R Markdown Data Science Final Project

Tina Vo

3/14/2021

This is an R Markdown report for my data science final project. This will provide information on the SATSA\_improved script that is uploaded into my github repository.

The materials for this R Markdown are found in [click here](https://github.com/tvo015/Rrepos). Or URL: <https://github.com/tvo015/Rrepos>

Contents in Rrepos:

* DataScience SATSA script.sas (This is my original workflow in SAS format. This is what will be improved upon through R. I have pasted chunks of my previous SAS code into this R markdown and the subsequent improved R code associated with that SAS chunk)
* R for SATSA\_improved.R (This is the R script associated with this R markdown report and where I made changes to improve efficiency, fidelity, and sharing/reproducibility)
* R Markdown Data Science.Rmd (This is the current R markdown script)
* R-Markdown-Data-Science.html (This is the resulting R Markdown output report)
* SATSA\_datascience.csv (This is the fake raw dataset)

Overview:

* This report will detail a previous workflow and steps done to improve that workflow using R to improve efficiency, fidelity, and reproducibility of my project. Originally, presented in the workflow critique assignment, I mentioned working with nine different datasets but upon more assessment, I realized that each dataset had similar issues that would greatly benefit from the same techniques (eg. automation, iterations, same naming conventions). Therefore, I have decided to improve my workflow in one dataset with the assumption that the same techniques will be applied to the remaining eight datasets (outside of this course assignment). As stated in my workflow self-critique assignment, I believe that my weakest link when it comes to data workflow is efficiency. Throughout this document, I will provide an explanation of how efficiency has increased as well as increasing the fidelity of my data.

### STEP 1: Importing in the dataset

This dataset is for my Swedish Adoption Twin Study of Aging (SATSA) study. Note: I have created a completely fake dataset for the purposes of this final project. The dataset is currently in a csv format. I first start by importing in my dataset to R and viewing the dataset.

#Load packages  
library(tidyverse)  
library(tidyverse)  
library(vroom)  
library(visdat)  
library(here)  
library(visdat)  
library(janitor)  
library(readr)  
  
SATSA\_datascience <- read\_csv("~/GitHub/Rrepos/SATSA\_datascience.csv")  
View(SATSA\_datascience)

### STEP 2: Attempt at understanding/visualizing the data

This next step uses vis\_dat to view this data set. We learned in class that visualization often times will yield additional information that we wouldn’t see if we just did not check or just assumed that the data is how we expect it to be.

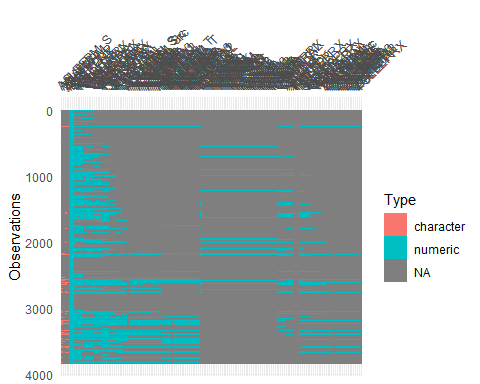
This code is what was previously done to attempt at visualizing the data in SAS:

\*\*PROC CONTENTS DATA=SATSAs VARNUM; RUN;

The issue with the above code is that it only prints out the variable names within the file. While that is good, it does not provide any insight on the structure of the data or any information regarding the actual data contained in the dataset.

This code is what was done in R to improve on the previous SAS code:

vis\_dat(SATSA\_datascience)



While this is quite hard to understand, it does provide some information on the data (e.g. which variables are string, which are numeric, how many observations are in each variable).

### STEP 3: Selecting only the variables of interest

This step is crucial as the datafiles that I often work with have a very large number of variables. Since the data for SATSA was not originally collected for the purposes of my sleep research, I have to go through and condense down the dataset to just the variables of interest. This is an improvement on my current workflow because I never even did this. I usually just worked with the very large dataset with over a hundred variables in there. The efficiency of my work was oftentimes compromised because I had to spend so much time scrolling through to find just the variables that I needed.

