Applied Demographic Data Analysis

Corey S. Sparks, PhD

Last Updated on 06 September, 2023

Table of contents

Preface

Why a book on statistics for demographers?

Demographers have always been a mixture of sociologists, economists, statisticians, health researchers, and other broad sub-disciples of social science. As such, we bring with us a large amount of baggage from our respective academic life courses, and we often are trained by a wide variety of mentors and professors. It's my perspective that our interdisciplinary experience is one of our greatest strengths as a group. Given that our training is often in one of a core set of home disciplines, we often have methodological training from said discipline, and this may not be a broad enough perspective to firmly ground us in the types of methods that demographers commonly employ. This is not the fault of the departments that trained us, it's just a historical fact. So, why am I writing a book on statistics and data analysis aimed at demographers? I will give you three reasons:

1. Demographers have to go beyond the sample. This is to say that our results and research is generally representative of a larger national or international population, and we do this explicitly in our models.
2. We demographers don't use random samples for our analysis. Statistics books the world over are based on assumptions of random sampling and independence, while the data that we often have to, or desire to use, comes more than likely from a data source that was collected using a complex survey design. This is a big deal and we have to have training materials that instill this in our students early on in their careers.
3. Weird data. As demographers, we often use data from lots of different places and if you were trained up to this point to believe that the linear model is the end-all be-all of statistical inference, I've got news for you friends, you've been misled. Categorical outcomes, counts, hierarchically structured, longitudinally collected, spatially referenced, just to name a few of such oddities, are ubiquitous in our field, and part of what makes our discipline so cool and interesting to newcomers.

My goal for this book is to take the lessons I've learned teaching statistics to a diverse and often cursorily trained group of students who have problems they care about, that they need to bring demographic data to bear upon. This is a challenge, and I have always been a stalwart proponent of teaching statistics and data analysis in a *very* applied manner. As such, this book won't be going into rigorous proofs of estimators or devoting pages to expositions of icky

algebra; instead it will focus on exploring modern methods of data analysis that in used by demographers every day, but not always taught in our training programs.

As someone who has learned much more of these methods by personal exploration than by formal study, I find that many of these methods are absent from the canon of social science statistics, but are both in great demand from people who hire us, and absolutely necessary to the demographer's analytic toolkit. It's a major goal of this book to de-mystify the process and to make it accessible to a wide audience, so I will always strive to illustrate the key aspects of the methods described herein, and ground the discussion of methods in applications.

is this a cookbook?

Broader picture of what i'll cover

What's a demographer?

What is applied demography?

Why we need to see this stuff?

Why R is a good option for applied demography?

why write code?

Mention packages earlier - talk about later

1 Preface

2 R in applied demography

2.1 Why R?

I've used R for twenty years. I was also trained in SPSS and SAS along the way, by various mentors. Some tried to get me to learn more general purpose languages like Delphi (of all things) or Perl, or Basic, and I've been chastised for not knowing the depths of Python, but R presents a nimble and rigorous platform to *do* demography. My top three reasons for teaching and using R are:

1. It's free - This is important, because, why should we pass along more costs to people, especially our students? This also make R code accessible to people, worldwide.
2. It's the hotbed of methodological development. The R ecosystem has thousands of packages that represent the bleeding edge of data analysis, visualization and data science. This makes R attractive because it can pivot quickly to adopt new methods, which often lag in their development in other environments.
3. It has a supportive community of users. While there are some debates over how friendly some R users are to new users, overall, after spending 20 years in the R community, I've personally assisted hundreds of users, and been personally helped by many others. The open source nature of R lends itself to sharing of ideas and collaboration between users.

2.1.1 My assumptions in this book

In statistics we always make assumptions, often these are wrong, but we adapt to our mistakes daily. My assumptions about who is reading this book are:

1. You are interested in learning more about R.
2. You are likely a student or professional interested in demography or population research.
3. You have likely been exposed to other statistical platforms and are curious about R, in conjunction with 1 and 2 above.
4. You may be an avid R user from another strange and exotic discipline, but are interested in how demographers do research.

5. You want to see *how* to do things instead of being bombarded with theoretical and often unnecessary gate-keeping mathematical treatments of statistical models.

I think if any of these assumptions are true, you're in the right place. That being said, this book *is not* a review of all of statistics, nor is it an encyclopedic coverage of the R language and ecosystem. I image the latter being on the same scale of hopelessness as the search for the Holy Grail or the fountain of youth. People have died for such fool hearty quests, I'm not falling on my sword here folks.

3 Introduction to R

4 Introduction to R

This chapter is devoted to introducing R to new users. R was first implemented in the early 1990's by Robert Gentleman and Ross Ihaka, both faculty members at the University of Auckland. It is an open source software system that specializes in statistical analysis and graphics. R is its own programming language and the R ecosystem includes over 18,000 user-contributed additions to the software, known as packages.

4.1 Welcome to R.

If you're coming to R from SAS, there is no data step. There are no procs. The SAS and R book Kleinman and Horton (2014) is very useful for going between the two programs.

If you're coming from SPSS and you've been using the button clicking method, be prepared for a steep learning curve. If you've been writing syntax in SPSS, you're at least used to having to code. There's a good book for SAS and SPSS users by Bob Meunchen at the Univ. of Tennessee here, which may be of some help.

Stata users have fewer official publications at their fingertips to ease the transition to R, but I have always thought that the two were very similar. If you search the internet for information related to R and Stata, you will find a myriad of (somewhat dated) blog posts on which one is better for "data science", how to "get started" with either and so forth. What is really needed is a text similar to Kleinman and Horton (2014) which acts as a crosswalk between the two programs.

4.2 R and Rstudio

The Rgui is the base version of R, but is not very good to program in. Rstudio is much better, as it gives you a true integrated development environment (IDE), where you can write code in one window, see results in others, see locations of files, and see objects you've created. To get started, you should download the R installer for your operating system. Windows and Mac have installer files, and Linux users will install R using their preferred package manager.

Download R from CRAN. If you're on Windows, I would also highly recommend you install Rtools, because it gives you c++ and Fortran compilers, which many packages need to be installed.

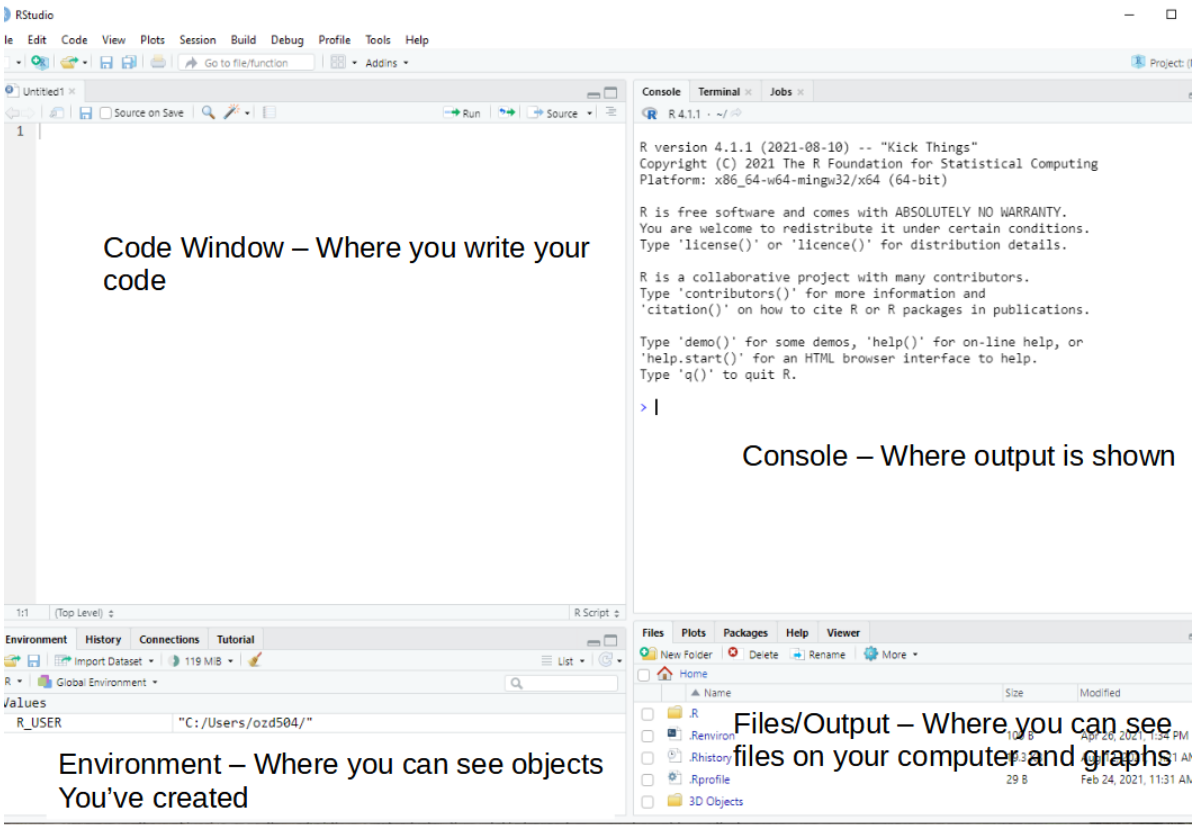
Rstudio can be downloaded for free here.

I would recommend installing the base R program from CRAN first then (for Windows users) install Rtools, then install Rstudio, in that order.

4.3 Introduction to Rstudio

Again, each operating system has its own binary for Rstudio, so pick the one that matches your operating system. Rstudio typically has 4 sub-windows open at any given time.

Rstudio is an open source Integrated Development Environment (IDE) for R. It is a much better interface for using R because it allows you to write code in multiple languages, navigate your computer's files, and see your output in a very nice single place. The Rstudio IDE has several components that we will explore.



4.3.1 Code window/Source editor pane

- This is where you write your R code. You can write R code in a few different file types (more on this later), but the basic one is an R script, with file extension .R
- The code window allows you to write and execute your code one line at a time, or to run an entire script at once. I use this to develop new code and when I want to test if things work (a VERY common exercise when writing any code).
- To run a single line of code, put your cursor on the line and hit Ctrl-Enter (on Mac CMD-Enter also does this)
- To run multiple lines of code, highlight the lines you want to run and do the same thing

4.3.2 Console Pane

- This is where most of your non-graphical output will be shown. Any numeric output will appear here, as well as any warnings or error messages. In R a warning doesn't necessarily mean something went wrong, its just R's polite way of telling you to pay attention.
- An Error means something did go wrong. This is often because you left off a) or a, sometimes because you misspelled something. I routinely spell `length` as `lenght` which causes R to print an error message. If you see an error, don't worry, R will print some kind of message telling you what went wrong.
- R's output is in plain text, although we can produce much prettier output using other output methods, and we'll talk more about that later.
- You can type commands or code into the console as well, and you'll immediately get the result, versus if you write it in the Source/Code window, you have to run it to see the result. I will often work in the console when I want to get "fast" answers, meaning little checks that I will often do to see the value of something.

4.3.3 Environment or Workspace browser pane

- The R environment is where any object you create is stored. In R, anything you read in or create with your code is called an object, and R is said to be an object oriented programming language.
- Depending on the type of object something is, you may be able to click on the object in the environment and see more about it.
- For instance if the object is a data frame, R will open it in a viewer where you can explore it like a spreadsheet, and sort and filter it as well.

- Other objects may not do anything when you click on them.
- There is also a useful History tab here that shows you recently executed lines of code from the console or the code pane.

4.3.4 Files/Output/Help pane

- The files and output area is where you can interact with files on your local computer, such as data files or code files, or images that R can open.
- This area also has a plots window that will display plots you create in R either via typing directly into the console or by running a line(s) of code from the source/code pane.
- There is also a very valuable part of this pane that lets you access the help system in R. If you are either looking for something, or you just want to explore the functions, you can get access to all of this here.

4.3.5 R file types

.R files R uses a basic text file with the .R extension. This type of file is useful if you're going to write a function or do some analysis and don't want to have formatted output or text. You can use these files for everything, but they are limited in their ability to produce reports and formatted output, so I recommend people work with R Markdown files instead.

.Rmd files Rstudio uses a form of the markdown formatting language, called R Markdown, for creating formatted documents that include code, tables, figures and statistical output. **This book is written in R Markdown!**

R Markdown is nice for lots of reasons, such as the ability to insert latex equations into documents.

$$y_i \sim Normal(x'\beta, \sigma_2)$$

or to include output tables directly into a document:

```
library(broom)
library(pander)
fit <- lm(imr~tfr+pcturban+pctlt15_2018+pctwomcontra_all,
          data = prb)
pander(broom::tidy(fit))
```

term	estimate	std.error	statistic	p.value
(Intercept)	6.209	6.551	0.9478	0.3446
tfr	3.392	2.006	1.691	0.09274
pcturban	-0.0923	0.04553	-2.028	0.04425
pctlt15_2018	0.8699	0.2441	3.564	0.0004798
pctwomcontra_all	-0.2114	0.06018	-3.512	0.0005757

This allows you to make tables in Rmarkdown without having to do non-repeatable tasks in Word or some other program. You can basically do your entire analysis, or a sideshow for a presentation, or an entire paper, including bibliography, in Rstudio.

4.3.6 R projects

Rstudio allows you to create a R project, which basically sets up a specific location to store R code for a given project you may be doing. For instance, this book is a single R project, which helps me organize all the chapters, bibliographies, figures, etc.

R projects also allow you to use version control, including Git and SVN, to collaborate and share code and data with others.

4.3.7 R data files

R allows you to read and write its own *native* data formats, as well as read and write text formatted files and data files from other statistical software packages. Two native R data formats are `.rds` and `.rdata` formats. `.rds` files allow you to save a single R object to an external files, while `.rdata` files allow you to save one or more objects to a file.

Here is a short example of doing this, where I create 2 vectors, `x` and `y` and save them.

```
x <- c(1, 2,3)

y <- c(4, 5, 6)

saveRDS(x,
        file="~/x.rds")

save(list=c("x","y"),
     file="xy.rdata")
```

I can also load these into R again:

```
readRDS(file = "~/x.rds")
```

```
[1] 1 2 3
```

```
load("xy.rdata")
```

Standard methods for importing text data such as comma separated value or tab delimited files can be read into R using `read.csv()` or `read.table()` and similar writing functions are available.

To read in a dataset from another statistical package, I recommend using the `haven` package. It allows you to read and write SAS (both sas7bdat and xpt files), Stata, SPSS (both .por and .sav files).

For example, here I write out a dataframe containing `x` and `y` from above to a SAS version 7 file:

```
xy <- data.frame(x = x, y = y)
xy
```

```
  x y
1 1 4
2 2 5
3 3 6
```

```
library(haven)

write_sas(data = xy,
          path = "~/xy.sas7bdat")
```

```
Warning: `write_sas()` was deprecated in haven 2.5.2.
i Please use `write_xpt()` instead.
```

I will describe dataframes more later in the chapter.

R also has packages for reading/writing such data formats as JSON, ESRI Shapefiles, Excel spreadsheets, Google Spreadsheets, DBF files, in addition to tools for connecting to SQL databases, and for interfacing with other statistics packages, such as Mplus, OpenBUGS, WinBUGS and various Geographic Information Systems.

4.4 Getting help in R

I wish I had a nickel for every time I ran into a problem trying to do something in R, that would be a lot of nickles. Here are some good tips for finding help in R:

1) If you know the name of a function you want to use, but just need help using it, try `?`

```
?lm
```

2) If you need to find a function to do something, try `??`

```
??"linear model"
```

3) You can also search the history of other R users questions by tapping into the RSiteSearch website, which is an archive of user questions to the R list serve. This can be used by tying `RSiteSearch()`

```
RSiteSearch("heteroskedasticity")
```

4) Speaking of which, there are multiple R user email list serves that you can ask questions to, or subscribe to daily digests from. These typically want an example of what you're trying to do, referred to as a *reproducible example*. I wish I also had nickles for each question I've asked and answered on these forums.

5) A good source for all things programming is the statistics branch of Stack Exchange, which has lots of contributed questions and answers, although many answers are either very snarky or wrong or for an old version of a library, so *caveat emptor*.

6) Your local R guru or R user group. You would be surprised at how many people are R users, there may be one just down the hall, or in the cubicle next door. I relish the opportunity to talk to other R users, mostly because, even though I've used R for more than 20 years, I still learn so much by talking to others about how they use R.

Lastly, I want to be clear that there are often **more than one way to do everything** in R. Simple things like reading and writing a CSV data file can be accomplished by any of a handful of different functions found in different packages. If someone tells you that there is only one way to do something, they are usually wrong in such a statement, regarding R at least.

4.5 R packages

R uses packages to store functions that do different types of analysis, so we will need to install lots of different packages to do different things. There are over 20,000 different packages currently for R. These are hosted on one of a number of *repositories*, such as the Comprehensive R Archive Network, or CRAN, which is the official repository for R packages. Other locations where authors store packages include R-Forge and BioconductoR. Many authors host packages in Github repositories, especially for development purposes.

Packages are often organized into *Task Views*, which CRAN uses to organize packages into thematic areas. You can find a list of these Task Views here. There is not a task view for Demography, but there are ones for the Social Sciences, Econometrics, and Spatial Data to name a few. Task views allow users to download a lot of thematically linked packages in a single command, through the package `ctv`, or Cran Task Views. You can install this package by typing:

```
install.packages("ctv")
```

into Rstudio. Then you have to load the package by using the `library()` command:

```
library(ctv)
```

which gives you access to the functions in that package. You don't need to install the package again, unless you update your R software, but each time you start a new session (i.e. open Rstudio), you will have to load the library again. If you want to install all of the packages in the Social Science task view, you would type:

```
install.views("SocialSciences")
```

into R and it will install all of the packages under that task view, as of the writing of this sentence, include over 80 packages.

I strongly recommend you install several packages prior to us beginning to use R, so you will not be distracted by this later. I've written a short script on my Github repository and you can use it by running:

```
source("https://raw.githubusercontent.com/coreysparks/Rcode/master/install_first_short.R")
```

This will install a few dozen R packages that are commonly used for social science analysis and some other packages I find of use.

You only have to install a package once, but each time you start a R session, you must load the package to use its functions. You should also routinely update your installed packages using `update.packages(ask = FALSE)`. This will update any packages that have new versions on CRAN. These often will contain bug fixes and new features. On CRAN, each package will have a README file that tells what has changed in the various versions. Here is one for one of my favorite packages `tidycensus`.

4.5.1 Functions within packages

Each package will have one or more functions within it, each doing a specific task. The default way to access all functions within a given package is to use the command `library(packagename)` to access the functions. Once loaded, all the functions will be accessible to you. Sometimes, different packages have functions with the same name, for example the base R library has the function `lag()`, which lag's a time series, the `dplyr` library also has a function `lag()`, which does a similar task, but with different function arguments. If you have the `dplyr` library loaded, R will default to use its `lag()` function. If you want to access a specific function within a specific library, sometimes it is safest to use the `library::function()` syntax. So if I want to use base R's `lag()` function, I could do

```
stats::lag()
```

to access that function specifically. How do you know when this happens? When you load a library, you will often see messages from R that functions have conflicts. For example, if I load `dplyr`, I see:

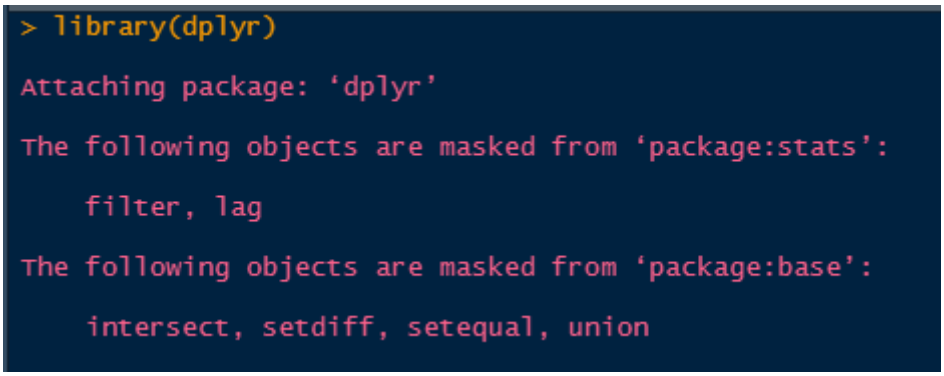


Figure 4.1: Conflict messages

As you use R more, you will learn which packages have conflicts, and often the developers of the packages will do this and rename the commonly conflicting functions. For example, the function to recode variables in the `car` package, `car::recode()` was renamed to `car::Recode()` to avoid conflicts with the `dplyr::recode()` function, as both are often used in the same analysis.

4.5.1.1 More notes on functions

Functions in R are bits of code that do something, what they do depends on the code within them. For instance, the `median()` function's underlying code can be seen by:

```
getAnywhere(median.default())
```

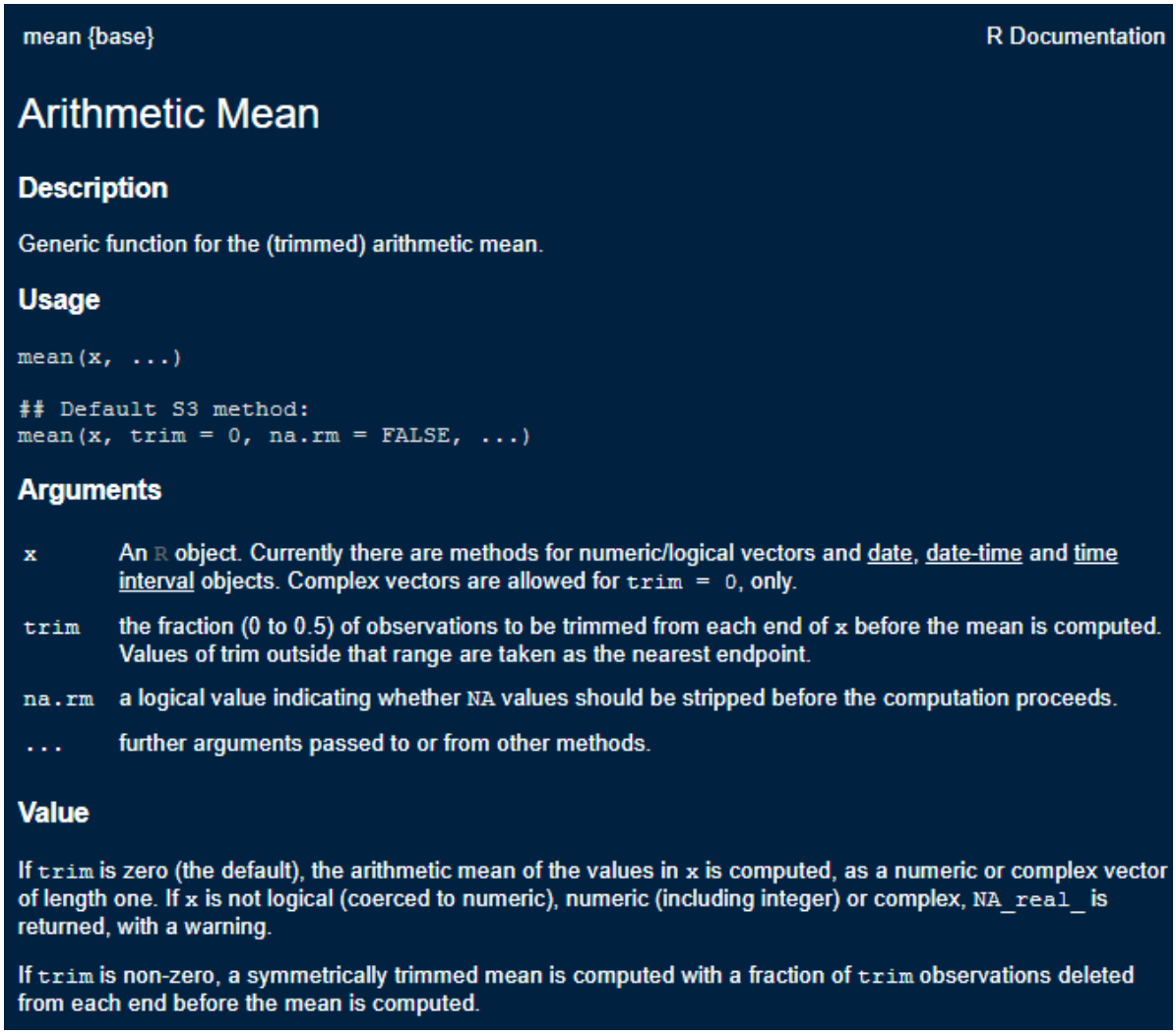
```
A single object matching 'median.default' was found
It was found in the following places
  package:stats
  registered S3 method for median from namespace stats
  namespace:stats
with value

function (x, na.rm = FALSE, ...)
{
    if (is.factor(x) || is.data.frame(x))
        stop("need numeric data")
    if (length(names(x)))
        names(x) <- NULL
    if (na.rm)
        x <- x[!is.na(x)]
    else if (any(is.na(x)))
        return(x[NA_integer_])
    n <- length(x)
    if (n == 0L)
        return(x[NA_integer_])
    half <- (n + 1L)%/%2L
    if (n%%2L == 1L)
        sort(x, partial = half)[half]
    else mean(sort(x, partial = half + 0L:1L)[half + 0L:1L])
}
<bytecode: 0x11105c358>
<environment: namespace:stats>
```

This seems like a lot, I know, but it allows you to see all of the code under the hood of any function. Obviously, the more complicated the function, the more complicated the code. For instance, I can write my own simple function to find the mean of a sample:

```
mymean <- function(x,
                   na.rm = FALSE){
  sx <- sum(x,
            na.rm = FALSE)
  nx <- length(x)
  mu <- sx/nx
  mu
}

mymean(x = c(1,2,3))
```

```
[1] 2
```

This function only includes the basic machinery to calculate the arithmetic mean of a vector x. The function has 2 **arguments**, `x` and `na.rm`. All R functions have one or more arguments that users must enter for the function to operate. Some arguments are required, while some are optional, also some arguments, such as the `na.rm = FALSE`, have default values. As mentioned earlier, to see all the information for a function's arguments, use the help operator, `?`. For example `?mean` will show you the help documents for the `mean()` function



mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

Figure 4.2: Mean Function Help

When using a new function, it's always advised to check out the help file to see all the arguments the function can take, because this is where you can choose alternative specifications for models

and methods. These help files also contain the original citations for methods, so you can immediately check the source of the algorithms. The help files also contain a working example of how to use the function on data contained in R.

4.6 Your R user environment

When you begin an R session (generally by opening Rstudio) you will begin in your home directory. This is traditionally, on Windows, at `'C:/Users/yourusername/Documents'` on Mac at `'/Users/yourusername'`, and on Linux at `'/users/yourusername'`. There are files you can add to your home directory to specify starting options for R.

