Programming Assignment 2 Solutions

SOC Methods Camp

September 4th and 5th, 2019

Step zero: load the data (finalaedf.csv)

Prompted by Andrew Wakefield's controversial, and since-retracted, article linking the MMR vaccine to autism, recent years have seen protests over childhood vaccination out of concerns that the vaccines will cause autism and other neurological problems.

The data we'll be using relates to these issues. It is from the CDC's *Wonder* database and is comprised of reports that parents and/or physicians submit to the CDC reporting that a vaccine caused symptoms of autism and/or Asperger's. For today's activity, load the cleaned version of the data: *finalaedf.csv* and store it as *autismae*.

```
#load data
autismae <- read.csv("finalaedf.csv")

library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------------------------------

## v ggplot2 3.2.0       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0

## -- Conflicts ---------------------------------------------------------------------------------------- t
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Step one: progress from a for loop with if/elseif/else statements to a function

You're interested in treating age as a continuous variable, but notice that the data currently codes the variable as a factor variable with different levels. You want to create a new variable (*agenumeric*) that codes the age categories with the midpoint of the corresponding age range.

We'll do variations of this task with a for loop and then with a function.

Task one: use a for loop and control flow statements (e.g., if/elseif/else) to explicitly outline each age category and manually enter its midpoint. So for instance, one of the conditionals, would be

if(age == “1-2 years”){
agenumeric <- 1.5
}

Store the results in a vector *agenumeric*. Code “Unknown” as NA and code months using the appropriate fractions.

```
#get a list of all the conditions by creating a table of that variable
#this will also help us make sure we recoded correctly by comparing
```

```
#counts on the original versus recoded variable
table(autismae$age)
```

```
## 
##  < 6 months   1-2 years 18-29 years   3-5 years 30-39 years 40-49 years 
##         206         717           9         136           3           3 
## 6-11 months  6-17 years 60-64 years     Unknown 
##         130          84           1         349
```

```
#initialize the empty vector
agenumeric <- c()

#
for(i in 1:nrow(autismae)){
  if(autismae$age[i] == "< 6 months"){
    agenumeric[i] <- 3/12
  } else if (autismae$age[i] == "1-2 years"){
    agenumeric[i] <- 1.5
  } else if (autismae$age[i] == "18-29 years"){
    agenumeric[i] <- 23.5
  } else if (autismae$age[i] == "3-5 years"){
    agenumeric[i] <- 4
  } else if (autismae$age[i] == "30-39 years"){
    agenumeric[i] <- 34.5
  } else if (autismae$age[i] == "40-49 years"){
    agenumeric[i] <- 44.5
  } else if (autismae$age[i] == "6-11 months"){
    agenumeric[i] <- 8.5/12
  } else if (autismae$age[i] == "6-17 years"){
    agenumeric[i] <- 11.5
  } else if (autismae$age[i] == "60-64 years"){
    agenumeric[i] <- 62
  } else {
    agenumeric[i] <- NA
  }
}

#compare tables - looks ok
table(autismae$age)
```

```
## 
##  < 6 months   1-2 years 18-29 years   3-5 years 30-39 years 40-49 years 
##         206         717           9         136           3           3 
## 6-11 months  6-17 years 60-64 years     Unknown 
##         130          84           1         349
```

```
table(agenumeric)
```

```
## agenumeric
##              0.25 0.708333333333333               1.5                 4 
##               206               130               717               136 
##              11.5              23.5              34.5              44.5 
##                84                 9                 3                 3 
##                62 
##                 1
```

```r
##attach to data and manually inspect
autismae$agenumeric <- agenumeric
head(autismae)
```

```
##         id      state        age gender year
## 1 133425-1   Missouri  1-2 years   Male 2000
## 2 133529-1    Arizona  1-2 years   Male 2000
## 3 133597-1    Indiana  1-2 years Female 1999
## 4 133907-1    Alabama  1-2 years   Male 2000
## 5 134256-1 California 6-17 years   Male 2000
## 6 134628-1   Arkansas  1-2 years Female 2000
##                                                            labdata
## 1                                       No lab data for this event
## 2                                       No lab data for this event
## 3                                       No lab data for this event
## 4 He was dx'd by a Neurologist as high functioning Autistic child
## 5                                       No lab data for this event
## 6                                                       CT/MRI-nml
##   agenumeric
## 1        1.5
## 2        1.5
## 3        1.5
## 4        1.5
## 5       11.5
## 6        1.5
```

You probably realized (or hoped!) when writing that loop that it was not the most efficient way to code that factor variable to the midpoint of the age categories. In particular, there are two issues:

- Length: it took awhile to write that, and imagine if the categories had been more fine grained!
- Potential for human error: regardless of your mental arithmetic skills or even if you confirmed each midpoint in R's console before inputting, it's inviting a mistake to happen to manually enter what the midpoint is rather than using a more reliable source (R's built-in calculator)

The latter issue is particularly problematic, so we are interested in writing a function that finds the midpoint of the age category variable for every category, which automates this process a bit.

The actual "meat" part of this function involves some advanced programming and manipulation with regular expressions, which is something you do NOT need to worry about at this point. Instead, we want you to just practice generalizing a series of operations/commands into a function that can be applied more widely.

Task two: First, to simplify this a bit, we want to focus only on reports about patients who are of known age and are at least one year old – so filter out all of people that are of "Unknown" age or are less than 1 year old.

Note: even after you remove these observations, the data structure still remembers those two age categories,or "levels". Those levels are just empty now. Drop the empty levels with the "droplevels()" command, which you can pipe after your filter step.

```r
#filter out Unknown and <6mo
autismae2 <- autismae %>%
  filter(age != "Unknown" & age != "< 6 months" & age!= "6-11 months") %>%
  droplevels()

#alternative way
autismae2 <- filter(autismae, age != "Unknown" & age != "< 6 months" & age != "6-11 months")
```

```
autismae2$age <- droplevels(autismae2$age)
```

Task three Below is some code that takes in an age-category ("1-2 years"), removes the word "year", splits out the dash between the two numbers, converts those two numbers from character into numeric data, and then takes the mean of those two numbers (the mean of two numbers is their midpoint). First, we provide these series of operations as just one line of code, and then we break down what each command is doing in case you are curious (you can run each one to see what it's doing).

(Again, it is totally okay if you don't fully understand what some of the commands below are doing! That is not the focus on this assignment.)