SATSA<-SATSA\_datascience %>% select(ID, AGE, ASLEEPN\_Src, ASLEEPM\_Src, ASLEEP1:ASLEEP12,  
 ZSLEEPNrc, ZSLEEPMrc, ZSLEEP1:ZSLEEP12)

Now I have a dataset called SATSA which just has the ID variable, age variable, and sleep variables for the first wave (wave A) and the second wave (wave Z). This also increases efficiency since I no longer need to manually go through and delete out variables that I didn’t need later on. Previously, I would have deleted variables in SPSS but now that I am selecting these variables, it is more of an automated process while reducing human error (e.g. I could accidentally miss a variable if I typed out ASLEEP1, ASLEEP2, ASLEEP3 etc. but now that I am doing ASLEEP1:ASLEEP12, I know that all of those variables in between will for sure be selected). Therefore, efficiency is increased but fidelity is increased as well.

### STEP 4: Creating a Function

This step creates a function for calculating sleep duration. How the data is currently set up is that I have an individual’s waketime and their bedtime and would have to calculate total sleep duration from those two variables.

Previously, in my SAS script, I had to convert the variables from string to numeric using the time10 format, rename the variables, and then calculate sleep duration for each individual wave through first converting the time to seconds.

Previous SAS Code:

/*set string to numeric for ASLEEP and ZSLEEP, BSLEEP already numeric*/ title ‘String to Numeric’; data SATSA\_num; set SATSAs; Abedtime= input (ASLEEPN\_Src, time10.); Awaketime= input (ASLEEPM\_Src, time10.); Zbedtime= input (ZSLEEPNrc, time10.); Zwaketime= input (ZSLEEPMrc, time10.); run;

proc contents data= SATSA\_num; run;

title ‘Renaming BSLEEP, convert to seconds’; /*renaming B sleep variables to be consistent with A and Z sleep items*/ data SATSA\_num1; set SATSA\_num; Bbedtime=BSLEEPN\_T; Bwaketime=BSLEEPM\_T; if Bbedtime=0 then Bbedtime=86400; run;

title ‘Calculating sleep duration based on bedtime and waketime’; /*calculate sleep duration based on waketime and bedtime*/ data SATSA\_num2; set SATSA\_num1; A\_hrs\_slept = mod(24+(Awaketime-Abedtime)/3600,24); B\_hrs\_slept = mod(24+(Bwaketime-Bbedtime)/3600,24); Z\_hrs\_slept = mod(24+(Zwaketime-Zbedtime)/3600,24); run;

title ‘frequency check for sleep duration’; proc freq data=SATSA\_num2; tables A\_hrs\_slept B\_hrs\_slept Z\_hrs\_slept; run;

Improved R Code This creates a function that will automatically calculate sleep duration based on the logic provided and return just the duration. When applied to the two wave A variables (bedtime and waketime), it yields SleepDurationA. When applied to the two wave Z variables (bedtime and waketime), it yields SleepDurationZ. This improves efficiency because previously (seen above), I would have had to type out the equation for each individual wave that I wanted to calculate sleep duration for. Typing the equation out each time makes room for human error (e.g. I could mistype 360 instead of 3600 and not notice and SAS would not give me an error or a warning because it would think that I intended to do that). With the new function, I am able to just make sure that I have correctly typed out the equation correctly one time and then can use the function as often as I need with the assurance that it would be calculated correctly.

HRS\_SLEPT <- function(a, b) {  
 duration <- (b-a)+24  
 return(duration)  
   
}  
SATSA$SleepDurationA<-HRS\_SLEPT(SATSA$ASLEEPN\_Src, SATSA$ASLEEPM\_Src)  
SATSA$SleepDurationZ<-HRS\_SLEPT(SATSA$ZSLEEPNrc, SATSA$ZSLEEPMrc)  
View(SATSA)

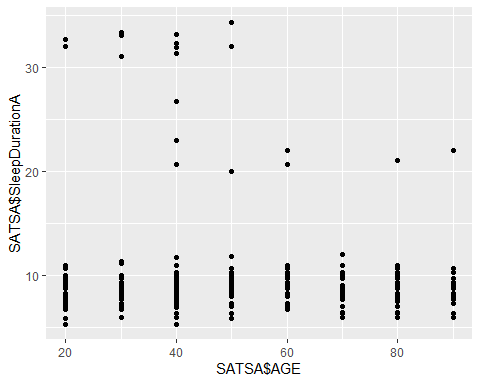
### STEP 5: Visualizing the data/Formally checking the data

While I did check means and sample size (N’s) of my data quiet often with my previous work flow, I never plotted the data to visually see what the data looks like. It is good to see the data through graphs because then you can clearly see any outliers.

