Demography 213
Calculating rates using tapply() and IPUMS

Carl Mason
carlm@demog.berkeley.edu

September 26, 2016

Contents

Abstract

This week we will use the American Community Survey (ACS) to calculate age specific rates. The example herein is to calculate ASFR and TFR for US resident women broken out by some interesting characteristic. Your job will be to find some other sort of rate to calculate – and make a table and perhaps a simple graph of it.

Monday we will have a show and tell and *next week* we'll spend some time producing clear and beautiful graphs of this week's results.

While it is perfectly reasonable to solve this week's problem using loops–it is modestly advantageous to use `tapply()`.

1 Introduction

The *Total Fertility Rate* is a synthetic measure that sort of more or less estimates the number of children a woman would have if she lived through her entire reproductive life and behaved—fertility wise– in the same way as women of each age *at a particular point in time* behave. This number is often reported in the media as the "number of children per woman". It is a nothing more than the sum of the *Age Specific Fertility Rates* (ASFR)s over the reproductive years. In other words, the work is in is calculating the ASFR.

Equation 1 shows the well known formulae for ASFR and TFR.

This week, we will calculate ASFRs and (trivially) TFRs for US resident, non-institutionalized women disaggregated by some characteristic or another. Although fertility is kind of interesting, what this class is really about is calculating rates – or more generally measures that require numerators and denominators. Proportions and probabilities are like this too. This comes up a lot in Demography. Demographers tend to see the world in terms of events (numerator) and risk sets (denominator). As Equation 1 makes "clear" an ASFR is the number of events (births to women of a certain age) divided by the risk set (the number of person years experienced by women of that age in that calendar year).

The `tapply()` function is particularly well suited to doing these sorts of calculations – and these sorts of calculations are what you will be doing this week.

$$\text{ASFR}_a = \frac{\sum \text{births to women of age a}}{\sum \text{woman years of exposure at age a}} \tag{1}$$

$$\text{TFR} = \sum_{a=15}^{45} \text{ASFR}_a = \sum_{a=15}^{45} E(\text{births}_a | \text{Survival through age a}) \tag{2}$$

$$a = \text{Single year of age} \tag{3}$$

2 Data

For this project we will use data from the Integrated Public Use Microseries (IPUMS). IPUMS is a fabulous resource for demographers so if you have not encountered it before, then you are in for treat.

If you have not already done so, visit `www.ipums.org/usa` and sign up for access. Do this by clicking on `Login` and then taking advantage of the obvious opportunity that follows (to Request an account).

I have already created an extract of the 2009 American Community Survey (ACS) which is adequate for doing the example. You **may/should** want to create your own extract for the second part of the assignment.

3 Assumptions and considerations

The ACS provides age and sex as well as a large number of other characteristics of each person in the sample (which is 1 percent of the US). Crucially, they also ask whether or not each person has had a child in the last year (actually, the Census is clever enough to only ask this question of women). It would have been nicer if they had asked for the *date of the last birth* but they did not. So it goes. We can estimate a set of ASFRs with what we have. (If we were ambitious we *might* be able to improve our estimate by using the year of birth variable – but let's not)

Some details that we shall need to consider are the following:

- The data are not a simple random sample. **We must use weights.**
- It is feasible and therefore desirable to use single year age cohorts for women at risk of birth.
- We would like to calculate ASFRs/TFRs for women disaggregated by some interesting characteristic. There are many to choose from.
- Mortality *could* have an effect–some women might have died before the Census Bureau could interview them. Their births and their person-years-at-risk are lost to us.
- Twin and multiple births pose problems (for parents as well as demographers). The possibility of multiple single births during the (one year) period of observation also creates problems for all concerned.
- We are calculating a ***period*** TFR rather than a *cohort* TFR, consequently, we would like to observe people over a particular calendar year. In a *perfect* world, that year of observation would also correspond exactly to the ages of those under observation. The ACS does not work that way:
  - The ACS surveys are collected throughout the year. The ACS sample is divided into 12 bits and surveys are administered to one bit every month over the year.
  - Relatively few women filled out the ACS survey on their birthday.