```
#one-line full version
mean(as.numeric(unlist(strsplit(gsub("years", "", "1-2 years"), "-"))))
```

```
## [1] 1.5
```

(breaking down in excruciating detail – optional reading)

```
#breaking it down: inner most nested command
#in this case, the "gsub" command searches the term "1-2 years"
#and removes ("subs") the word "years"
gsub("years", "", "1-2 years")

#next: we use "strsplit" to split the character value we got
#from the previous step into substrings and remove the dash
strsplit(gsub("years", "", "1-2 years"), "-")

#the previous step returns a list, which we then un-list
unlist(strsplit(gsub("years", "", "1-2 years"), "-"))

#now it's just a two-element character vector,
#which we turn into numeric
as.numeric(unlist(strsplit(gsub("years", "", "1-2 years"), "-")))

#now that we have a two-element numeric vector,
#take the mean of the two numbers to find midpoint
mean(as.numeric(unlist(strsplit(gsub("years", "", "1-2 years"), "-"))))
```

It is always good practice to test the commands in your function before you write it in function format. Try to run that series of commands on "6-17 years" instead of "1-2 years" to make sure it works.

Task four Now, generalize the above commands into a function that can be run on any element of the "age" vector in the *autismae* dataset. Save this function as *agemidfunc*.

```
#potential function
age.mean.function <- function(age.level){
  mean(as.numeric(unlist(strsplit(gsub("years", "", age.level), "-"))))
}



age.mean.function("6-17 years")
```

```
## [1] 11.5
```

```
agemidfunc <- function(x){
  mid <- mean(as.numeric(unlist(strsplit(gsub("years", "", x), "-"))))
  return(mid)
}
```

Task five: use sapply to apply the function to every element of the data's age vector. Store the result (so don't transform directly) as a new variable in your data as *agemidfuncresult*

```
# base R
autismae2$agemidfuncresult <- sapply(X = autismae2$age, FUN = age.mean.function)

# dyplr alternative
autismae2 <- mutate(autismae2, agemidfuncresult = sapply(autismae2$age, agemidfunc))
```

Step two: using functions to structure data in a way useful for plotting

We're going to be creating a function that creates and arrays plots of an *individual state's* trends in the counts of parent reports of vaccine events over time.

Task one: to make the next steps easier, restrict the data to exclude observations that are missing the year ("Unknown Date")

```
austimae.plot.data <- filter(autismae2, age != "Unknown Date")

autismaecompyear <- autismae[autismae$year != "Unknown Date", ]
```

Task two: before moving to the function, we're going to get the data in a format that is easier to feed the function. Create a new data.frame, *stateyearcounts*, that indicates the number of cases per year for each state. Practice doing this using dplyr and pipes.

```
#load dplyr
library(dplyr)

stateyearcounts <- austimae.plot.data %>% group_by(year, state) %>%
  select(id) %>%  summarise(reports = n())



#use dplyr to obtain a count of cases by state and year
#here we specified the package for summarise because we had another
#package where a command shares that name that was causing errors
stateyearcounts <- autismaecompyear %>%
  group_by(state, year) %>%
  dplyr::summarise(reports = n())
```

Step three: using functions to plot

Task zero Load the data: stateyearreports.csv. Is this data tidy? Why or why not? (hint: can you imagine easily plotting the number of reports by year for every state?). If not, tidy the data.

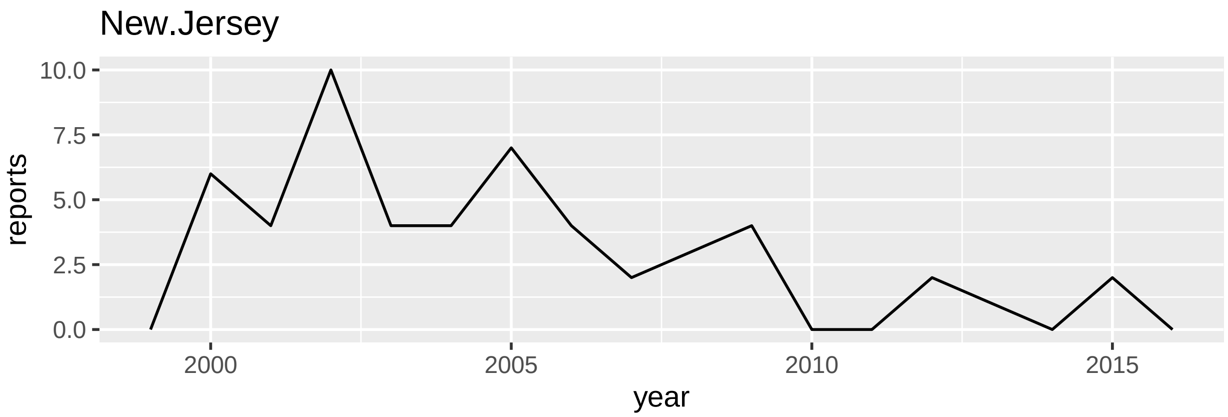
```
dirty <- read.csv("stateyearreports.csv")
library(tidyr)