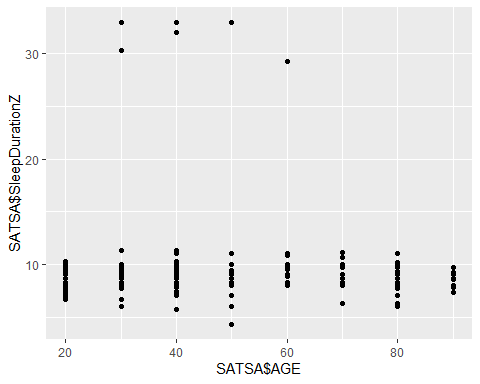
Previous SAS Code title ‘Means’; PROC MEANS DATA=SATSA\_3; VARS; RUN;

Improved R Code to Visualize and formally check the data

SATSA %>% ggplot(aes(x = SATSA$AGE, y = SATSA$SleepDurationA)) + geom\_point()



SATSA %>% ggplot(aes(x = SATSA$AGE, y = SATSA$SleepDurationZ)) + geom\_point()



#checking data  
range(SATSA$SleepDurationA, na.rm=T)

## [1] 5.3 34.3

range(SATSA$SleepDurationZ, na.rm=T)

## [1] 4.3 33.0

With this, I am able to see that there are some outliers in my two waves of sleep duration.Not only can I see them in the graphs, but I also did a formal check through checking their range. Based on this information, I will create a new dataset to set individuals below two hours of sleep and above 15 hours of sleep to NA. With my previous workflow, I would only have caught the outliers if I was super careful in looking at the output of the means.

SATSA\_cleaned <- SATSA %>% mutate(  
 SleepDurationA = ifelse(SleepDurationA >= 2 & SleepDurationA < 15, SleepDurationA, NA),  
 SleepDurationZ = ifelse(SleepDurationZ >= 2 & SleepDurationZ < 15, SleepDurationZ, NA)  
)  
  
range(SATSA\_cleaned$SleepDurationA, na.rm = T)

## [1] 5.3 12.0

range(SATSA\_cleaned$SleepDurationZ, na.rm = T)

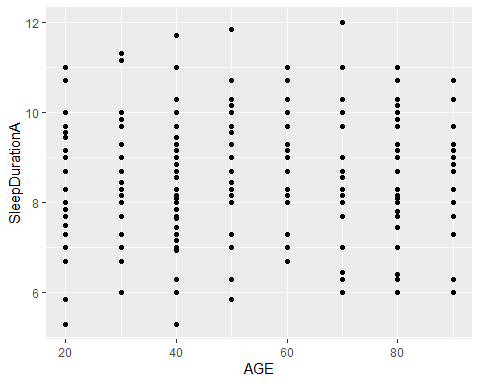
## [1] 4.3 11.3

Now the range for SleepDurationA is 5.3 and 12 and SleepDurationZ is 4.3 and 11.3.

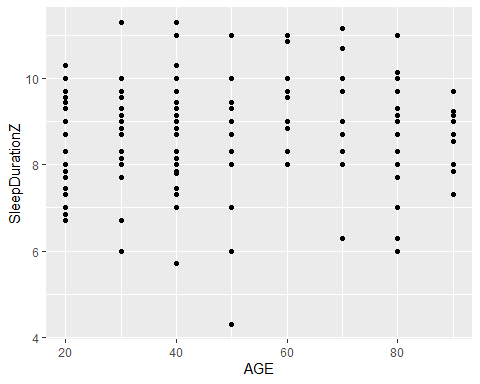
### STEP 6: Re-visualizing the data

Now lets look at the data again.

