strike
      strikes <- strikes + 1
    } else if (i %in% c("B", "I", "Q", "P", "Q", "R","V")) {
      # ball
      balls <- balls + 1
    } else if (i == "N") {
      return(TRUE)
    }
    # check if we're done
    if (balls == 4) {
      return(TRUE)
    } else if (strikes == 3) {
      return(FALSE)
    }
  }
  # if we're here, we gotta get some more random numbers
  while (TRUE) {
    r <- runif(n = 1, min = 1, max = 1000)
    if (r < 588) {
      balls <- balls + 1
    }
    else {
      strikes <- strikes + 1
    }
    # check if we're done
    if (balls == 4) {
      return(TRUE)
    } else if (strikes == 3) {
      return(FALSE)
    }
  }
}
```

okay, this is basically the same as before, just with an extra thing at the end. time to throw on the simulation machine once again.

```r
# get the strikeouts
ks <- barry[k_ind,]

# simulate the strikeouts
k_to_walk <- sum(sapply(ks$V8, k_sim))

# how many can we add to the walk total?
print(k_to_walk)
```

```
## [1] 3
```

```r
# add them to what we have
bb <- bb + k_to_walk

# remove the strikeouts from the plate appearances - they're taken care of
barry <- barry[-k_ind,]
```

a couple more walks. '04 barry sure needs them.

```r
obp()
```

```
## [1] 0.3776337
```

is a .377 obp good?

balls in play

the meat of the piece. batless barry is already a very good player, but this could put him over the edge. 335 plate appearances left for when barry put the ball in play. we change our simulation function once again to account for all outcomes. only two more letters in the pitch sequence have to be accounted for.

letter	meaning	what to do with it
X	ball put into play by batter	simulate
Y	ball put into play on pitchout	ball

see, we could've done all of our work in one function from the get-go, but i wanted to show you how to construct this piece-by-piece. furthermore, this allowed me to show you the results like in the video. and that's the reason why this last function is called `full_sim`.

```r
# the ball in play simulation function.
# return true if it's a walk, false otherwise.
# i know we could cut some corners here, but i don't want to right now.
full_sim <- function(pitch_sequence) {
  # strike and ball counter
  balls <- 0
  strikes <- 0
  # iterate through the sequence
  for (i in strsplit(pitch_sequence,"")[[1]]) {
    # we got a simulated ball
    if (i %in% c("F","L","M","O","S","T","X")) {
      # generate random number between 1 and 1000
      r <- runif(n = 1, min = 1, max = 1000)
      # ball
      if (r < 192) {
        balls <- balls + 1
      } else {
        # strike
        strikes <- strikes + 1
      }
    } else if (i %in% c("C","K")) {
      # called strike, unknown strike
      strikes <- strikes + 1
    } else if (i %in% c("B","I","Q","P","Q","R","V","Y")) {
      # ball
      balls <- balls + 1
    } else if (i %in% c("H","N")) {
      # hbp or balk - automatic walks
      return(TRUE)
    }
    # check if we're done
    if (balls == 4) {
      return(TRUE)
    } else if (strikes == 3) {
      return(FALSE)
    }
  }
  # if we're here, we gotta get some more random numbers
  while (TRUE) {
    r <- runif(n = 1, min = 1, max = 1000)
    if (r < 588) {
      balls <- balls + 1
    }
    else {
      strikes <- strikes + 1
    }
    # check if we're done
    if (balls == 4) {
      return(TRUE)
    } else if (strikes == 3) {
      return(FALSE)
    }
  }
}
```

okay, let's see what happens.