We deserve a simpler world than the one we forced to study. Fortunately, as social scientists, we can fight back against the forces of unreasonable complexity with our one most powerful weapon: **The Assumption**.

While we will explicitly deal with sample weights– we can assume our way out of many difficulties by:

- Defining "birth" as one *or more babies*.
- Ignoring the possibility of two singleton birth events in a single year. (known sometimes impolitically as *Irish Twins*)

- Assuming that the probabilities of dying and of giving birth are **not** correlated.
- Assuming that womens' birthdays are randomly distributed throughout the year.
- Defining our ASFRs as applying in some way to 2008 and 2009.

Pause briefly to consider the implications. OK that's long enough.

4 Example: calculating ASFR and TFR

The rest of this document includes a bunch of Rcode intersperced with pithy remarks. Many serious young scientists have found that running the code in R as you read is a better way to learn than simply to read and forget. If that's you, why not browse to `rstudio.demog.berkeley.edu` and, as always, create a NEW project in the already existing 213/WeekN directory. You may also wish to copy `~carlm/213/TFRI/demonstration.r` into your WeekN directory – although cutting and pasting, or re-typing are also options.

For this example we'll read in an IPUMS data extract that I already created. In Section 5.1, it is revealed, how you will create your own IPUMS data extract and prepare it to be read into R. **NOTE that the procedure for reading in the data in this section is different from the one you will pursue on your own.** Refer to Section 5.1 when you are ready to import your own data extract.

```
##############################################################################
## Read the data into R AND
##
## CAREFULLY examine each and every variable that we use to make
## sure that we understand what each value means
##############################################################################

## the "foreign" library has the code to read and process stata .dta files
## NOTE this is not a raw file simply downloaded from the IPUMS I had to
## either use stata to create it OR download it AS A STAT .dta file
## from IMPUS...  just as you will do later...

library(foreign)
## acs09.dta is file that I downloaded long ago and saved as a
## stata .dta file.  The fun of downloading data from IPUMS and
## storing it as a stata .dta file is something that you enjoy later
## -- in order to get to the point faster we'll just read in the one
## that I have stored.

## note that the path is /data/carlm  NOT ~carlm
```

```
acs09<-read.dta(file=paste("/data/carlm/ACS2009/acs09.dta"))
names(acs09)
```

```
##  [1] "year"      "datanum"   "serial"    "hhwt"      "region"
##  [6] "statefip"  "gq"        "pernum"    "perwt"     "famsize"
## [11] "nchild"    "nchlt5"    "age"       "sex"       "marst"
## [16] "marrno"    "marrinyr"  "widinyr"   "divinyr"   "fertyr"
## [21] "race"      "raced"     "bpl"       "bpld"      "citizen"
## [26] "yrnatur"   "yrimmig"   "hispan"    "hispand"   "hcovany"
## [31] "hcovpriv"  "hinsemp"   "hinspur"   "occ"       "migrate1"
## [36] "migrate1d" "diffmob"
```

```
## Once the data have in read into the object acs09, we can explore it and
## clean it up

## sum the person weight to see how many people the observations represent
sum(acs09$perwt) ## 75.9 million women between age 15 and 50
```

```
## [1] 75974672
```

```
################################################################################
## IPUMS GOTCHA # 1 --
## It is less than obvious that the acs09$age variable is NOT
## numeric.
################################################################################
is.numeric(acs09$age)
```

```
## [1] FALSE
```

```
is.factor(acs09$age)
```

```
## [1] TRUE
```

```
## levels(acs09$age)
## look at all these stupid levels -- many of which in
##this case have no observations.



## create a numeric age variable
acs09$ageN<-as.numeric(acs09$age)
## make sure it looks right
table(acs09$ageN)
```

```
##
##    16    17    18    19    20    21    22    23    24    25    26    27
## 20231 20318 20815 20648 19442 17924 17160 16530 17209 17110 17088 17239
##    28    29    30    31    32    33    34    35    36    37    38    39
```