SATSA\_cleaned %>% ggplot(aes(x = AGE, y = SleepDurationA)) + geom\_point()



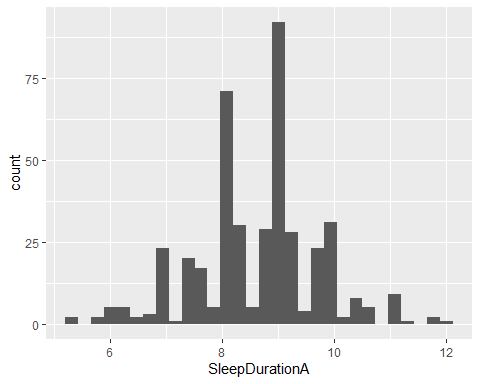
SATSA\_cleaned %>% ggplot(aes(x = AGE, y = SleepDurationZ)) + geom\_point()



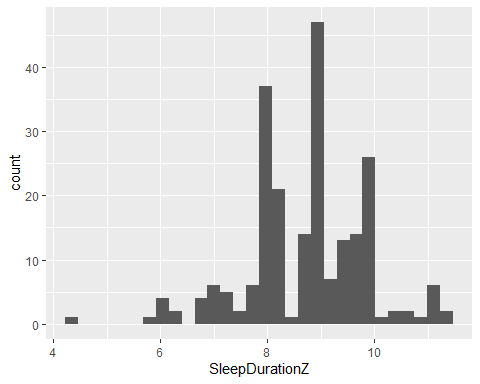
It appears that people are sleeping more hours in the second wave (Z) compared to first wave (A).

Another way to look at the data:

SATSA\_cleaned %>% ggplot(aes(x = SleepDurationA)) + geom\_histogram()