stateyear <- dirty %>% gather(state, reports, -c(1:2))
```

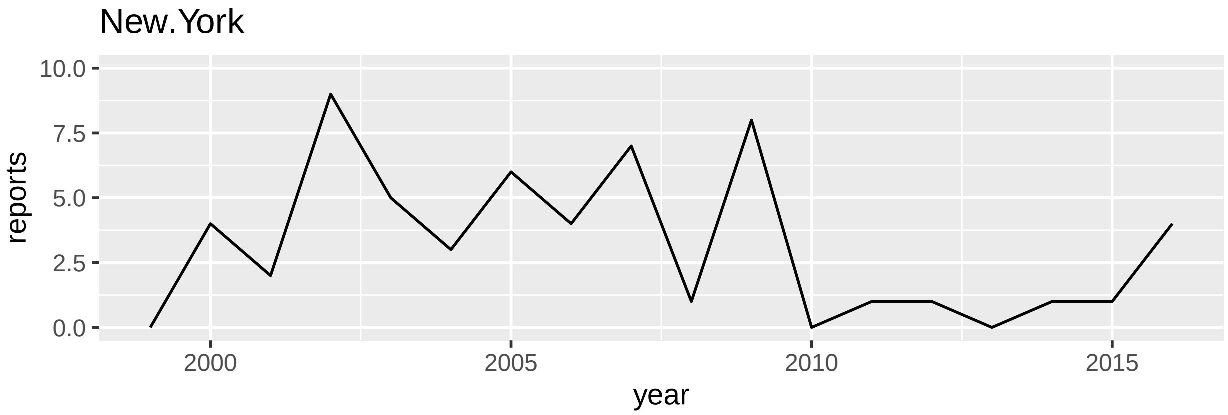
Task one: We're going to plot in two steps:

1. Creating two plots outside a function to get the code correct
2. Generalizing to a function that will plot the counts by year for any group of states you choose

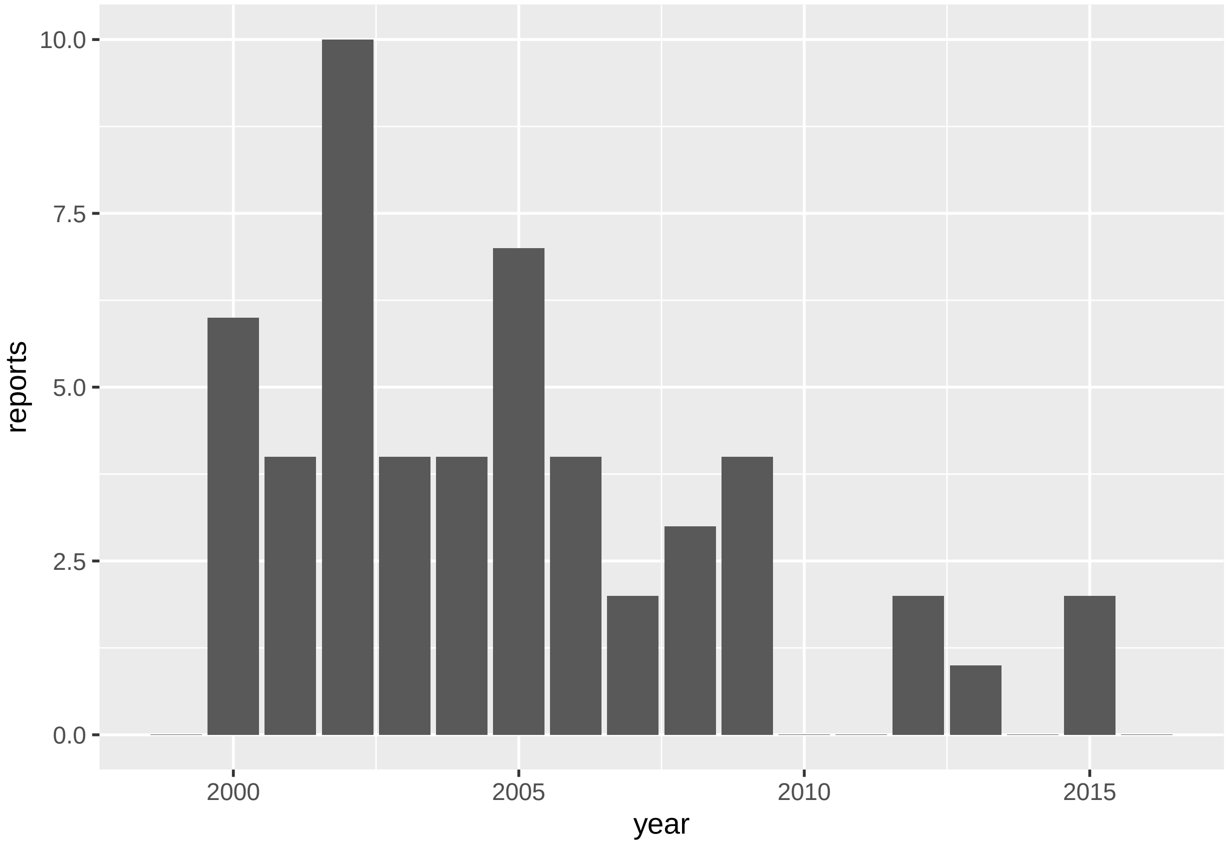
First, use ggplot to create separate plots for the counts of autism-related vaccine reports by year for two states– New Jersey and New York–side by side. You can either do a bar or line graph. Make sure the title indicates which state it is and make sure the two plots have the same y axis range for comparability purposes (0 to the maximum reports out of the two)

```
library(ggplot2)

##create vector with two state names (not necessary but can be useful for max)
statesofinterest <- c("New.Jersey", "New.York")


##create and store plots
njplot <- ggplot(stateyear[stateyear$state == statesofinterest[1], ],
       aes(x = year, y = reports)) +
      geom_line()+
      ggtitle(statesofinterest[1]) +
      ylim(0, max(stateyear$reports[stateyear$state
                                           %in% statesofinterest],
                  na.rm = TRUE))
nyplot <- ggplot(stateyear[stateyear$state == statesofinterest[2], ],
       aes(x = year, y = reports)) +
      geom_line(stat = "identity") +
      ggtitle(statesofinterest[2])  +
      ylim(0, max(stateyear$reports[stateyear$state
                                           %in% statesofinterest],
                  na.rm = TRUE))