```
## 17532 17824 18005 18402 17777 17792 17265 17429 17828 17983 18853 20534
##    40    41    42    43    44    45    46    47    48    49    50    51
## 20579 20932 19360 20286 20679 22050 23425 22969 23360 23886 23918 24931

table(as.character(acs09$age))

##
##    15    16    17    18    19    20    21    22    23    24    25    26
## 20231 20318 20815 20648 19442 17924 17160 16530 17209 17110 17088 17239
##    27    28    29    30    31    32    33    34    35    36    37    38
## 17532 17824 18005 18402 17777 17792 17265 17429 17828 17983 18853 20534
##    39    40    41    42    43    44    45    46    47    48    49    50
## 20579 20932 19360 20286 20679 22050 23425 22969 23360 23886 23918 24931

   #########
   ## WTF?##
   #########

acs09$ageN<-as.numeric(as.character(acs09$age))
table(acs09$ageN)

##
##    15    16    17    18    19    20    21    22    23    24    25    26
## 20231 20318 20815 20648 19442 17924 17160 16530 17209 17110 17088 17239
##    27    28    29    30    31    32    33    34    35    36    37    38
## 17532 17824 18005 18402 17777 17792 17265 17429 17828 17983 18853 20534
##    39    40    41    42    43    44    45    46    47    48    49    50
## 20579 20932 19360 20286 20679 22050 23425 22969 23360 23886 23918 24931

## HEADS UP -- in this case we have no zero age observations because
## such folks were not asked about thier fertility.  IF YOUR DATA SET
## CONTAINS SUCH PEOPLE there is one extra step:

## acs09$ageN[acs09$age=="Less than 1 year old"]<-0

###########################################################################
## IPUMS GOTCHA #2  --
## fertyr is the answer to the question: did respondent have birth in
## the last year?  One would think that this was a yes/no question... but
###########################################################################
is.logical(acs09$fertyr)

## [1] FALSE

is.factor(acs09$fertyr)

## [1] TRUE
```

```r
table(acs09$fertyr)
```

```
##
##    N/A     No    Yes
##      0 665705  39608
```

```r
## The "N/A"  here is NOT the same as NA..
## create a new "logical" ie TRUE/FALSE variable
acs09$fertyrL<-ifelse(acs09$fertyr=="Yes",TRUE,FALSE)

table(acs09$fertyrL)
```

```
##
##  FALSE   TRUE
## 665705  39608
```

```r
##############################################################################
## IPUMS GOTCHA # 3
## The citizen variable has several irrelevant categories an
## one category that is not very well named:
##############################################################################
table(acs09$citizen)
```

```
##
##                                          N/A
##                                       596433
##               Born abroad of American parents
##                                         7146
##                             Naturalized citizen
##                                        41749
##                                  Not a citizen
##                                        59985
##  Not a citizen, but has received first papers
##                                            0
## Foreign born, citizenship status not reported
##                                            0
```

```r
## same trick as above to get rid of irrelevant lavels
acs09$citizenF<-factor(as.character(acs09$citizen))
## change "N/A" to what it really means
## levels(acs09£citizenF) is just a vector of strings
levels(acs09$citizenF)
```

```
## [1] "Born abroad of American parents" "N/A"
## [3] "Naturalized citizen"             "Not a citizen"
```

```
## which we can change
levels(acs09$citizenF)[2]<-"US Born"

table(acs09$citizenF)

##
## Born abroad of American parents                        US Born
##                            7146                         596433
##             Naturalized citizen                  Not a citizen
##                           41749                          59985
```

So now we have the data in usable shape, how shall we calculate the ASFRs?

It is useful to separate this problem into two parts: first that of calculating the total number of events (births) to each woman of each age and category; and second that of calculating the "risk set" or total person years at risk (of child birth) for women of each relevant age and category. These two quantities (matrices) can be thought of as a *numerator* and a *denominator*.