SATSA\_cleaned %>% ggplot(aes(x = SleepDurationZ)) + geom\_histogram()

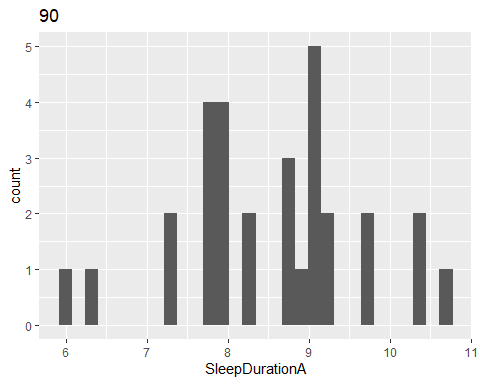
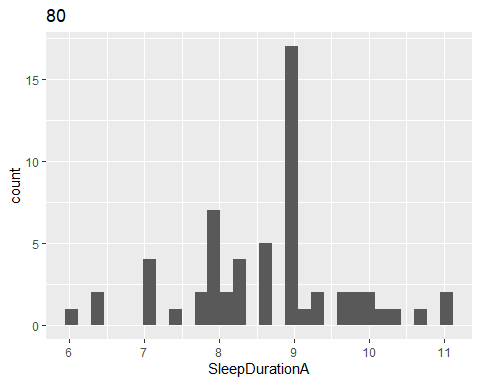
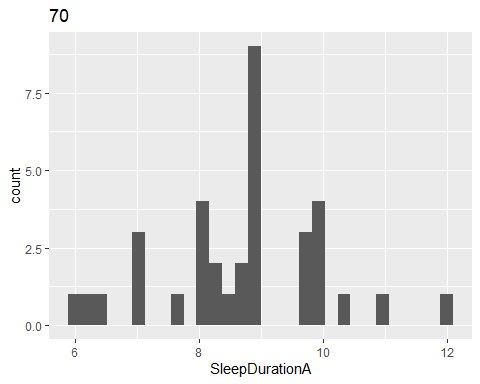
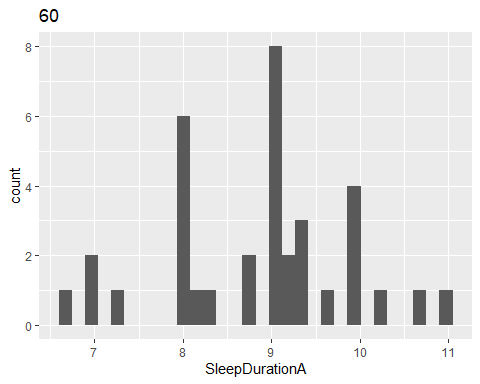
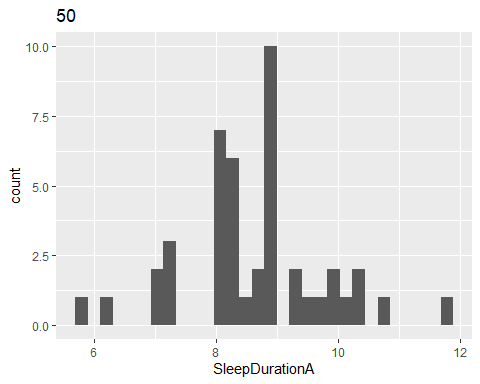
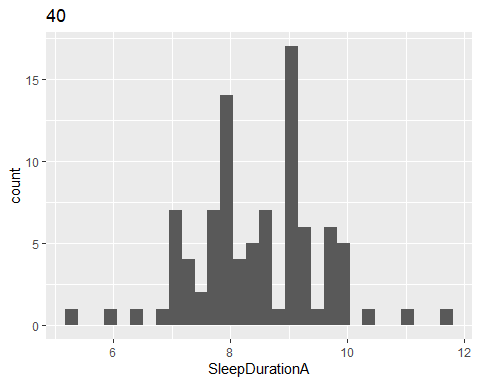
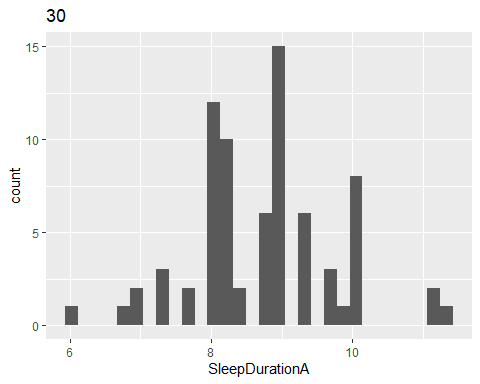
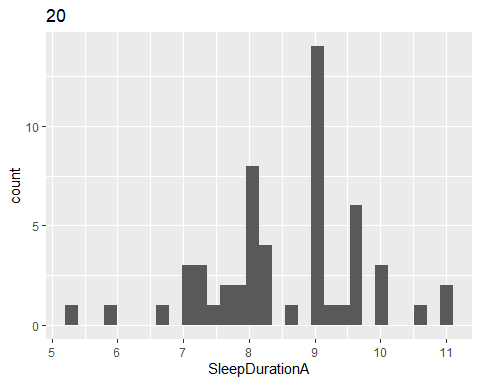