##arrange side by side
library(gridExtra)
grid.arrange(njplot, nyplot)
```

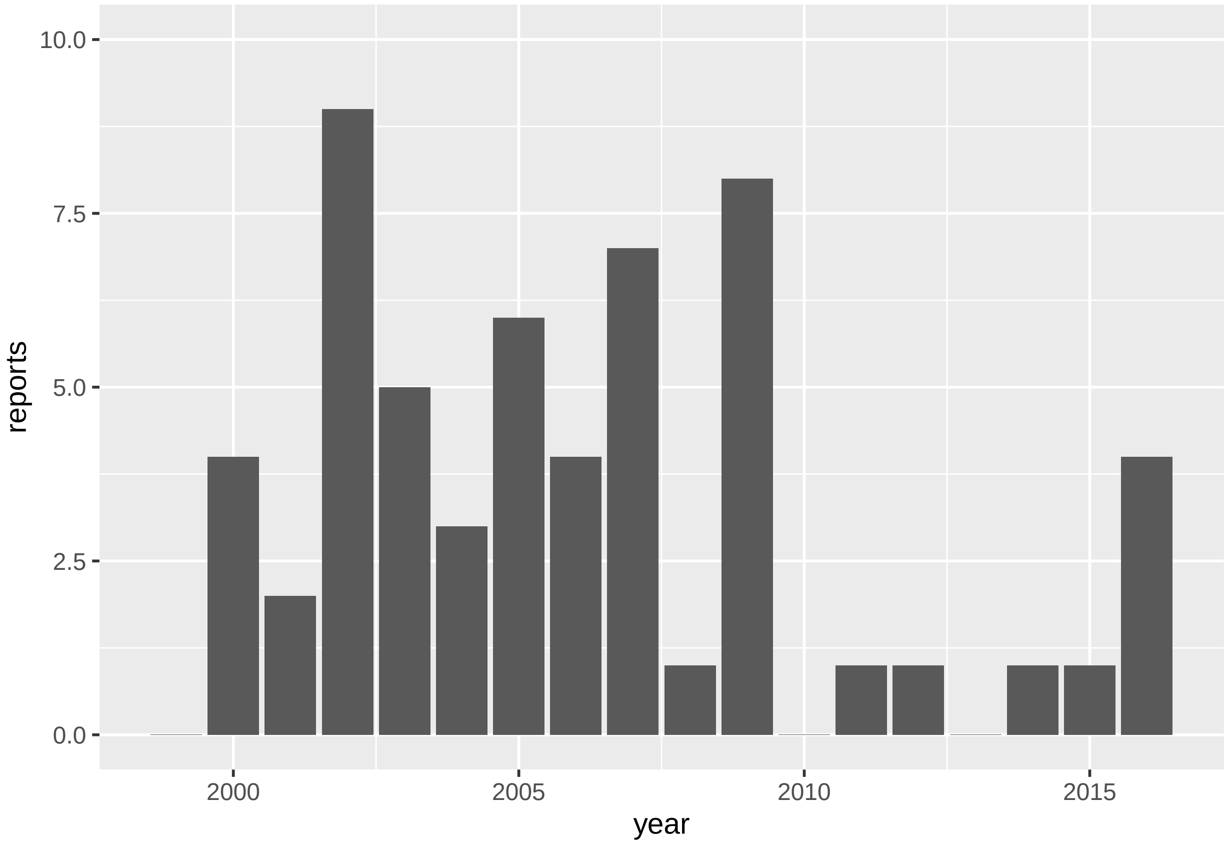




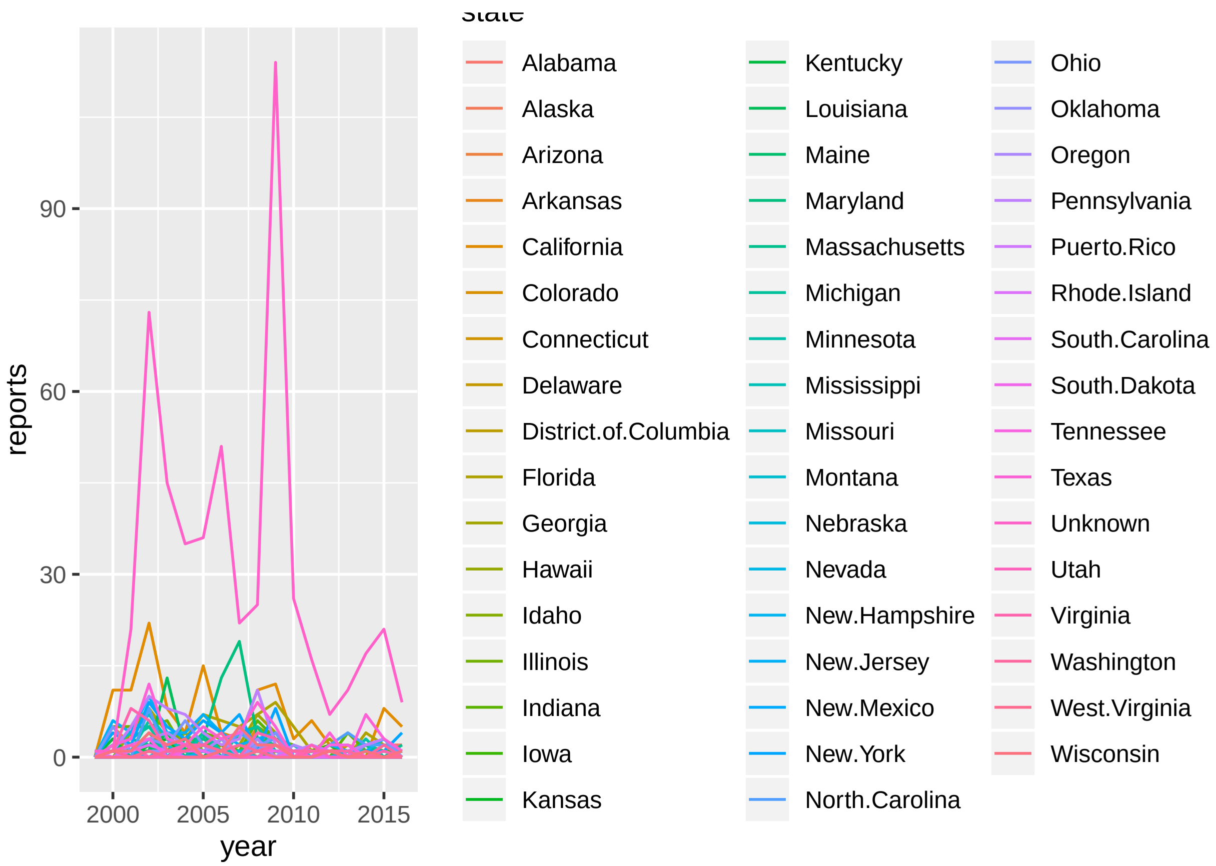
```r
# dplyr to do the same thing
stateyear %>% filter(state == statesofinterest[1]) %>%
  ggplot(aes(x = year, y = reports)) +
  geom_bar(stat = "identity") +
  ylim(0, 10)
```



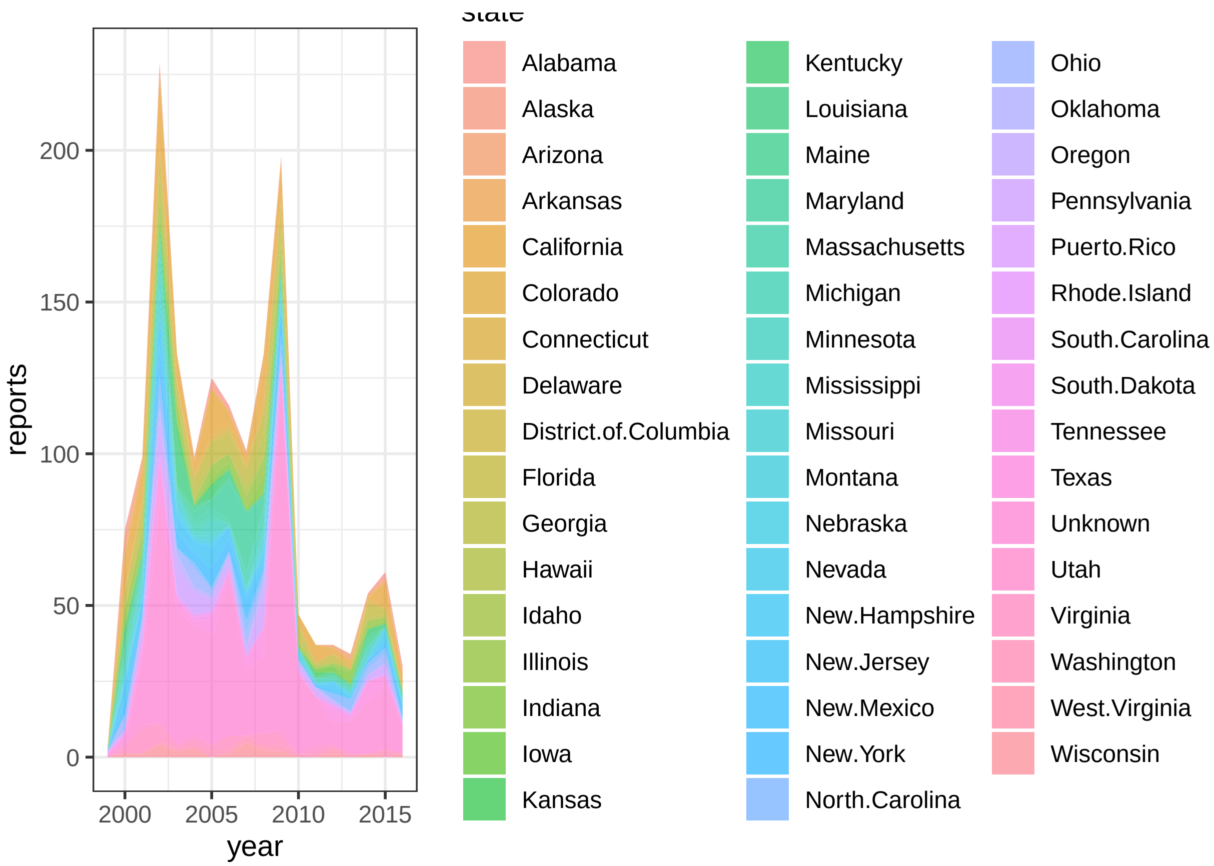
```
stateyear %>% filter(state == statesofinterest[2]) %>%
  ggplot(aes(x = year, y = reports)) +
  geom_bar(stat = "identity") +
  ylim(0, max(stateyear$reports[stateyear$state
                                           %in% statesofinterest],
                  na.rm = TRUE))
```



