Beeminder WL Project

Kimberly Ebel

July 21, 2017

The Project

I was given a bunch of data from the productivity app Beeminder to look at. This was old data from a UCLA study that was conducted at some point. I believe this study is not yet published.

I took this challenge on for several reasons.

1. It was an opportunity to communicate with an actual working team and hopefully provide value to an existing organization.
2. It was opportunity to work with real data and not the kinds of tailored datasets that are often found in exercises, competitions, and most of our instructional materials. This was at least partially informed by the information that I've gotten from numerous people in the industry who kept telling me that "Data Munging/Wrangling/Processing is 80% of what you do as a Data Scientist." It seemed prudent to have a good idea of where abilities in this area lay, so that I could enter into interviews with confidence.
3. There are a lot of tracking apps and products and a lot of data in the areas covered by this and the QS communities. I imagine there are lots of exciting things to find out too. I've used the Beeminder app in the past to track different goals, and I found it to be helpful in setting up a physical activity routine. I wanted to find out how common my experiences were and - more broadly - how one goes about handling data of this nature.

Importing the Data

"The data came in 2068 files, one per user. In order to process it I had to import each file separately and then combine all of them together.

To do this I used the package jsonlite, and the function list.files() - plus some sweet substr() moves - to read these json files by filename. Finally, I use the list2env() function to take the names of the files and populate them in my environment, all at once.

```
library(jsonlite)
# load('Markdown_Environment.RData')


setwd("C:/Users/kim/Google Drive/001-SIGNAL/Week11/Beeminder Project/jmcn")

#getting and setting names
temp<-list.files(pattern="*.json")
names=substr(temp,nchar(temp)-28, nchar(temp)) #removes front parts of filename
names=substr(names,1,nchar(names)-5) #removes .json extension
names=lapply(names, function(x) {paste0('a',x)})   #adds a letter to the front of the name so R
 will read them as variables when they're converted
names=unlist(names)
head(names)

#pulls files in by filename
myfiles=lapply(temp,read_json)

#renames myfiles for actual user names
names(myfiles)=names

#extracting individual files to environment
list2env(myfiles,envir=.GlobalEnv)  #<environment: R_GlobalEnv>

ls()  #2068 entries
```

Transforming Data

These Json files were a mess! Lists inside lists inside lists inside…. well, you get the picture. Not exactly the easiest thing to wrangle. There was data that was attached at the user level and for each user, there were goals that had additional data underneath. I decided to separate the files into a user dataframe and a goals dataframe. Ultimately, my goal was to reach the point where I had a dataframe for each level that contained additional sets of data.

First, I decided to separate out the top layer by creating a dataframe where most of the columns were the user-level pieces of information. I named this dataframe user, and I left the other layered items in as a element for each user ID that contained a list of lists to be extracted later.

```
library(devtools)
library(DSR)
library(dplyr)
library(tidyr)

#creating user dataframe
user=as.data.frame(matrix(data=NA,nrow=2065, ncol=23))
rownames(user)=names
colnames(user)=names(a4db75efe86f2245b2800000b)

#run loop to populate all the columns
for (i in 1:length(colnames(user))) {
  colname=colnames(user)[i]
  column=sapply(names, function(x) {get(x)[colname]})
  column[sapply(column, is.null)]=NA
  user[[i]]=column
}
View(user)
rownames(user)=NULL
```

The next layer down was goal data, so I first simply extracted the columns for goal data out of the user df and created a new df with them. I then unlisted them once using the attribute recursive=FALSE for the unlist() function.

The weight loss data also had nested information inside of it. My next goal was to take the raw extracted weight loss data and transform it into a dataframe with information that would be more readily viewed and explored.

Within the wl_goals layer, there were both empty lists and NULL values which needed to be removed. I took advantage of the fact that both are considered to have zero length by R and wrote a function to replace everything with length < 1 with NA.

```
#pulling out wl_goals info

#pulls wl_goals from user df
wl_goals=user$wl_goals


#restructures list so that each element is a separate goal
wl_goals=unlist(wl_goals,recursive=FALSE)


#function to test for empty lists and NULL values and replace them with NA
NArepl=function(list) {
  for (j in 1:length(list)) {
    elem=list[[j]]
    for (i in 1:length(elem)) {
      # print(paste0("testing element ",i," of ",length(elem)))
      len=length(elem[[i]])
      if (len==0) {
        elem[[i]]=NA
      }
      list[[j]]=elem
    }
  }
  return(list)
}

#replacing NULLS for all of wl_goals
wl_goals=NArepl(wl_goals)
```