This is an improvement to my previous workflow because previously I had not graphed my data to visualize it. Visualizing it through scatterplots or histograms improves the ability to potentially catch any errors or any outliers. In addition, it also yields additional interpretation and understanding of the data.

### STEP 7: For Loop (graphing to visualize sleep duration by age group)

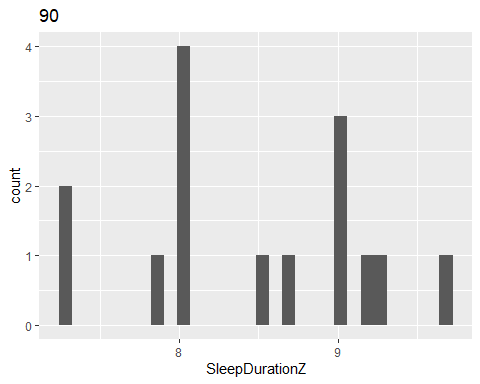
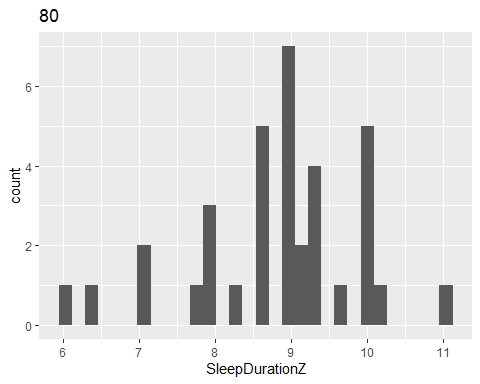
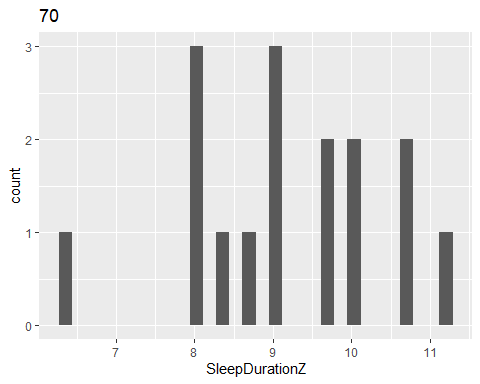
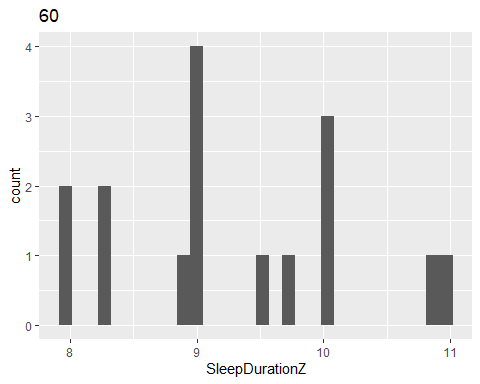
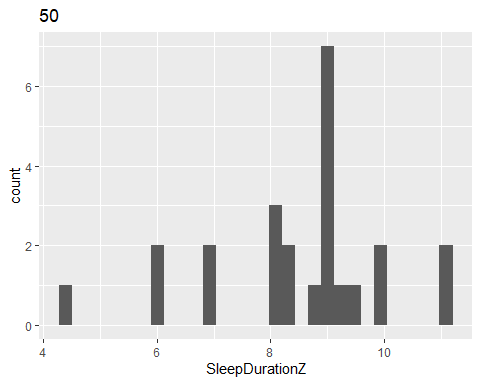
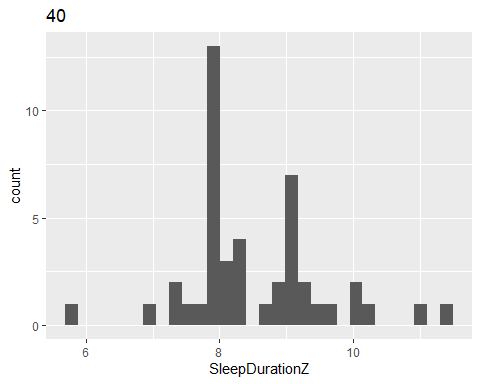
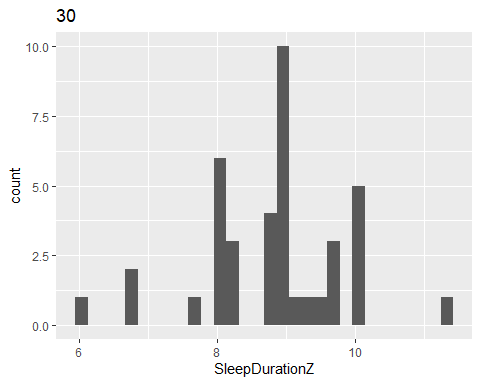
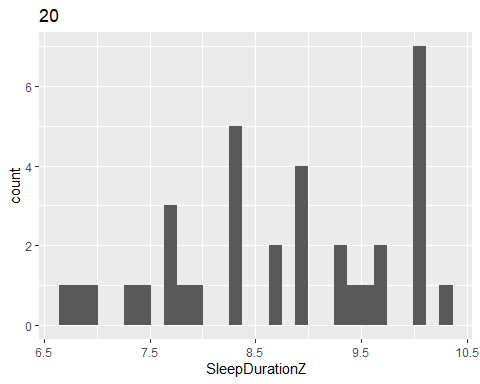
Additionally, what might be important to assess is how sleep duration may look by certain age group bins. This for loop is able to increase efficiency by graphing all age groups rather than me having to graph each age group individually.

graphA<-function(i){  
 p <-ggplot(i, aes(x=SleepDurationA)) + geom\_histogram()+  
 ggtitle(i$AGE)  
 print(p)  
}  
  
  
SATSA\_agegroup <-split(SATSA\_cleaned, as.factor(SATSA\_cleaned$AGE))  
for(x in SATSA\_agegroup) {  
 graphA(x)  
   
}



Repeated for wave Z

graphZ<-function(q){  
 z <-ggplot(q, aes(x=SleepDurationZ)) + geom\_histogram()+  
 ggtitle(q$AGE)  
 print(z)  
}  
  
  
SATSA\_agegroup <-split(SATSA\_cleaned, as.factor(SATSA\_cleaned$AGE))  
for(h in SATSA\_agegroup) {  
 graphZ(h)  
   
}



Using for loops to automate this graphing process greatly improves my previous workflow. Previously, in order to do this, I would have either had to graph each plot for each age group individually through chunks of code (roughly 100’s of lines of code with room for human error in typing in SAS) or graphed using drop down menus in SPSS. Doing so through for loops in R leaves syntax for other collaborators to have a document of what was done thereby increasing potential reproducibility of my code and data to generate the same exact graphs. Additionally, this decreases errors and increases efficiency (I can graph many in just around 10 lines of code as opposed to writing hundreds of lines of code).