```
# Unrelated to homework, ggplot exploration for fun
# Extra plots for fun
stateyear %>%
  ggplot(aes(x = year, y = reports, color = state)) +
  geom_line()
```



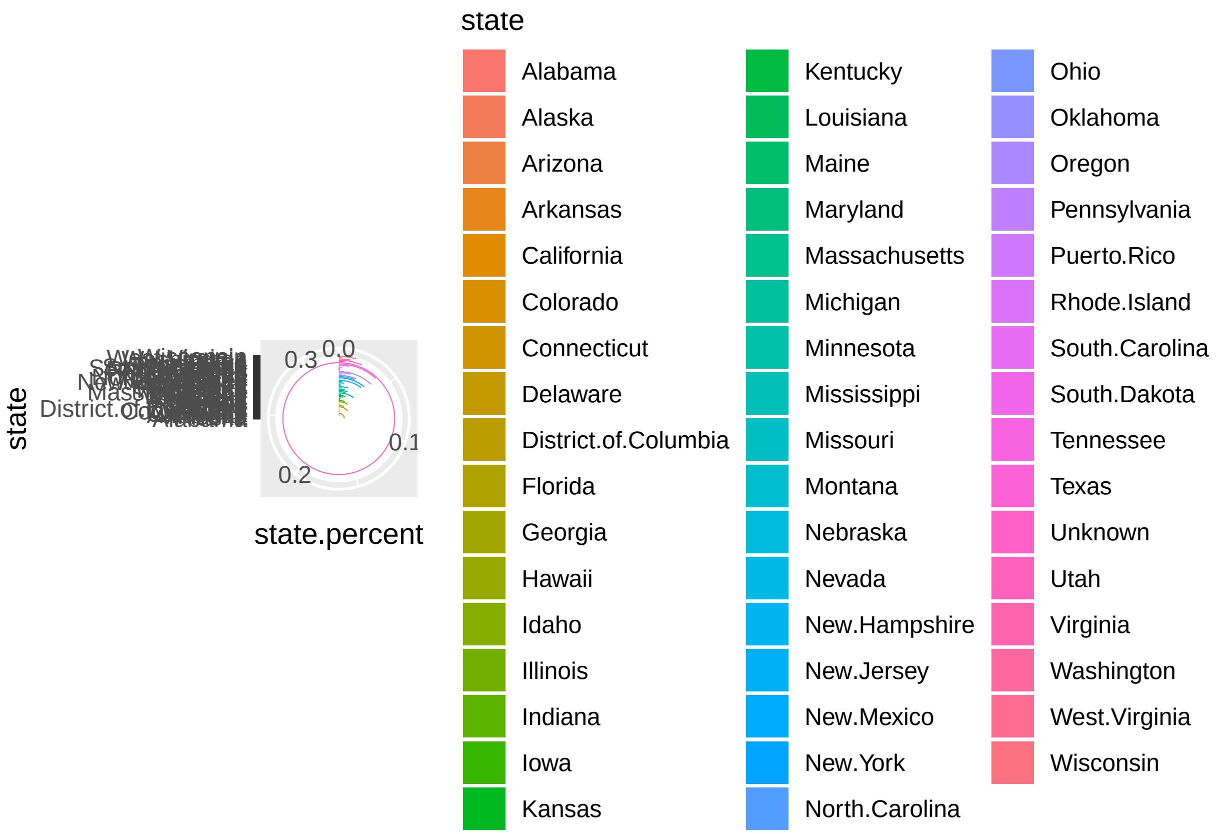
```r
# you can also use geom_area in this which may look more fluid/nicer
state.plot <- stateyear %>%
  ggplot(aes(x=year, y=reports, fill=state)) +
  geom_area(alpha = 0.6)

state.plot + theme(legend.position="bottom") + theme_bw()
```



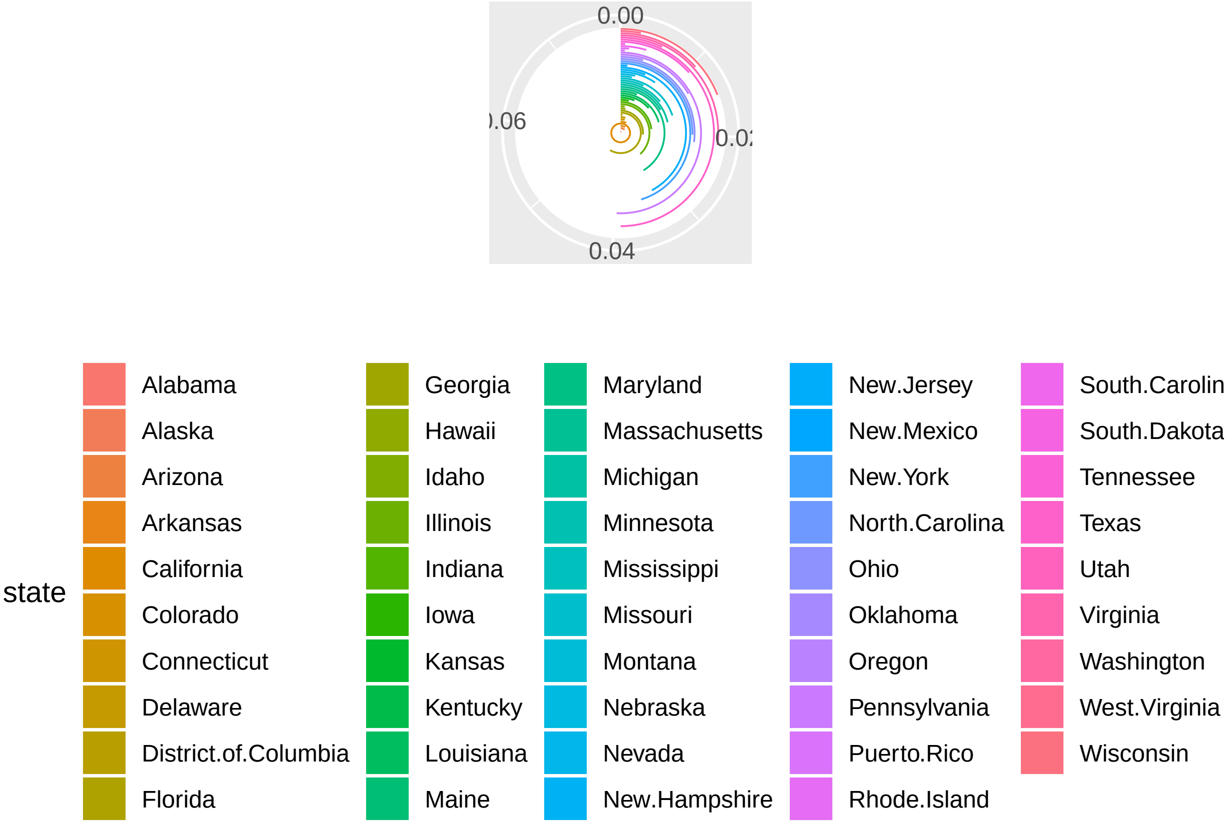
```
# what if we just want to see which state is the biggest reporter
state.report <-stateyear %>%
  group_by(state) %>%
  summarise(state.percent = sum(reports)/sum(stateyear$reports)) %>%
  arrange(desc(state.percent))
head(state.report)
```