If we are clever enough to generate a numerator and denominator which happen to have the same structure – matrices or 13 dimensional arrays or whatever, then one could simply write:

```
> ASFR <- numerator / denominator
```

It turns out that `tapply()` makes this is pretty easy to do – we'll review how `tapply()` works in a minute, but it will be instructive to first do this job with loops:

4.1 Most obvious loopy solution

The most straight forward way of getting a numerator using loops would be to create a matrix of zeros to hold the result and then loop through *each row of the acs09 data.frame* and increment the appropriate cell of our result matrix. Here is what that approach might look like:

```
## The numerator should be a matrix of counts of births that occurred to   women
## by age .. and how about citizenship status


## (1) create an empty array to hold the results
## note that since acs09 includes only women 15-50, we
## only NEED an array with 35 rows -- however it is
## convenient if row 43 of our array refers to people who are
## 43 years old soo consequenlty we're going to create an array
## with some extra unusedr rows
range(acs09$ageN)
```

```
max(acs09$ageN) > length(unique(acs09$ageN))  ## ?
numerL0<-array(0,c(max(acs09$ageN),
                   length(levels(acs09$citizenF))))
## naming the columns is useful when they refer to categorical values
colnames(numerL0)<-levels(acs09$citizenF)
## Loop through and visit each row/observation in acs09 (boring)
for(i in 1:nrow(acs09)){
  ## to save on typing grab the current row of acs09
  ego<-acs09[i,]
  ## increment the cell of the numerL0 matrix that coresponds to
  ## ego's age and citizenship status
  numerL0[ego$ageN,as.character(ego$citizenF)]<-
    numerL0[ego$ageN,as.character(ego$citizenF)] + ego$perwt *ego$fertyrL
}
```

Unfortunately, this loop takes a rather long time to run. R is just not all that fast with loops.

4.2 Better but less obvious looping strategy

A much better loop-based approach (from a performance in R point of view) would be to loop through the cells of the **result matrix** and select and count relevant people in the corresponding age and citizenship category. Here's how that might look:

```
numerL1<-array(0,c(max(acs09$ageN),
                   length(levels(acs09$citizenF))))

colnames(numerL1)<-levels(acs09$citizenF)
## nested set of loops age visits each row
for(age in unique(acs09$ageN)){
  ## citz visits each citizenship category within each age
  for(citz in levels(acs09$citizenF)){

    ## create a vector for selecting those women who are of the
    ## current age and citizenship category
    sel<-acs09$ageN == age & acs09$citizenF==citz

    ## count the selected women and store the result
    numerL1[age,citz]<-sum(acs09$perwt[sel]*acs09$fertyrL[sel])

  }
}
```

This works a lot faster because in R, its the number of iterations that hurt performance. R can do square-bracket selections and arithmetic on parts of ar-

rays very quickly. But in order to let you cancel a loop after it starts, R is careful to store all kinds of intermediate results–and that takes a lot of work/time.

4.3 The tapply() function (review)

The `tapply()` function is very much like the second kind of loop, but `tapply()` saves you from the boring task of setting up the results array full of zeros, and is generally less error prone, and always faster.

The first argument of to `tapply()` is a vector (column of a data.frame) which will be broken into subsets each of which to be acted upon separately. In the example below, this is `acs09$perwt* acs09$fertyrL` , the each person-weight multiplied the TRUE/FALSE vector "fertyrL" which is coerced into 1 if the women had a baby in the last year and zero otherwise.