Following this, I was able to turn the extracted goals data into the goals dataframe. While this still had nested information, I was able to keep the nested information by testing to see if each element was a list. If it was, I kept it as such. If it wasn't, I transfered over the value to the dataframe as is. This also gave me another piece of information: this dataset included data from 3807 goals. I then went through and cleaned up the rownames and did other dataframe housekeeping so that this information could me more easily accessed in the future for further analysis.

```
#creating weight loss goals dataframe
goals=as.data.frame(matrix(data=NA,nrow=3807, ncol=24))
rownames(goals)=names(wl_goals)
colnames(goals)=names(wl_goals$a4db75efe86f2245b2800000b.wl_goals1)



#creating function to populate dataframe

list2df=function(df,list){
  lstlen=length(list)
  dfcol=ncol(df)
  for (j in 1:dfcol){
    for (i in 1:lstlen) {
      print(paste0("i,j is ",i,", ",j))
      if (is.list(list[[i]][[j]])==TRUE) {
        df[[i,j]]=list[[i]][j]
      }
      else {
        df[[i,j]]=list[[i]][[j]]
      }
    }
  }
  return(df)
}


goalsBKUP=goals
goals=list2df(goals,wl_goals)

#clearing up ID info in goals
colnames(goals)[[1]]="goal_id"

user_id=rownames(goals)
user_id=sapply(X = user_id,FUN = substr,2,nchar(user_id))
user_id=sapply(X = user_id,FUN = substr,1,nchar(user_id)-10)
goals=as.data.frame(cbind(user_id,goals,stringsAsFactors = FALSE))

rownames(goals)=NULL
```

I've also noticed that some goals have actual datapoints associated with them and some don't. These are individual weight and date entries, and seem like an idea raw data source to use for the bulk of my exploration as I go forward. Since a lot of what I discussed exploring with the Beeminder team was comparing weight change on goal and off goal, it seems clear that I'll need actual data points in order to explore this.

I create a dataframe for only those goals that have a non-zero amount of information in the datapoints column and go through additional processing on it. As I create the datapoints df, I remove the created_at, goaldate, road, and runits variables, as those have more to do with graphing on the beeminder site and won't be used for this analysis. Then I change the date data into date data format.

This takes my number of goals from 3807 to 2833. Out of curiosity, I check to see how many users this makes. The answer is 2012.

```
datapoints=goals[is.na(goals$datapoints)==FALSE,-c(3,5,17:19)]



#changing datapoints data into date format in R
for (i in 1:nrow(datapoints)) {
  for (j in 1:length(datapoints[[i,4]])){
    datapoints[[i,4]][[j]][[1]]=as.Date(datapoints[[i,4]][[j]][[1]],origin='1970-01-01')
  }
}


length(unique(datapoints$user_id)) #2012
colnames(goals)
colnames(datapoints)

#cleaning up datapoints gunits for later
unique(datapoints$gunits)  #[1] ""   NA   "kg" "lb"
datapoints$gunits[[13]]
unit_ind=which(datapoints$gunits==datapoints$gunits[[13]])
datapoints[unit_ind,'gunits']=NA
```

I then pull out derails and pledges data as separate lists to do further processing on later.

```
#pulls pledges from dataframes df
pledges=datapoints$pledges

#restructures list so that each element is a separate goal
pledges=unlist(pledges,recursive=FALSE)

#replacing NULLS for all of pledges
pledges=NArepl(pledges)



#pledges is now at the point that datapoints is, where further nestings are of differing length
s.


#pulls derails from dataframes df
derails=datapoints$derails

#restructures list so that each element is a separate goal
derails=unlist(derails,recursive=FALSE)


#replacing NULLS for all of derails
derails=NArepl(derails)


#derails is now at the point that datapoints is, where further nestings are of differing length
s.
```

Rabbits, Rabbits Everywhere, and Nary a Carrot in Sight

At this point, a few days have passed. I've learned a lot about importing data into R that hadn't been covered previously, which has been great. I'm also still trying to figure out the ultimate structure I want this data to take so that I can glean meaningful insights from it.

Even though I'm not actually done transforming the data, I'm curious about what sorts of insights I might be able to ferret out from just the things I've done. So I put together a few vectors and graphed them, just to give myself my first taste of what this data looks like.

```
# Number of weight loss goals by user
num_goals=sapply(user$wl_goals,length)
names(num_goals)=user$id
hist(num_goals)


#number of other goals by user
other_goals=sapply(user$other_goals,length)
names(other_goals)=user$id
hist(other_goals)


#number of total goals by user
total_goals=num_goals+other_goals
names(total_goals)=user$id
hist(total_goals)


#has pledge history
has_pledge_hist=is.na(goals$pledges)
names(has_pledge_hist)=goals$goal_id
table(has_pledge_hist)
barplot(table(has_pledge_hist), main='Has Pledge History')


#number of derails by goal
zeroOut=which(is.na(goals$datapoints)==TRUE)  #getting the indices of NAs to zero out later
goals$datapoints=unlist(goals$datapoints,recursive = FALSE) #unlisting datapoints to get length
 back
num_drail=sapply(goals$datapoints,length)
num_drail[zeroOut]=0  #zeroing out the goals we know are NAs
names(num_drail)=goals$goal_id
hist(num_drail)
```

Back to the Trenches

After this brief visual interlude in which I gain a glimpse of some of the tasty inner workings of the multi-layered wall of text I was originally sent, I go back to figuring out what it is I want to do about the data that I have.