```
## # A tibble: 6 x 2
##   state        state.percent
##   <chr>                <dbl>
## 1 Unknown             0.330
## 2 California          0.0782
## 3 Florida             0.0459
## 4 Pennsylvania        0.0397
## 5 Texas               0.0391
## 6 New.York            0.0354
```

```
# what is the pink, check data visually
ggplot(state.report, aes(x = state, y = state.percent, fill = state)) +
  geom_bar(width = 0.85, stat="identity") +
  # To use a polar plot and not a basic barplot
  coord_polar(theta = "y")
```



```r
#filter out unknown state
state.report.filtered <- stateyear %>%
  group_by(state) %>%
  summarise(state.percent = sum(reports)/sum(stateyear$reports)) %>%
  arrange(desc(state.percent)) %>%
    filter(state!= "Unknown")

# what is the pink, check data visually
ggplot(state.report.filtered, aes(x = state, y = state.percent, fill = state)) +
  geom_bar(width = 0.85, stat="identity") +
# To use a polar plot and not a basic barplot
  coord_polar(theta = "y") +
  #Remove useless labels of axis
  xlab("") + ylab("")  +
  #Remove useless legend, y axis ticks and y axis text
  theme(legend.position =  "bottom", axis.text.y = element_blank() ,
        axis.ticks = element_blank())
```



```
my.list <- list(c(1:5), c("New York", "Hello", "New Jersey"), c(FALSE, TRUE), nyplot)
```

Task two: now generalize into a function that can do the following:

- Take in a vector of state names
- Iterate through that vector and with each state, create and store a plot of that state's autism reports per year
- Return a list containing all the stored plots (the length of the list will be equal to the number of states you specified in the vector of state names)

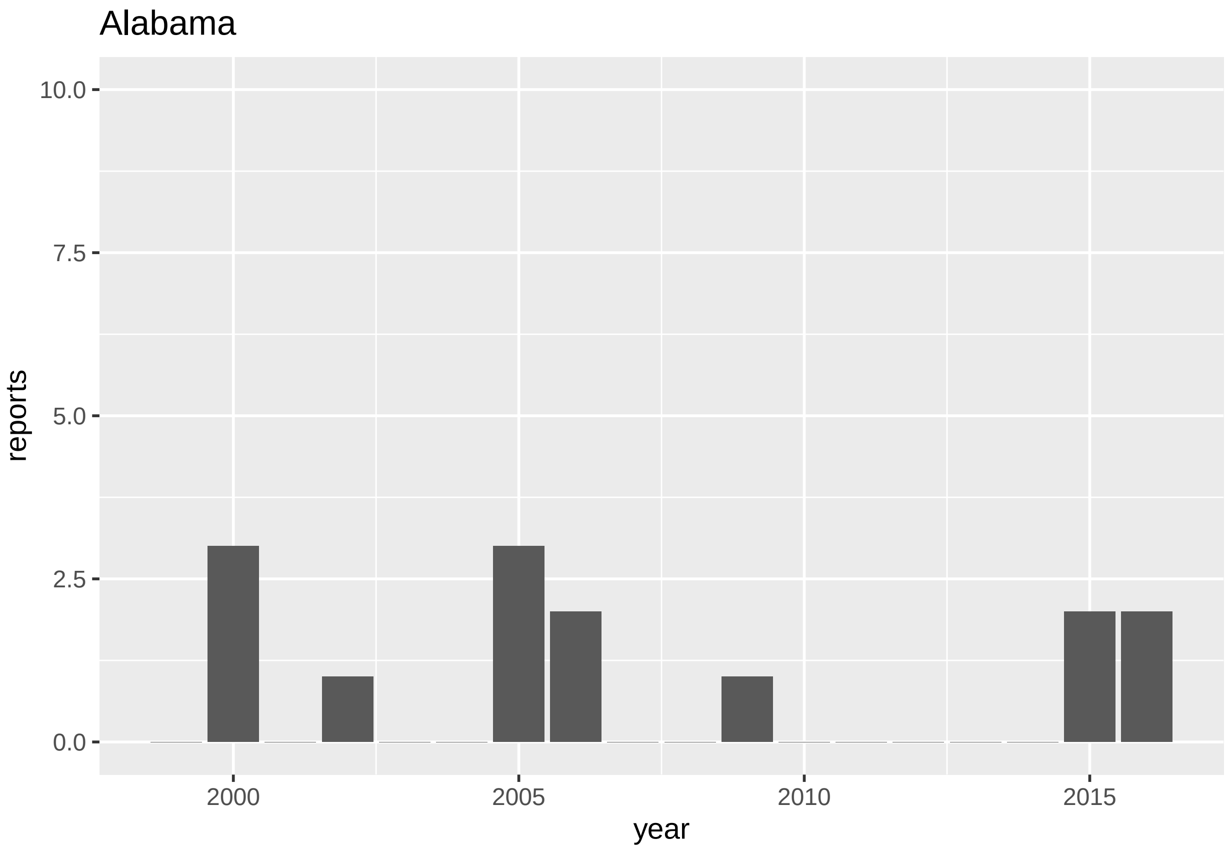
Store the function

Hint: Look at the above code for the two states. What did you change when copying and pasting? How can you subset the vector to give you the name of the state for the title and plot?

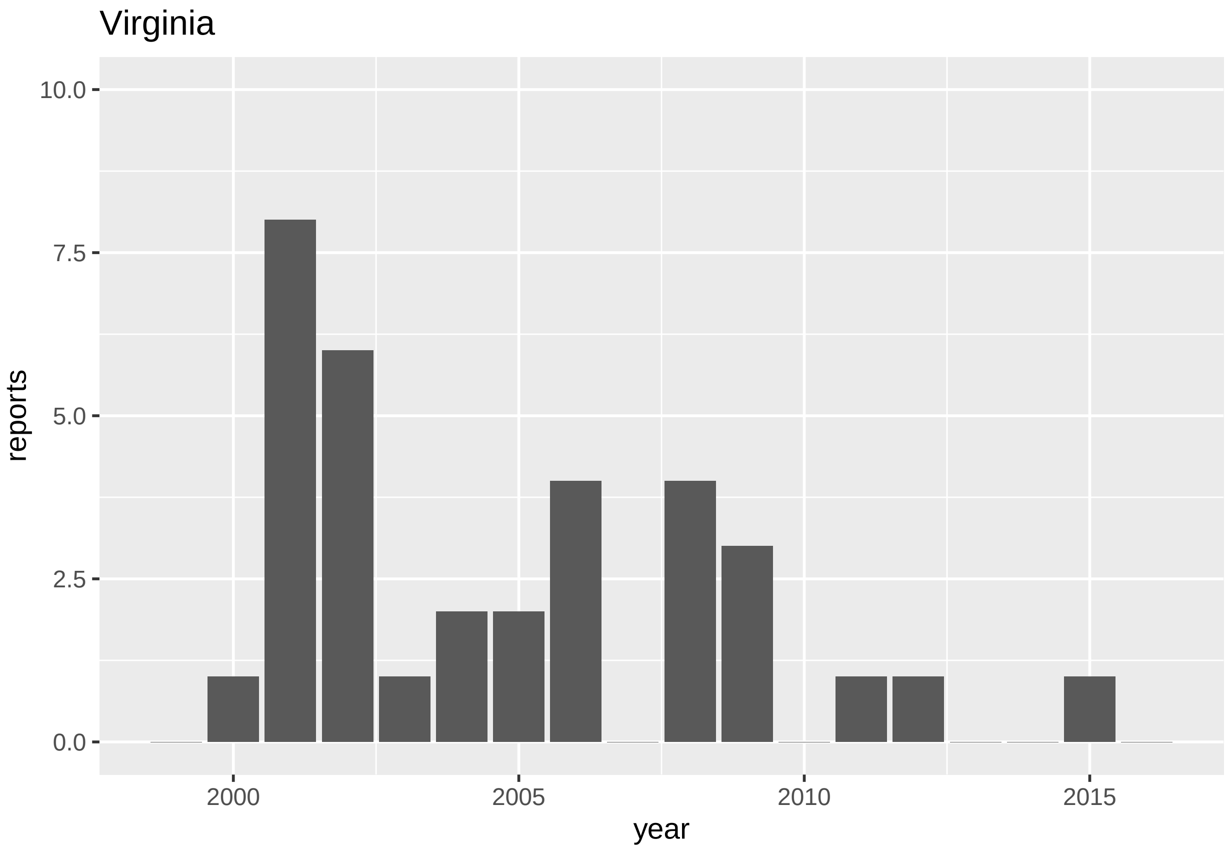
```
counthistfunc <- function(my.state){
stategraph <- ggplot(stateyear[stateyear$state == my.state, ], #subsets the stateyearcounts to state of
aes(x = year, y = reports)) +
geom_bar(stat = "identity") +
ggtitle(my.state) +
ylim(0, max(stateyear$reports[stateyear$state
%in% statesofinterest],
na.rm = TRUE)) #makes y axis range go from 0 to max across plots- y axis should almost always start
#at zero to avoid exaggerating differences by deflating the axis, we're maxing it go to the max of the
#across the states of interest so that we can mor easily compare states
return(stategraph)
}
```