You can find information on setting up `.Rprofile` and `.Renviron` files on CRAN's website. This allows you to setup packages that load every time R starts, to save API keys and other various options. These are completely optional and many R users never touch these.

If you're not sure where you are you can type `getwd()`, for get working directory, and R will tell you:

```
getwd()
```

If you don't like where you start, you can change it, by using `setwd()`, to set your working directory to a new location.

```
setwd("~")
getwd()
```

R projects will typically set the home folder for the project at the directory location of the project, so files associate with the project will always be in the same place. You can set this at the beginning of your R code file to ensure the code will look for data in a specific location.

4.7 Some Simple R examples

Below we will go through a simple R session where we introduce some concepts that are important for R. I'm running these in an Rstudio session, in the

4.7.1 R is a calculator

```
#addition and subtraction
3+7
```

```
[1] 10
```

```
3-7
```

```
[1] -4
```

```
#multiplication and division
3*7
```

```
[1] 21
```

```
3/7
```

```
[1] 0.4285714
```

```
#powers
3^2
```

```
[1] 9
```

```
3^3
```

```
[1] 27
```

```
#common math functions
log(3/7)
```

```
[1] -0.8472979
```

```
exp(3/7)
```

```
[1] 1.535063
```

```
sin(3/7)
```

```
[1] 0.4155719
```

R allows users to write custom functions as well. In general, if you find yourself writing the same code over and over again, you should probably just write a function and save it to your local user environment.

For example a very simple function is given below, it takes a variable x as an argument, and then exponentiates the value of the variable.

```
#custom functions
myfun <- function(x){
  exp(x)
}

myfun(.5)
```

```
[1] 1.648721
```

```
myfun(-.1)
```

```
[1] 0.9048374
```

You may want to save this function for future use, so you don't have to write it over again. In general, this is why people write R packages, to store custom functions, but you can also save the function to an R script. One such way to do this is to use the `dump()` command.

```
dump("myfun",
     file="myfun1.R")
```

One way to load this function when you want to use it is to use the `source()` command, which loads any code in a given R script.

```
source("myfun1.R")
```

Which will load this function into your local environment and you can use it. If you are interested in writing your own packages, I would highly recommend reading Wickham (n.d.).

4.8 Variables and objects

In R we assign values to objects (object-oriented programming). These can generally have any name, but some names are reserved for R. For instance you probably wouldn't want to call something 'mean' because there's a 'mean()' function already in R. For instance:

```
x <- 3
y <- 7
x+y
```

```
[1] 10
```

```
x*y
```

```
[1] 21
```

```
log(x*y)
```

```
[1] 3.044522
```

The `[1]` in the answer refers to the first element of a *vector*, which brings us to…

4.8.1 Vectors

R thinks many objects are like a matrix, or a vector, meaning a row or column that contains either numbers or characters. One of R's big selling points is that much of it is completely vectorized. Meaning, I can apply an operation along all elements of a vector without having to write a *loop*.

For example, if I want to multiply a vector of numbers by a constant, in SAS, I could do:

```
for (i in 1 to 5) x[i] <- y[i]*5 end;
```

but in R, I can just do:

```r
x <- c(3, 4, 5, 6, 7)
#c() makes a vector
y <- 7

x*y
```

```
[1] 21 28 35 42 49
```

R is also very good about using vectors, let's say I wanted to find the third element of x:

```r
x[3]
```

```
[1] 5
```

or if I want to test if this element is 10

```r
x[3] == 10
```

```
[1] FALSE
```

```r
x[3] != 10
```

```
[1] TRUE
```

or is it larger than another number:

```r
x[3] > 3
```

```
[1] TRUE
```

or is any element of the whole vector greater than 3

```r
x > 3
```

```
[1] FALSE  TRUE  TRUE  TRUE  TRUE
```

If you want to see what's in an object, use `str()`, for `str`ucture

```
  str(x)
```

```
 num [1:5] 3 4 5 6 7
```

and we see that x is numeric, and has the values that we made.

We can also see different characteristics of x

```
  #how long is x?
  length(x)
```

```
[1] 5
```

```
  #is x numeric?
  is.numeric(x)
```

```
[1] TRUE
```

```
  #is x full of characters?
  is.character(x)
```

```
[1] FALSE
```

```
  #is any element of x missing?
  is.na(x)
```

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

```
  #now i'll modify x
  x <- c(x, NA) #combine x and a missing value ==NA
  x
```

```
[1]  3  4  5  6  7 NA
```

```
#Now ask if any x's are missing
is.na(x)
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

4.8.1.1 Replacing elements of vectors

Above, we had a missing value in X, let's say we want to replace it with another value. He we will use basic conditional logic, which exists in any programming language. The `ifelse()` function will evaluate a `test` statement, and depending on if that statement is true, it will assign a value, if the statement is false, R will assign another value. Here, we replace the missing value with $\sqrt{7.2}$, and leave the other values as they are.

```
x <- ifelse(test = is.na(x) == TRUE,
            yes =  sqrt(7.2),
            no =  x)
x
```

```
[1] 3.000000 4.000000 5.000000 6.000000 7.000000 2.683282
```

4.9 Variable types

R stores data differently depending on the type of information contained. Common variables types in R are numeric, character, integer and factor.

Numeric variables are just that, numbers. They can be whole numbers or decimal values. The best way to see if a variable is numeric is to use `is.numeric(x)`, and R will return TRUE if the variable is numeric and FALSE if it is not.

```
is.numeric(x)
```

```
[1] TRUE
```

Likewise, you can use `is.character()`, `is.integer()`, and `is.factor` to identify if a variable is of a given type. The `class()` function will also do this more generally:

```
class(x)
```

```
[1] "numeric"
```

Character and factor variables often store the same kind of information, and R (until recently) would always convert character variables to factors when data were read into R. This is the option `getOption("stringsAsFactors")`, which used to default to True, but has recently changed. What's the difference you ask? Character variables store information on strings, or text. This is one way to code categorical variables that are strings. Factors, on the other hand can store strings OR numbers as categorical variables, and can be ordered or unordered. Factors also allow for specific categories of the variable to be considered as reference categories, as are often used in many statistical procedures. Factor variables have "levels" which are the different values of the categorical variable, this implied a more complicated structure than simple character variables, which lack these qualities.

You can manipulate variables of one type into another, with some notable things to watch out for.

Here are some examples:

```
#create at numeric vector

z <-  c(1,2,3,4)
class(z)
```

```
[1] "numeric"
```

We can convert this to a character vector using `as.character()`

```
zc <-  as.character(z)
zc
```

```
[1] "1" "2" "3" "4"
```

Likewise, we can convert it to a factor type:

```
zf <- as.factor(z)
zf
```

```
[1] 1 2 3 4
Levels: 1 2 3 4
```

```
  class(zf)
```

```
[1] "factor"
```

```
  is.ordered(zf)
```

```
[1] FALSE
```

and as an ordered factor:

```
  zfo <- factor(zf,
                ordered = TRUE)
  zfo
```

```
[1] 1 2 3 4
Levels: 1 < 2 < 3 < 4
```

Another very useful variable type is the *logical* type. In R a logical variable is either a `TRUE` or `FALSE` value. I personally use this a lot in my work in both preliminary data analysis and data checking. We saw this used above, when we did `is.na(x) == TRUE` to check if the `x` variable was missing. We can see how this translates into a logical variable here:

```
  x <- c(3, 4, 5, 6, 7, NA)

  z<-is.na(x) #check if x is missing

  z
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

```
  class(z)
```

```
[1] "logical"
```

In practice, I use this with the `I()` function (more on this below) to do quick binary codes of a value:

```r
x <- c(3, 4, 5, 6, 7)

table( I(x >= 5) )
```

```
FALSE  TRUE
    2     3
```

4.10 Dataframes

Traditionally, R organizes variables into data frames, these are like a spreadsheet. The columns can have names, and the *dataframe* itself can have data of different types.

Here we make a short data frame with three columns, two numeric and one factor:

```r
mydat <- data.frame(
  x = c(1,2,3,4,5, 6, 7, 8),
  y = c(10, 20, 35, 57, 37, 21, 23, 25),
  group = factor(c("A", "A" ,"A", "B", "B", "C","C","C"))
)

#See the size of the dataframe
dim(mydat)
```

```
[1] 8 3
```

```r
#Open the dataframe in a viewer and just print it
print(mydat)
```

```
  x  y group
1 1 10     A
2 2 20     A
3 3 35     A
4 4 57     B
5 5 37     B
6 6 21     C
7 7 23     C
8 8 25     C
```

4.10.1 Accessing variables in dataframes

R has a few different ways to get a variable from a data set. One way is the `$` notation, used like `dataset$variable`, and another is to provide the column index or name of the variable. These three methods are illustrated below. The first tells R to get the variable named `group` from the data. The second tells R to get the column named `group` from the data, and the third tells R to get the third column from the data.

```
mydat$group
```

```
[1] A A A B B C C C
Levels: A B C
```

```
mydat['group']
```

```
  group
1     A
2     A
3     A
4     B
5     B
6     C
7     C
8     C
```

```
mydat[,3]
```

```
[1] A A A B B C C C
Levels: A B C
```

The `names()` function is very useful for seeing all the column names of a dataset, without having to print any of the data.

```
names(mydat)
```

```
[1] "x"     "y"     "group"
```

R has several useful function for previewing the contents of a dataframe or variable. The `head()` function shows the first 6 observations of a dataframe or variable, and `tail()` shows the last 6 observations. You can also show a custom number of observations by using the `n=` argument in either function. These are illustrated below:

```
head(mydat)
```

```
  x  y group
1 1 10     A
2 2 20     A
3 3 35     A
4 4 57     B
5 5 37     B
6 6 21     C
```

```
head(mydat, n = 2)
```

```
  x  y group
1 1 10     A
2 2 20     A
```

```
head(mydat$group)
```

```
[1] A A A B B C
Levels: A B C
```

```
tail(mydat)
```

```
  x  y group
3 3 35     A
4 4 57     B
5 5 37     B
6 6 21     C
7 7 23     C
8 8 25     C
```

```
tail(mydat, n = 2)
```

Table 4.2: My basic table

x	y	group
1	10	A
2	20	A
3	35	A
4	57	B
5	37	B
6	21	C
7	23	C
8	25	C

```
  x  y group
7 7 23     C
8 8 25     C
```

```
tail(mydat$group)
```

```
[1] A B B C C C
Levels: A B C
```

4.10.2 Nicer looking tables

R can also produce nicely formatted HTML and LaTeX tables. There are several packages that do this, but the **knitr** package has some basic table creation functions that do a good job for simple tables.

```
library(knitr)

kable(mydat,
      caption = "My basic table",
      align = 'c',
      format = "html")
```

```
library(knitr)
kable(mydat,
      caption = "My basic table",
      align = 'c',
      format="latex"  )
```

Much more advanced tables can be created using the `gt` package Iannone, Cheng, and Schloerke (2020), which allows for highly customized tables.

```
library(gt,
        quietly = TRUE)
library(dplyr,
        quietly = TRUE)
```

```
Attaching package: 'dplyr'
```

```
The following objects are masked from 'package:stats':

    filter, lag
```

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
mydat%>%
  gt()%>%
  tab_header(title= "My simple gt table",
             subtitle = "With a subtitle")
```

My simple gt table

With a subtitle

x	y	group
1	10	A
2	20	A
3	35	A
4	57	B
5	37	B
6	21	C
7	23	C
8	25	C

4.11 Real data example

Now let's open a 'real' data file. This is the 2018 World population data sheet from the Population Reference Bureau. It contains summary information on many demographic and population level characteristics of nations around the world in 2018.

I've had this entered into a **Comma Separated Values** file by some poor previous research assistant of mine and it lives happily on Github now for all the world to see. CSV files are a good way to store data coming out of a spreadsheet, because R can read them without any other packages. R can also read Excel files, but it requires external packages to do so, such as `readxl`.

I can read it from Github directly by using a function in the `readr` library, or with the base R function `read.csv()`, both accomplish the same task.

```
prb <- read.csv(file = "https://github.com/coreysparks/r_courses/raw/master/data/2018_WPDS
    stringsAsFactors = TRUE)
```

That's handy. If the file lived on our computer in your working directory, I could read it in like so:

```
prb <- read_csv("path/to/file/2018_WPDS_Data_Table_FINAL.csv")
```

Same result.

The `haven` library Wickham and Miller (2020) can read files from other statistical packages easily, so if you have data in Stata, SAS or SPSS, you can read it into R using those functions, for example, the `read_dta()` function reads Stata files, `read_sav()` to read SPSS data files.

```
library(haven)
prb_stata <- read_dta("path/to/file/prb2018.dta")

prb_spss <- read_sav("path/to/file/prb_2018.sav")
```

4.12 Basic Descriptive analysis of data

One of the key elements of analyzing data is the initial descriptive analysis of it. In subsequent chapters, I will go into more depth about this process, but for now, I want to illustrate some simple but effective commands for summarizing data.

4.12.1 Dataframe summaries

The `summary()` function is very useful both in terms of producing numerical summaries of individual variables, but also for shows summaries of entire dataframes. Its output differs based on the type of variable you give it, for character variables it does not return any summary. For factor variables, it returns a frequency table, and for numeric variables, it returns the five number summary plus the mean.

```
summary(prb$region)
```

```
      CARIBBEAN  CENTRAL AMERICA     CENTRAL ASIA        EAST ASIA
             17                8                5                8
  EASTERN AFRICA   EASTERN EUROPE    MIDDLE AFRICA  NORTHERN AFRICA
             20               10                9                7
NORTHERN AMERICA  NORTHERN EUROPE          OCEANIA    SOUTH AMERICA
              2               11               17               13
     SOUTH ASIA   SOUTHEAST ASIA  SOUTHERN AFRICA  SOUTHERN EUROPE
              9               11                5               15
  WESTERN AFRICA     WESTERN ASIA   WESTERN EUROPE
             16               18                9
```

```
summary(as.factor(prb$continent))
```

```
         AFRICA            ASIA          EUROPE NORTHERN AMERICA
             57              51              45              27
        OCEANIA    SOUTH AMERICA
             17              13
```

```
summary(prb$tfr)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.600   2.300   2.709   3.750   7.200
```

I find this function to be very useful when I'm initially exploring a data set, so I can easily see the min/max values of a variable. There are many alternatives to this base function, including `psych::describe()`, `Hmisc::describe()`, and `skimr::skim()`, all of which produce summaries of dataframes or variables

```
desc1  <-  psych::describe(prb[, 1:8],
                           fast = FALSE)
print(desc1,
      short = TRUE)
```

```
    vars n mean sd median trimmed mad min max range skew kurtosis se
 [ reached 'max' / getOption("max.print") -- omitted 8 rows ]
```

```
desc2 <-  Hmisc::describe(prb[, 1:8],
                          tabular= FALSE)
head(desc2)
```

```
desc3 <- skimr::skim(prb[, 1:8])
desc3
```

The `skimr::skim()` function is very good at doing summaries of both numeric and categorical data, while the other functions are perhaps best suited to numeric data.

The `summary()` function, as well as the other three functions in other packages can be used on a single variable within a dataframe as well, or on a simple vector:

```
summary(prb$tfr)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.600   2.300   2.709   3.750   7.200
```

```
summary(zf)
```

```
1 2 3 4
1 1 1 1
```

From this summary, we see that the mean is 2.7085714, there is one country missing the Total fertility rate variable. The minimum is 1 and the maximum is 7.2 children per woman.

4.12.2 Frequency tables

A basic exploration of data, especially if your data have categorical or nominal variables, includes the extensive use of frequency tables. If you're simply looking at the number of observations in each level of a categorical variable, or using frequency tables to aggregate data, they are some of the most useful basic statistical summaries around. The basic function for constructing simple tables is `table()` in base R. More sophisticated table construction is allowed in `xtabs()`

Let's have a look at some descriptive information about the data:

```
#Frequency Table of # of Countries by Continent
table(prb$continent)
```

```
        AFRICA           ASIA         EUROPE NORTHERN AMERICA
            57             51             45             27
       OCEANIA  SOUTH AMERICA
            17             13
```

Frequency of TFR over 3 by continent:

```
table(I(prb$tfr > 3),
      prb$continent)
```

```
       AFRICA ASIA EUROPE NORTHERN AMERICA OCEANIA SOUTH AMERICA
FALSE      11   40     45               27       7            12
TRUE       46   11      0                0      10             1
```

Two things to notice in the above code, first we have to use the `$` operator to extract each variable from the `prb` dataframe. Second, the `I()` operator is used. This is honestly one of my favorite things in base R. `I()` is the indicator function, it evaluates to `TRUE` or `FALSE` depending on the argument inside of it. This also allows for fast construction of binary variables on the fly in any function. Here's another example:

```
x <- c(1, 3, 4, 5, 7, 19)
I(x > 5)
```

```
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE
```

```
table(I(x > 5))
```

```
FALSE  TRUE
    4     2
```

So we see how this works, I checks if `x` is greater than 5, if it is, `I()` returns `TRUE`. When we feed this to `table()`, we can count up the `TRUE` and `FALSE` responses.

Later in the book, we will see how to employ the `xtabs()` function to quickly aggregate data from individual level to aggregate level.

4.12.3 More basic statistical summaries

Now, we will cover some basic descriptive statistical analysis including basic measures of central tendency and variability.

4.12.4 Measures of central tendency

We can use graphical methods to describe what data 'look like' in a visual sense, but graphical methods are rarely useful for comparative purposes. In order to make comparisons, you need to rely on a numerical summary of data vs. a graphical one.

Numerical measures tell us a lot about the form of a distribution without resorting to graphical methods. The first kind of summary statistics we will see are those related to the measure of *central tendency*. Measures of central tendency tell us about the central part of the distribution

4.12.5 Mean and median

Here is an example from the PRB data.

```
mean(prb$tfr)
```

```
[1] 2.708571
```

Whoops! What happened? This means that R can't calculate the mean because there's a missing value, which we saw before. We can tell R to automatically remove missing values by:

```
mean(prb$tfr,
     na.rm = TRUE)
```

```
[1] 2.708571
```

Which works without an error. Many R functions will fail, or do listwise deletion of observations when NAs are present, so it's best to look at the documentation for the function you're wanting to use to see what it's default na action is. The `mean()` function defaults to `na.rm = FALSE`, which indicates that it does not remove missing values by default.

We can also calculate the median TFR

```
median(prb$tfr,
       na.rm = TRUE)
```

```
[1] 2.3
```

4.12.6 Measures of variation

One typical set of descriptive statistics that is very frequently used is the so-called **five number summary** and it consists of : the Minimum, lower quartile, median, upper quartile and maximum values. This is often useful if the data are not symmetric or skewed. This is what you get when you use the `fivenum()` function, or we can include the mean if we use the `summary()` function.

```
fivenum(prb$tfr)
```

```
[1] 1.0 1.6 2.3 3.8 7.2
```

```
summary(prb$tfr)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.600   2.300   2.709   3.750   7.200
```

4.12.6.1 Variance

To calculate the variance and standard deviation of a variable:

```
var(prb$tfr,
   na.rm = TRUE) #variance
```

```
[1] 1.806338
```

```
sd(prb$tfr,
   na.rm = TRUE) #standard deviation
```

```
[1] 1.344001
```

```
sqrt(var(prb$tfr)) #same as using sd()
```

```
[1] 1.344001
```

The above sections have shown some basic ways to summarize data in R, along with many handy functions that are pervasive in my own general work flow. Is this everything R will do, No. Are these the only way to do things in R? Never. I'm constantly marveled at how many new functions I see my students using in their own work and this reminds me how much of the R ecosystem I have yet to explore, even after twenty-plus years of using it.

4.13 The tidyverse

So far, most of the functions I have discussed have been from the base R ecosystem, with some specific functions from other downloadable packages. One of the biggest changes to R in recent years has been the explosion in popularity of the **tidyverse** Wickham et al. (2019). The tidyverse is a large collection of related packages that share a common philosophy of how data and programming relate to one another and work together to produce a more streamlined, literate way of programming with data.

To get the core parts of the tidyverse, install it using `install.packages("tidyverse")` in your R session. This will install the core components of the tidyverse that can then be used throughout the rest of the book [1].

Two of the workhorses in the tidyverse are the packages `dplyr` Wickham et al. (2020) and `ggplot2` Wickham (2016). The `dplyr` package is very thoroughly described in the book *R for Data Science* Wickham and Grolemund (2017), and the `ggplot2` package also has a book-length description in the book *ggplot2: Elegant Graphics for Data Analysis* Wickham (2016),

[1] If you followed the script at the beginning of this chapter, the tidyverse will already be installed.

so I won't waste time and space here with complete descriptions. Instead, I will show some pragmatic examples of how these work in my own work flow, and also use these packages together to produce some descriptive data visualizations.

4.13.1 Basic dplyr

The `dplyr` package has many functions that work together to produce succinct, readable and highly functional code. I often say about base R packages in comparison to things like SAS, that I can do something in R in about 10 lines of code compared to 50 in SAS. Using dplyr, you can do even more, faster.

The package consists of core "verbs" that are used to clean, reshape, and summarize data. Using "pipes", the user can chain these verbs together so that you only have to name the data being used once, which makes for more efficient code, since you're not constantly having to name the dataframe. The pipes also allow for all variables within a dataframe to be accessed, without using the `$` or `[]` notation described earlier in this chapter.

Perhaps a short tour of using dplyr would be good at this point, and we will see it used throughout the book. In the following code, I will use the `prb` data from earlier, and I will do a series of tasks. First, I will create a new variable using the `mutate()` function, then group the data into groups (similar to SAS's 'by' processing) , then do some statistical summaries of other variables using the `summarise()` function.

Here we go:

```
library(dplyr)

prb %>%
  mutate(high_tfr = ifelse(test = tfr > 3,
                           yes =  "high",
                           no =  "low") )%>%
  group_by(high_tfr) %>%
  summarise(mean_e0 = mean(e0male, na.rm = TRUE))
```

```
# A tibble: 2 x 2
  high_tfr mean_e0
  <chr>      <dbl>
1 high        62.8
2 low         73.6
```

The `prb%>%` line says, take the prb data and feed it into the next verb using the pipe. The next line `mutate(high_tfr = ifelse(test = tfr > 3,yes =  "high", no =  "low")`

`)%>%` tells R to create a new variable called `high_tfr`, the value of the variable will be created based on conditional logic. If the value of the tfr is over 3, the value will be `"high"` and if the value of the tfr is less than 3, the value of the variable will be `"low"`.

The `group_by(high_tfr)%>%` line tells R to form a "grouped data frame", basically this is how `dplyr` segments data into discrete groups, based off a variable, and then performs operations on those groups. This is the same thing as stratification of data.

The final command `summarise(mean_e0 = mean(e0male, na.rm = TRUE))` tells R to take the mean of the `e0male` variable, in this case it will be calculated for each of the `high_tfr` groups.

Finally, we `ungroup()` the dataframe to remove the grouping, this is customary whenever using the `group_by()` verb.

We can also summarize multiple variables at the same time using the `across()` command. In the code below, I find the mean (specified by `.fns = mean`) for each of the four variables `e0male`, `e0female`, `gnigdp` and `imr` for each of the `high_tfr` groups.

```
prb %>%
  mutate(high_tfr = ifelse(test = tfr > 3,
                           yes =  "high",
                           no =  "low") )%>%
  group_by(high_tfr) %>%
  summarise(n = n(),
            across(.cols = c(e0male, e0female, gnigdp, imr),
                   .fns = mean,
                   na.rm = TRUE))%>%
  ungroup()
```

```
Warning: There was 1 warning in `summarise()`.
i In argument: `across(...)`.
i In group 1: `high_tfr = "high"`.
Caused by warning:
! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
Supply arguments directly to `.fns` through an anonymous function instead.