The problem I run into is that each goal has a different number of datapoints, and I have over 2000 goals. My options are clutter up my environment with that many objects, or find some other way to get at the data.

After examining things further, I decide to extract just the date and the weight from the datapoints data, along with some additional information from the users table, and create one massive dataframe with one line per datapoint. Since I'll have the dates, I won't need the created dates. This process takes a while to run.

```
#more prepping of datapoints df for extraction
length(datapoints$datapoints) #2833
length(datapoints$datapoints[[1]]) #1
length(datapoints$datapoints[[1]][[1]])  #1531 - we want one element up to be this length, so...
datapoints$datapoints=unlist(datapoints$datapoints,recursive=FALSE)




#calculating total number of rows in final dataframe
totaldp=c(rep(0,2833))  #creates vector to sum later
singledp=c()  #records indices of rows with only one datapoint
for (goal in 1:nrow(datapoints)){
  totaldp[[goal]]=length(datapoints$datapoints[[goal]])
  if (totaldp[[goal]]==1) {
    singledp=c(singledp,goal)
  }
  print(paste0('goal ',goal,' has ',totaldp[[goal]],' datapoints.'))
}
sum(totaldp)  #1167525, then 1167353
singledp

#because we'll be comparing datapoints, we'll remove the goals that have only one datapoint.
datapoints=datapoints[-singledp,]
#datapoints now has 2661 observations


dat=data.frame(matrix(nrow=1167353,ncol=9,dimnames=list('rows'=list(),'columns'=c('user_id','goa
l_id','date','weight','u_metric','g_metric','comment','ideal_wt','is_payer'))))
dat$date=as.Date(dat$date,origin='1970-01-01')  #sets the data type so it imports alright




#WARNING:  this takes ~36 hours to run.
counter=0
for (rw in 1:nrow(datapoints)) {  #2661
  print(paste0('populating goal ',rw,' of 2833'))
  uid=datapoints[[rw,1]]
  gid=datapoints[[rw,2]]
  u_met=user[user$id==datapoints[[rw,1]],'sci_units']
  i_weight=user[user$id==datapoints[[rw,1]],"ideal_weight"]
  payer=user[user$id==datapoints[[rw,1]],"is_payer"]
  g_met=datapoints$gunits[[rw]]
  for (dp in 1:length(datapoints[[rw,'datapoints']])) {
    counter=counter+1
    if (counter %% 500==0) { #500
      print(paste0('populating new df row ',
                   counter,
                   ' with datapoint ',
                   dp,' of ',
                   length(datapoints[[rw,'datapoints']])))
    }
```

```
    for (dp_elem in 1:length(datapoints[[rw,'datapoints']][[dp]])) {
      if (is.null(datapoints[[rw,'datapoints']][[dp]][[dp_elem]])==TRUE){
        datapoints[[rw,'datapoints']][[dp]][[dp_elem]]=NA
      }
      else {
        datapoints[[rw,'datapoints']][[dp]][[dp_elem]]=datapoints[[rw,'datapoints']][[dp]][[dp_e
lem]]
      }
    }
    dat$date[[counter]]=as.Date(datapoints[[rw,'datapoints']][[dp]][[1]],origin=1970-01-01)
                                          #1970-01-01 is the beginning of unix time
    dat$weight[[counter]]=as.numeric(datapoints[[rw,'datapoints']][[dp]][[2]])
    dat$comment[[counter]]=datapoints[[rw,'datapoints']][[dp]][[3]]
    dat$g_metric[[counter]]=g_met
    dat$user_id[[counter]]=uid
    dat$goal_id[[counter]]=gid
    dat$u_metric[[counter]]=u_met
    dat$ideal_wt[[counter]]=i_weight
    dat$is_payer[[counter]]=payer
    }
  }


#1167353 obs of 9 variables

#unlisting u_metrics as it populated as a list of single element lists
dat$u_metric=unlist(dat$u_metric, recursive=FALSE)
```

Once I've done this, I do some more processing on the data frame I've created. Some of the things that come to mind as needing done are : convert measurements into single unit. Identify and deal with outliers and measurement errors.