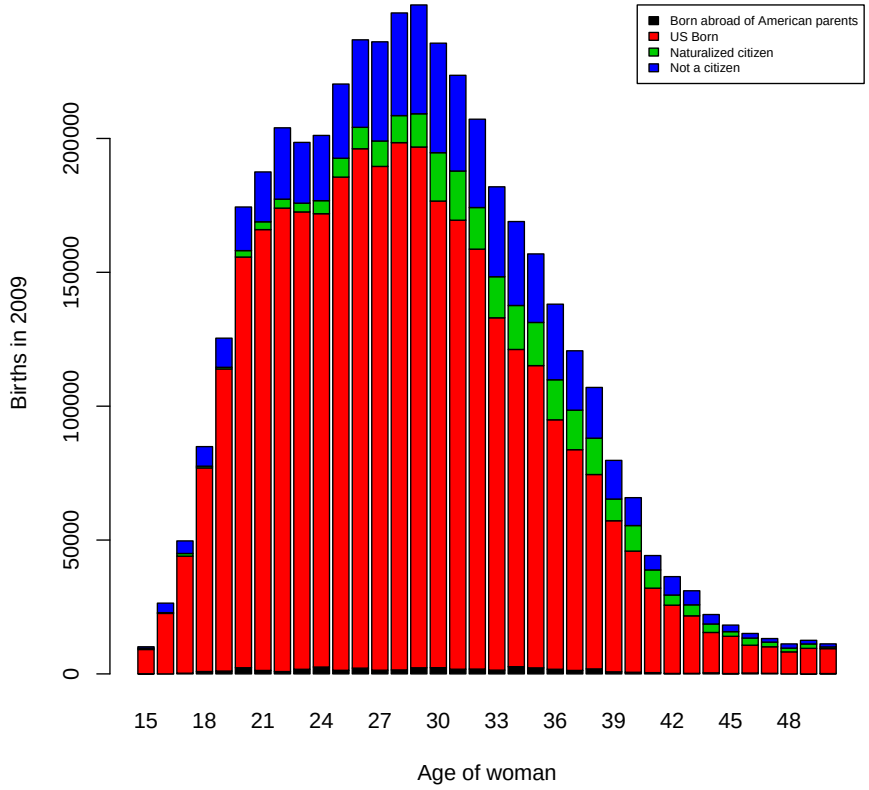
The second argument determines how the subsets are to be created. In the example below, each unique combination of acs09$ageN and acs09$citizenF will form a subset.

The third argument tells `tapply()` what to do to each of the subsets of the first argument. In the example below, we just take the sum. To pass arguments to `sum()` (or whatever function specified in the third argument to `tapply()`) just tack them on separated by commas. In the example below, `na.rm` is an argument to `sum()`.

`tapply()` returns a matrix (with row and column names) and one element for each subset as determined by the second argument.

```
numer<-tapply(acs09$perwt*acs09$fertyrL,
              list(acs09$ageN,acs09$citizenF),
              sum,na.rm=TRUE)

dim(numer)

## [1] 36  4

head(numer)

##    Born abroad of American parents US Born Naturalized citizen
## 15                              68    9056                 426
## 16                               0   22597                 283
## 17                             257   43721                1000
## 18                             889   76056                 652
## 19                            1060  112845                 621
## 20                            2296  153410                2403
##    Not a citizen
## 15           587
## 16          3552
## 17          4698
## 18          7304
```

```
## 19          10884
## 20          16299

## A barplot would be nice right now
barplot(t(numer),xlab="Age of woman",ylab="Births in 2009",
        col=(numercol<-1:length(levels(acs09$citizenF))))
legend(x="topright",fill=numercol,legend=levels(acs09$citizenF),
       cex=.6)
```




```
#tail(numer)
#tail(numerL1)
```

It is very convenient to use `tapply()` for the denominator too. Especially because the second argument is the same. That ensures that the numerator and denominator will have the same dimensionality.