  # Previously
  across(a:b, mean, na.rm = TRUE)

  # Now
  across(a:b, \(x) mean(x, na.rm = TRUE))

# A tibble: 2 x 6
```

```
  high_tfr     n e0male e0female gnigdp   imr
  <chr>    <int>  <dbl>    <dbl>  <dbl> <dbl>
1 high        68   62.8     66.4  5329.  43.2
2 low        142   73.6     78.9 27216.  11.9
```

The line `summarise(n=n() , across(.cols = c(e0male, e0female, gnigdp, imr), .fns = mean, na.rm = TRUE))` tells R to first count the number of cases in each group `n = n()`, then summarize multiple variables, in this case male and female life expectancy at birth, GDP, and the infant mortality rate, by each of the levels of the `high_tfr` variable. The summary I want to do is the mean of each variable, being sure to remove missing values before calculating the mean.

We see then the estimates of the four other indicators for countries that have TFR over 3, versus countries with a TFR under 3.

This is a basic `dplyr` use, but it is far from what the package can do. Throughout the rest of the book, this process will be used to do calculations, aggregate data, present model results and produce graphics. This example was trying to show a simple workflow in dplyr, and introduce the pipe concept.

Next, we will explore some basic uses of `dplyr` in conjunction with the `ggplot2` package.

4.14 Basic ggplot

Let's say that we want to compare the distributions of income from the above examples graphically. Since the `ggplot2` library is part of the tidyverse, it integrates directly with dplyr and we can do plots within pipes too.

In generally, `ggplot()` has a few core statements.

1) `ggplot()` statement - This tells R the data and the basic aesthetic that will be plotted, think x and y axis of a graph. The aesthetic is defined using the `aes()` function. This is where you pass values to be plotted to the plot device.
2) Define the geometries you want to use to plot your data, there are many types of plots you can do, some are more appropriate for certain types of data
3) Plot annotations - Titles, labels etc. This allows you to customize the plot with more information to make it more easily understandable.

Now I will illustrate some basic ggplot examples, and I'm going to use the PRB data that I have been using for other examples. In order to better illustrate the code, I will walk through a *very* minimal example, line by line.

`library(ggplot2)` Loads the ggplot package

`ggplot(data = prb, mapping = aes(x = tfr))+` Use the ggplot function, on the prb dataframe. The variable we are plotting is the total fertility rate, `tfr`. In this case, it is the only variable we are using. I include a `+` at the end of the line to tell R that more elements of the plot are going to be added.

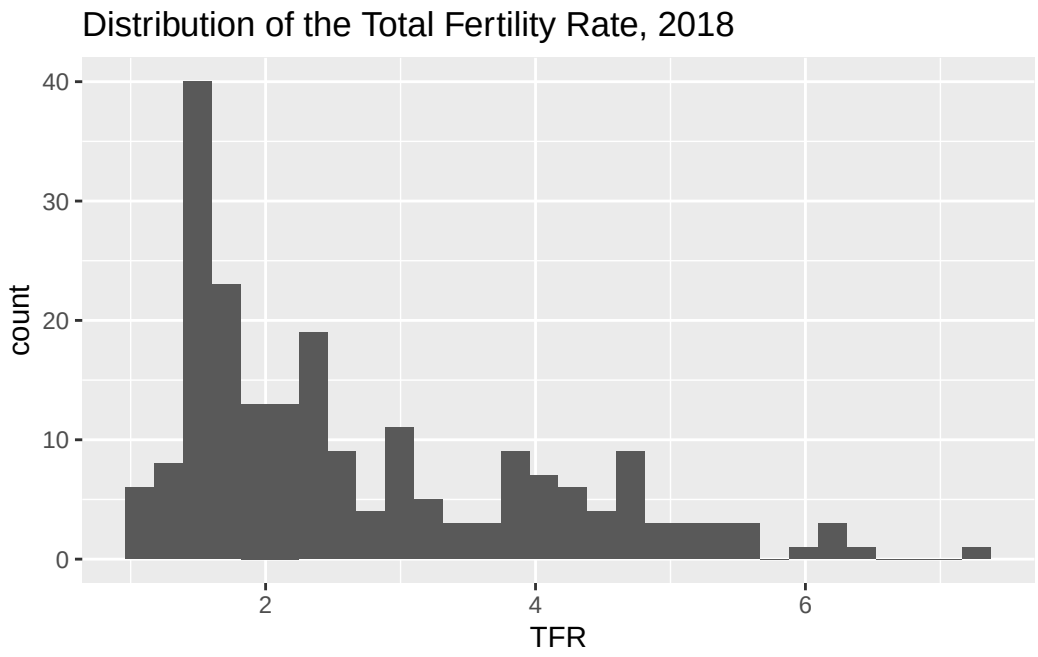
`geom_histogram()+` Tells R that the geometry we are using is a histogram, again we have the `+` at the end of the line to indicate that we will add something else to the plot, in this case a title.

`ggtitle(label = "Distribution of the Total Fertility Rate, 2018")` Tells R the primary title for the plot, which describes what is being plotted. I'm also going to add an additional annotation to the x-axis to indicate that it is showing the distribution of the TFR:

`xlab(label = "TFR")`

Now, let's see all of this together:

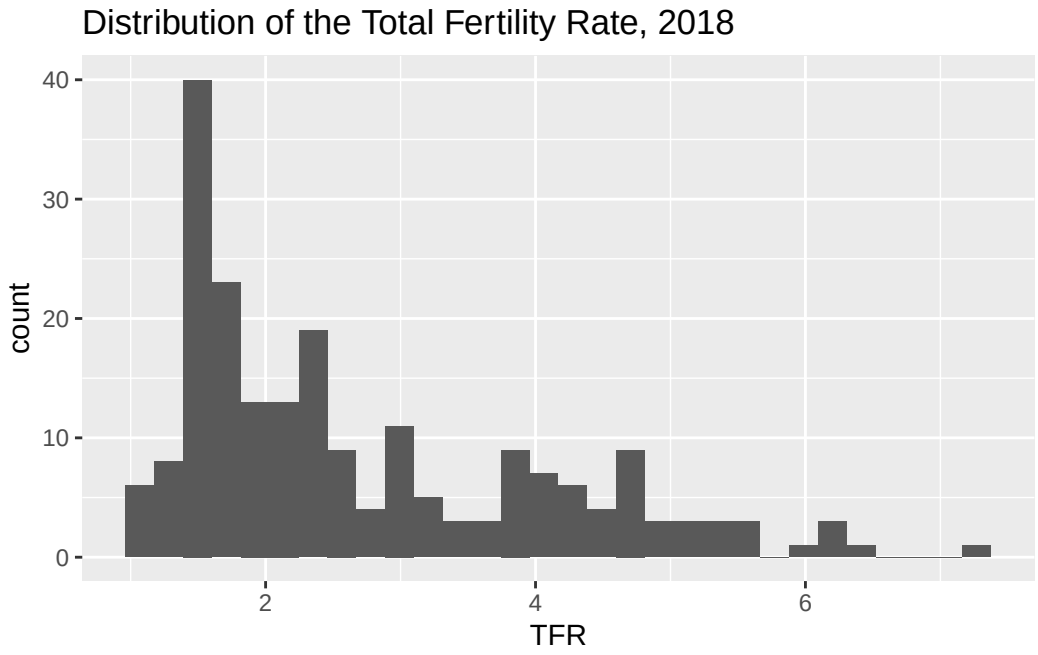
```
library(ggplot2)

ggplot(data=prb,
       mapping=aes(x = tfr))+
  geom_histogram()+
  ggtitle(label = "Distribution of the Total Fertility Rate, 2018")+
  xlab(label = "TFR")
```



The above example named the data frame explicitly in the `ggplot()` call, but we can also use `dplyr` to pipe data into the plot:

```
prb%>%
  ggplot(mapping=aes(x = tfr))+
  geom_histogram()+
  ggtitle(label = "Distribution of the Total Fertility Rate, 2018")+
  xlab(label = "TFR")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



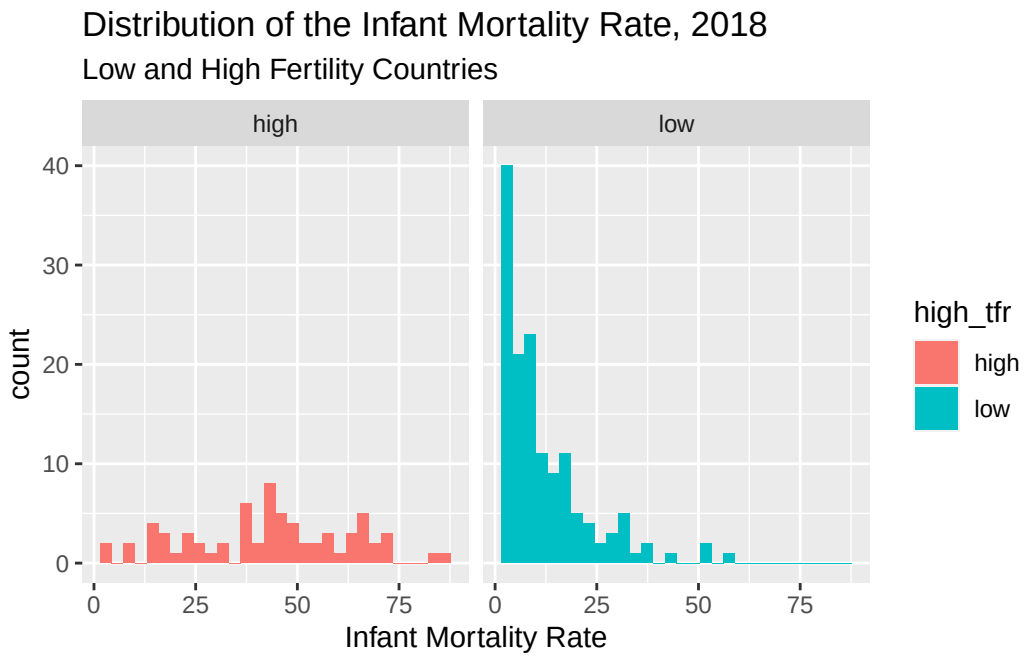
We can likewise incorporate a `dplyr` workflow directly into our plotting, using the example from before, we will create histograms for the high and low fertility groups using the `facet_wrap()` function.

```
prb%>%
  mutate(high_tfr = ifelse(test = tfr > 3,
                           yes = "high",
                           no = "low") )%>%
  group_by(high_tfr)%>%
  ggplot(mapping=aes(x = imr))+
```

```
    geom_histogram(aes( fill = high_tfr))+
    facet_wrap( ~ high_tfr)+
    ggtitle(label = "Distribution of the Infant Mortality Rate, 2018",
            subtitle = "Low and High Fertility Countries")+
    xlab(label = "Infant Mortality Rate")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
Warning: Removed 1 rows containing non-finite values (`stat_bin()`).
```



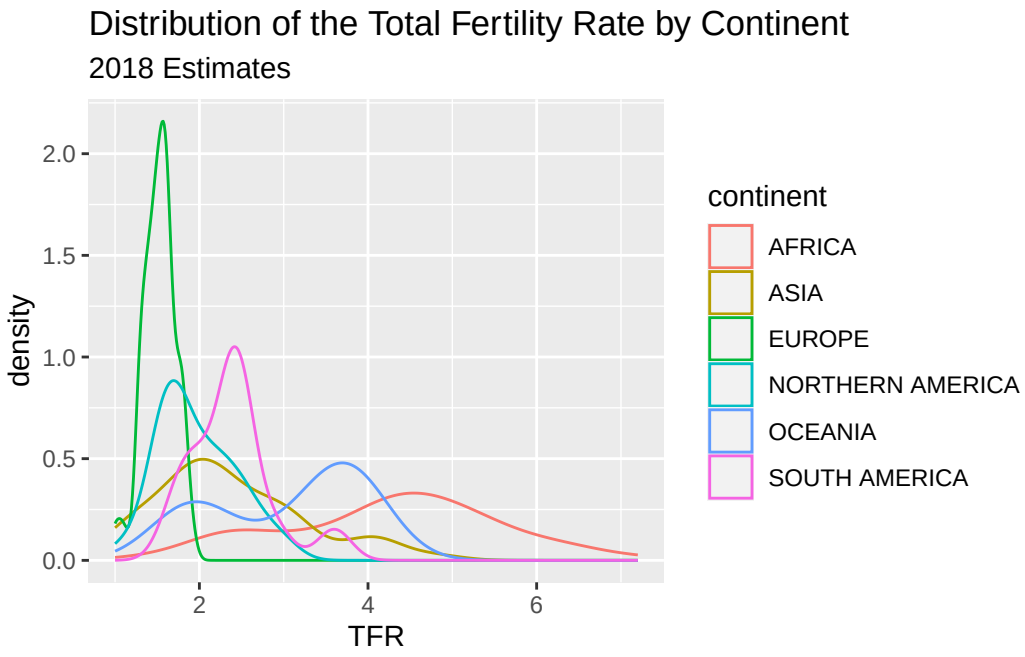
You also notice that I used the `aes(fill = high_tfr)` to tell R to color the histogram bars according to the variable `high_tfr`. The `aes()` function allows you to modify colors, line types, and fills based of values of a variable.

Another way to display the distribution of a variable is to use `geom_density()` which calculates the kernel density of a variable. Again, I use a variable, this time the continent a country is on, to color the lines for the plot.

```
prb%>%
ggplot(mapping = aes(tfr,
                     colour = continent,
```

```
                    stat = ..density..))+
  geom_density()+
  ggtitle(label = "Distribution of the Total Fertility Rate by Continent",
          subtitle = "2018 Estimates")+
  xlab(label = "TFR")
```

```
Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(density)` instead.
```

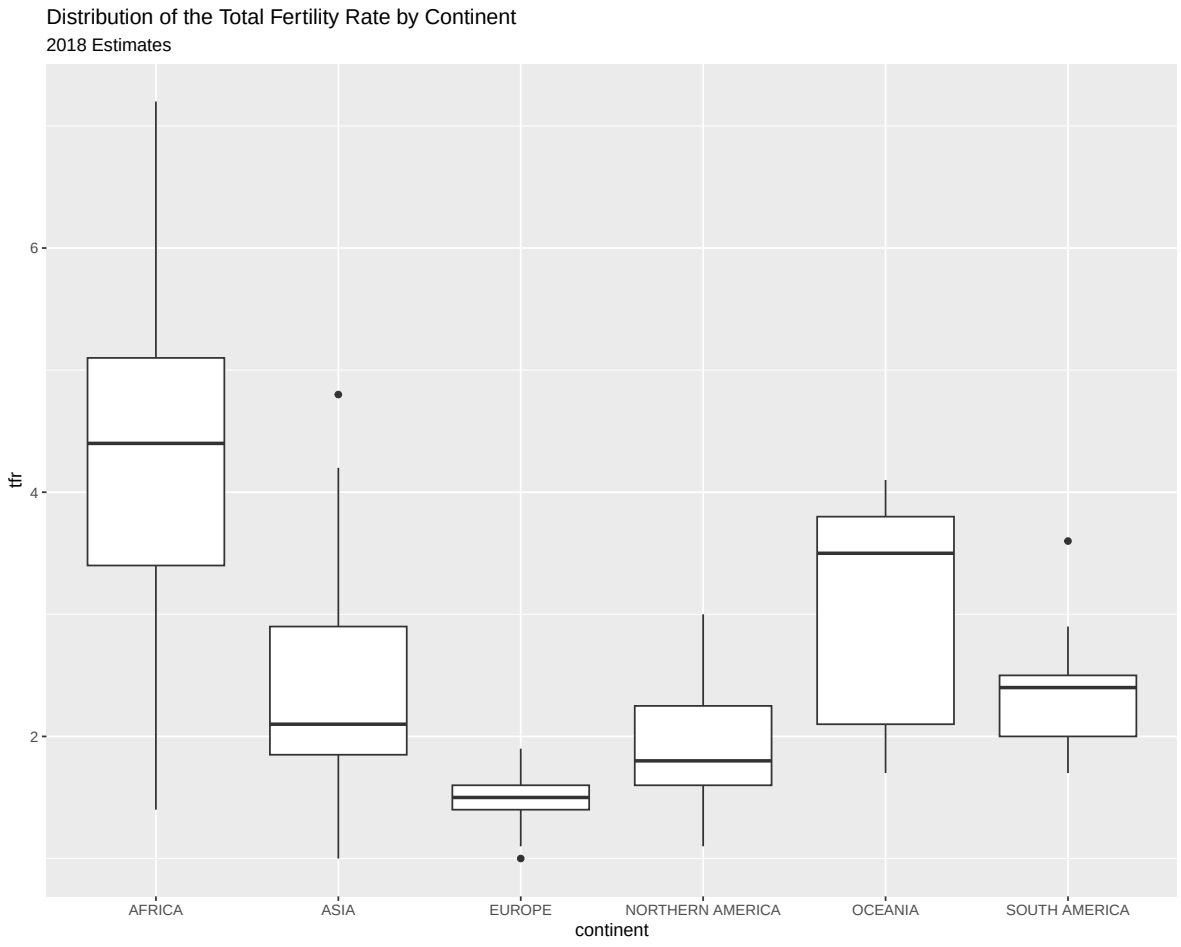


4.14.1 Stem and leaf plots/Box and Whisker plots

Another visualization method is the stem and leaf plot, or box and whisker plot. This is useful when you have a continuous variable you want to display the distribution of across levels of a categorical variable. This is basically a graphical display of Tukey's 5 number summary of data.

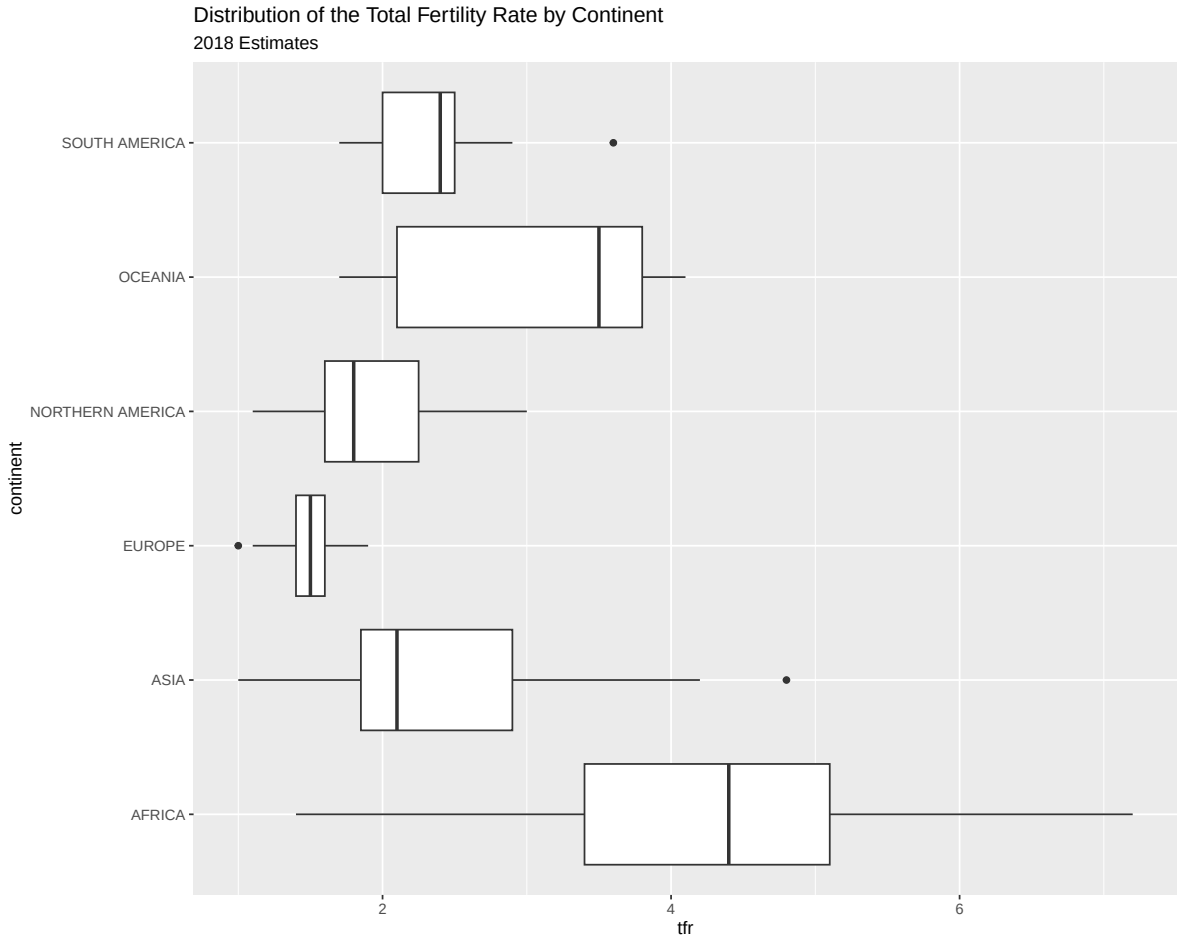
```
prb%>%
  ggplot( mapping = aes(x = continent, y = tfr))+
  geom_boxplot()+
  ggtitle(label = "Distribution of the Total Fertility Rate by Continent",
```

```
          subtitle = "2018 Estimates")
```



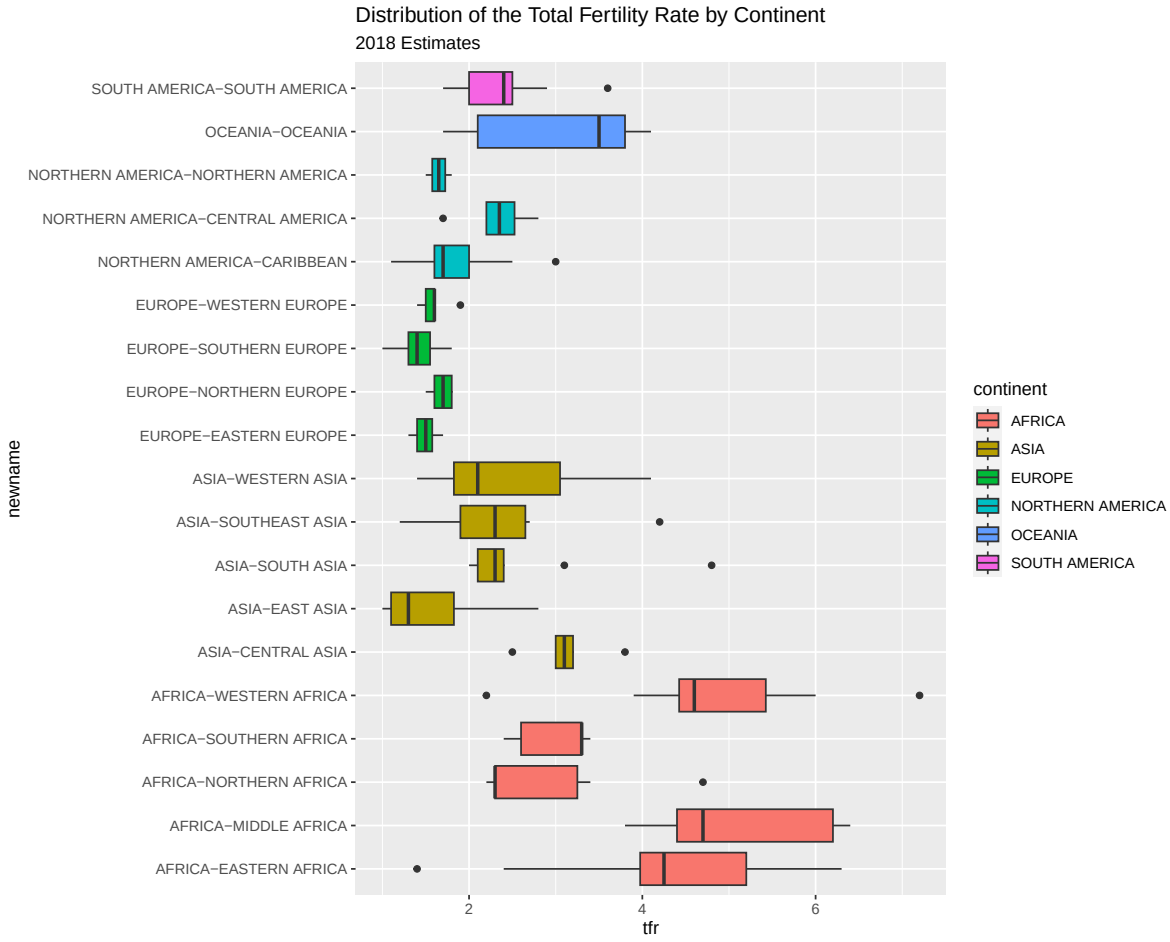
You can flip the axes, by adding `coord_flip()`

```
prb%>%
ggplot( mapping = aes( x = continent,
                       y = tfr))+
  geom_boxplot()+
  ggtitle(label = "Distribution of the Total Fertility Rate by Continent",
          subtitle = "2018 Estimates")+
  coord_flip()
```



You can also color the boxes by a variable, Here, I will make a new variable that is the combination of the continent variable with the region variable, using the `paste()` function. It's useful for combining values of two strings.

```
prb%>%
  mutate(newname = paste(continent, region, sep = "-"))%>%
  ggplot(aes(x = newname,
             y = tfr,
             fill = continent))+
  geom_boxplot()+
  coord_flip()+
  ggtitle(label = "Distribution of the Total Fertility Rate by Continent",
          subtitle = "2018 Estimates")
```



4.14.2 X-Y Scatter plots

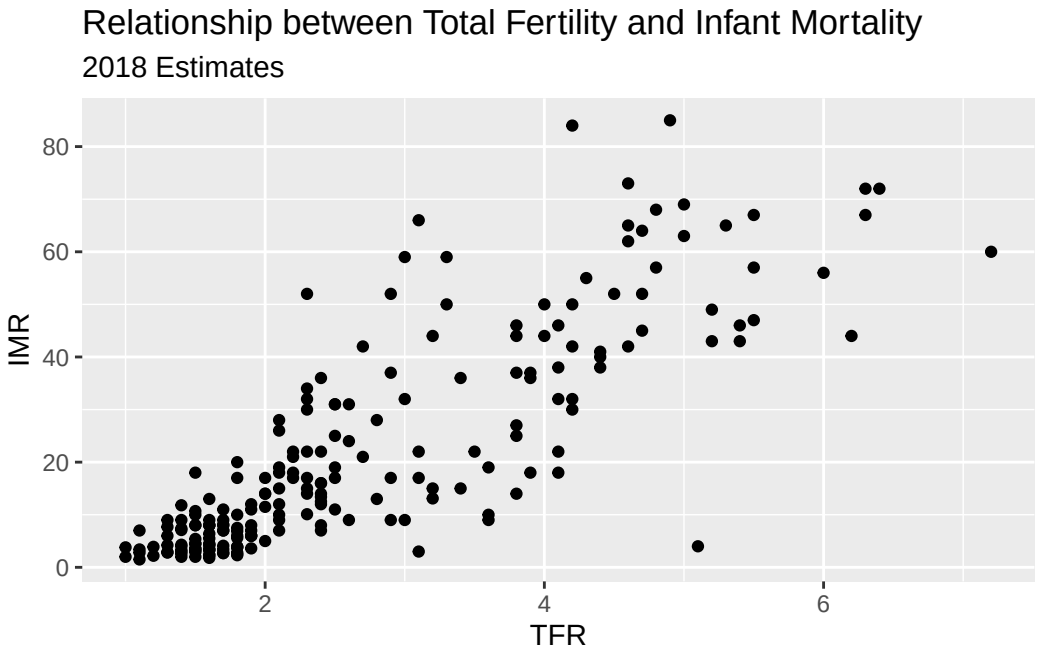
These are useful for finding relationships among two or more continuous variables. `ggplot()` can really make these pretty. The `geom_point()` geometry adds points to the plot.

Here are a few riffs using the PRB data:

```
prb%>%
ggplot(mapping= aes(x = tfr,
                    y = imr))+
  geom_point()+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality",
          subtitle = "2018 Estimates")+
  xlab(label = "TFR")+
```

```
  ylab(label = "IMR")
```

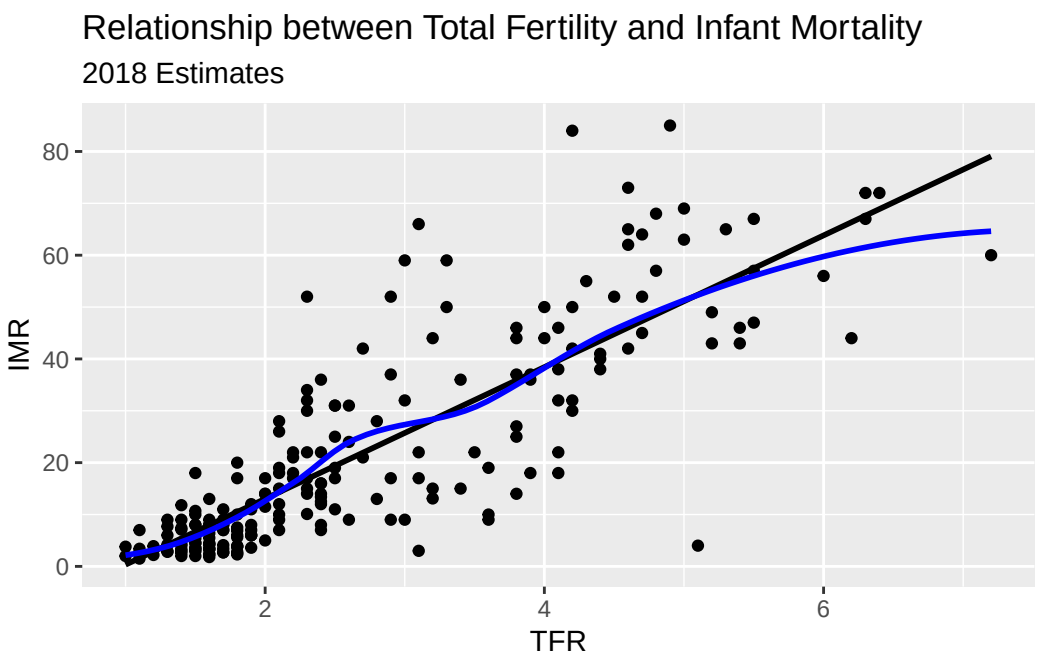
```
Warning: Removed 1 rows containing missing values (`geom_point()`).
```



R also makes it easy to overlay linear and spline smoothers for the data (more on splines later).