```
state.list <- lapply(c("Alabama", "Virginia" ,"Wisconsin"), counthistfunc)
state.list
```

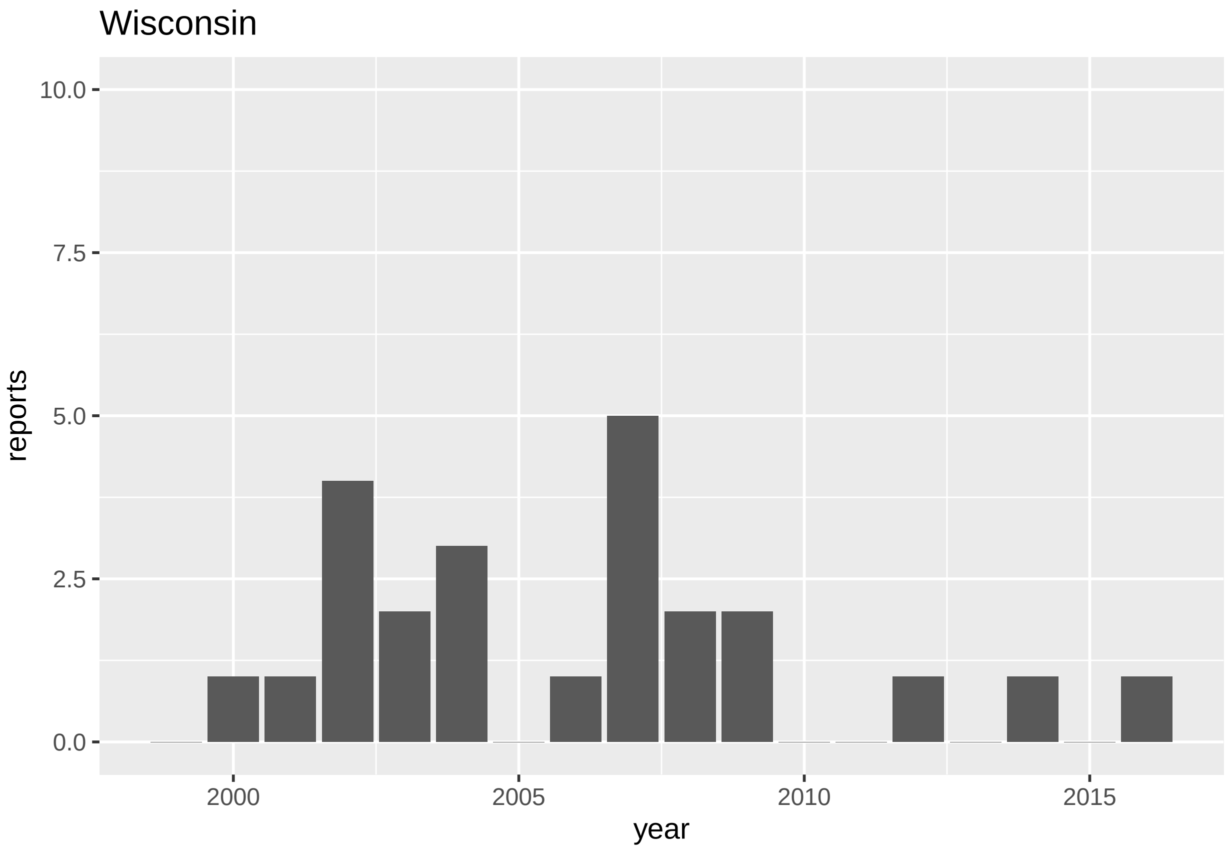
```
## [[1]]
```



```
##
## [[2]]
```



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



Task two: Run the function with different groups of states that you're interested in comparing (e.g., you could create a vector with the state where you grew up, the state where you went to college, and the state you're in now, and plot graphs for each).

Try with at least two different groups of states and store the results.