### STEP 8: Automation (recoding)

The next step deals with increasing efficiency and fidelity through improving ways of recoding data. The sleep variables originally are on a 1-5 scale but need to be converted to a 0-1 scale for the purposes of analyses (0=no endorsement, 1=some endorsement).

Previously done in SAS

#if ASLEEP3=1 then ASLEEP3\_h=0; #if ASLEEP3 ge 2 then ASLEEP3\_h=1; #if ASLEEP1=1 then ASLEEP1\_h=0; #if ASLEEP1 ge 2 then ASLEEP1\_h=1; #if ASLEEP4=1 then ASLEEP4\_h=0; #if ASLEEP4 ge 2 then ASLEEP4\_h=1; #if ASLEEP8=1 then ASLEEP8\_h=0; #if ASLEEP8 ge 2 then ASLEEP8\_h=1; #if ASLEEP10=1 then ASLEEP10\_h=0; #if ASLEEP10 ge 2 then ASLEEP10\_h=1; #if ZSLEEP1=1 then ZSLEEP1\_h=0; #if ZSLEEP1 ge 2 then ZSLEEP1\_h=1; #if ZSLEEP3=1 then ZSLEEP3\_h=0; #if ZSLEEP3 ge 2 then ZSLEEP3\_h=1; #if ZSLEEP4=1 then ZSLEEP4\_h=0; #if ZSLEEP4 ge 2 then ZSLEEP4\_h=1; #if ZSLEEP8=1 then ZSLEEP8\_h=0; #if ZSLEEP8 ge 2 then ZSLEEP8\_h=1; #if ZSLEEP10=1 then ZSLEEP10\_h=0; #if ZSLEEP10 ge 2 then ZSLEEP10\_h=1;

Note: The example shown for SAS above has already been improved. Previously, it was even more clunky (ex: if ASLEEP 3=1 then ASLEEP3\_h=0, if ASLEEP3=2 then ASLEEP3\_h=1, if ASLEEP3=3 then ASLEEP3\_h=1, if ASLEEP3\_h=4 then ASLEEP3\_h=1, if ASLEEP3\_h=5 then ASLEEP3\_h=1). Previously, I would take roughly five lines to recode just one variable. Learning the “ge” to mean greater than meant that I could reduce the five lines into two lines now (seen in SAS example code above).

I could use automation to further reduce the lines of code even further (shown in R)

Improved R Code

library(dplyr)  
range(SATSA$ASLEEP1, na.rm=T)

## [1] 1 5

a1<-c(5:16, 19:30)  
as.character(a1)

## [1] "5" "6" "7" "8" "9" "10" "11" "12" "13" "14" "15" "16" "19" "20" "21"  
## [16] "22" "23" "24" "25" "26" "27" "28" "29" "30"

#automation of recoding rather than if then statements  
SATSA\_RC<-SATSA %>%  
 mutate\_at(5:16, recode, '2'='1', '3'='1', '4'='1','5'='1', '1'='0') %>%  
 mutate\_at(19:30, recode, '2'='1', '3'='1', '4'='1','5'='1', '1'='0')  
range(SATSA\_RC$ASLEEP1, na.rm=T)

## [1] "0" "1"

With using mutate\_at, I am able to reduce my recoding code down to just three lines. This greatly increases effiency through saving me time from typing out each if-then statement while also ensuring that fidelity of the data is not compromised since it does not leave much room for human error since I do not have to individually type out each recoding. Additionally, a formal check implemented through looking at the range of the recoded variable ensures that the minimum is 0 and the maximum is 1 as intended with the recoding.

### STEP 9: Addressing naming conventions

As mentioned previously, I need to merge multiple datasets together later on. One issue that I have run into is that my naming conventions are not comparable, making merges quite difficult. My previous workflow included renaming variables by hand at the end of all data recodings. This gets quite tedious and I could accidentally misspell something and end up renaming a variable incorrectly which would affect my merging of datasets later on.