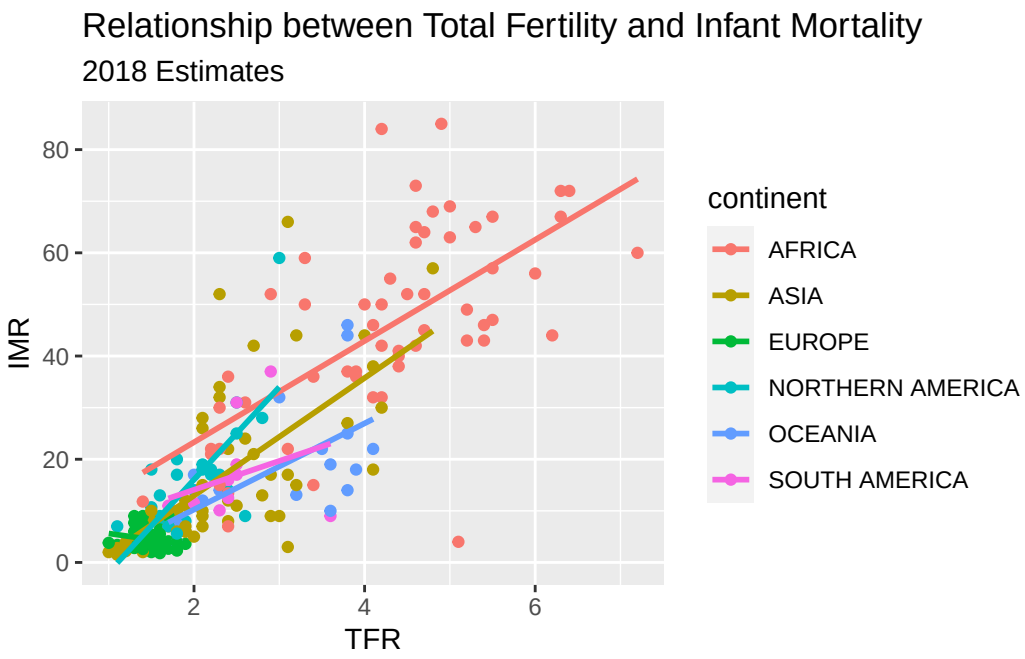
```
prb%>%
ggplot(mapping = aes(x = tfr,
                     y = imr))+
  geom_point()+
  geom_smooth(method = "lm",
              color = "black",
              se = F)+ #linear regression fit
  geom_smooth(color = "blue",
              method = "loess",
              se = FALSE)+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality",
          subtitle = "2018 Estimates")+
  xlab(label = "TFR")+
```

```
  ylab(label = "IMR")
```



Now we color the points by continent

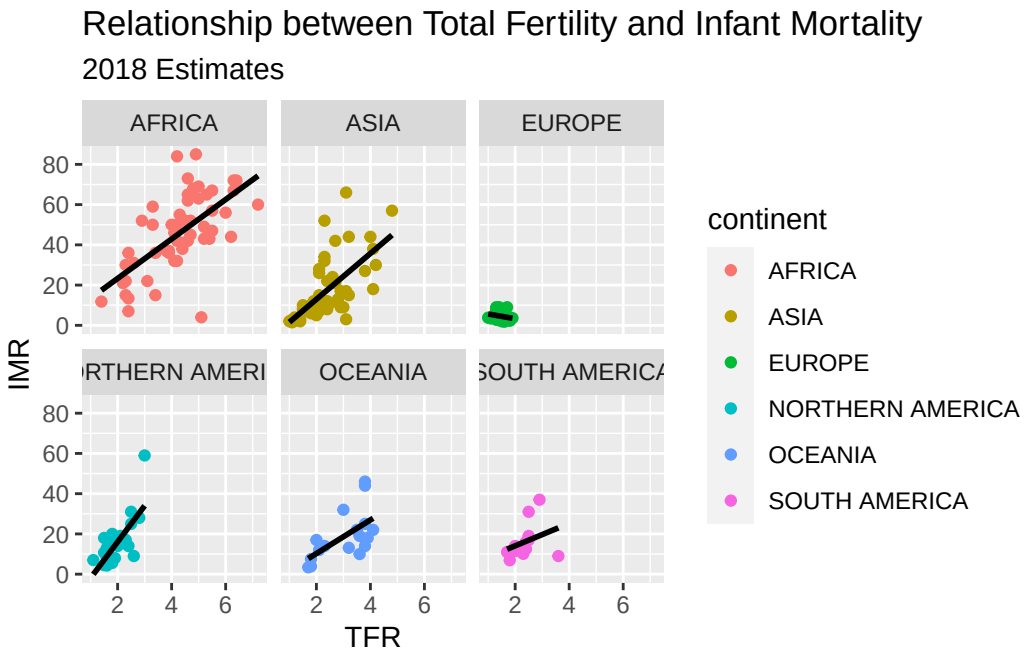
```
prb%>%
ggplot(mapping = aes(x = tfr,
                     y = imr,
                     color =continent))+
  geom_point()+
  geom_smooth(method = "lm",
              se = FALSE)+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality",
          subtitle = "2018 Estimates")+
  xlab(label = "TFR")+
  ylab(label = "IMR")
```



4.14.3 Facet plots

Facet plots are nice, they allow you to create a plot separately based on a grouping variable. This allows you to visualize if the relationship is constant across those groups. Here, I repeat the plot above, but I facet on the continent, and include the regression line for each continent.

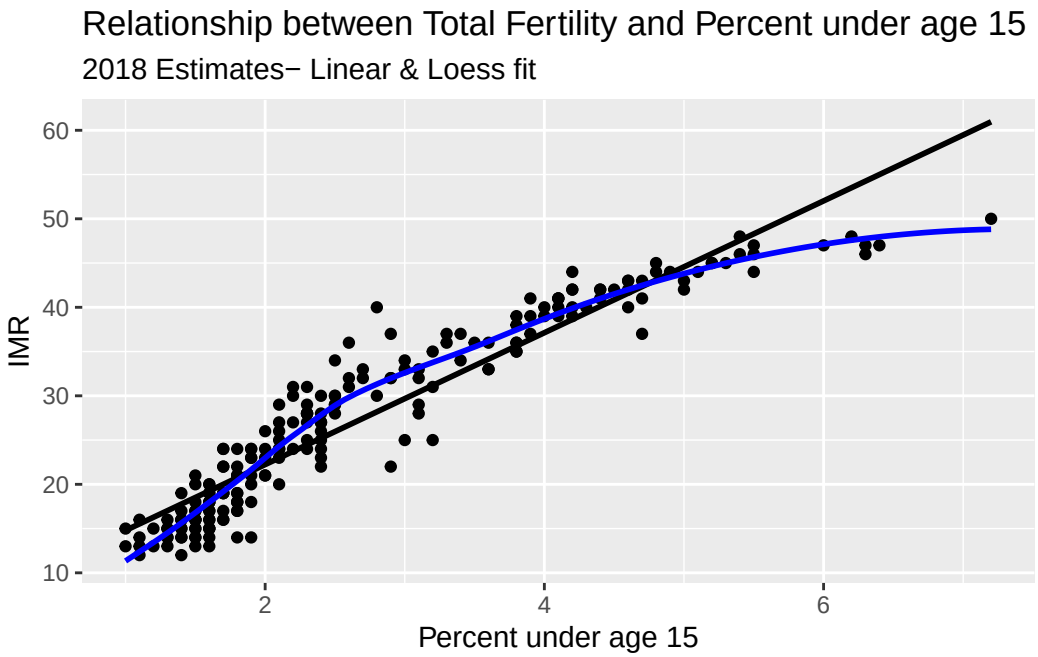
```
prb%>%
ggplot(mapping= aes(x = tfr,
                    y = imr,
                    color = continent))+
  geom_point()+
  geom_smooth(method = "lm",
              se = FALSE,
              color = "black")+
  facet_wrap( ~ continent)+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality",
          subtitle = "2018 Estimates")+
  xlab(label = "TFR")+
  ylab(label = "IMR")
```



Another example, this time of a bad linear plot! `ggplot` makes it easy to examine if a relationship is linear or curvilinear, at least visually.

```
ggplot(data = prb,mapping = aes(x = tfr, y = pctlt15_2018))+
  geom_point()+
  geom_smooth( method = "lm",
               se = FALSE,
               color = "black")+
  geom_smooth( method = "loess",
               se = FALSE,
               color = "blue")+
  ggtitle(label = "Relationship between Total Fertility and Percent under age 15",
          subtitle = "2018 Estimates- Linear & Loess fit")+
  xlab(label = "Percent under age 15")+
  ylab(label = "IMR")
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```



4.15 Chapter summary

In this chapter, I have introduced R and Rstudio and some basic uses of the software for accessing data and estimating some summary statistics. The R ecosystem is large and complex, and the goal of this book is to show you, the user, how to use R for analyzing data from demographic data sources. In the chapters that follow, I will show how to use R within two large universes of data, the macro and the micro. The *macro* level sections will focus on using R on data that come primarily from places - nations, regions, administrative areas. The *micro* level sections will focus on analyzing complex survey data on individual responses to demographic surveys. The final section will discuss approaches that merge these two levels into a multi-level framework and describe how such models are estimated and applied.

4.16 References

5 Analysis of Survey Data

6 Survey Data

6.1 Demographic Survey data

The majority of demographic research relies on two or three main sources of information. First among these are population enumerations or censuses, followed by vital registration data on births and deaths and last but not least, data from surveys. Censuses and other population enumerations are typically undertaken by federal statistical agencies and demographers use this data once it's disseminated from these agencies. Similarly, vital registration data are usually collected by governmental agencies, who oversee the collection and data quality for the data. Survey data on the other hand can come from a wide variety of sources.

It's not uncommon for us to go and collect our own survey data specific to a research project we have, typically on a specialized population that we are interested in learning about, but surveys can also be quite general in their scope and collect information on a wide variety of subjects. Owing to the mix of small and large-scale survey data collection efforts, survey data are often available on many different topics, locales and time periods. Of course we as demographers are typically interested in population-level analysis or generalization from our work, so the survey data we try to use are collected in rigorous manners, with much attention and forethought paid to ensure the data we collect can actually be representative of the *target population* we are trying to describe.

In this chapter, I will introduce the nature of survey sampling as is often used in demographic data sources, and describe what to look for when first using a survey data source for you research. These topics are geared towards researchers and students who have not worked with survey data much in the past and will go over some very pragmatic things to keep in mind. Following this discussion, I will use a specific example from the US Census Bureau's American Community Survey and illustrate how to apply these principals to this specific source. The final goal of this chapter is to show how to use R to analyze survey data and produce useful summaries from our surveys, both tabular and graphically.

My goal in this chapter is to introduce you to common statistical sampling terms and principles, and show how to analyze complex survey design data using R. I hope that the general workflow I use in this chapter allows you to identify the key elements of your survey data source, use R to begin analyzing your data.

6.2 Basics of survey sampling

To begin this section, I want to go over some of the simple terms from sampling that are very important to those of us who rely on survey data for our work. For many of the concepts from this chapter, I strongly recommend Lohr (2019) for the theoretical portions and Lumley (2010) for discussion of how R is used for complex survey data.

The **target population** is the population that our survey has been designed to measure. For large national surveys, these are typically the population of the country of interest. For example, the Demographic and Health Survey (DHS) has it's primary target population as women of childbearing ages in women of reproductive age and their young children living in households. Our **observational units** are the level at which we are collecting data, for surveys this is typically a person or a household, and our survey documentation will tell us what its unit of observation is. **Sampling Units** refer to the units that can serve for us to collect data from, for example we may not have a list of every school age child, but we may have a list of schools, so we may use schools as our sampling units and sample children within them. The **sampling frame** is the set of sampling units containing distinct sets of population members, this is usually the most recent population census, ideally the entire population, or following our school example from above, the entire listing of schools.

These terms are ubiquitous in sampling, but other terminology also exists in many surveys and these terms relate to the nature of how the survey was actually carried out. Many times the surveys we end up using are not themselves **simple random samples**, but are instead some blend of stratified or cluster sample. Simple random samples are the basis for much of statistical insight, and most methods in statistics assume your data are drawn from a population at random. Certainly, methods for collecting random samples exist, such as random digit dialing for phone samples, but often times these samples themselves are not truly random, they are stratified by a geographic area or by type of phone (mobile vs land line). For example, the DHS uses a stratified, cluster sample to collect its information. **Strata** refer to relatively homogeneous areas within the place we are trying to collect data. In the DHS, these are typically rural or urban areas of a country, as identified by the census. Within each strata, the DHS will choose **clusters** from which to sample from, this is a **two-stage sampling method**, where first the sampling frame is stratified, then clusters are selected. Clusters in the DHS are usually neighborhoods in urban areas and smaller towns or villages in rural areas.

Figure 1 shows a cartoon of how this process works, with multiple potential cluster that can be sampled (boxes), and within the cluster are our observational units, some of which are sampled, and some of which are unsampled.

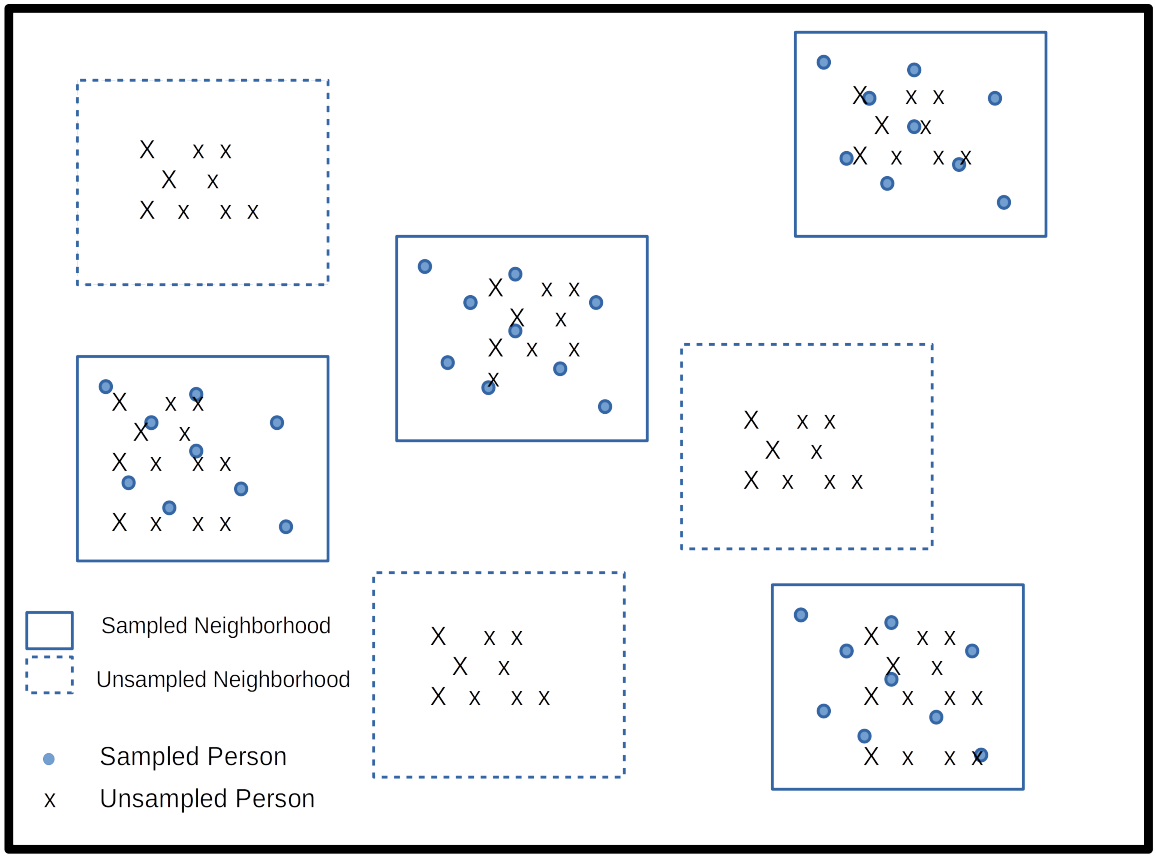


Figure 6.1: Figure 1

6.3 Simple versus complex survey designs

How the data we're using is sampled has a major implication for how we analyze it. The majority of statistical tools assume that data come from simple random samples, because most methods assume independence of observations, regardless of which distribution or test statistic you are using. Violations of this assumption are a big problem when we go to analyze our data, because the non-independence of survey data are automatically in violation of a key assumption of any test. The stratified and clustered nature of many survey samples may also present problems for methods such as linear regression analysis which assume errors in the model are **homoskedastic**, or constant. When data are collected in a stratified or clustered method, the data may have less variation than a simple random sample, because individuals who live closely to one another often share other characteristics in common as well. Our statistical models don't do well with this type of reduction in variation and we often have to resort to manipulations of our model parameters or standard errors of our statistics in order to make them coincide with how the data were collected.

Not to fear! Data collected using public funds are typically required to be made available to the public with information on how to use them. Most surveys come with some kind of code book or user manual which describes how the data were collected and how you should go about using them. In these cases, it pays to read the manual because it will tell you the names of the stratification and clustering variables in the survey data. This will allow you to use the design of the survey in your analysis so that your statistical routines are corrected for the non-randomness and homogeneity in the survey data.

He's not heavy, he's my brother

Another important aspect of survey data are the use of weighting variables. Whenever we design a survey, we have our target population, or universe of respondents in mind. In the DHS, again, this is traditionally[1] women of childbearing age and their children (International 2012). When we collect a sample from this population, or sample may be, and typically is, imperfect. It is imperfect for many reasons, owing to the difficulty of sampling some members of the population, or their unwillingness to participate in our study. Part of designing an effective survey is knowing your universe or population, and its characteristics. This will let you know the probability of a particular person being in the sample. Of course, the more complicated the survey, the more complicated it is to know what this probability is. For example, if we were to sample people in the United States, using a stratified design based on rural and urban residence, we would need to know how many people lived in rural and urban areas within the country, as this would effect the probability of sampling a person in each type of area. This **inclusion probability** tells us how likely a given person is of being sampled. The inverse of the inclusion probability is called the **sampling weight**:

$w_i = \frac{1}{\pi_i}$

[1]The modern DHS collects information on couples, men and children, so the universe has been expanded away from just women of childbearing age and their children.

where π_i is the inclusion probability.

Sampling weights are what we use to make our analyses of a survey representative of the larger population. They serve many purposes including unequal inclusion probabilities, differences in sample characteristics compared to the larger population, and differences in response rates across sample subgroups. All of these situations make the sample deviate from the population by affecting who the actual respondents included in the survey are. Differences in our sample when compared to the larger population can affect most all of our statistical analysis since again, most methods assume random sampling. The weights that are included in public data are the result of a rigorous process conducted by those who designed and implemented the survey itself, and most surveys in their user manuals or code books describe the process of how the weights are created. For example, the US Center for Disease Control and Prevention's Behavioral Risk Factor Surveillance System (BRFSS) provides a very thorough description of how their final person weights are calculated (CDC 2020). These weights include three primary factors, the *stratum weight*, which is a combination of the number of records in a sample strata and the density of phone lines in a given strata, combined with the number of phones in a sampled household and the number of adults in the household to produce the final design weight. These weights are then **raked** to eight different marginal totals, based on age, race/ethnicity, education, marital status, home ownership, gender by race/ethnicity, age by race/ethnicity and phone ownership(CDC 2020). After this process, weights are interpretable as the number of people a given respondent in the survey represents in the population. So, if a respondent's weight in the survey data is 100, they actually represent 100 people in the target population.

Other types of weights also exist, and are commonly seen in federal data sources. A common kind of weight that includes information on both the probability of inclusion **AND** the stratified design of the survey are **replicate weights**. Replicate weights are multiple weights for each respondent, and there are as many weights as there are different levels of the stratification variable. Later in this chapter, we will discuss how replicate weights are used, as compared to single design weights in an example.

6.4 Characteristics of YOUR survey

Survey data that come from reputable sources, such as most federal agencies or repositories such as the Inter-university Consortium for Political and Social Research (ICPSR) at the University of Michigan in the United States, are accompanied by descriptions of the data source including when and where it was collected, what it's target population is, and information on the design of the survey. This will include information on sample design, such as stratum or cluster variables, and design or replicate weights to be used when you conduct your analysis. I cannot stress enough that learning how your particular survey data source is designed, and how the designers recommend you use provided survey variables for your analysis, is imperative to ensure your analysis is correctly specified.

6.5 Example from the American Community Survey

Let's look at an example of these ideas in a real data source. Throughout the book I will use several complex survey design data sources to illustrate various topics, in this chapter I will use data from the US Census Bureau's American Community Survey (ACS) public use microdata sample (PUMS). We can actually use the `tidycensus` package (Walker and Herman 2021) to download ACS PUMS directly from the Census Bureau.

This example shows how to extract the 2018 single-year PUMS for the state of Texas, and only keep variables related to person-records. The ACS has information on both people and households, but for now we'll only look at the person records. Help on these functions can be found by typing `?pums_variables` and `?get_pums` in `R`

```
library(tidycensus)
library(tidyverse)

pums_vars_18<- pums_variables %>%
  filter(year== 2018, survey == "acs1") %>%
  distinct(var_code, var_label, data_type, level) %>%
  filter(level == "person")

TX_pums <- get_pums(
  variables = c("PUMA", "SEX", "AGEP", "CIT", "JWTR","JWRIP", "HISP"),
  state = "AL",
  survey = "acs1",
  year = 2018)
```

```
knitr::kable(
  head(TX_pums),
  format = 'html'
  )
```

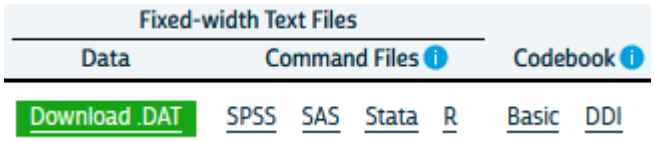
These data are also easily available from the Integrated Public Use Microdata Series (IPUMS) project housed at the University of Minnesota (Ruggles et al. 2021). The IPUMS version of the data adds additional information to the data and homogenizes the data across multiple years to make using it easier. The following example will use the `ipumsr` package to read in an extract from IPUMS-USA.

After you create an IPUMS extract, right click on the DDI link and save that file to your computer. Then repeat this for the .DAT file. If you need help creating an IPUMS extract, their staff have created a tutorial for doing so (https://usa.ipums.org/usa/tutorials.shtml).

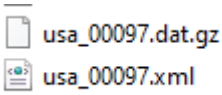
This will save the xml file that contains all the information on the data (what is contained in

the data file) to your computer. When using IPUMS, it will have a name like `usa_xxxxx.xml` where the x's represent the extract number.

You will also need to download the data file, by right clicking the **Download.DAT** link in the above image. This will save a .gz file to your computer, again with a name like: `usa_xxxxx.dat.gz`. Make sure this file and the xml file from above are in the same folder.



The fundamentals of using `ipumsr` is to specify the name of your `.xml` file from your extract, and as long as your `.tar.gz` file from your extract is in the same location, R will read the data. The files on my computer:




```
library(ipumsr)

ddi <- read_ipums_ddi(ddi_file = "data/usa_00097.xml")
ipums <- read_ipums_micro(ddi = ddi)
```

Will read in the data, in this case, it is a subset of the 2008 to 2012 single year ACS. This extract is not all of the variables from the ACS, as that would be a very large file and for my purposes here, I don't need that. My goal for the rest of the chapter is to illustrate how to use the IPUMS as an example of a complex survey design data set and the steps necessary to do so.

6.6 Basics of analyzing survey data

A fundamental part of analyzing complex survey data are knowing the variables within the data that contain the survey design information. The US Census Bureau has documented the design of the survey in a publication (US Census Bureau 2014) The IPUMS version of the ACS has two variables `STRATA` and `CLUSTER` that describe the two stage process by which the data are collected. Here are a the first few lines of these from the data:

```
options(scipen = 999)
library(knitr)
```

```
kable(head(ipums[, c("SERIAL", "STRATA", "CLUSTER")],
           n=10),
      digits = 14 )
```

For the ACS, the strata variable is named, ironically `STRATA` and the cluster variable `CLUSTER`. The IPUMS creates the `STRATA` variable based on the sampling strata in the ACS, and the `CLUSTER` variable based on households within a stratum. Often in surveys, the clusters may not be households, they could be smaller population aggregates, such as neighborhoods and villages, as in the DHS.

The data also come with housing unit weights and person unit weights, so your analysis can be either representative of housing units or people.

```
kable(head(ipums[, c("SERIAL", "STRATA", "CLUSTER", "HHWT", "PERWT")],
           n=10),
      digits = 14 )
```

SERIAL	STRATA	CLUSTER	HHWT	PERWT
1189369	330148	2019011893691	67	67
1189370	231948	2019011893701	43	43
1189371	690048	2019011893711	114	114
1189372	430248	2019011893721	34	34
1189373	650048	2019011893731	35	35
1189374	680548	2019011893741	19	19
1189375	340048	2019011893751	18	18
1189376	60048	2019011893761	37	37
1189377	440048	2019011893771	76	76
1189378	462348	2019011893781	10	10

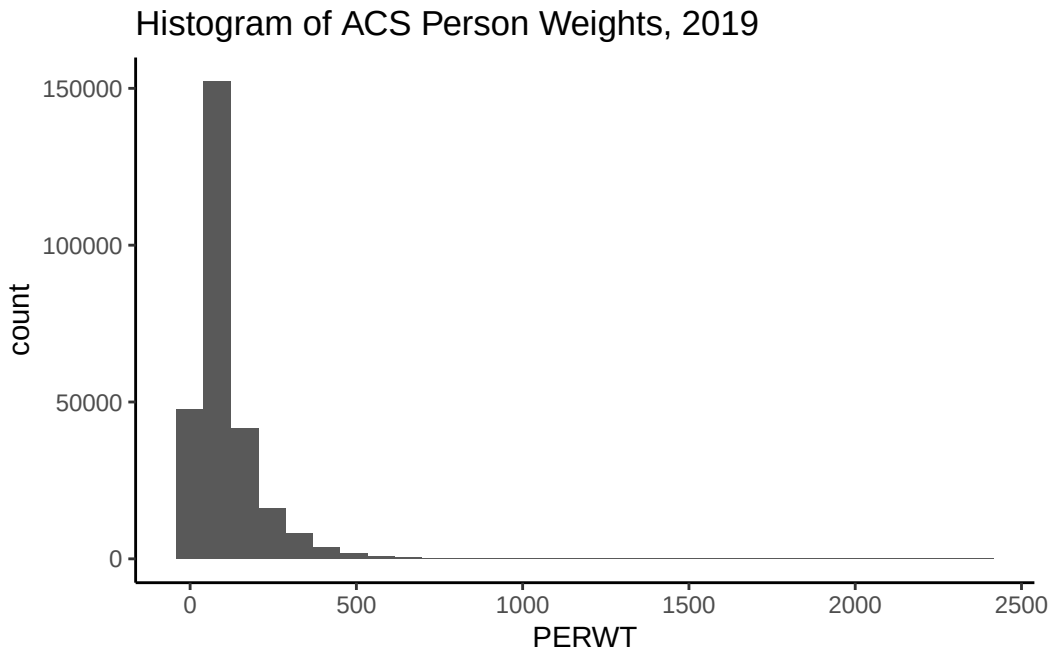
As can be seen in the first few cases, the `HHWT` variable is the same for everyone in a given household, but each person has a unique person weight showing that they each represent different numbers of people in the population. Further investigation of the housing and person weights allow us to see what these values actually look like.