```
#create vector with states of interest
statesofinterest <- c("Illinois", "California", "New.Jersey")

#another state of interests
statesofinterest2 <- c("Iowa", "Texas", "Mississippi", "Florida")

listofstategraphs <- sapply(statesofinterest, counthistfunc)

listofstategraphs2 <- sapply(statesofinterest2, counthistfunc)
```

Step four: troubleshooting errors in functions

You can use conditionals inside the function to print informative error messages when something goes wrong. Here, we're going to practice using those conditionals in the context of potential errors.

We provide you with a function, *sample.subset* that does the following:

- Takes the following arguments: data, number of draws (numdraws), the variable we want to take the mean of during the draw (varofinterest), i (number of times to draw from the sample), an empty vector to store the means (vecformeans), a logical flag to indicate whether the sampling should be with or without replacement (replaceornot)

- The function uses the sample command with the following arguments: sample from a numeric vector with 1. . . number of rows in the data, the number of samples = the number of draws argument, and whether it replaces or not equals the replaceornot argument
- Then, the function subsets data based on those row ids and find the mean age from the numeric version of the variable (there are NA's in the data so make sure the arguments for mean still allow you to ignore those and still calculate)
- The function repeats that process *i* times and store the means in a vector
- The function returns that vector of means that's equal to the number of iterations

```
#ran outside the function once
rows_sampled <- sample(1:nrow(autismae),
                       100, replace = FALSE)

dfsamp <- autismae[rows_sampled, ]
# agenumeric needs to be appended to the dataset
mean(dfsamp$agenumeric, na.rm = TRUE)
```

```
## [1] 2.830628
```

```
#generalized to a function, but be careful you need to include
# an unused i in the arguments if you want to use it in sapply somehow
sample.subset <- function(varofinterest, data, numdraws,
                       replaceornot = FALSE, i,
                          vecformeans){
  rows_sampled <- sample(1:nrow(data),
                  numdraws, replace = replaceornot)
  df_sample <- data[rows_sampled, varofinterest]
  sample_mean <- mean(df_sample, na.rm = TRUE)
  vecformeans <- c(vecformeans, sample_mean)
  #return(vecformeans)
}
```

Task one: use a function in the apply family to run the function for five iterations (i = 5) with 100 draws each iteration. Store the results in a vector.

```
#five times with 100 draws each time
set.seed(123)
i <- 1:5
means5x100draws <- sapply(1:5, sample.subset,
               varofinterest = "agenumeric",
              data = autismae,
              numdraws = 100, replaceornot = FALSE,
              vecformeans = c())




means5x100draws
```

```
## [1] 2.961538 2.167254 1.904321 3.028571 2.340955
```

```
# if you think the i indexing is crazy or confusing, can try tidyverse:: purrr
# so first no need to define i in the function arg.
# why? I HAVE NO IDEA!!#$%@#
```

```
sample.subset.noI <- function(varofinterest, data, numdraws,
                       replaceornot = FALSE,
                          vecformeans){
  rows_sampled <- sample(1:nrow(data),
                  numdraws, replace = replaceornot)
  df_sample <- data[rows_sampled, varofinterest]
  sample_mean <- mean(df_sample, na.rm = TRUE)
  vecformeans <- c(vecformeans, sample_mean)
  return(vecformeans)
}
set.seed(123)
purrr.5draws <-
  5 %>% rerun(
          sample.subset.noI(
              varofinterest = "agenumeric",
              data = autismae,
              numdraws = 100, replaceornot = FALSE,
              vecformeans = c()
            ))
purrr.5draws
```

```
## [[1]]
## [1] 2.961538
##
## [[2]]
## [1] 2.167254
##
## [[3]]
## [1] 1.904321
##
## [[4]]
## [1] 3.028571
##
## [[5]]
## [1] 2.340955
```

```
# the results are a list, what if we want them as a vector instead
unlist(purrr.5draws) # same result as above coz we set seed here
```

```
## [1] 2.961538 2.167254 1.904321 3.028571 2.340955
```

```
# even within the apply family, we could also ignore the i index
# and use the replicate command instead, part of lapply
# note that the answer is obviously different from previous because
# we're doing random sampling and didn't set seed
meanwithrep <- replicate(5, sample.subset.noI(varofinterest = "agenumeric",
                       data = autismae,
                       numdraws = 100,
                       replaceornot = FALSE,
                       vecformeans = c()))
meanwithrep
```

```
## [1] 4.595833 2.119118 1.789502 2.242089 2.378289
```

Task two: now, run the function for five iterations (i = 5) with 2000 draws each iteration and the replacement flag still set to false. It should return an error. What's happening?

Answer: error since we're not replacing the samples we're taking and we're trying to take more samples than there are rows. Also for R markdown purposes, note that I specified error = TRUE at the top of this code chunk to allow me to knit despite the presence of an error. You might want to check with your Soc 500 preceptors about whether using this option is permissible to allow you to knit a problem set to submit despite an error in the code that you can flag to preceptors.

```
test2 <- sapply(i, sample.subset, data = autismae,
                numdraws = 2000, replaceornot = FALSE,
                vecformeans = c())
```

```
## Error in sample.int(length(x), size, replace, prob): cannot take a sample larger than the population
```

Task three: the error message is already pretty informative so no need to write our own. Instead, add a condition to the function that checks if the number of draws is greater than the number of rows in the dataframe. If that is the case, then the function should round the number of draws down to 0.9 x the number of rows in the data.frame and proceed with the rest of the steps (sampling, subsetting the data, finding the mean age)

```
sample.subset2 <-
  function(data, numdraws, replaceornot = FALSE, i,
                          vecformeans){
    if(numdraws > nrow(data)) {
      newnumdraws <- 0.9 * nrow(data)
      rows_sampled <- sample(1:nrow(data), newnumdraws,
                           replace = replaceornot)
      df_sample <- data[rows_sampled, ]
      meanage <- mean(df_sample$agenumeric, na.rm = TRUE)
      vecformeans <- c(vecformeans, meanage)
  } else {
    rows_sampled <- sample(1:nrow(data), numdraws, replace = replaceornot)
    df_sample <- data[rows_sampled, ]
    meanage <- mean(df_sample$agenumeric, na.rm = TRUE)
    vecformeans <- c(vecformeans, meanage)
  }
 # return(vecformeans)
}
```

Task four: practice applying that function to two cases: a case where the number of draws exceeds the number of rows in the data and a case where the number of draws is less than the number of rows in the data to confirm that the function works in either case.

```
#exceeds rows
i <- 1:5
test3 <- sapply(i, sample.subset2, data = autismae,
                numdraws = 5000, replaceornot = FALSE,
                vecformeans = c())
head(test3)
```

```
## [1] 2.473485 2.474184 2.510551 2.453510 2.449520
```

```
#less than nrows
i <- 1:5
test4 <- sapply(i, sample.subset2, data = autismae,
                numdraws = 300, replaceornot = FALSE,
                vecformeans = c())
head(test4)
```

```
## [1] 2.180265 2.229867 2.283408 2.592593 2.592250
```