Previously done in SAS

/*create datafile with the naming conventions*/ data SATSA\_sleep\_names; set SATSA\_sleepcat3; slp\_dur\_fu16=A\_hrs\_slept; slp\_dur\_fu18=B\_hrs\_slept; slp\_dur\_fu17=Z\_hrs\_slept; slpcat\_fu16=slp\_ave\_cat1; slpcat\_fu18=slp\_ave\_cat2; slpcat\_fu17=slp\_ave\_cat3; latency\_fu16=ASLEEP5\_h; latency\_fu18=BSLEEP5\_h; latency\_fu17=ZSLEEP5\_h; RClatency\_fu16=ASLEEP5\_RC; RClatency\_fu18=BSLEEP5\_RC; RClatency\_fu17=ZSLEEP5\_RC; latency\_in=E60; slpRX\_in=SLEEPRX; slpRX\_fu1=IPT1SLEEPRX; slpRX\_fu2=VSLEEPRX; slpRX\_fu3=QSLEEPRX; slpRX\_fu4=GSLEEPRX; slpRX\_fu5=XSLEEPRX; slpRX\_fu6=HSLEEPRX; slpRX\_fu7=FSLEEPRX; slpRX\_fu8=RSLEEPRX; slpRX\_fu9=SSLEEPRX; slpRX\_fu10=JSLEEPRX; slpRX\_fu11=USLEEPRX; slpRX\_fu13=WSLEEPRX; wake\_earlyfu16=ASLEEP1\_h; wake\_earlyfu18=BSLEEP1\_h; wake\_earlyfu17=ZSLEEP1\_h; RCwake\_earlyfu16=ASLEEP1\_RC; RCwake\_earlyfu18=BSLEEP1\_RC; RCwake\_earlyfu17=ZSLEEP1\_RC;

wake\_nightfu16=ASLEEP4\_h; wake\_nightfu18=BSLEEP4\_h; wake\_nightfu17=ZSLEEP4\_h; RCwake\_nightfu16=ASLEEP4\_RC; RCwake\_nightfu18=BSLEEP4\_RC; RCwake\_nightfu17=ZSLEEP4\_RC;

snorefu16=ASLEEP10\_h; snorefu18=BSLEEP10\_h; snorefu17=ZSLEEP10\_h; RCsnorefu16=ASLEEP10\_RC; RCsnorefu18=BSLEEP10\_RC; RCsnorefu17=ZSLEEP10\_RC;

restless\_in=M6\_h; restless\_fu2=VM6\_h; restless\_fu3=QM6\_h; restless\_fu4=GM6\_h; zygos=PRZYGUP; study=1; run;

Note: as you can see, it requires a lot of lines of code to rename everything and even then, the naming conventions or the naming styles aren’t all consistent (some are snake case, some are upper and lowercase, some have underscores while other variables do not)

Improved R Code

SATSA\_clean\_names<-SATSA\_RC%>%  
 janitor::clean\_names()

Simply just using the clean names in janitor will automatically apply the same naming convention to all variables within my dataset without having the need to type out each individual variable. This greatly improves efficiency. Additionally, fidelity is improved because there is no room for error. Lastly, reproducibility and sharing is also impacted positively because now individuals can easily merge my datasets together since naming conventions will all be the same. I can also easily share my data because the variable names will be understandable to other people.

### Ending Comments

Overall, through this class I was able to learn ways to improve my efficiency, fidelity, and sharing/reproducibility of my data. Efficiency was improved through the reduction of repetitive code (if-then statements) by using mutate\_at. A custom function to calculate sleep duration and also a custom function to generate graphs by age group (through a for loop) greatly improved efficiency and fidelity of the data by allowing visualization from plotted graphs to catch any errors (data checks) but also to further understand the data. Additionally, the implementation of checking the range of the data allowed for more frequent data checks as well. Improved readibility of the code was done through adapting a consistent naming convention which will also allow for easy merging of datasets later on. Lastly, improved documentation of the project is shown through this R markdown report. Previous documentation of data steps was just done directly in the SAS script through comments but implementing a dynamic document like R markdown improves understanding of the research project and also sharing/reproducibility of the project since each code chunk was well-documented. Documentation of the project was also improved upon through pushing and committing changes to github.