```
summary(ipums$PERWT)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.0    49.0    78.0   106.3   128.0  2376.0
```

Here we see the minimum person weight is 1 and the maximum is 2376, which tells us that at least on person in the data represents 2376 people in the population that year. A histogram of the weights can also show us the distribution of weights in the sample.

```
library(ggplot2)

ipums%>%
  ggplot(aes(x = PERWT)) +
    geom_histogram() +
  labs(title = "Histogram of ACS Person Weights, 2019")
```



We can see how the weights inflate each person or household to the population by summing the weights. Below, I sum the person weights for the state of Texas, the sum is 28,995,881 million people, which is the same as the official estimate of the population in 2019 (https://www.census.gov/quickfacts/TX), we also see, by using the `n()` function, that there were 272,776 persons in the sample in 2019 living in Texas.

```
library(dplyr)
ipums%>%
  filter(STATEFIP == 48)%>%
  summarize(tot_pop = sum( PERWT ) , n_respond = n())
```

```
# A tibble: 1 x 2
   tot_pop n_respond
     <dbl>     <int>
1 28995881    272776
```

For housing units, we have to select a single person from the household in order for the same process to work, otherwise we would misrepresent the number of households in the state. We see there are 10,585,803 million housing units, and 114,016 unique households in the data.

```
ipums%>%
  filter(STATEFIP == 48,
         PERNUM == 1)%>%
  summarize(tothh = sum( HHWT ) , n_housing = n())
```

```
# A tibble: 1 x 2
     tothh n_housing
     <dbl>     <int>
1 10585803    114016
```

This total is nearly identical to that from the Census's ACS estimate.

This exercise shows that by using the provided weights in the survey, we can estimate the population size the sample was supposed to capture effectively. The `survey` package and the newer tidyverse package `srvyr` are designed to fully implement survey design and weighting and perform a wide variety of statistical summaries.

The way these packages work, is that you provide the name of your data frame, and the survey design variables that are in your data and the package code performs the requested analysis, correcting for survey design and weighting to the appropriate population. The code below illustrates how to enter the survey design for the IPUMS-USA ACS. Some surveys will not have both a cluster and stratification variable, so again, it's important to consult your survey documentation to find these for your data.

The function `as_survey_design()` in this case takes three arguments, since we are piping the 2019 ACS into it, we don't have to specify the name of the data. `ids` is the argument for the cluster variable, if your survey doesn't have one, just leave it out. `strata` is where you specify the name of the survey stratification variable, and `weights` is where you specify the name of the appropriate weighting variable. In this case, I'm replicating the estimate of the housing units in Texas from above, so I'll use the `HHWT` variable. The easiest way to get a total population estimate is to use the `survey_total()` function, which is equivalent to summing the weights as shown above, although in the case of the survey analysis commands in the `survey` and `srvyr` packages, the total will also be estimated with a standard error of the estimate.

```
library(srvyr)
library(survey)

ipums%>%
  filter(STATEFIP == 48, PERNUM == 1)%>%
  as_survey_design(cluster= CLUSTER,
                   strata = STRATA,
                   weights = HHWT)%>%
  summarize(tothh = survey_total())
```

A short aside about survey design options The core definition of the ACS survey design is shown in the code above, and I highly recommend that you inspect the help file for the survey design functions `?as_survey_design` or `?svydesign`. An important option that often has to be specified is the `nest=TRUE` option. This if often necessary if PSU identifiers are not unique across strata. For example, the fictional data shown below has the PSU's values the same across strata.}

```
fake_survey<- data.frame(
  strata = c(1,1,1,1,1,1,
             2,2,2,2,2,2),
  psu = c(1,1,1,2,2,3,
          1,1,2,2,3,3),
  weight = rpois(n = 12, lambda = 20))

knitr::kable(fake_survey)
```

strata	psu	weight
1	1	21
1	1	20
1	1	18
1	2	21
1	2	14
1	3	15
2	1	14
2	1	21
2	2	9
2	2	18
2	3	25
2	3	12

If we attempt to make a survey design from this, R would show an error.

```
fake_design<- fake_survey%>%
  as_survey_design(ids = psu,
                   strata=strata,
                   weights = weight)
```

```
Error in svydesign.default(ids, probs, strata, variables, fpc, .data, : Clusters not nested
```

But if we include the `nest = TRUE` option, R doesn't give us the error:

```
fake_design<- fake_survey%>%
  as_survey_design(ids = psu,
                   strata=strata,
                   weights = weight,
                   nest = TRUE)
fake_design
```

```
Stratified 1 - level Cluster Sampling design (with replacement)
With (6) clusters.
Called via srvyr
Sampling variables:
 - ids: psu
 - strata: strata
 - weights: weight
Data variables: strata (dbl), psu (dbl), weight (int)
```

The ACS from IPUMS has unique CLUSTERs across strata, so we don't have to specify that argument when we declare our survey design.

Back to our housing estimates.

In this case, our `tothh` estimate is identical to summing the weights, but new we also have an estimate of the precision of the estimate, so we could produce a more informed statistical estimate that in 2019, there were 10,585,803 $\pm$ 27,274.96 occupied housing units in the state.

If your data come with replicate weights instead of strata and cluster variables, this can be specified using the `as_survey_rep()` command instead `as_survey_design()`. In this case, we have to specify all of the columns which correspond to the replicate weights in the data. There are likely many ways to do this, but below, I use a method that matches the column names using a regular expression, where we are looking for the string `REPWT`, followed by any number of numeric digits, that is what the `[0-9]+` portion tells R to do. Also, the ACS uses a balanced replicate weight construction, which also requires the case weight as well (Ruggles

et al. 2021), so we specify the replicate weight type as `BRR`. Again, this is specific to the ACS, and you need to consult your own code book for your survey for your design information.

In this case, we get the same estimate for the total number of housing units, but a smaller variance in the estimate, which is often seen when using replicate weights.

```
ipums%>%
  filter(STATEFIP == 48, PERNUM == 1)%>%
  as_survey_rep(weight = HHWT,
                repweights =matches("REPWT[0-9]+"),
                type = "JK1",
                scale = 4/80,
                rscales = rep(1, 80),
                mse = TRUE)%>%
  summarize(tothh = survey_total())
```

```
# A tibble: 1 x 2
     tothh tothh_se
     <dbl>    <dbl>
1 10585803   15479.
```

```
rw<-ipums%>%
      filter(STATEFIP==48, PERNUM==1)%>%
      select(REPWT1:REPWT50)

t1<-survey::svrepdesign(data=ipums[ipums$STATEFIP==48&ipums$PERNUM==1,],
                        repweights = rw,
                        weights = ipums$HHWT[ipums$STATEFIP==48&ipums$PERNUM==1],
                        type="JK1",scale = .05, rscales = rep(1, ncol(rw)), mse= TRUE)

t1$variables$ones<-1
library(survey)

svytotal(~ones, design=t1)
```

```
        total    SE
ones 10585803 11850
```

We can also define the survey design outside of a dplyr pipe if we want using the `survey` package.

```
acs_design <- svydesign(ids = ~ CLUSTER,
                        strata= ~ STRATA,
                        weights = ~ PERWT,
                        data=ipums)

acs_design
```

```
Stratified 1 - level Cluster Sampling design (with replacement)
With (114016) clusters.
svydesign(ids = ~CLUSTER, strata = ~STRATA, weights = ~PERWT,
    data = ipums)
```

Of course we typically want to do more analysis than just estimate a population size, and typically we are interested in using survey data for comparisons and regression modeling. To carry out any sort of statistical testing on survey data, we must not only weight the data appropriately but we must also calculate all measures of variability correctly as well. Since surveys are stratified, the traditional formula for variances is not correct because under stratified sampling, all estimates are not only a function of the total sample, but also the within-strata sample averages and sample sizes. We can estimate the variances in our estimates using the design variables and sample weights in the survey analysis procedures, but there are options.

6.7 Replicates and jack knifes and expansions, oh my!

When conducting your analysis, you may not have any choices of whether you should use replicate weights or design weights, because your survey may only have one of these. There are two main strategies to estimate variances in survey data, the *Taylor Series Approximation* also referred to as *linearization* and the use of *replicate weights*. The Taylor Series, or linearization method is an approximation to the true variance, but is likely the most commonly used technique when analyzing survey data using regression methods. Lohr (2019) describes the calculation of variances from simple and clustered random samples in her book, and by her admission, once one has a clustered random sample the variance calculations for simple calculations becomes much more complex.

The problem is that we often want much more complicated calculations in our work and the variance formulas for anything other than simple ratios are not analytically known. The Taylor series approximation to the variance for complex and nonlinear terms such as ratios or estimates of regression parameters. The `survey` package in R will do this if you specify a survey design that includes strata or clusters, while if you specify replicate weights then it will use an appropriate technique depending on how the data were collected.

Typical replicate methods include balanced replicates, where there are exactly two clusters within each stratum, jackknife methods, which effectively remove one cluster from the strata and perform all calculations without that cluster in the analysis, then average across all replicates, and bootstrap methods which randomly sample clusters within strata with replacement a large number of times to get an estimate of the quantities of interest.

6.8 Descriptive analysis of survey data

The `survey` library allows many forms of descriptive and regression analysis.

6.9 Weighted frequencies and rates

Basic frequency tables are very useful tools for examining bivariate associations in survey data. In the survey analysis packages in R, the basic tools for doing this are the `svytable()` function in `survey`, or via the `survey_total()` function in `srvyr`. First I will recode two variables in the ACS, the employment status to indicate if a respondent is currently employed, and the `MET2013` variable, which is the metropolitan area where the respondent was living. This will give us the ACS estimate for the employed and unemployed population in each Texas MSA. I first have to filter the data to be people of working age, who are in the labor force and living in a MSA.

```
ipums %>%
  filter(EMPSTAT %in% 1:2,
         AGE >= 16 & AGE <= 65,
         MET2013 != 0) %>%
  mutate(employed = as.factor(case_when(.$EMPSTAT == 1 ~ "Employed",
                                .$EMPSTAT == 2 ~ "Unemployed" )),
         met_name = haven::as_factor(MET2013)) %>%
  as_survey_design(cluster = CLUSTER,
                   strata = STRATA,
                   weights = PERWT) %>%
  group_by(met_name, employed)%>%
  summarize(emp_rate = survey_total()) %>%
  head()
```

```
# A tibble: 6 x 4
# Groups:   met_name [3]
  met_name                 employed   emp_rate emp_rate_se
  <fct>                    <fct>         <dbl>       <dbl>
```

```
1 Amarillo, TX             Employed     114201       2824.
2 Amarillo, TX             Unemployed     3761        843.
3 Austin-Round Rock, TX    Employed    1193654      10735.
4 Austin-Round Rock, TX    Unemployed    47998       3247.
5 Beaumont-Port Arthur, TX Employed     163936       3513.
6 Beaumont-Port Arthur, TX Unemployed     5851        774.
```

This is OK, but if we want the totals in columns versus rows, we need to reshape the data. To go from the current "long" form of the variables to a wide form, we can use `pivot_wider` in `dplyr`.

```
wide<-ipums %>%
  filter(EMPSTAT %in% 1:2,
         AGE >= 16 & AGE <= 65,
         MET2013 != 0) %>%
  mutate(employed = as.factor(case_when(.$EMPSTAT == 1 ~ "Employed",
                                .$EMPSTAT == 2 ~ "Unemployed" )),
         met_name = haven::as_factor(MET2013)) %>%
  as_survey_design(cluster = CLUSTER,
                   strata = STRATA,
                   weights = PERWT) %>%
  group_by(met_name, employed)%>%
  summarize(emp_rate = survey_total()) %>%
  pivot_wider(id_cols = met_name,
              names_from = employed,
              values_from = c(emp_rate, emp_rate_se) ) %>%
  head()

wide
```

```
# A tibble: 6 x 5
# Groups:   met_name [6]
  met_name            emp_rate_Employed emp_rate_Unemployed emp_rate_se_Employed
  <fct>                           <dbl>               <dbl>                <dbl>
1 Amarillo, TX                   114201                3761                2824.
2 Austin-Round Rock,~           1193654               47998               10735.
3 Beaumont-Port Arth~            163936                5851                3513.
4 Brownsville-Harlin~            161922                8153                3500.
5 College Station-Br~            111576                3517                3132.
6 Corpus Christi, TX             208801               12365                4222.
# i 1 more variable: emp_rate_se_Unemployed <dbl>
```

A note about pivoting data The previous example used the `pivot_wider()` function to take observations that were in rows and pivot them into columns. This process, along with the `pivot_longer()` function are fundamental to working with many types of demographic data where we either need to format the data so observations are distributed over columns (as in the above example), or where we have multiple columns and we need to put those into multiple rows. The `pivot_wider()` function has lots of arguments that can be used, but the key ones are `id_cols` which identifies the unique identifier for each observation, here being the city or `met_name`. After that is identified, the `names_from` option tells R what the labels are in the current rows that will become separate column names. Here these are `Employed` and `Unemployed`. From these new columns, we can have multiple values provided to `values_from`, provided as a vector of current column names. Here these are `emp_rate` and the standard error `emp_rate_se`.

To return the data to their original form, with repeated observations in the rows, we can use the `pivot_longer()` function. In this case I will just provide the columns in the wide data that I want to be in rows in the long data:

```
wide%>%
  pivot_longer(cols = starts_with("emp_rate_"),
               names_to = c(".value", "rate"),
               names_sep = "\\._")%>%
  select(-rate)
```

```
# A tibble: 6 x 5
# Groups:   met_name [6]
  met_name            emp_rate_Employed emp_rate_Unemployed emp_rate_se_Employed
  <fct>                           <dbl>               <dbl>                <dbl>
1 Amarillo, TX                   114201                3761                2824.
2 Austin-Round Rock,~           1193654               47998               10735.
3 Beaumont-Port Arth~            163936                5851                3513.
4 Brownsville-Harlin~            161922                8153                3500.
5 College Station-Br~            111576                3517                3132.
6 Corpus Christi, TX             208801               12365                4222.
# i 1 more variable: emp_rate_se_Unemployed <dbl>
```

Which gets me back to a very similar data frame that I started with above. We will also see in the chapter on micro demography examples of using these functions when assemlbing longitudinal data sets.

Of course, if we want rates, this would imply us having to divide these columns to calculate the rate, but we can also get R to do this for us using `survey_mean()`. Since the `employed` variable is dichotomous, if we take the mean of its various levels, we get a proportion, in this case the employment and unemployment rates, respectively.

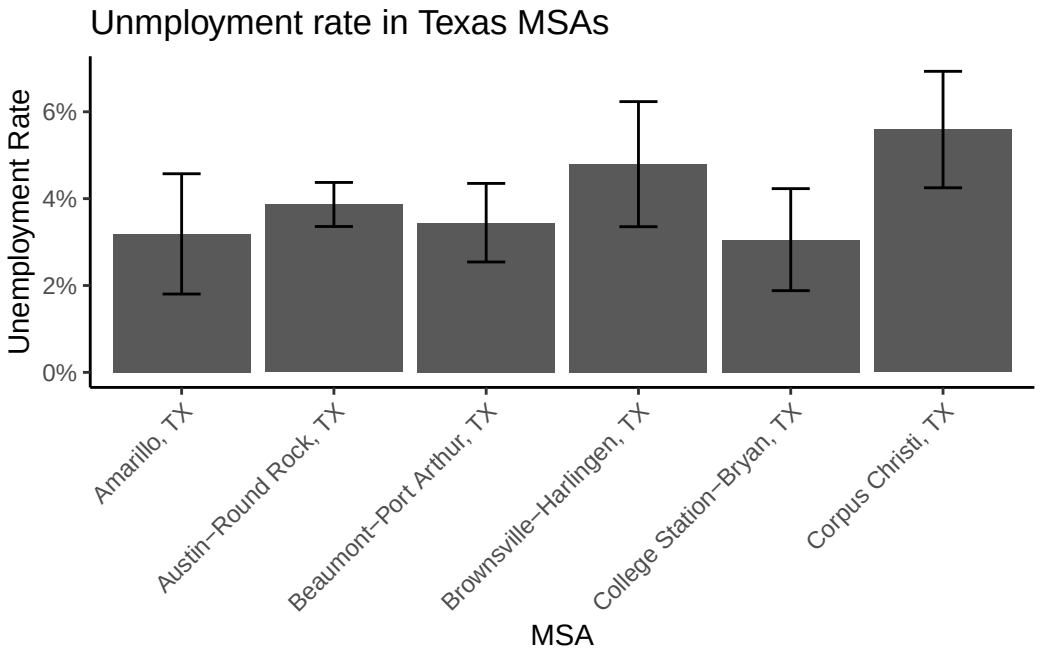
```
ipums %>%
  filter(EMPSTAT %in% 1:2,
         AGE >= 16 & AGE <= 65,
         MET2013 != 0) %>%
  mutate(employed = as.factor(case_when(.$EMPSTAT == 1 ~ "Employed",
                                .$EMPSTAT == 2 ~ "Unemployed" )),
         met_name = haven::as_factor(MET2013)) %>%
  as_survey_design(cluster = CLUSTER,
                   strata = STRATA,
                   weights = PERWT) %>%
  group_by(met_name, employed)%>%
  summarize(emp_rate = survey_mean()) %>%
  pivot_wider(id_cols = met_name,
              names_from = employed,
              values_from = c(emp_rate, emp_rate_se) ) %>%
  head()
```

```
# A tibble: 6 x 5
# Groups:   met_name [6]
  met_name            emp_rate_Employed emp_rate_Unemployed emp_rate_se_Employed
  <fct>                           <dbl>               <dbl>                 <dbl>
1 Amarillo, TX                    0.968              0.0319               0.00706
2 Austin-Round Rock,~             0.961              0.0387               0.00259
3 Beaumont-Port Arth~             0.966              0.0345               0.00461
4 Brownsville-Harlin~             0.952              0.0479               0.00735
5 College Station-Br~             0.969              0.0306               0.00599
6 Corpus Christi, TX              0.944              0.0559               0.00684
# i 1 more variable: emp_rate_se_Unemployed <dbl>
```

Which gets us the employment rate and the unemployment rate for each metropolitan area, with their associated standard errors. This is a general process that would work well for any grouping variable. If we create an object from this calculation, in this case I'll call it `tx_rates`, then we can also easily feed it into `ggplot` to visualize the rates with their associated 95% confidence intervals. The `geom_errorbar` addition to a `ggplot` object can add errors to estimates, which are great because we convey the uncertainty in the rates.

```
tx_rates%>%
  ggplot()+
  geom_bar(aes(x=met_name, y = emp_rate_Unemployed), stat = "identity")+
  geom_errorbar(aes(x=met_name,
                    ymin=emp_rate_Unemployed-1.96*emp_rate_se_Unemployed,
```

```
                    ymax= emp_rate_Unemployed+1.96*emp_rate_se_Unemployed),
                width=.25)+
  scale_y_continuous(labels = scales::percent)+
  labs(x = "MSA",
       y = "Unemployment Rate",
       title = "Unmployment rate in Texas MSAs")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

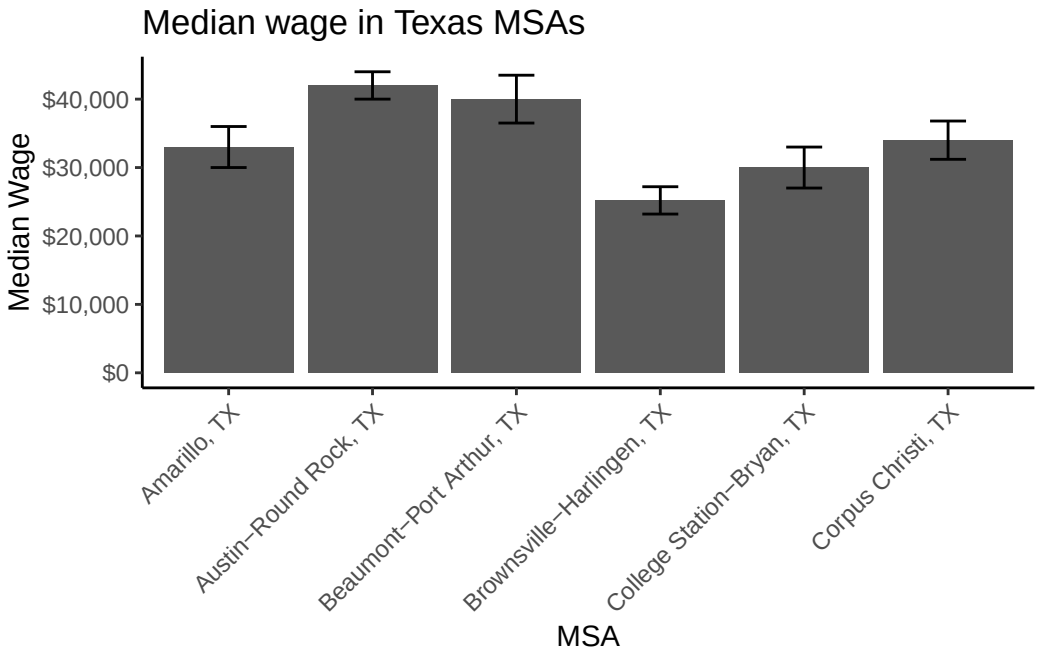


We see that among the first 6 MSAs in the state (that are in the ACS microdata), Corpus Christi has the highest unemployment rate a %, and College Station-Bryan has the lowest unemployment rate at %.

We can also use these functions for continuous variables, say with incomes. In the code below, I pipe all the elements of the analysis together to illustrate the workflow that we can do to calculate the median income in each MSA and plot it along with its error.

```
ipums %>%
  filter(EMPSTAT %in% 1:2,
         AGE >= 16 & AGE <= 65,
         MET2013 != 0) %>%
  mutate(income = ifelse(INCWAGE <= 0, NA, INCWAGE),
```

```
       met_name = haven::as_factor(MET2013)) %>%
as_survey_design(cluster = CLUSTER,
                 strata = STRATA,
                 weights = PERWT) %>%
group_by(met_name)%>%
summarize(median_wage = survey_median(income, na.rm=T)) %>%
head() %>%
ggplot()+
geom_bar(aes(x=met_name, y = median_wage), stat = "identity")+
geom_errorbar(aes(x=met_name,
                  ymin=median_wage-1.96*median_wage_se,
                  ymax= median_wage+1.96*median_wage_se),
              width=.25)+
scale_y_continuous(labels = scales::dollar)+
labs(x = "MSA",
     y = "Median Wage",
     title = "Median wage in Texas MSAs")+
theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Which shows that Austin-Round Rock has the highest median wage and Brownsville-Harlingen has the lowest median wage.

Table 6.3: Median Wages in Texas MSAs

MSA Name	Median Wage	Median Wage SE
Amarillo, TX	33000	1529
Austin-Round Rock, TX	42000	1020
Beaumont-Port Arthur, TX	40000	1784
Brownsville-Harlingen, TX	25200	1020
College Station-Bryan, TX	30000	1529
Corpus Christi, TX	34000	1428

6.10 Creating tables from survey data analysis

Tabular output from our survey data analysis is possible through several different means. When using dplyr, our intermediate output of the analysis is always a data frame, and so any R method for printing data frames would work for simple tabular display. For instance, if we just use `knitr::kable()` on our workflow from above instead of piping into a plot we would get something like this:

```
ipums %>%
  filter(EMPSTAT %in% 1:2,
         AGE >= 16 & AGE <= 65,
         MET2013 != 0) %>%
  mutate(income = ifelse(INCWAGE <= 0, NA, INCWAGE),
         met_name = haven::as_factor(MET2013)) %>%
  as_survey_design(cluster = CLUSTER,
                   strata = STRATA,
                   weights = PERWT) %>%
  group_by(met_name)%>%
  summarize(median_wage = survey_median(income, na.rm=T)) %>%
  head() %>%
  knitr::kable(format = "latex",
               digits = 0,
               caption = "Median Wages in Texas MSAs",
               align = 'c',
               col.names =c("MSA Name", "Median Wage", "Median Wage SE"))
```

Which is OK, but there are other ways to make tables for reports. The `gt` package (Iannone, Cheng, and Schloerke 2021) is built using tidyverse principles, and build tables in much the same way that `ggplot` builds plots, and fits easily into a dplyr workflow. Here, I use `gt` to produce a similar table to that from `knitr::kable` from above.

```r
library(gt, quietly = T)

ipums %>%
  filter(EMPSTAT %in% 1:2,
         AGE >= 16 & AGE <= 65,
         MET2013 != 0) %>%
  mutate(income = ifelse(INCWAGE <= 0, NA, INCWAGE),
         met_name = haven::as_factor(MET2013)) %>%
  as_survey_design(cluster = CLUSTER,
                   strata = STRATA,
                   weights = PERWT) %>%
  group_by(met_name)%>%
  summarize(median_wage = survey_median(income, na.rm=T)) %>%
  head()%>%
  gt() %>%
    tab_header(title = "Median Wages in Texas MSAs")%>%
    cols_label(met_name = "MSA Name",
               median_wage = "Median Wage",
               median_wage_se = "Median Wage SE")%>%
    fmt_number(columns = c( median_wage,  median_wage_se),
               decimals = 0, use_seps = TRUE)
```

Median Wages in Texas MSAs

MSA Name	Median Wage	Median Wage SE
Amarillo, TX	33,000	1,529
Austin-Round Rock, TX	42,000	1,020
Beaumont-Port Arthur, TX	40,000	1,784
Brownsville-Harlingen, TX	25,200	1,020
College Station-Bryan, TX	30,000	1,529
Corpus Christi, TX	34,000	1,428

In general, the `gt` tables are much easier to make look nice, compared to basic tables, because they're much more customizable. The `gtsummary` package extends the table functionality by combining the summary functions like `dplyr` with the table structures of `gt`. Additionally, it will recognize survey design objects so that information can also be integrated into your workflow. The `gtsummary` presents a more descriptive statistical summary of the variables included, and actually uses `dplyr` tools under the hood of the package.

In the code below, I first filter and mutate the IPUMS data to contain working age people who are employed, and who live in the six Texas cities featured in the examples above. I also create a new income variable that excludes all zero incomes, and drop levels of the `MET2013` variable

that aren't in the list I specified. This pipes into the survey design function from the `survey` package, which pipes into the `tbl_svysummary` function which summarizes income for each MSA. This function has a lot of options to specify its output, and I recommend you consult the examples at the author's website[2].