For measurements, I decide to go with US vs metric, as those are more intuitive to me.

```
#clearing out environment

rm(dp)
rm(dp_elem)
rm(g_met)
rm(gid)
rm(goal)
rm(counter)
rm(rw)
rm(singledp)
rm(uid)
rm(u_met)
rm(zeroOut)

# getting the userIDs for the users that have measurements in the metric system
met_ind=which(dat$u_metric=='MET') #216767
length(unique(dat[met_ind,1])) #271 users


#checking to see if there are any obs where user measurement unit is set to something different
 from goal measurement unit
which(dat$g_metric=='lb' & dat$u_metric=='MET')  #0
which(dat$g_metric=='kg' & dat$u_metric=='US')  #0
length(which(dat$g_metric=='kg' & dat$u_metric=='MET'))  #4636
length(which(dat$g_metric=='lb' & dat$u_metric=='US'))  #7994

#creating these lists for error checking later
dbl_cnt_met=which(dat$g_metric=='kg' & dat$u_metric=='MET')
dbl_cnt_US=which(dat$g_metric=='lb' & dat$u_metric=='US')

length(unique(dat$user_id))  #1949 users in dat
length(unique(dat$goal_id))  #2661 goals in dat


#1kg=2.20462lbs.  converting kg to lbs in original dat
dat[met_ind,4]=round(2.20462*dat[met_ind,4],2)
dat[met_ind,5]='US'  #changing the df to reflect the change in data
g_met_ind=which(dat$g_metric=='kg')
dat[g_met_ind,6]='kg2lb'
length(dat[dat$g_metric=='kg2lb',6])
```

Additional Cleaning

Finally, wanted to make sure that we were only comparing adults here. Per the CDC Anthropometric Reference Data for Children and Adults: United States, 2011-2014 (https://www.cdc.gov/nchs/data/series/sr_03/sr03_039.pdf (https://www.cdc.gov/nchs/data/series/sr_03/sr03_039.pdf)) only 5% of adult women 80 years or older weigh < 100 lbs, so we removed all datapoints where the weight in US lbs was under 100. This removed 248 users and 419 goals from our data, leaving us with 1701 unique users and 2242 unique goals.

Next we wanted to identify drastic weight changes over small periods of time, as these are likely measurement errors of some kind. We set a threshold of changes that are over 50# in a 14 day period. First we ordered dat by user_id, goal_id, and date so that when we compare entries we know they're arranged chronologically and they're

still for the correct users/goals.

This leaves us with information on 42 users/95 goals, which we remove, leaving us with 1659 users and 2147 goals to examine.

```
#ordering dat
dat_ord=dat[order(dat$user_id,dat$goal_id,dat$date),]
dat=dat_ord
rm(dat_ord)

#identifying drastic weight changes. This takes 2-3 hours to run.
ind=1
wt_ch_prob_usr=c()
for (change in 1:(nrow(dat)-1)) {  #nrow: 949105
  if (dat$user_id[[change]] %in% wt_ch_prob_usr) {
    ind=ind
  }
  else if ((dat$user_id[[change+1]]==dat$user_id[[change]]) &
           (dat$date[[change+1]]-dat$date[[change]] <= 14)  &
           (dat$weight[[change+1]]-dat$weight[[change]]>=50)){
    wt_ch_prob_usr[[ind]]=dat$user_id[[change]]
    print(ind)
    ind=ind+1
  }
}

#this list contains 42 users
length(unique(dat[dat$user_id %in% wt_ch_prob_usr,2])) # 95 goals would be removed by removing t
hese 42 users

rmv=which(dat$user_id %in% wt_ch_prob_usr)
length(which(dat$user_id %in% wt_ch_prob_usr))  #47229
dat=dat[-rmv,]
#dat now has 901877 obs of 9 variables

length(unique(dat$user_id))  #now 1659 from 1701 from 1949 users in dat
length(unique(dat$goal_id))  #now 2147 from 2242 from 2661 goals in dat
```

Transformation, ahoy

Up until now we've been playing with the structure of the data and getting things to where we can get to them more easily. In the last chunk, we did some conversions for measurements, and a whole lot of extraction. Now we'll be looking at the data and seeing what it has to say.

One of the things I discussed doing with the Beeminder team was looking at how much users' weight had changed while currently tracking a goal vs how much it changed in between goals. In order to do this, I needed to do a bit of feature engineering, the details of which are set out below.

Weight Loss/Gain Feature Engineering

First, I set a threshold of 14 days (2 weeks), and ran a loop to extract dates which came before and after. This gives me the dates which bookend 'time on goal' and 'time off goal'.