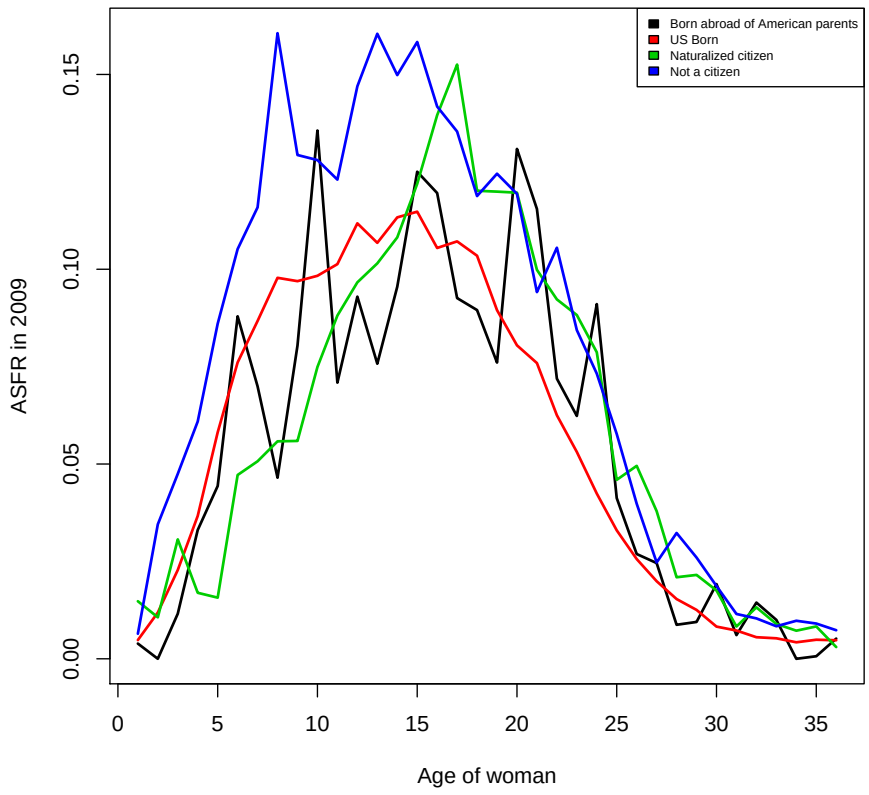
```r
## the denominator is easier -- All we want is a weighted count of the
  ## number of women of each age who were "at risk" of giving
  ## birth. We are neglecting mortality -- because dead women do not
  ## fill out census forms so all we need to do is count

denom<-tapply(acs09$perwt,
              list(acs09$ageN,acs09$citizenF),
              sum,na.rm=TRUE)
dim(denom)
```

```
## [1] 36  4
```

```r
dim(numer)
```

```
## [1] 36  4
```

```r
ASFR<-numer/denom

matplot(ASFR,type='l',lty=1,lwd=2,
xlab="Age of woman",ylab="ASFR in 2009",
        col=(numercol<-1:length(levels(acs09$citizenF))))
legend(x="topright",fill=numercol,legend=levels(acs09$citizenF),
       cex=.6)
```




```
## and apply (NOT t-apply()) takes column sums
TFR<-apply(ASFR,2,sum); TFR

## Born abroad of American parents                    US Born
##                        1.989181                   2.004676
##            Naturalized citizen                 Not a citizen
##                        2.142767                   2.865514
```

5 Assignment

So now you know how to calculate ASFRs, your job is to find, in IPUMS, another *even more interesting rate* (or proportion or probability) to calculate. And (next week) to plot the result in an informative and attractive manner. **On Monday please be ready to share your excellent work with your**

excellent colleagues: A five minute story and a table will be quite sufficient.

The easiest way to go would be to look for variables that ask "did respondent do X in the last year". There are several of those. Ambitious students will want to consider other options as well. Since the IPUMS provides ACS data for each year between 2001 and 2012, you can – use calendar time in place of age – or in addition to age. (the rate school attendance by 17 year olds each year for example). If you choose to pursue something longitudinal, consider ways to keep your data set reasonably small **before downloading it from IPUMS**:

- Take only the variables that you need.
- Exclude (during the IPUMS extraction process) observations that you do not need (men for example if you were looking at fertility)
- Limit your analysis to one state or region – California is interesting enough.
- The IPUMS website allows you to take a random sample of your sample – look for the Customize Sample Sizes