```
library(gtsummary)

ipums %>%
 filter(EMPSTAT == 1,
        AGE >= 16 & AGE <= 65,
        MET2013 != 0,
        MET2013 %in% c(11100, 12420, 13140, 15180, 17780, 18580)) %>%
  mutate(income = ifelse(INCWAGE <= 0, NA, INCWAGE),
        met_name = haven::as_factor(MET2013))%>%
  select(met_name,  income, CLUSTER, STRATA, PERWT)%>%
  droplevels()%>%
  survey::svydesign(id = ~ CLUSTER,
                    strata = ~ STRATA,
                    weights = ~ PERWT,
                  data = .) %>%
   tbl_svysummary(by = "met_name",
                  missing = "no",
                  include = c(met_name, income),
                  label = list(income = "Median Wage"))%>%
  as_hux_table()
```

TX, N = 1,193,654	**Beaumont-Port Arthur, TX**, N = 163,936	**Brownsville-Harlingen, TX**, N = 161,922	**Colleg**
, 75,000)	40,000 (20,000, 67,000)	26,000 (14,000, 47,000)	

6.10.1 How this differs from simple random sampling

So the big question I often get from students is "Do I really need to weight my analysis?" and of course, I say, "Of course!". Weights don't just inflate your data to the population, they serve a very important role in making sure your sample data aren't biased in their scope. Bias can enter into survey data in many forms, but nonresponse bias can dramatically affect population based estimates if key segments of our target population respond at low rates. Surveys will often deal with this by rigorous data collection strategies, but often the survey designers have

[2]http://www.danieldsjoberg.com/gtsummary/

to account for the added probability of nonresponse by modification of their basic weights. Lohr (2019) describes how this is done using a variety of methods including raking and post stratification, which typically separate the sample into subdivisions based on one or more demographic characteristic and produce weights based on how the sample deviates from the population composition. These methods are robust and are commonly used in large national surveys including the Behavioral Risk Factor Surveillance system (BRFSS).

Other reasons for weighting and why it matters are oversampling. Many surveys will do this in their data collection because an important element of their target population may be small in overall size, so they will sample more respondents from that population subgroup than their proportion in the larger population would predict. For example, if a survey wanted to get detailed data on recent refugees to a country, and this group is small, say .1 percent of the overall population, they may design their study to have 10% of their survey respondents be refugees. This oversampling can be accounted for in the survey weights so the additional respondents are down-weighted to represent their fraction of the target population, while still allowing the researchers to get a large sample from this group. Surveys such as the Early Childhood Longitudinal Surveys and the National Longitudinal Study of Adolescent to Adult Health (AddHealth) routinely over-sample specific groups. For example the AddHealth over-samples black adolescents with college educated parents, Cuban, Puerto Rican, Chinese, and physically disabled adolescents.

6.10.2 How do weights affect our estimates?

In the code example below, I illustrate how not including sample weights affects the estimates generated. This in effect is how traditional statistical analysis assuming random sampling would do things. So in order to compare the weighted estimates to the unweighted estimates, all we need to do is use a non-survey design oriented method to produce our estimate, and compare it to the survey design oriented method. Note that many surveys are designed to be *self-weighting*, so weights are not provided nor necessary, again, read the documentation for your specific survey for what it recommends.

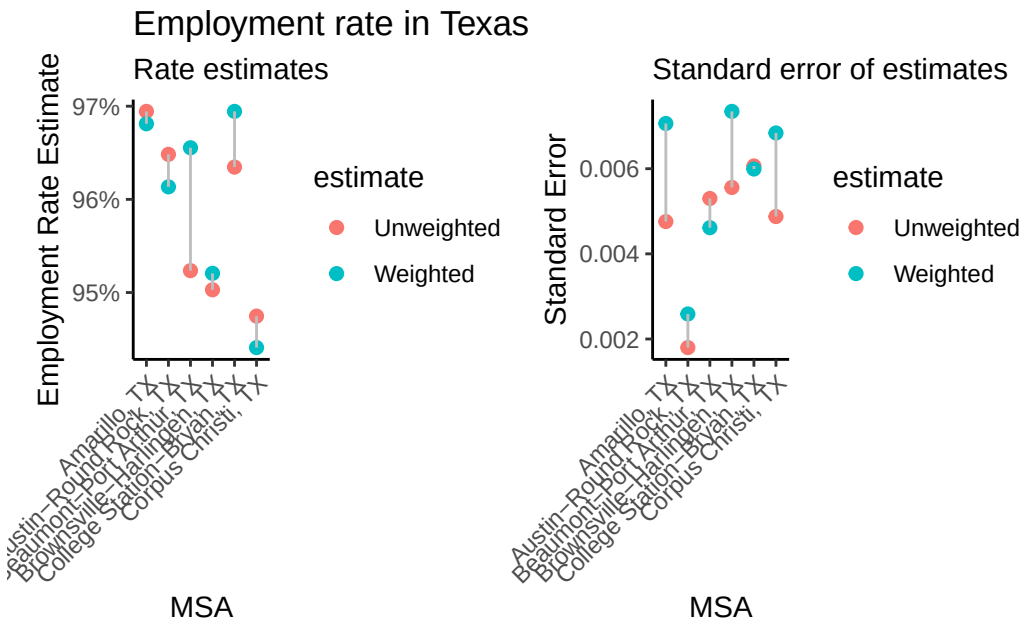
```
srs<-ipums %>%
  filter(EMPSTAT %in% 1:2,
         AGE >= 16 & AGE <= 65,
         MET2013 != 0) %>%
  mutate(employed = as.factor(case_when(.$EMPSTAT == 1 ~ "Employed",
                                .$EMPSTAT == 2 ~ "Unemployed" )),
         met_name = haven::as_factor(MET2013)) %>%
  group_by(met_name)%>%
  summarize(emp_rate = mean(I(employed)=="Employed"),
            emp_rate_se = sd(I(employed)=="Employed")/sqrt(n()))%>%
   # pivot_wider(id = met_name,
```

```
   #             names_from = employed,
   #             values_from = c(emp_rate, emp_rate_se) ) %>%
  head()
srs$estimate<-"Unweighted"
```

```
surv.est<-ipums %>%
  filter(EMPSTAT %in% 1:2,
         AGE >= 16 & AGE <= 65,
         MET2013 != 0) %>%
  mutate(employed = as.factor(case_when(.$EMPSTAT == 1 ~ "Employed",
                                .$EMPSTAT == 2 ~ "Unemployed" )),
         met_name = haven::as_factor(MET2013)) %>%
  as_survey_design(cluster = CLUSTER,
                   strata = STRATA,
                   weights = PERWT) %>%
  group_by(met_name)%>%
  summarize(emp_rate = survey_mean(I(employed)=="Employed")) %>%
  #rename(emp_rate_surv = emp_rate, emp_rate_se_surv = emp_rate_se)%>%
  head()

surv.est$estimate<-"Weighted"
```

```
merged <- rbind(srs, surv.est)
p1<-merged%>%
  ggplot(aes(y = emp_rate, x = met_name , color = estimate))+
  geom_point(stat="identity",
             cex=2)+
  geom_line(aes(group = met_name),
            col = "grey")+
  scale_y_continuous(labels = scales::percent)+
  scale_color_discrete( name = "Estimate Type")%>%
  labs(x = "MSA",
       y = "Employment Rate Estimate",
       title = "Employment rate in Texas MSAs",
       subtitle = "Rate estimates")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p2<-merged%>%
  ggplot(aes(y = emp_rate_se, x = met_name , color = estimate))+
  geom_point(stat="identity",
```

```
            cex=2)+
  geom_line(aes(group = met_name),
            col = "grey")+
  scale_color_discrete()%>%
  labs(x = "MSA",
       y = "Standard Error",
       title = "",
       subtitle = "Standard error of estimates")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

library(patchwork)

(p1 + p2)
```



We see that the difference in the weighted and unweighted estimates do vary by MSA, as do the standard errors of the estimates, with the survey-design calculated standard errors nearly always being higher than those assuming simple random sampling. If our data came from a survey with more extreme oversampling we would likely see larger differences in these estimates, and it is generally advisable to include both weights and survey design elements in all analysis of complex survey data.

6.11 Basic statistical testing on survey data.

To perform basic bivariate statistical tests for frequency tables, the `srvyr::svychisq` and `survey::svy_chisq` are the primary tools you need. These will test for basic independence among rows and columns of a frequency table. Below is an example on how you would test for independence of labor force participation and gender in the IPUMS ACS data.

```
ipums %>%
  filter(EMPSTAT != 0,
         AGE >= 16 & AGE <= 65) %>%
  mutate(lab_force_part = ifelse (test = EMPSTAT %in% c(1,2),
                                  yes = 1,
                                  no = 0),
         gender = ifelse(test = SEX ==1,
                         yes = "Male",
                         no = "Female")) %>%
  as_survey_design(cluster = CLUSTER,
                   strata = STRATA,
                   weights = PERWT) %>%
  svychisq(~lab_force_part+gender,
                    design = .)
```

```
    Pearson's X^2: Rao & Scott adjustment

data:  NextMethod()
F = 1955.4, ndf = 1, ddf = 172976, p-value < 0.00000000000000022
```

The output above is for the survey adjusted chi square test (Rao and Scott 1984), which is actually calculated as an F-test in this case. We see a large F-test value, and a very small p value, which indicates that men and women have different labor force participation rates. Using our workflow from above, we can calculate the actual rates as well.

```
ipums %>%
  filter(EMPSTAT != 0,
         AGE >= 16 & AGE <= 65) %>%
  mutate(lab_force_part = ifelse (test = EMPSTAT %in% c(1,2),
                                  yes = 1,
                                  no = 0),
         gender = ifelse(test = SEX ==1,
                         yes = "Male",
```

```
                        no = "Female")) %>%
as_survey_design(cluster = CLUSTER,
                 strata = STRATA,
                 weights = PERWT) %>%
group_by(gender)%>%
summarize(lf_part_rate = survey_mean(lab_force_part, na.rm=T)) %>%
head()%>%
gt() %>%
  tab_header(title = "Labor Force Participation Rates in Texas")%>%
  cols_label(gender = "Gender",
             lf_part_rate = "Labor Force Participation Rate",
            lf_part_rate_se = "SE")%>%
  fmt_number(columns = c( lf_part_rate,  lf_part_rate_se),
             decimals = 3, use_seps = TRUE)
```

Labor Force Participation Rates in Texas

Gender	Labor Force Participation Rate	SE
Female	0.674	0.002
Male	0.797	0.002

So we see that males have a much higher labor force participation rate, compared to females, and this puts the differences that we observed from the chi square test into better context.

6.11.1 Regression and survey design

The design of our surveys affect the most basic estimates we do, and likewise, the design affects the more complicated analysis as well. Regression models are the work horse of social science research and we will spend a significant amount of the chapters that follow on thorough inspection of them. In the context of this chapter, I felt like I need to show both how to include survey design in a regression model and illustrate that weighting and survey design matters in terms of the output from out models. This section is *NOT* a total coverage of these models, and is at best a short example. This example will go in a different direction and use data from the Demographic and Health Survey instead of the ACS.

The Demographic and Health Survey (DHS) data have been collected since the mid 1980's in over 90 countries around the world, and the DHS provides a public model data set that represents data on real households, without a specific national context. These data are provided to let people learn how to use the data before applying for access. The model data can be downloaded freely from the DHS[^surveydata-3] as a SAS or STATA format, or from my Github site[^surveydata-4] for this book. Below, I will read in the data from Github and

recode child growth stunting relative to the WHO standard as an outcome[^surveydata-5] , and child age, rural residence and gender as predictors.

The DHS household file is arrayed with a column for every child, so we must reshape the data from wide to long format using `pivot_longer` for the variables for child height relative to the WHO standard (`hc70`), child gender `hc27`, and child age `hc1`. The other variables are common to the household, so we do not have to reshape them (per the `cols=` line below). We then recode the outcome and the predictors for our regression example.

```
dhs_model_hh <- readRDS(
  url("https://github.com/coreysparks/data/blob/master/dhs_model_hh.rds?raw=true")
  )

dhs_model_hh_sub <- dhs_model_hh%>%
  select(hc27_01:hc27_20,
         hc70_01:hc70_20,
         hc1_01:hc1_20,
         hv021, hv025, hv270, hv005, hv021, hv022)%>%
  pivot_longer(cols = c(-hv021, -hv025, -hv270, -hv005, -hv021, -hv022),
               names_to  = c(".value", "child"),
               names_sep = "_") %>%
  na.omit()%>%
  mutate(stunting = car::Recode(hc70, recodes = "-900:-200 = 1; 9996:9999 = NA; else = 0")
         gender = ifelse(test = hc27 == 1, yes = "male", no = "female"),
         hh_wealth = as.factor(hv270),
         age = hc1,
         age2 = hc1^2,
         rural = ifelse(test = hv025 ==2, yes = "rural", no = "urban"),
         wt = hv005/1000000,
         psu = hv021,
         strata = hv022)
```

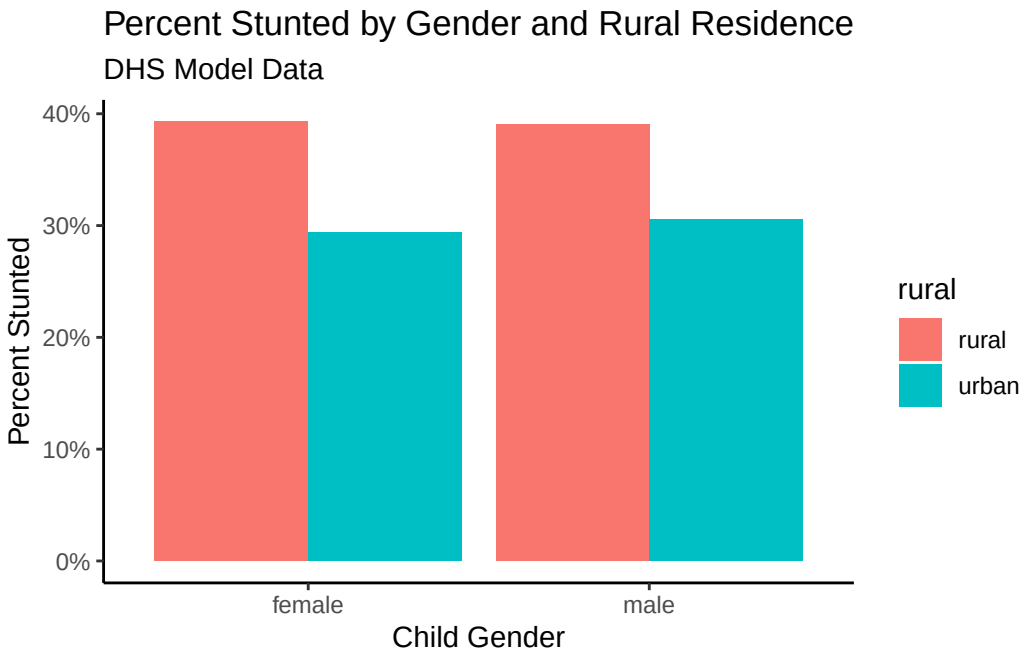
```
dhs_model_des<- dhs_model_hh_sub%>%
  as_survey_design(cluster = psu,
                   strata = strata,
                   weights = wt,
                   nest = TRUE)

summary(dhs_model_des)
```

```
Stratified Independent Sampling design (with replacement)
Called via srvyr
```

```
Probabilities:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1552  0.7872  1.2943  1.5297  2.0336  6.6422
Stratum Sizes:
             1   2  3  4   5  6   7   8   9  10  11 12  13 14  15  16  17 18  19
obs         33 136 52 41 159 23 108 117 185 266 218 44 244 25 125 183 152 80 144
design.PSU  33 136 52 41 159 23 108 117 185 266 218 44 244 25 125 183 152 80 144
actual.PSU  33 136 52 41 159 23 108 117 185 266 218 44 244 25 125 183 152 80 144
            20 21 22 23 24 25  26  27
obs         62 83 74 74 73  6 122 124
design.PSU  62 83 74 74 73  6 122 124
actual.PSU  62 83 74 74 73  6 122 124
Data variables:
 [1] "hv021"     "hv025"     "hv270"     "hv005"     "hv022"     "child"
 [7] "hc27"      "hc70"      "hc1"       "stunting"  "gender"    "hh_wealth"
[13] "age"       "age2"      "rural"     "wt"        "psu"       "strata"
```

Just for completeness, I create a bar chart showing the percent stunted by the two main variables, gender and rural residence.

```
dhs_model_des%>%
  group_by(gender, rural)%>%
  summarise(stunting = survey_mean( stunting,design =.,
                                    na.rm=T) )%>%
  ggplot()+
  geom_bar(aes(x = gender, y  = stunting, fill = rural),
           stat="identity",
           position="dodge")+
  labs(x = "Child Gender",
       y = "Percent Stunted",
       title = "Percent Stunted by Gender and Rural Residence",
       subtitle = "DHS Model Data")+
  scale_y_continuous(labels = scales::percent)
```



Now I estimate two logistic regression models for the stunting outcome. The first assumes random sampling and the second includes the full survey design information.

For the unweighted regular logistic regression, we use the `glm()` function with `family = binomial (link = "logit")`. The`glm()` function will be used extensivley in the chapters that follow.

```
library(broom)
m1<-glm(stunting ~ age + age2 + rural + gender,
              data = dhs_model_hh_sub,
              family = binomial(link = "logit"))
m1%>%
  tidy()
```

```
# A tibble: 5 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept) -1.67     0.158      -10.6   4.47e-26
2 age          0.0976   0.0111       8.82  1.10e-18
3 age2        -0.00145  0.000176    -8.23  1.91e-16
4 ruralurban  -0.484    0.0934      -5.19  2.15e- 7
5 gendermale   0.0461   0.0837       0.551 5.82e- 1
```

For the survey design model, you specify the design versus the data set name, otherwise the same code works just fine.

```
m2 <- dhs_model_des%>%
  svyglm(stunting ~ age + age2 +rural + gender,
         design = .,
         family = binomial)
```

```
Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```

```
m2%>%
  tidy()
```

```
# A tibble: 5 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept) -1.77     0.180      -9.83   2.03e-22
2 age          0.108    0.0128      8.39   8.18e-17
3 age2        -0.00164  0.000209   -7.84   6.77e-15
4 ruralurban  -0.432    0.128      -3.38   7.25e- 4
5 gendermale   0.00938  0.103       0.0906 9.28e- 1
```

There are lots of ways to make a table from a regression model, but `stargazer` (Hlavac 2018) is a simple way to present multiple models side by side.

```
stargazer::stargazer(m1, m2,
                     keep.stat = "n",
                     model.names = F,
                     column.labels = c("Unweighted model", "Weighted Model"),
                     type = "latex",
                     header = FALSE,
                     style = "demography",
                     title = "Output from Unweighted and Weighted Regression Models")
```

In this case, we see that the coefficient estimates are very similar between the two models, but the coefficient standard errors are all smaller in the unweighted model. This is commonly what you see in this situation, because the survey deign model is actually using *clustered standard errors* instead of asymptotic standard errors Ibragimov and Müller (2016). While this example does not show an extreme difference, it is commonplace for t-statistics generated from clustered standard errors to have higher p-values than those using asymptotic standard errors. As a

Table 6.4: Output from Unweighted and Weighted Regression Models

	stunting	
	Unweighted model	Weighted Model
	Model 1	Model 2
age	0.098***	0.108***
	(0.011)	(0.013)
age2	−0.001***	−0.002***
	(0.0002)	(0.0002)
ruralurban	−0.484***	−0.432***
	(0.093)	(0.128)
gendermale	0.046	0.009
	(0.084)	(0.103)
Constant	−1.670***	−1.770***
	(0.158)	(0.180)
N	2,570	2,570

*p < .05; **p < .01; ***p < .001

result, if the t-statistic is lower, and the p-value higher, you can easily get differences in your hypothesis tests for regression parameters. This is always something to be aware of as you analyze survey data.

[^surveydata-3] https://dhsprogram.com/data/model-datasets.cfm [^surveydata-4] https://github.com/coreyspa stats-book [^surveydata-4] Per the guide to DHS statistics

6.12 Chapter summary

The goal of this chapter was to review the complexities of demographic survey data sources. The analysis of these kinds of data have to proceed in a way that adheres to the design of the survey as well as the goal of population-level inference from the survey data. The use of person or housing unit weights are necessary to ensure that our statistical summaries and analysis are representative of the population that the survey was designed to measure. The incorporation of the sample design elements of primary sampling units and sampling strata are integral to the correct calculation of standard errors for any survey-based estimates we create. R, like most major statistical programs have several ways of dealing with these issues, and the libraries `survey` and `srvyr` are very flexible in the types of analysis they can perform, and can incorporate any survey design into the subsequent analysis that we carry out.

6.13 References

7 Macrodemographic Analysis of Places

8 Macro demographic data analysis

Prior to the advent in the 1960's of large scale social surveys like the General Social Survey (GSS), most demographic research was done not on individuals but on aggregates, because that's how data were available. If you look at texts such as Keyfitz (1968), all of the examples are for national level calculations, and many nations did not have sufficient data availability to produce quality statistical summaries of their populations, resulting in publications such as the United Nations Population Division's famous Manual X (1983), which gave pragmatic formulas to measure a wide variety of demographic indicators at the national level using basic inputs, usually available from census summaries.

Paul Voss (2007) describes most demography (and certainly most demographic studies prior to the 1970's and 1980's) as **Macro** demography. Voss also mentions that prior to the availability of individual level microdata, all demography was macro-demography, and most demographic studies were spatial in nature, because demographic data were only available in spatial units corresponding to administrative areas. Typical types of geographic areas would be counties, census tracts, ZIP codes, state or nations.

In the macro-demographic perspective on demography, observations are typically places, areas, or some other aggregate level of individuals. We do not observe the individual people themselves often times. An example of this is if you were to have access to an aggregate count of deaths in a region, even if the deaths were classified by age and sex, you still would be dealing with data that ignores, or has no index to the more nuanced characteristics of the individual decedents themselves. That being said, data such as these are invaluable, and most demographic summaries of individual-level data would aggregate based on the characteristics of the individuals any way. The macro scale principal is illustrated below, where all of the variables we observe are a scale above the individual person.

Such **macro-level propositions** are hypothesized relationships among variables ($\rightarrow$) measured at a macro scale (Z and Y), which ignores individual level data, mostly because we don't observe individuals (x and y) in many of these kinds of analysis.

If all we looked at were the individuals within the population, we would be overwhelmed by the variation that we would see, and we wouldn't be doing statistics anymore, we would be trying to process a million anecdotes, and the plural of anecdote is not data. By aggregating across basic demographic groups, such as age and sex, demographers begin to tease apart the differences that we are interested in. If we go a little further and, data willing, aggregate not only across these fundamental demographic groups, but also across some kind of place-based

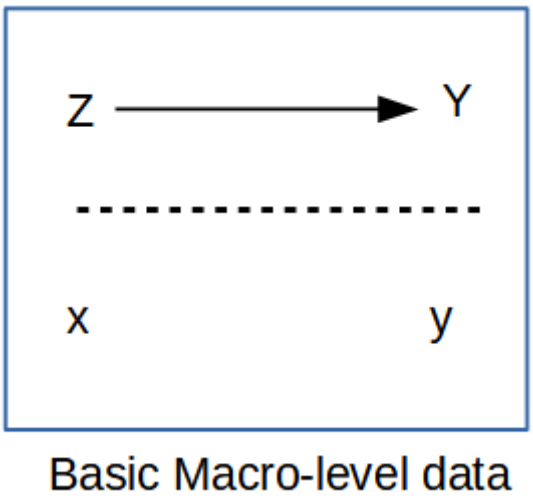


Figure 8.1: Macro Level Proposition

areal unit, then we adding an extremely important part of human existence: the **where** part of where we live.

This presents an attractive view of populations and typically data on places are more widely available, but there are caveats we must be aware of. If we are using purely aggregate data in our analysis, meaning that we do not have access to the individual level microdata, then our ability to observe variation within a place is extremely limited, if not impossible.

The goal of this chapter is to illustrate how places are a special unit of analysis, and the types of data we often see at the place level are very different from individual level surveys. Additionally, the analysis of place-based data is similar to survey data in that places are do not necessarily represent random observations, and so analyzing data on places often requires special modifications to statistical models. In this chapter, I show how the the linear regression model can be expanded in several ways and illustrate the generalized linear model as a very useful and extendable tool to analyze data on places and especially when we are analyzing rates as demographers often do.

8.1 Getting data on places

In the macro-demographic perspective on demography, observations are typically places, areas, or some other aggregate level of individuals. We do not observe the individual people themselves often times. An example of this is if you were to have access to an aggregate count of deaths in a region, even if the deaths were classified by age and sex, you still would be dealing with data that ignores, or has no index to the more nuanced characteristics of the individual decedents themselves. That being said, data such as these are invaluable, and most

demographic summaries of individual-level data would aggregate based on the characteristics of the individuals any way. The macro scale principal is illustrated below, where all of the variables we observe are a scale above the individual person.

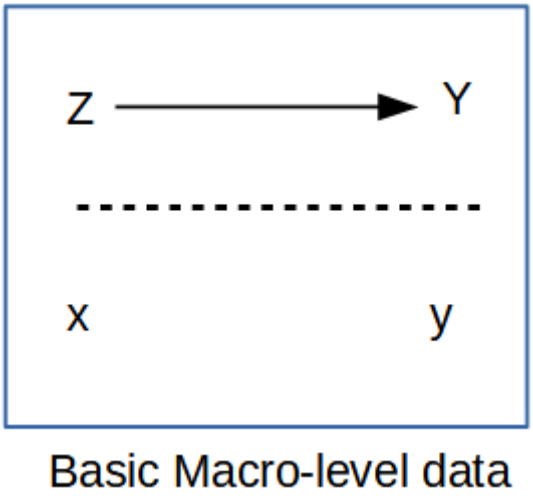


Figure 8.2: Macro Level Proposition

Such **macro-level propositions** are hypothesized relationships among variables ($\rightarrow$) measured at a macro scale (Z and Y), which ignores individual level data, mostly because we don't observe individuals (x and y) in many of these kinds of analysis.