I write a function called ‘datemark’ that checks for gaps of 14 days or longer when a user hasn’t tracked any data. When it finds those gaps, it extracts the dates before and after into a dataframe called imp_dates (for Important Dates) and indicates if this date is the day the user stops tracking their weight (‘goes off goal’, which is marked in the data with a FALSE) or the day the user starts tracking their weight again (‘goes on goal’, which is marked in the data with at TRUE).

```
#getting important dates

#important: use the subset function to create the df to pass into here. If you use bracket subse
tting the nrow function will grab NAs as well.  Don't know why. It's stupid.  Also, make sure yo
u order the df you're passing in by date.



#getting important dates

#important: use the subset function to create the df to pass into here. If you use bracket subse
tting the nrow function will grab NAs as well.  Don't know why. It's stupid.  Also, make sure yo
u order the df you're passing in by date.



datemark=function(df) {
  #this relies on the df having the same columns and column values as the                dat da
taframe has. currently:  "user_id"  "goal_id"  "date"     "weight"                  "metric"   "s
ource"   "ideal_wt" "is_payer"
  library(plyr)

  #getting the number of rows of the passed-in df to pass into the for loop later
  Ldf=nrow(df)


  #creating the dataframe we'll return
  imp_dates=data.frame('date'=c(0),'weight'=c(0),'status'=c(NA))

  if (Ldf <=1){
    return(imp_dates)
  }

  #setting the data type for date column
  imp_dates$date=as.Date(imp_dates$date, origin='1970-01-01')

  #populating the first row of the returning dataframe
  imp_dates[[1,1]]=df$date[[1]]
  imp_dates[[1,2]]=df$weight[[1]]
  imp_dates[[1,3]]=TRUE

  #setting the counter for the returning dataframe rows
  cnt=2

  #checking for gaps of longer than 14 days between dates and extracting that info into the retu
rning dataframe
  for (date in 1:(Ldf-1)){
    if (df$date[[date+1]]-df$date[[date]] > 14) {
      imp_dates[[cnt,1]]=df$date[[date]]
      imp_dates[[cnt,2]]=df$weight[[date]]
      imp_dates[[cnt,3]]=FALSE
      imp_dates[[cnt+1,1]]=df$date[[date+1]]
      imp_dates[[cnt+1,2]]=df$weight[[date+1]]
```

```
      imp_dates[[cnt+1,3]]=TRUE
      cnt=cnt+2
    }
  }

  #populating the final row
  imp_dates[[cnt,1]]=df$date[[Ldf]]
  imp_dates[[cnt,2]]=df$weight[[Ldf]]
  imp_dates[[cnt,3]]=FALSE

  imp_dates=imp_dates[order(imp_dates$date), ,drop=FALSE]
  return(imp_dates)
}
```

I then construct a for-loop that computes aggregate statistics on that weight change over time. It calculates the number of days in the user is ‘on goal’ and the weight change over that time, as well as the number of days the user is ‘off goal’ and the weight change over that time. I create a data frame to hold this information and I call it g_stats, as the program batches the lines by unique goal.

I then construct another function called stats_pop_goal that uses the for-loop and the previous datemark function to populate this data frame using all the data extracted and run it. Over the course of running it, I realize that there were a number of goals that had one and only one data point in them. I replace the info for those goals with NA (as there is no weight change that can be measured with that data.) I also constructed NAs for drastic important_dates weight changes over 100# as an additional layer of error check.

```
#this is using the datapoints dataframe created above and the dat as it is at this point in the
 process.  If those have been changed, parts of this function will not work.


stats_pop_goal=function(df) {

  #creating a list of unique goal IDs left in dat
  gid=unique(dat$goal_id)

  #creating a list of unique user IDs left in dat
  uid=unique(dat$user_id)

  #creating a reference df for getting the info for each gid/uid combo we have
  dp=datapoints[datapoints$goal_id %in% gid,]
  all.equal(uid,unique(dp$user_id))   #TRUE


  #creating the df to populate:
  goal_stats=data.frame('user'=(rep(0,nrow(dp))),
                        'goal'=(rep(0,nrow(dp))),
                        'total_time_on'=c(rep(0,nrow(dp))),
                        'total_wt_ch_on'=c(rep(0,nrow(dp))),
                        'total_time_off'=c(rep(0,nrow(dp))),
                        'total_wt_ch_off'=c(rep(0,nrow(dp))),stringsAsFactors = FALSE)




  for (goal in 1:nrow(dp)) { #

    #subsetting the larger data-frame
    batch=subset(dat,(user_id==dp$user_id[[goal]] & goal_id==dp$goal_id[[goal]]))
    goal_stats$user[[goal]]=dp$user_id[[goal]]
    goal_stats$goal[[goal]]=dp$goal_id[[goal]]
    print(paste0('creating dates for batch ',goal,' of ',nrow(dp)))

    # creating the important dates data for each subset
    imp_dates=datemark(batch)

    for (change in 1:(nrow(imp_dates)-1)) {
      #removing goals with only one important date
      if((nrow(imp_dates) <= 1)) {
        goal_stats$total_time_on[[goal]]=NA
        goal_stats$total_wt_ch_on[[goal]]=NA
        goal_stats$total_time_off[[goal]]=NA
        goal_stats$total_wt_ch_off[[goal]]=NA
        print(paste0('line ',goal,' has only one datapoint and will be removed from consideratio
n'))
      }
      #calculating the goal stats with info from each batch's important dates
      else if (imp_dates$status[[change]]==TRUE & imp_dates$status[[change+1]]==FALSE){
        goal_stats$total_time_on[[goal]]=
          goal_stats$total_time_on[[goal]]+
```

```
          as.numeric(imp_dates$date[[change+1]]-imp_dates$date[[change]])
        goal_stats$total_wt_ch_on[[goal]]=
          goal_stats$total_wt_ch_on[[goal]]+
          (imp_dates$weight[[change+1]]-imp_dates$weight[[change]])
      }
      else if (imp_dates$status[[change]]==FALSE & imp_dates$status[[change+1]]==TRUE){
        goal_stats$total_time_off[[goal]]=
          goal_stats$total_time_off[[goal]]+
          as.numeric(imp_dates$date[[change+1]]-imp_dates$date[[change]])
        goal_stats$total_wt_ch_off[[goal]]=
          goal_stats$total_wt_ch_off[[goal]]+
          (imp_dates$weight[[change+1]]-imp_dates$weight[[change]])
      }
    }
  }
  return(goal_stats)
}
```