5.1 Getting data from IPUMS to R

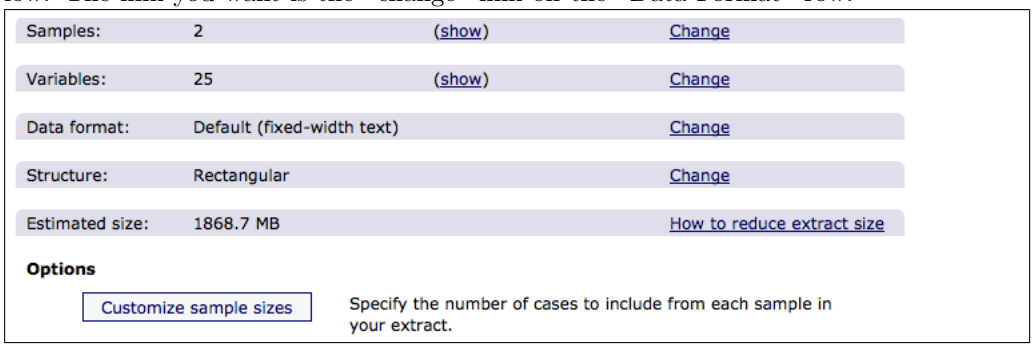
So now you're ready to do science on your own IPUMS data extract. This section is about how to get that extract into R – without polluting your home directory. The gist is that you must make sure that your stata .dta file winds up in a directory such as /90days/username. It's good for everyone if you get used to using the /90days directory for storing things files that are both large *and* easily reproduced. IPUMS makes it **very** easy for you to download another copy of you dataset–so easy in fact that you will find yourself downloading it again rather than searching your disorganized directories to find your old copy. It's also good for everyone else in the universe, because making backup copies of stuff takes resources and /90days isn't backed up. Thanks for being a good citizen.

The IPUMS website is pretty straightforward – if I wrote instructions on how to use it, you wouldn't read them. There are, however, two non-obvious things about it:

1. Size matters. A single year of ACS data has about 3 million records. This is manageable, but it is like hiking through mud. If you include a variable like `statefip` then you will be able to easily reduce the scope of your project to a single US state. Consider also removing records by age and sex if it makes sense for your project. You can "select cases" in IPUMS, or you can do the data reduction step in R.

2. You have to look for the check box that formats your dataset as a .dta file. In order to find that check box you first have to find and click on one of the "change" links on the screen that comes up right after

you hit the "create extract button". A part of that screen is shown below. The link you want is the "change" link on the "Data Format" row.



So skipping detailed instructions on the IPUMS website...I'll assume that you have managed on your own to:

- Visit `ipums.org/usa` (and acquire a userid/password)
- Visit `ipums.org/usa` again to select variables and samples.
- Receive email from IPUMS telling you that your extract is ready to download.

Once you have done all of this then read on.

5.1.1 Download your data directly to /90days/¡your-useridIPUMS¿

The data file that you are about to download is probably on the order 300MB. That's a moderately large file you could get away with putting a few of those in your home directory, but sooner or later your conscience will catch up with you and make you feel bad. The smarter, more virtuous place to put it is in `/90days/<useridIPUMS>` where `<userid>` is your userid – or really any word that no one else is likely to choose. If you are up on your "12 most important Unix commands" you'll remember that you can use ***ln -s*** to create a symbolic link so that your /90days/¡useridIPUMS¿ directory acts like a subdirectory of you WeekN project directory. What I'd do (in a shell terminal window) is:

```
@:> mkdir /90days/carlmIPUMS
```

```
@:> cd ~/213/WeekN
```

```
@:> ln -s /90days/carlm IPUMS
```

THEN browse to where the email from IPUMS tells you to go by opening the link that you were sent by IPUMS with the announcement that your data are ready and download the STATA formatted data file. This will be the left most link marked "STATA" under the "Formatted Data" column. **Do not**

bother to download the fixed width text data file. Use the "save link as" feature of your modern browser to make sure that the compressed stata .dta.gz file winds up in your /90days/¡useridIPUMS¿ folder.

Its not a bad idea to keep your browser open to that download page as the code book file – and the rest of the IPUMS site will be quite handy in figuring out what you have.

5.2 Getting your IPUMS data into R

Once your dataset is downloaded and resting comfortably in your /90days/useridIPUMS folder you may turn your attention to loading the data into R. This will involve a couple of UNIX commands, but it's easiest for both you and your biographer, if you do those from within R(studio) using the `system()` function.

Presumeablye, you have already browsed to Rstudio and created a new project in your pe-existing WeekN direcotry but if no, now would be an excellent time to do so.