If all we looked at were the individuals within the population, we would be overwhelmed by the variation that we would see, and we wouldn't be doing statistics anymore, we would be trying to process a million anecdotes, and the plural of anecdote is not data. By aggregating across basic demographic groups, such as age and sex, demographers begin to tease apart the differences that we are interested in. If we go a little further and, data willing, aggregate not only across these fundamental demographic groups, but also across some kind of place-based areal unit, then we adding an extremely important part of human existence: the **where** part of where we live.

This presents an attractive view of populations and typically data on places are more widely available, but there are caveats we must be aware of. If we are using purely aggregate data in our analysis, meaning that we do not have access to the individual level microdata, then our ability to observe variation within a place is extremely limited, if not impossible.

The goal of this chapter is to illustrate how places are a special unit of analysis, and the types of data we often see at the place level are very different from individual level surveys. Additionally, the analysis of place-based data is similar to survey data in that places are do not necessarily represent random observations, and so analyzing data on places often requires special modifications to statistical models. In this chapter, I show how the the linear regression model can be expanded in several ways and illustrate the generalized linear model as a very

useful and extendable tool to analyze data on places and especially when we are analyzing rates as demographers often do.

8.2 Getting data on places

Typically when thinking about data on places, we are really referring to some sort of administrative geography, such as nations, states, region, and census tracts. While these are often readily available (and I'll show some R package that can easily get data from the web), we often have to use these as proxy measures of more interesting social spaces like neighborhoods and other types of activity spaces. These social spaces are harder to get data on, typically because they are more fluid in their definitions, and there is generally not a systematic effort to produce data on socially defined spaces on national scales. This is a big part of doing macro demography, defining the scale and the unit of analysis, both because we need to define the scope of our work, but also we are very much constrained by the availability of data for our projects. For instance, I may want to look at national scale inequality in mortality risk in neighborhoods in the United States, but you immediately face a couple of hurdles. No national data source identifies sub-city residential location for death certificates, also, what are neighborhoods? Again, they're probably some socially defined space that may not be available from a national scale source. To get around this, we may have to settle for a state-level analysis, because state vital registration systems will often allow researchers to use more fine-scale geographic data on death certificates (such as latitude/longitude of the decedent's residence), and once we have very fine scale geographic data on the vital events, we could potentially find data on some more socially defined spaces, perhaps from cities who often maintain geographic data on neighborhoods specific to that city. OK, so that's fine, but then you still run into the "what's my denominator" problem, where you have no baseline population data on the age and sex breakdown of the population, or even the population size of these places, because federal agencies don't produce estimates for such small scale areas. *This is frustrating.* Often when advising students on their dissertation projects, I have to have this moment of truth where I lay out the problems of the mixing of geographic scales for their projects, and the hard reality of the lack of data on so many things they would like to study. Often what happens is that we have to proxy our ideal places with places for which we can find data. You see this a lot in the population health literature, where people want to analyze *neighborhoods* but all they have are census tracts. Tracts aren't social spaces! They're arbitrary areas of 3 to 5 thousand people, that change every 10 years, that the Census uses to count people. Likewise, counties are very rich areas to find data for, but they are not really activity spaces or neighborhoods, but they may be areas that have some policy making authority (such as county health departments) that *could* be relevant for something. States are also nice geographies, they're very large, so you loose the ability to contextualize behavior on a fine spatial scale, but states make a lot of decisions that affect the lives of their residents, often more than national decisions. States have become very popular units of analysis in the health literature again, primarily as a result of differential adoption of portions of the Patient Protection and Affordable Care Act

of 2010 (Soni, Hendryx, and Simon 2017; Courtemanche et al. 2019). This being said, many times when we do an analysis on places, that analysis has lots of limitations, which we must acknowledge, and analyses such as these are often called *ecological* analyses because we are examining associations at the macro scale, and we do not observe individual level outcomes.

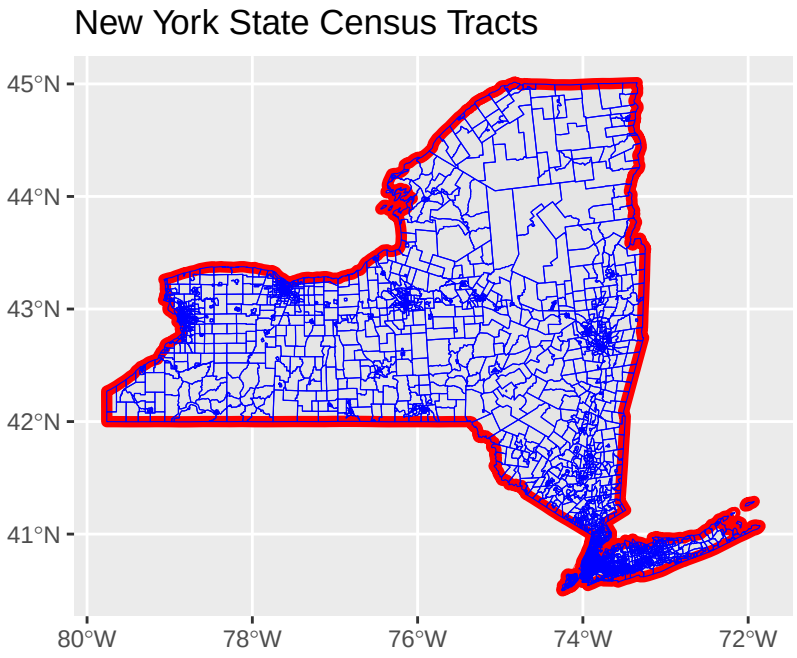
8.3 US contexts

The US Census bureau produces a wide variety of geographic data products that are the most widely used forms of geographic data for demographic studies in the United States. The TIGER Line Files data consist of geographic data with census bureau GEOIDs attached so they can be linked to any number of federal statistical products. They do not contain demographic data themselves, but are easily linked. The `tigris` package in R provides a direct way to download any TIGER line file data type directly in a R session as either a *simple feature* class or as a *Spatial_DataFrame* (Walker 2021).

Using the `tigris` package is very easy and its functions fit directly into the tidyverse as well. Below, I download two layers of information, first the state polygon for New York state, and the census tracts within the state and overlay the two datasets on each other. The package has a function for each type of geography that you would want, for example `states()` downloads state level geographies and `tracts()` does the same for census tracts. The functions have some common arguments, including `cb = TRUE/FALSE` so you can choose cartographic boundary files or not. Cartographic boundary files are lower resolution, smaller files that are often used for thematic mapping. Also `year =` will allow you to get different annual vintages of the data. The `tracts()` function also allows you to obtain geographies for specific counties within a state.

```
library(tigris)

nyst <- states(cb = TRUE,
               year = 2010) %>%
  filter(NAME == "New York")

nyst_ct <- tracts(state = "NY",
                  cb = TRUE,
                  year = 2010)

ggplot(data=nyst)+
  geom_sf(color = "red",
          lwd = 2)+
   geom_sf(data = nyst_ct,
           fill = NA,
```

```
          color = "blue") +
ggtitle(label = "New York State Census Tracts")
```



8.3.1 Tidycensus

Another package the provides access to the US Census Bureau Decennial census summary file , the American Community Survey, Census population estimates, migration flow data and Census Public Use Microdata Sample (PUMS) data is `tidycensus` (Walker and Herman 2021). The `tidycensus` package primarily works to allow users to use the Census Bureau's Application Programming Interface (API) to download Census summary file data for places within an R session. This removes the need to download separate files to your computer, and allows users to produce visualizations of Census data easily. The package is actively maintained and has several online tutorials on how to use it [1]. Depending on which data source you are interested in, there are functions that allow extracts from them. The ACS data is accessed through the `get_acs()` function, likewise the decennial census data is accessed using the `get_decennial()` function. The package also allows users to test for differences in ACS estimates either across time or between areas using the `significance()` function.

The package requires users to obtain a developer API key from the Census Bureau's developer page[2] and install it on your local computer. The package has a function that helps you install

[1] $\bar{y}$ = mean of y, $\bar{x}$ = mean of x
[2] http://api.census.gov/data/key_signup.html

the key to your `.Renviron` file. It is used like this:

```
census_api_key(key = "yourkeyhere", install = TRUE)
```

which only needs to be done once.

A basic use of the `tidycensus` package is to get data and produce maps of the indicators. This is done easily because `tidycensus` fits directly into general `dplyr` and `ggplot2` workflows. Below is an example of accessing 2019 ACS data on poverty rate estimates for New York census tracts from New York county, New York. The syntax takes several arguments indicating what level of census geography you want, the year of the estimates, the details of states and counties you may want, and which ACS tables you want. Here I use the Data Profile table for the percentage estimate of families with incomes below the poverty line. The `output = "wide"` option is useful if you get multiple estimates, as it arranges them into columns, one for each estimate.

```
library(tidycensus)

nyny <- get_acs(geography = "tract",
                year = 2018,
                state = "NY",
                county = "061",
                variables = "DP03_0119PE",
                output = "wide",
                geometry = TRUE)
```

```
Getting data from the 2014-2018 5-year ACS
```

```
Downloading feature geometry from the Census website.  To cache shapefiles for use in future
```

```
Using the ACS Data Profile
```

The tabular output shows the Estimate column ending in *E* and the ACS margin of error column ending in *M*.

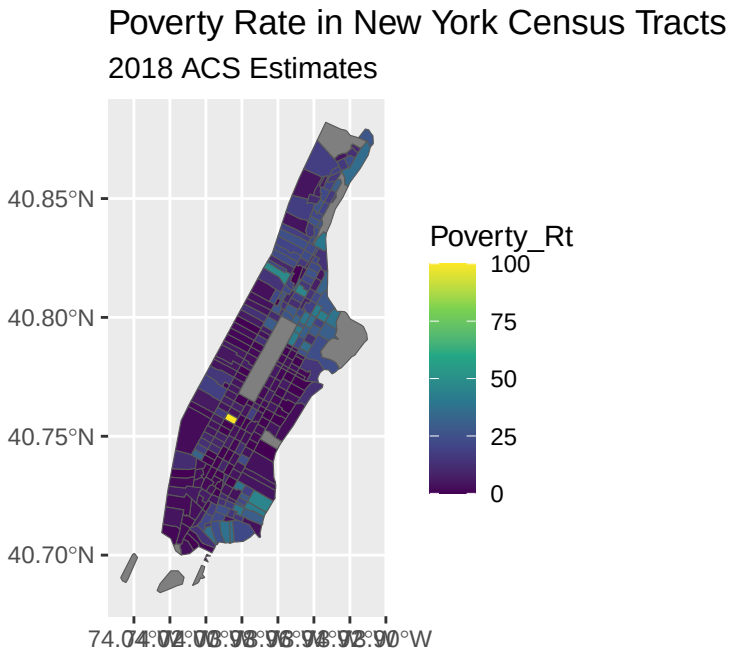
```
knitr::kable(x = head(nyny),
             format = "html")
```

GEOID	NAME	DP03_0119PE	DP03_0119PM	geometry
36061020101	Census Tract 201.01, New York County, New York	0.0	20.0	MULT

GEOID	NAME	DP03_0119PE	DP03_0119PM	geometry	
36061020701	Census Tract 207.01, New York County, New York	24.1	17.0	MULT	
36061022200	Census Tract 222, New York County, New York	18.5	9.1	MULT	
36061022600	Census Tract 226, New York County, New York	15.5	9.4	MULT	
36061000600	Census Tract 6, New York County, New York	37.5	9.8	MULT	
36061001600	Census Tract 16, New York County, New York	22.2	8.5	MULT	

The `geometry = TRUE` option also download the TIGER line file for the requested geography and merges it to the ACS estimates. This allows you to immediately map the estimates for the requested geographies.

```
# Create map of estimates
nyny %>%
  rename (Poverty_Rt = DP03_0119PE)%>%
  ggplot(aes(fill = Poverty_Rt))+
  geom_sf()+
  scale_fill_viridis_c()+
  ggtitle ( label = "Poverty Rate in New York Census Tracts",
            subtitle = "2018 ACS Estimates")
```

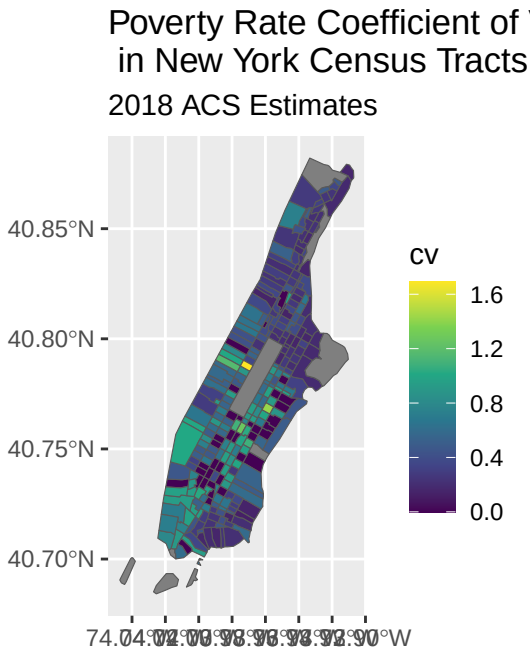


The `tidycensus` package had a great array of functions and the author Kyle Walker has

published a book on using it *FILL IN CITATION* which covers its many uses.

One common task that we should do when visualizing ACS estimates is to examine the coefficient of variation in the estimates. This gives us an idea of how stable the estimates are. This can be particularly problematic as we use smaller and smaller geographies in our analysis. Below, I calculate the coefficient of variation for the estimates and map it. To get the standard error of the ACS estimate, I divide the margin of error by 1.645, following Census Bureau recommendations (Bureau 2019).

```
nyny %>%
  mutate ( cv =ifelse(test = DP03_0119PE==0,
                      yes = 0,
                      no = (DP03_0119PM/1.645) / DP03_0119PE))%>%
  ggplot(aes(fill = cv))+
  geom_sf()+
  scale_fill_viridis_c()+
  ggtitle ( label = "Poverty Rate Coefficient of Variation\n in New York Census Tracts",
            subtitle = "2018 ACS Estimates")
```



which shows areas with the highest coefficient of variations mostly adjacent to Central Park and on the lower west side of Manhattan. These are also the areas with the lowest poverty rates in the city, so the estimates have low precision because so few respondents report incomes below the poverty line.

8.3.2 IPUMS NHGIS

The IPUMS NHGIS project [3] is also a great source for demographic data on US places, and allows you to select many demographic tables for census data products going back to the 1790 census (Manson et al. 2021). When you perform an extract from the site, you can get both data tables and ESRI shapefiles for your requested geographies. The IPUMS staff have created several tutorials which go through how to construct a query from their site [4]. Below, I use the `sf` library to read in the geographic data from IPUMS and the tabular data and join them.

```
library(sf)
ipums_co <- read_sf("data/US_county_2020.shp")
```

```
Error: Cannot open "data/US_county_2020.shp"; The file doesn't seem to exist.
```

```
im_dat <- readr::read_csv("data/nhgis0025_ds231_2005_county.csv")
```

```
Rows: 3143 Columns: 12
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (7): GISJOIN, AREANAME, STATE, STATEA, COUNTY, COUNTYA, DATAFLAG
dbl (5): YEAR, NOTECODE, AGWE001, AGWI001, AGWJ001

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
m_dat <- left_join(x = ipums_co,
                   y = im_dat,
                   by = c("GISJOIN" = "GISJOIN"))
```

```
Error in eval(expr, envir, enclos): object 'ipums_co' not found
```

```
m_dat %>%
  filter(STATE == "New York" )%>%
  ggplot()+
  geom_sf(aes (fill = AGWJ001))+
```

[3] Link to AHRF codebook - "https://data.hrsa.gov/DataDownload/AHRF/AHRF_USER_TECH_2019-2020.zip"

[4] More on this below

```
    scale_fill_viridis_c()+
    ggtitle(label = "Infant Mortality Rate per 10,000 Live Births",
            subtitle = "New York, 2005")
```

```
Error in eval(expr, envir, enclos): object 'm_dat' not found
```

8.3.3 International data

Sources of international data exist in numerous sites on the internet. Personally, I frequently will use the DHS Spatial Data repository [5] to access data from DHS sampled countries. This repository allows you to obtain both spatial administrative boundary data, as well as key indicators of maternal and child health at sub-national levels. Additionally, the `rdhs` package allows you to perform queries from the spatial data repository and from the DHS microdata as well directly via the DHS API from within an R session (Watson, FitzJohn, and Eaton 2019), assuming you have registered with the DHS and have an approved project with them.

8.4 Statistical models for place-based data

Data on places is often analysed in the same ways as data on individuals, with some notable complications. The remainder of this chapter introduces the regression framework for analyzing data at the macro level, first by a review of the linear model and its associated pitfalls, and then the generalized linear model with a specific focus on the analysis of demographic count outcomes that are commonly observed for places.

In the example below, I use data from the U.S. Health Resources and Services Administration Area Health Resource File (AHRF), which is a produced annually and includes a wealth of information on current and historical data on health infrastructure in U.S. counties, as well as data from the Census Bureau, and the National Center for Health Statistics. The AHRF is publicly available, and we can read the data directly from the HHS website as a SAS format `.sas7bdat` data set within a ZIP archive. R can read this file to your local computer then extract it using the commands below. I would strongly encourage you consulting the AHRF codebook available from the HRSA website[6].

```
#create temporary file on  your computer
temp <- tempfile()

#Download the SAS dataset as a ZIP compressed archive
```

[5] More on this below

[6] Link to AHRF codebook - "https://data.hrsa.gov/DataDownload/AHRF/AHRF_USER_TECH_2019-2020.zip"

```
download.file("https://data.hrsa.gov/DataDownload/AHRF/AHRF_2019-2020_SAS.zip", temp)

#Read SAS data into R
ahrf<-haven::read_sas(unz(temp,
                          filename = "ahrf2020.sas7bdat"))

rm(temp)
```

Next, I remove many of the variables in the AHRF and recode several others. In the analysis examples that follow in this chapter, I will focus on the outcome of low birth weight births, measured at the county level.

```
library(tidyverse)

ahrf2<-ahrf%>%
  mutate(cofips = f00004,
         coname = f00010,
         state = f00011,
         popn =  f1198416,
         births1618 =  f1254616,
         lowbw1618 = f1255316,
         fampov14 =  f1443214,
         lbrate1618 = 1000*(f1255316/f1254616),  #Rate per 1000 births
         rucc = as.factor(f0002013),
         hpsa16 = case_when(.$f0978716 == 0 ~ 'no shortage',
                            .$f0978716 == 1 ~ 'whole county shortage',
                            .$f0978716 == 2 ~ 'partial county shortage'),
         obgyn15_pc= 1000*( f1168415 / f1198416 ) )%>%
  mutate(rucc = droplevels(rucc, ""))%>%
  dplyr::select(births1618,
                lowbw1618,
                lbrate1618,
                state,
                cofips,
                coname,
                popn,
                fampov14,
                rucc,
                hpsa16,
                obgyn15_pc)%>%
  filter(complete.cases(.))%>%
```

```
  as.data.frame()
```

In order to make a nice looking map of the outcome, I use the `tigris` package to fetch geographic data for US states and counties, then merge the county data to the AHRF data using `left_join()`

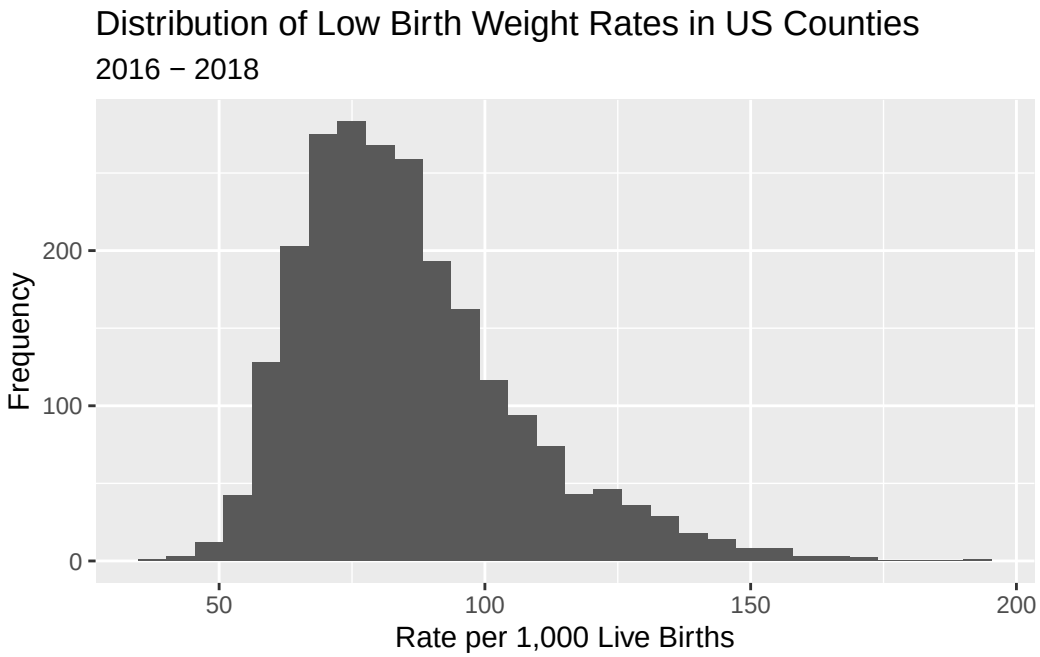
```
options(tigris_class="sf")
library(tigris)
library(sf)
usco<-counties(cb = T, year= 2016)

usco$cofips<-usco$GEOID

sts<-states(cb = T, year = 2016)

sts<-st_boundary(sts)%>%
  filter(!STATEFP %in% c("02", "15", "60", "66", "69", "72", "78"))%>%
  st_transform(crs = 2163)

ahrf_m<-left_join(usco, ahrf2,
                    by = "cofips")%>%
  filter(is.na(lbrate1618)==F,
         !STATEFP %in% c("02", "15", "60", "66", "69", "72", "78"))%>%
  st_transform(crs = 2163)

glimpse(ahrf_m)
```

There are a total of 2,418 observations in the data, because the HRSA restricts some counties with small numbers of births from the data.

Here is a `ggplot()` histogram of the low birth weight rate for US counties.

```
ahrf_m%>%
  ggplot()+
  geom_histogram(aes(x = lbrate1618))+
  labs(title = "Distribution of Low Birth Weight Rates in US Counties",
       subtitle = "2016 - 2018")+
       xlab("Rate per 1,000 Live Births")+
  ylab ("Frequency")
```

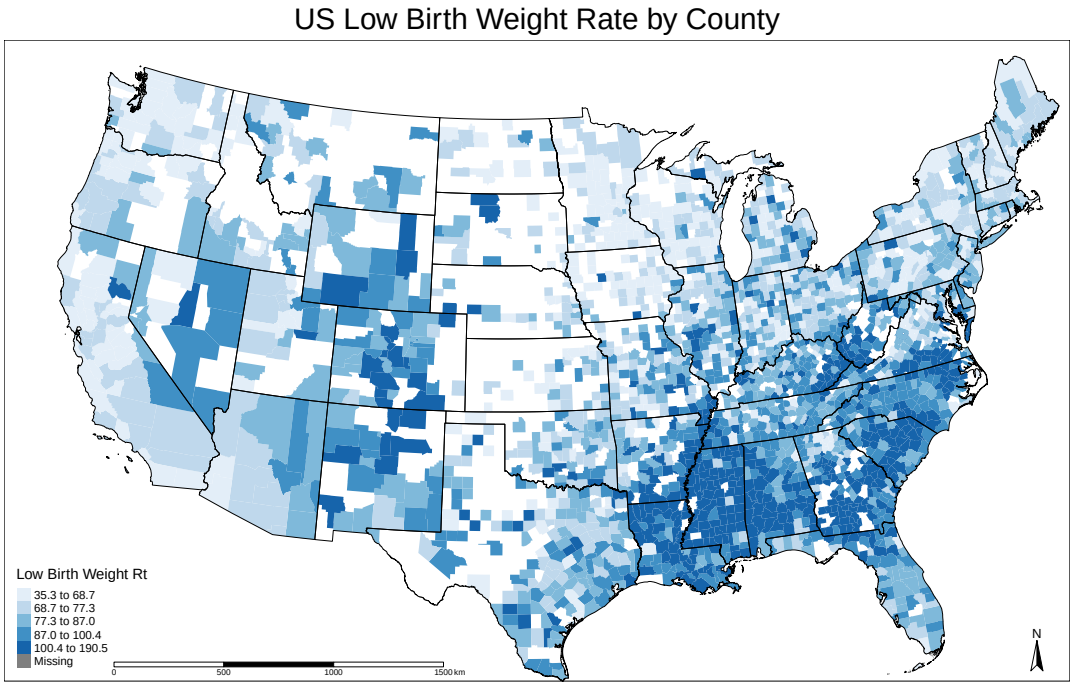
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Here, we do a basic map of the outcome variable for the continental US, and see the highest rates of low birth weight births in the US occur in the southeastern areas of the country. Notice, I do not color the boundaries of the counties in order to maximize the reader's ability to see the variation, instead I show lines between states by overlaying the `sts` layer from above. I also add cartographic options of a scale bar and a north arrow, which I personally believe should be on any map shown to the public.

```
library(tmap)

tm_shape(ahrf_m)+
  tm_polygons(col = "lbrate1618",
              border.col = NULL,
              title="Low Birth Weight Rt",
              palette="Blues",
              style="quantile",
              n=5,
              showNA=T, colorNA = "grey50")+
   tm_format(format= "World",
             main.title="US Low Birth Weight Rate by County",
            legend.position =  c("left", "bottom"),
            main.title.position =c("center"))+
  tm_scale_bar(position = c(.1,0))+
```

```
  tm_compass()+
tm_shape(sts)+
  tm_lines( col = "black")
```



When doing analysis of place-based data, maps are almost a fundamental aspect of the analysis and often convey much more information about the distribution of the outcome than either distribution plots or summary statistics.

8.5 The linear model framework

Probably the most used and often abused statistical model is the linear regression model, sometimes called the OLS model because it is typically estimated by the method of least squares. I do not plan on spending a lot of real estate in this this book talking about the linear model, mostly because lots of times it doesn't get us very far, and there are much more thorough books on this subject, one of my personal favorites being John Fox's text on applied regression (Fox 2016).