With this completed, we run the stats_pop_goal on dat and remove the rows with NAs generated by the program. This leaves us with information for 2131 goals and 1650 users.

```
g_stats=stats_pop_goal(dat)

#removing rows with NAs
remove_ind=which(is.na(g_stats$total_time_on)==TRUE)
length(remove_ind) #16
g_stats=g_stats[-remove_ind,]


length(unique(dat$user_id))  #now 1659 from 1701 from 1949 users in dat
length(unique(dat$goal_id))  #now 2147 from 2242 from 2661 goals in dat

nrow(g_stats)    # 2131 observations/goals left
length(unique(g_stats$user))  #1650 users
```

Finally, I aggregate the by-goal data by user to generate user statistics as well.

```
library(ggplot2)

# setwd('C:\Users\kim\Google Drive\001-SIGNAL\Week11\Beeminder Project\Week2')
load('Markdown_Environment.RData')
stats_uid=unique(g_stats$user)

u_stats=data.frame('user'=stats_uid,
                   'num_goal'=c(rep(0,length(stats_uid))),
                   'total_time_on'=c(rep(0,length(stats_uid))),
                   'total_wt_ch_on'=c(rep(0,length(stats_uid))),
                   'total_time_off'=c(rep(0,length(stats_uid))),
                   'total_wt_ch_off'=c(rep(0,length(stats_uid))),stringsAsFactors = FALSE)

for (user in 1:length(stats_uid)) {
  u_stats$num_goal[[user]]=length(g_stats[g_stats$user==stats_uid[[user]],'goal'])
  u_stats$total_time_on[[user]]=sum(g_stats[g_stats$user==stats_uid[[user]],'total_time_on'])
  u_stats$total_wt_ch_on[[user]]=sum(g_stats[g_stats$user==stats_uid[[user]],'total_wt_ch_on'])
  u_stats$total_time_off[[user]]=sum(g_stats[g_stats$user==stats_uid[[user]],'total_time_off'])
  u_stats$total_wt_ch_off[[user]]=sum(g_stats[g_stats$user==stats_uid[[user]],'total_wt_ch_off']
)
}


u_stats[[7]]=round(u_stats$total_wt_ch_on/u_stats$total_time_on,2)
colnames(u_stats)[[7]]="wt_ch_per_day_ON"
u_stats[[8]]=round(u_stats$total_wt_ch_off/u_stats$total_time_off,2)
colnames(u_stats)[[8]]="wt_ch_per_day_OFF"
u_stats[[9]]=rowSums(u_stats[,c('total_wt_ch_off','total_wt_ch_on')])
colnames(u_stats)[[9]]="TOTAL_wt_ch"
u_stats[[10]]=u_stats$wt_ch_per_day_ON*7
colnames(u_stats)[[10]]="Avg_wt_ch_by_week_on"
u_stats[[11]]=u_stats$wt_ch_per_day_OFF*7
colnames(u_stats)[[11]]="Avg_wt_ch_by_week_off"
u_stats[[12]]=rowSums(u_stats[,c('total_time_off','total_time_on')])
colnames(u_stats)[[12]]="TOTAL_time"
u_stats[[13]]=round((u_stats$TOTAL_wt_ch/u_stats$TOTAL_time)*7,2)
colnames(u_stats)[[13]]="avg_TOTAL_wt_ch_by_week"
length(which(u_stats$avg_TOTAL_wt_ch_by_week < 0) )  #950
```

```
## [1] 950
```