5.2.1 Uncompressing your data file

The STATA formatted data file that you just downloaded is zipped (or compressed). The foreign library in R does not read compressed files so how about if we uncompress it.

In the following two commands, I create a new and uniquely named directory in /90days and then I unzip a compressed IPUMS dta file from my /data/commons folder into the new directory in /90days that I just created **and** I give the resultant uncompressed (ready to read) stata .dta file a new and exciting name.

You will want to copy these commands into your R source file and then change them to suit your circumstances.

```
## By now you should have created a directory in /90days to hold your data
## AND you should have downloaded into it a compressed .dta.gz file if you
## haven't done that already, go back a few pages and make that happen.

## I also assume here that you have created a symbolic link as suggested
## if not,  this will not work
## uncompress your data into a file in the new directory
system("gunzip  ./IPUMS/usa_00086.dta.gz")
## Note your file will not be called usa_00086.dta.gz, but something similar.
system("ls ./IPUMS")
## should show that you now have
```

It will take a few moments for this command to execute because uncompressing is just slow. While it is spinning, consider the coolness of this procedure:

- It is not unlikely that, for reasons that you could not possibly have foreseen, you will need revise and re-download your IPUMS data set. When

you do that it will have a slightly different name. All you will need to do in R(studio) to get at that yummy new data will be to change probably one character in the above (gunzip) command, and then run it.

- Although the file that you are working with in this exercise may not be huge, someday you will find yourself downloading entire full enumeration censues of very large countries for multiple years. Those will be huge and you will surely want to feel the inner glow of virtue that derives from not cluttering your home directory with huge but easily reproduced files.
- Note that the system() command, which you have seen before, is very useful for running UNIX commands from within R.

5.2.2 Reading the data into R

At this point you should be able to read the data into R as I do in the commands bellow.

```
## add the library that knows how to read .dta files
library(foreign)
## without any arguments other than the file this can work. You might
## wish to consult the help screen to see what options exist.


## change carlmWeek6/happyScience.dta  to something smarter
## if you want this to work:
acs<- read.dta(file="./IPUMS/happyScience.dta")

names(acs) ## do the column names look familiar ?

##  [1] "year"      "datanum"   "serial"    "hhwt"      "region"
##  [6] "statefip"  "gq"        "pernum"    "perwt"     "famsize"
## [11] "nchild"    "nchlt5"    "age"       "sex"       "marst"
## [16] "marrno"    "marrinyr"  "widinyr"   "divinyr"   "fertyr"
## [21] "race"      "raced"     "bpl"       "bpld"      "citizen"
## [26] "yrnatur"   "yrimmig"   "hispan"    "hispand"   "hcovany"
## [31] "hcovpriv"  "hinsemp"   "hinspur"   "occ"       "migrate1"
## [36] "migrate1d" "diffmob"

sum(acs$perwt) ##how many people do the observations represent? sum

## [1] 75974672

               ##the person weight to find out
```

If you are worried about your data disappearing from /90days before you are able to complete this exercise, don't be. First "90days" really means "180 days" around here. And further, that means 180 days without being touched. You

can extend the time that files stay in "/90days" by simply "touching" them. `touch` is the UNIX command that does that.

Second, and even more to the point, who cares if your file disappears from /90days? Your R code is save in your home directory and IPUMS, as far as I can tell, never deletes your data request. To be fair, once you have some publishable results, you will want to keep your data and your code all together somewhere so you can easily (read:possibly) reproduce your work – but that is later. With IPUMS and data sources like it, you will inevitably download several versions of your data before you get the one you want to keep.

The above should also serve as a suggestion that when you return to this project after a brief hiatus, you do not need to run the gunzip command again. Your .dta file should still be waiting in /90days¡useridIPUMS¿ aka 213/WeekN/IPUMS.

5.3 Do Science

So now you have created an IPUMS extract; downloaded it; and read it into R. All that's left is to calculate an interesting measure using the `tapply()` based procedure that you just learned.

For Monday please have a table of results and a short story about what you have found. A graph would be fine – but we'll spend time graphing next week.