```
# get the balls in play pas
bip <- barry$V8

# simulate balls in play
bip_to_walk <- sum(sapply(bip, full_sim))

# how many balls in play turned into walks?
print(bip_to_walk)
```

```
## [1] 141
```

```
# add them to our total
bb <- bb + bip_to_walk

# output final obp
print(obp())
```

```
## [1] 0.6061588
```

a .606 obp. that's impressive, but not *quite* what jon got. fortunately for us, we have automated the process, so we can pack this into one big ugly function and run this as often as we like.

turn on the bullshit machine

so we do what we said we would do. cram it all into one function for simulation. that's where our barry copy will come in. we use the last function we described and put it into the simulator.

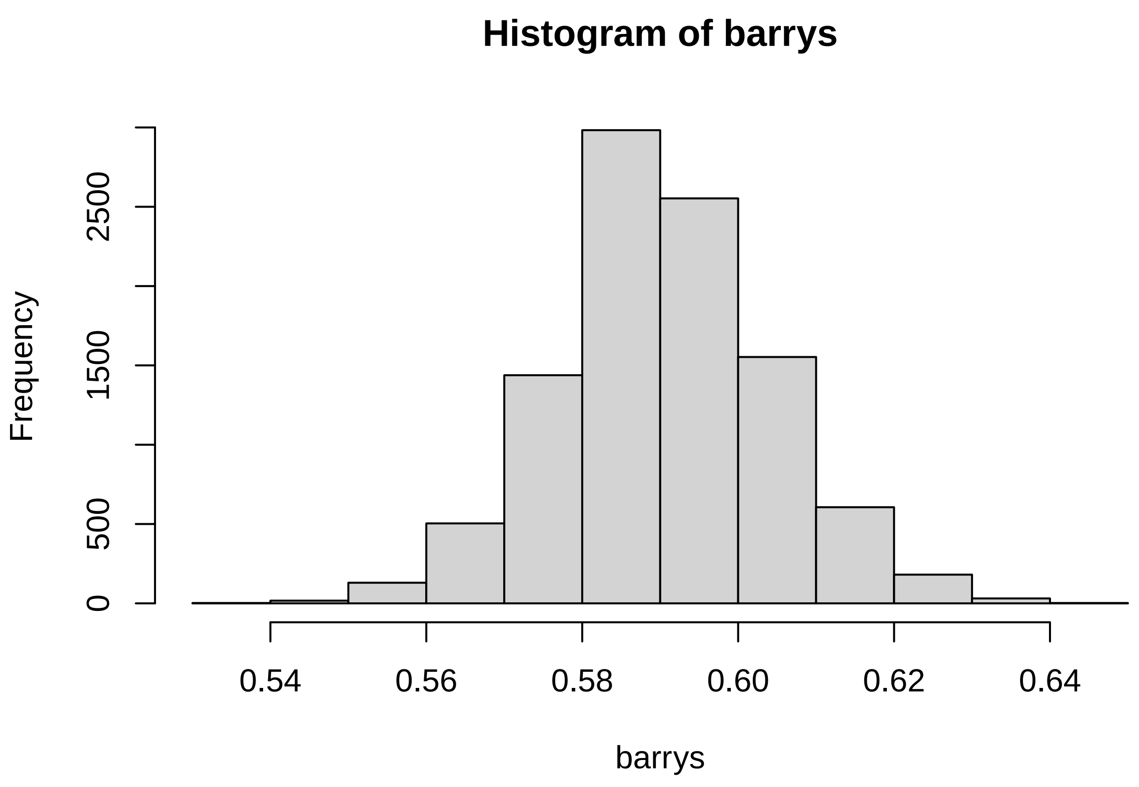
```
barry_bonds_without_a_bat <- function(pbp) {
  onbase <- sum(sapply(pbp, full_sim))

  return((onbase)/length(pbp))
}
```

and now for the grand finale. throw on the simulation simulator.

```
# simulate jon's simulation N=10000 times
N <- 10000
set.seed(1)
barrys <- replicate(N, barry_bonds_without_a_bat(barry_copy$V8))
```

what happens when we visualize the on-base percentages in a histogram?

```
hist(barrys)
```



so batless barry has a pretty good shot at being the player with the highest obp in history. we ran this simulation 10000 times, and more often than not batless barry is better than his 2003 self in terms of on-base percentage.

post script

so i found this reddit post by u/cg2916 after i wrote this. they took more than 5 minutes to read the retrosheet documentation. this helped me find some errors in my calculations. they claim that there's a 67.6% chance that batless barry gets the obp record. we can put that claim to the test:

```
sum(barrys > 0.5817)/N
```

```
## [1] 0.7506
```

additionally, they claim that barry has a 3.1% chance of beating his actual obp that year, which was 0.6094:

```
sum(barrys > 0.6094)/N
```

```
## [1] 0.1001
```

uh maybe not? it's higher? maybe i made a mistake. or maybe not. who knows. what does it say about me that i spent an afternoon reverse engineering this? nothing good