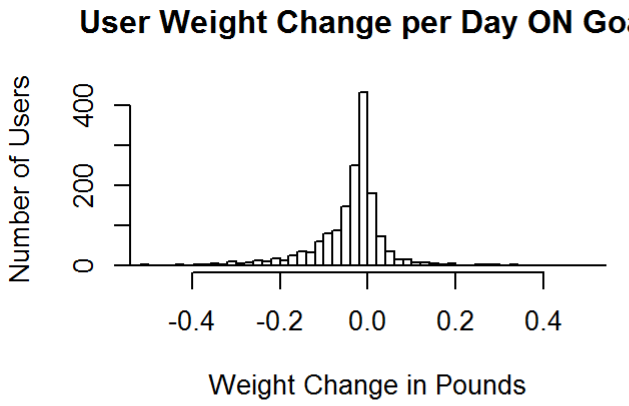
```
u_stats[[14]]=round((u_stats$TOTAL_wt_ch/u_stats$TOTAL_time),2)
colnames(u_stats)[[14]]="avg_TOTAL_wt_ch_by_day"
length(which(u_stats$avg_TOTAL_wt_ch_by_day < 0) )  # 789
```

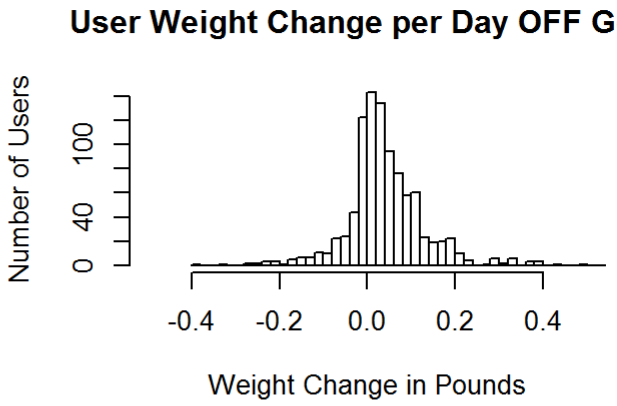
```
## [1] 789
```

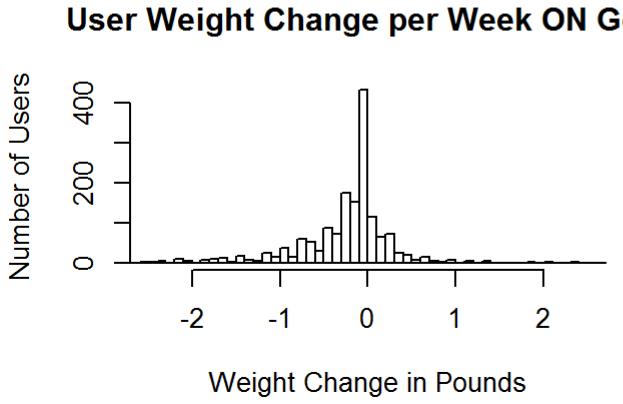
```
avg_num_goals=mean(u_stats$num_goal)  #1.291515 for wl_goals
avg_wt_change=mean(u_stats$TOTAL_wt_ch) #-3.662159
num_u_lost=as.numeric(length(which(u_stats$TOTAL_wt_ch < 0))) #974
num_u_gained=as.numeric(length(which(u_stats$TOTAL_wt_ch > 0))) #648
```

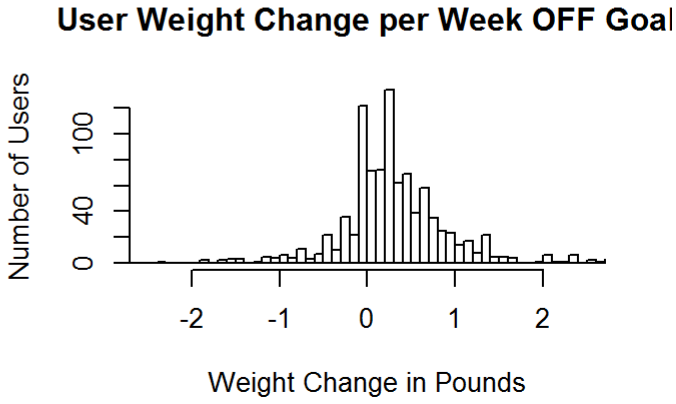
Results

After all of this, let's take a look at some of the information our data can give us:





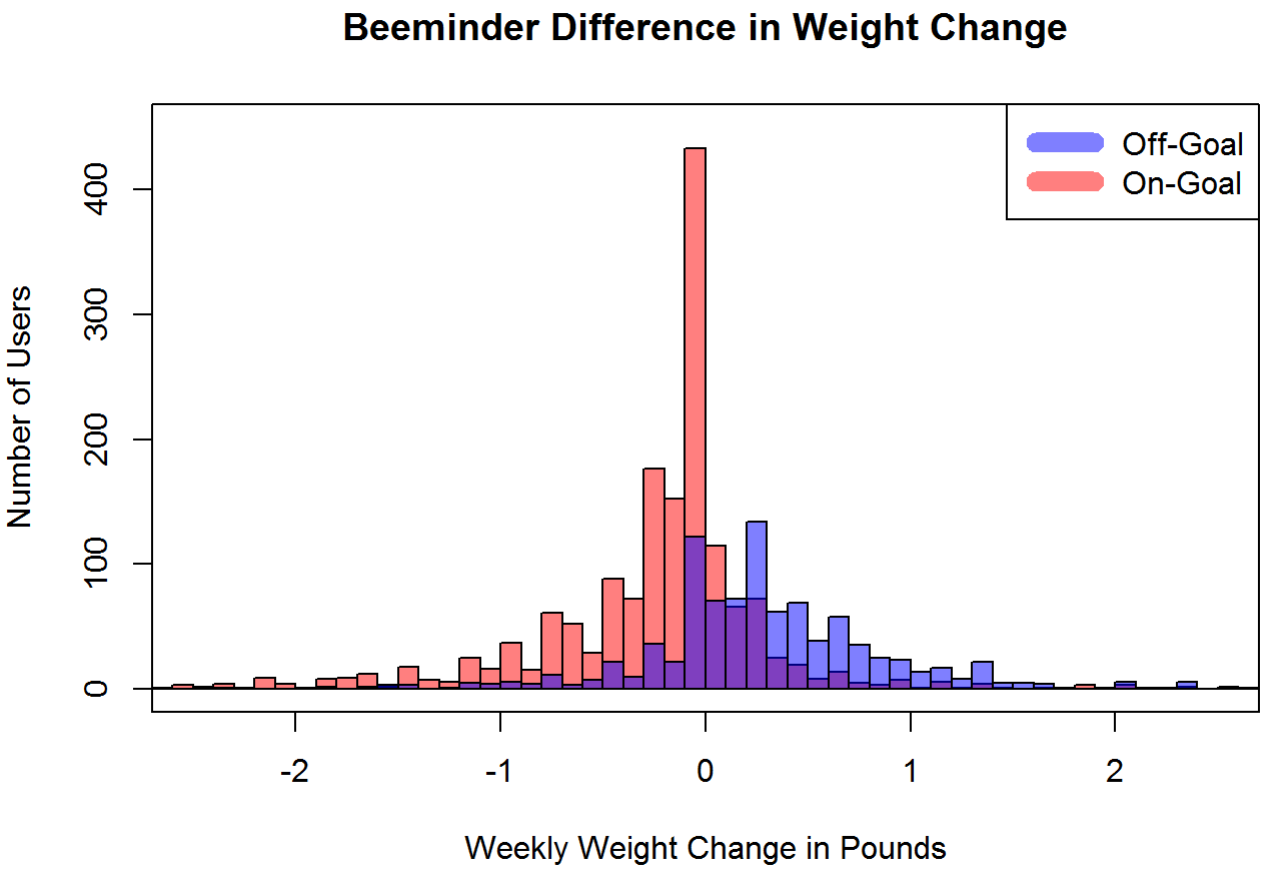




As you can see, users lost more weight while tracking than they did while not tracking. This is true both on a day-by-day basis as well as a weekly basis. Indeed, if we look at the raw numbers, we can see that of the 1650 users remaining, the average weight change was -3.66 pounds and 974 users lost weight compared to the 648 who didn't.

But perhaps, in the end, a picture may be worth more than words. So I leave you with the following.

```
{hist(u_stats$Avg_wt_ch_by_week_on, breaks=500, col=rgb(1,0,0,0.5),xlim=range(-2.5:2.5), ylim=c(
0,450), main='Beeminder Difference in Weight Change', xlab='Weekly Weight Change in Pounds', yla
b="Number of Users")
par(new = TRUE)
hist(u_stats$Avg_wt_ch_by_week_off, breaks=500, col=rgb(0,0,1,0.5), xlim=range(-2.5:2.5), ylim=c
(0,450), add=T)
legend("topright", c("Off-Goal", "On-Goal"), col=c(rgb(0,0,1,0.5), rgb(1,0,0,0.5)), lwd=10)
box()}
```


Beeminder Difference in Weight Change
Off-Goal
On-Goal
Number of Users
0
100
200
300
400
-2
-1
0
1
2
Weekly Weight Change in Pounds