This model is typically shown as:

$$y_i = \beta_0 + \sum_k \beta_k x_{ki} + \epsilon_i$$

with the β's being parameters that define the linear relationship between the independent variables x_k, and y, and ϵ_i being the unexplained, or residual portion of y that is included in the model. The model has several assumptions that we need to worry about, first being normality of the residuals, or

$$\epsilon_i \sim Normal(0, \sigma_\epsilon)$$

Where σ_ϵ^2 is the residual variance, or mean square error of the model. Under the strict assumptions of the linear model, the Gauss-Markov theorem says that the unbiased and minimum variance estimates of the β's is:

$$\beta_k = \frac{\sum (y_i - \bar{y})(x_i - \bar{x})}{\sum x_i - \bar{x}^2} = \frac{Cov(x, y)}{Var(x)}$$

Which is often shown in the more compact matrix form:

$$\beta_k = (X'X)^{-1}X'Y$$

We could just as directly write the model in it's distributional form as:

$$y_i \sim Normal(\beta_0 + \sum_k \beta_k x_{ki}, \sigma_\epsilon)$$

or even as:

$$y_i \sim Normal(X'\beta, \sigma_\epsilon)$$

Which I prefer because it sets up the regression equation as the **linear predictor**, or linear combination (in the linear algebra sense) of the predictors and the model parameters for the mean of the outcome. This term, linear predictor, is a useful one, because as you get more and more accustomed to regression, you will see this same structure in every regression model you ever do. This is the fundamental workhorse of regression, where we get an estimated value for every combination of the observed predictors. Moreover, below when I present the **Generalized Linear Model**, it will be apparent that the linear predictor can be placed within a number of so-called **link functions** to ensure that the mean of the outcome agrees with the assumed distribution for the outcome.

8.6 Estimating the linear model in R

The linear model is included in the base R `stats` package, and is accessed by the `lm()` function. In the example below, I use data from the U.S. Health Resources and Services Administration Area Health Resource File (AHRF) to estimate a model of the associations between the poverty rate, the rurality of the county and whether the county is a healthcare shortage area. This is a mixture of continuous (or partially continuous) and categorical predictors.

The basic `lm()` model syntax is specified as `lm ( y ~ x_1 + x_2, data = dataname)` with the `~` operator representing the formula for the model equation, with the outcome on the left and the predictors on the right. You also provide the name of the dataframe which contains the variables specified in the formula in a `data=` argument. For help on the function and to see the other potential arguments use `?lm`.

```
lm1 <- lm (lbrate1618 ~  fampov14 + rucc + hpsa16, data = ahrf_m)
```

This stores the model data and parameter estimates in the object called `lm1`. You can name the object anything you wish, just try to avoid using other R commands as object names. For instance, I wouldn't want to call an object `mean` or `sd` because those are names of functions. The basic way to see the model results is to use the `summary()` function on the model fit.

```
summary(lm1)
```

```
Call:
lm(formula = lbrate1618 ~ fampov14 + rucc + hpsa16, data = ahrf_m)

Residuals:
    Min      1Q  Median      3Q     Max
-90.114 -10.345  -1.229   9.287  77.778

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                 62.26920    1.18901  52.371  < 2e-16 ***
fampov14                     2.25805    0.06846  32.981  < 2e-16 ***
rucc02                      -1.91854    1.20082  -1.598 0.110249
rucc03                      -3.38658    1.25055  -2.708 0.006818 **
rucc04                      -6.65483    1.40639  -4.732 2.36e-06 ***
rucc05                      -5.96283    1.93739  -3.078 0.002110 **
rucc06                      -3.94180    1.14317  -3.448 0.000575 ***
rucc07                      -4.00730    1.28166  -3.127 0.001790 **
rucc08                       0.45112    2.11062   0.214 0.830770
```

```
rucc09                          -3.88365    2.29118  -1.695 0.090202 .
hpsa16partial county shortage -0.66219    1.00767  -0.657 0.511148
hpsa16whole county shortage    2.36214    1.25335   1.885 0.059601 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 16.36 on 2312 degrees of freedom
Multiple R-squared:  0.3697,    Adjusted R-squared:  0.3667
F-statistic: 123.3 on 11 and 2312 DF,  p-value: < 2.2e-16
```

We see that there is a significant positive association between family poverty and the low birth weight rate, we also see that there is a tendency for more rural areas to have lower low birth weight rates than the largest cities (Reference level = `rucc01`). There is a marginally significant association between a county being a healthcare shortage area and the low birth weight rate. Overall the model is explaining about a third of the variation in the outcome, as seen in the adjusted R-Square value of .3667.

8.6.0.1 A not on interpretation

It is important to remember, when describing results for place-based data, to avoid using language centered on individuals. For instance, with reference to the `fampov14` variable, we cannot say that families living in poverty are more likely to have a low birth weight birth, instead, we must focus on discussion on *places* with higher rates of poverty having a higher rate of low birth weight births. Ascribing individual risk from an ecological analysis is an example of the *ecological fallacy* often seen when doing place-based analysis, and we must be aware of it when framing our questions and describing our results.

The `gtsummary` package (Sjoberg et al. 2021) provides a very nice interface to produce much better looking summaries of models.

```
library(gtsummary)

lm1%>%
  tbl_regression(add_estimate_to_reference_rows = TRUE)
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**Beta**	**95% CI**	**p-value**
% Families Below Poverty Level 2014-18	2.3	2.1, 2.4	<0.001
rucc			
01	0.00	—	
02	-1.9	-4.3, 0.44	0.11
03	-3.4	-5.8, -0.93	0.007
04	-6.7	-9.4, -3.9	<0.001
05	-6.0	-9.8, -2.2	0.002
06	-3.9	-6.2, -1.7	<0.001
07	-4.0	-6.5, -1.5	0.002
08	0.45	-3.7, 4.6	0.8
09	-3.9	-8.4, 0.61	0.090
hpsa16			
no shortage	0.00	—	
partial county shortage	-0.66	-2.6, 1.3	0.5
whole county shortage	2.4	-0.10, 4.8	0.060

8.7 Assumptions of the OLS model

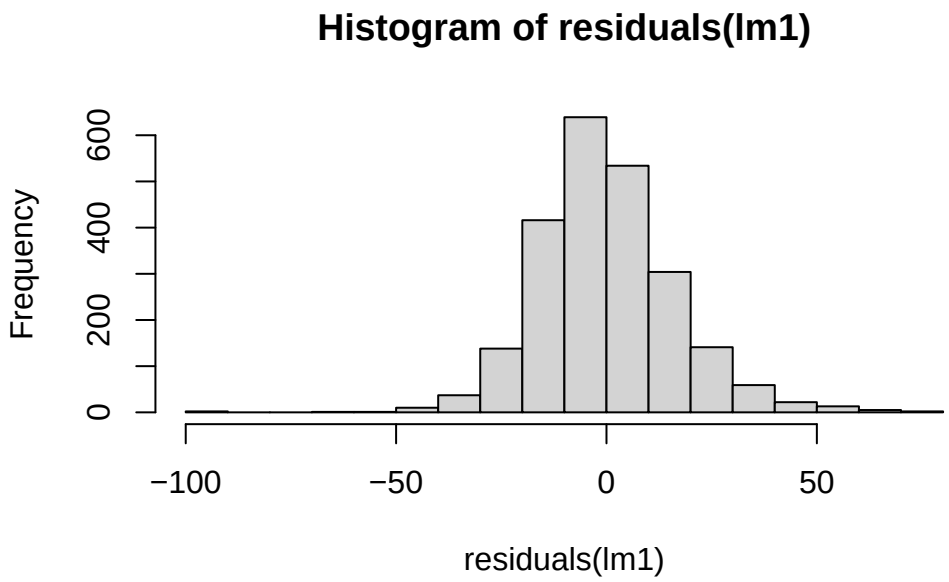
The linear model has several assumptions that we need to be concerned with, the big four are

1) *Normality of residuals*,
2) *Constant variance in residuals, or* homoskedasticity, and
3) *Linearity of the regression function*, and
4) *Independence of observations*

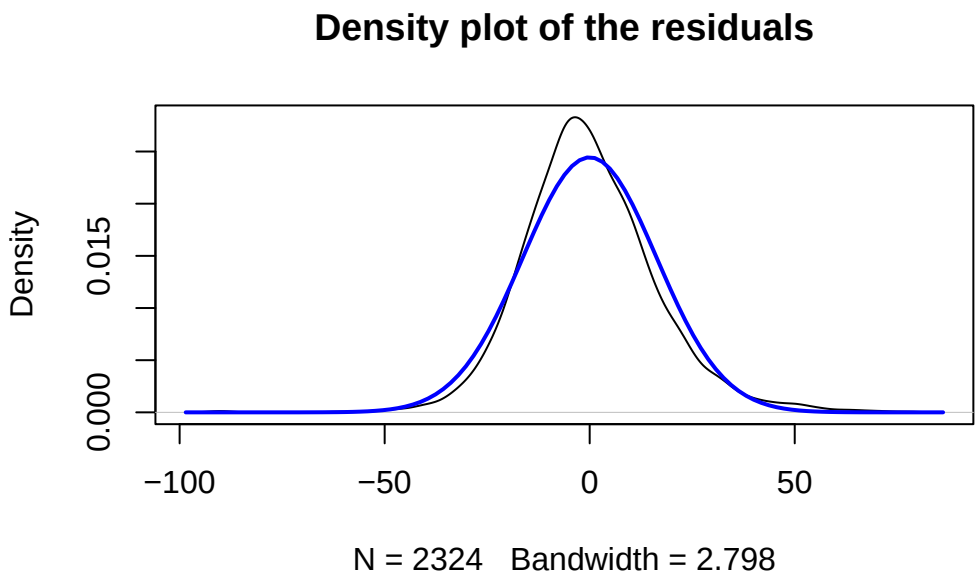
The normality assumption is linked to distributional assumption underlying the linear regression model. This states that the model residuals, calculated as $e_i = (\beta_0 + \sum_k \beta_k x_{ki}) - y_i$, or more compactly as $e_i = \hat{y}_i - y_i$ follow a normal distribution. If the errors around the mean function are not normally distributed, this can be an indicator that the linear model is not appropriate for the outcome under consideration. A commonly used graphical check of this is the *quantile-quantile* or Q-Q plot, which plots the residuals from a model against the hypothetical quantiles from a normal distribution.

We can check these for our model above easily:

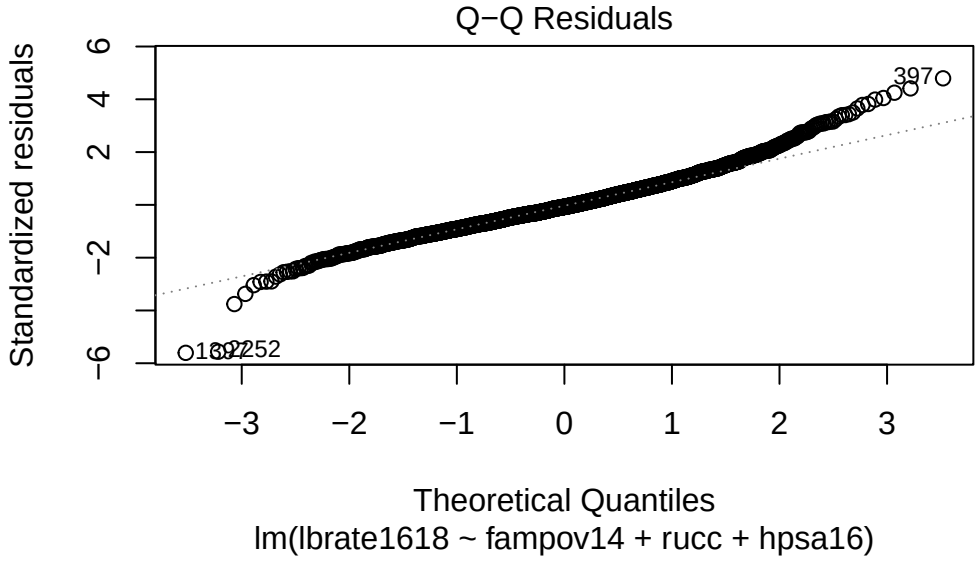
```
hist(residuals(lm1))
```




```
plot(density(resid(lm1)),
     main = "Density plot of the residuals")
curve(dnorm(x,0,sd(resid(lm1))),
       col = "blue", lwd =2, add=TRUE)
```




```
plot(lm1, which = 2)
```

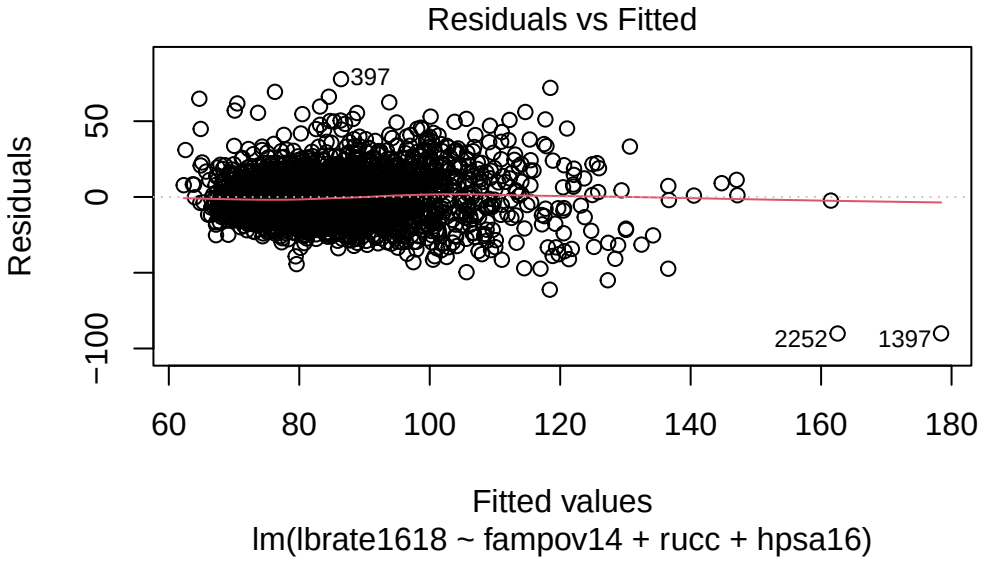


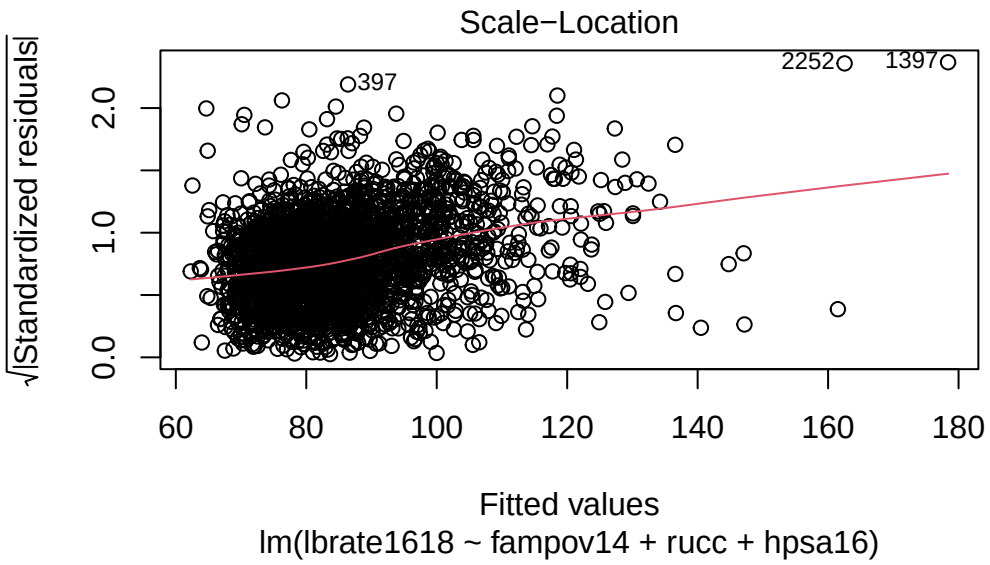
While the overall distribution of the residuals is fairly normal based on the histogram and the comparison to the normal density plot, the q-q plot shows that the tails of the distribution are not well modeled by the normal distribution because there are several observations that are too far below or above the theoretical line (dotted line).

The *homoskedasticity* assumption is also tied to the normal distributional assumption of the model, as see above, if we write the model in its distributional form, $y_i \sim Normal(X'\beta, \sigma_\epsilon)$, the term $ _ $ is a single parameter, meaning that we only have one of these in a linear regression model. This parameter determines the spread of the variation around the mean function. Larger values equal more spread around the mean, and smaller values equal less spread. A commonly used graphical procedure to detect lack of homoskedasticity, or *heteroskedasticity*, is an envelope plot, or a plot of the residuals against the fitted values from the model. Formal tests also exist including the *Breusch-Pagan test* and the modified version of this test developed by Cook and Weisberg [cite]

A graphical check of this assumption is easily done from the model fit:

```
plot(lm1, which = c(1,3))
```





These plots show some evidence that the error variances are non-constant. The first plot has the very characteristic "fish" or "cone" shape, where the error variances increase as the fitted values increase. We can also do a formal test using functions from the `car` package:

```
library(car)
```

```
Loading required package: carData
```

```
Attaching package: 'car'
```

```
The following object is masked from 'package:dplyr':

    recode
```

```
The following object is masked from 'package:purrr':

    some
```

```
ncvTest(lm1)
```

```
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 469.4844, Df = 1, p = < 2.22e-16
```

Which again shows more statistical evidence of the non-constant variance in residuals.

8.7.0.1 Correction for non-constant variance

The assumption of homoskedasticity is important for two reasons, first is related to prediction from the model, but the second is related to the test statistics derived from the model. In order to test our hypothesis that x is related to y, we form the ratio of the β_1 parameter to its error, this is typically either a z, t or Wald χ^2 statistic, depending on which procedure you're using.

$$t = \frac{\hat{\beta}_1}{se(\beta_1))}$$

The term $se(\beta_1)$ is the estimated standard error of the parameter, and is calculated using the ratio of the residual standard deviation and the square root of the sums of squares of the x:

$$se(\beta_1) = \frac{\sigma_\epsilon}{\sqrt{\sum (x - \bar{x})^2}}$$

or in the matrix terms:

$$Var(\beta) = \sigma_\epsilon (X'X)^{-1}$$

if the term σ_ϵ is not constant then the standard error of each parameter in the model is incorrect. Corrections for heteroskedasticity are commonplace in the social sciences, and are usually attributed to White (1980) and MacKinnon and White (1985) with many additions since the original publication, notably Long and Ervin (2000). These corrections use the empirically observed error terms and avoid the assumption of common variance in all residuals.

The `coeftest()` function in the `lmtest` package is one option to correct for heteroskedasticity in regression models. It allows for various correction types, with the "HC3" type (Long and Ervin 2000) being the default for linear models. Below, I show the default tests assuming constant variance and the corrected tests.

```
library(sandwich)
library(lmtest)
```

```
Loading required package: zoo


Attaching package: 'zoo'

The following objects are masked from 'package:base':

    as.Date, as.Date.numeric
```

```
coeftest(lm1)
```

```
t test of coefficients:

                               Estimate Std. Error t value  Pr(>|t|)
(Intercept)                  62.269198   1.189010 52.3706 < 2.2e-16 ***
fampov14                      2.258054   0.068465 32.9813 < 2.2e-16 ***
rucc02                       -1.918536   1.200819 -1.5977 0.1102488
rucc03                       -3.386577   1.250553 -2.7081 0.0068176 **
rucc04                       -6.654825   1.406391 -4.7318 2.359e-06 ***
rucc05                       -5.962826   1.937392 -3.0778 0.0021101 **
rucc06                       -3.941799   1.143175 -3.4481 0.0005746 ***
rucc07                       -4.007304   1.281664 -3.1266 0.0017901 **
rucc08                        0.451120   2.110622  0.2137 0.8307703
rucc09                       -3.883651   2.291182 -1.6950 0.0902020 .
hpsa16partial county shortage -0.662194   1.007671 -0.6572 0.5111478
hpsa16whole county shortage    2.362138   1.253348  1.8847 0.0596007 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
coeftest(lm1, vcov  = vcovHC(lm1, type = "HC3"))
```

```
t test of coefficients:

                             Estimate Std. Error t value  Pr(>|t|)
(Intercept)                  62.26920    1.14941 54.1749 < 2.2e-16 ***
fampov14                      2.25805    0.11349 19.8966 < 2.2e-16 ***
rucc02                       -1.91854    0.97531 -1.9671 0.0492896 *
```

```
rucc03                          -3.38658    1.06364 -3.1839 0.0014722 **
rucc04                          -6.65483    1.20922 -5.5034 4.137e-08 ***
rucc05                          -5.96283    1.95622 -3.0481 0.0023288 **
rucc06                          -3.94180    1.09704 -3.5931 0.0003336 ***
rucc07                          -4.00730    1.30130 -3.0795 0.0020981 **
rucc08                           0.45112    2.42803  0.1858 0.8526203
rucc09                          -3.88365    3.29696 -1.1779 0.2389381
hpsa16partial county shortage -0.66219    0.85442 -0.7750 0.4384068
hpsa16whole county shortage    2.36214    1.26921  1.8611 0.0628553 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The take away in this example is that non-constant variance can affect the standard errors of the model parameter estimates, and in turn affect the test statistics that we base all of our hypothesis tests on. In the particulars of this model there is not a lot of change, the `rucc02` parameter is barley significant once using the corrected standard errors, otherwise we see a very similar pattern in terms of what is significant in the model.

8.7.0.2 Clustered standard errors

Another commonly used correction in regression modeling is the *clustered standard error*. These are commonplace and almost the default in the Stata programming environment, and are widely used in the field of economics. Clustering of standard errors attempts to correct for clustering in the residuals from the regression model. Clustering can happen for a wide variety of reasons, and as we saw in the previous chapter on survey data analysis, is often an artifact of how the data are collected. With place-based data, we may have clustering because the places are close to each other, or because the share some other characteristic that we have not measured in our regression model. In the case of our regression model, and in our descriptive analysis of our outcome, the map shown prior may indicate some form of *spatial correlation* in the outcome. While there are models to deal with such non-independence in place-based data, they are not a subject I will touch on here. Instead, we may use the state which each county is in as a proxy for the spatial clustering in the outcome, as one example of potential of a clustering term.

```
coeftest(lm1,
         vcov  = vcovCL(lm1, cluster = ahrf_m$state))
```

```
t test of coefficients:

                               Estimate Std. Error t value  Pr(>|t|)
```

```
(Intercept)                    62.26920    2.43737 25.5477 < 2.2e-16 ***
fampov14                        2.25805    0.28830  7.8322 7.243e-15 ***
rucc02                         -1.91854    1.15345 -1.6633  0.096387 .
rucc03                         -3.38658    1.42381 -2.3785  0.017462 *
rucc04                         -6.65483    1.35050 -4.9277 8.912e-07 ***
rucc05                         -5.96283    2.60314 -2.2906  0.022075 *
rucc06                         -3.94180    1.28117 -3.0767  0.002118 **
rucc07                         -4.00730    2.11299 -1.8965  0.058017 .
rucc08                          0.45112    2.59621  0.1738  0.862069
rucc09                         -3.88365    4.73238 -0.8207  0.411927
hpsa16partial county shortage -0.66219    1.18695 -0.5579  0.576971
hpsa16whole county shortage    2.36214    1.80407  1.3093  0.190549
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this case, the `rucc02` and `rucc07` terms become marginally significant and the healthcare shortage areas lose their marginal significance.

8.7.0.3 Weighted Least Squares

Another method of dealing with non-constant error variance is the method of weighted least squares. This method modifies the model somewhat to be:

$$y \sim Normal(\beta_0 + \sum_k \beta_k x_{ki}, \sigma_{\epsilon_i})$$

Where the term σ_{ϵ_i} represents the different variances for each observation. The weights in the model are often variables that represent an underlying factor that affects the variance in the estimates. In demography this is often the population size of a place, as places with smaller population sizes often have more volatility to their rate estimates. This approach has been used in the spatial demographic modeling of county mortality rates in the United States by several authors (McLaughlin et al. 2007; Sparks and Sparks 2010).

```
lm2 <- lm(lbrate1618 ~  fampov14 + rucc + hpsa16,
          data = ahrf_m,
          weights = popn)
summary(lm2)
```

```
Call:
lm(formula = lbrate1618 ~ fampov14 + rucc + hpsa16, data = ahrf_m,
```

```
    weights = popn)

Weighted Residuals:
   Min     1Q Median     3Q    Max
-42957  -2122   -106   2333  27618

Coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                   65.54503    0.98536  66.519  < 2e-16 ***
fampov14                       1.83332    0.06338  28.928  < 2e-16 ***
rucc02                        -0.63464    0.66623  -0.953 0.340901
rucc03                        -1.67371    0.93793  -1.784 0.074479 .
rucc04                        -4.38387    1.31520  -3.333 0.000872 ***
rucc05                        -3.31398    2.16308  -1.532 0.125642
rucc06                        -1.87074    1.35213  -1.384 0.166630
rucc07                        -2.12596    1.80385  -1.179 0.238693
rucc08                         3.92060    4.33118   0.905 0.365452
rucc09                        -0.21442    4.93030  -0.043 0.965315
hpsa16partial county shortage -1.85070    0.93518  -1.979 0.047938 *
hpsa16whole county shortage    0.21786    1.65417   0.132 0.895229
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4620 on 2312 degrees of freedom
Multiple R-squared:  0.2832,    Adjusted R-squared:  0.2798
F-statistic: 83.05 on 11 and 2312 DF,  p-value: < 2.2e-16
```

We see that by including the population size weights in the model, most of the parameters are no longer significant in the analysis, but the weights have not dealt with the non-constant variance issue totally:

```
ncvTest(lm2)
```

```
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 84.49228, Df = 1, p = < 2.22e-16
```

But the overall size of the non-constant variance test is much lower than it was for the original model.

8.7.1 Linearity assumption

The linearity assumption of the model assumes that the true underlying relationship in the data can be modeled using a linear combination of the predictors and the parameters. I think a lot of people think this means that you cannot include square or polynomial terms in a regression model, but that is not the case. The assumption is concerned with the linearity of the parameters, not the predictor variables themselves. For example the standard linear model with one predictor, x is written:

$$y = \beta_0 + \beta_1 x_1 + \epsilon$$

Which is clearly the equation for a straight line with y-intercept β_0 and slope β_1. We also see that the parameters combine in a linear (additive) fashion. This is the assumption of the model, and can also be seen when expressing this equation using vector notation

$$y = x'\beta$$

Because the term $x'\beta$ is the inner product of the β parameters and the information from x. If we include the square of x in the model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \epsilon$$

The same inner, additive product of the βs and the xs is still the same. If we constructed a model that was non-linear function of the βs, such as:

$$y = \beta_0 + \beta_1 x_1 * e^{\beta_2 x_1^2} + \epsilon$$

The the model would no longer be linear in the parameters because we introduce a nonlinearity by exponentiating the β_2 parameter inside the mean function (not that this is a real model, just as an example).

When this actually comes into our experience is when our data are actually generated by a nonlinear process, such as a time series with seasonality included, which may oscillate between seasons (such as temperature, or rainfall), for instance a situation such as this arises:

Where the variable y is actually generated from a cosine curve with noise. Since the linear regression of y on x forces the y to change linearly with x, the model is absolutely unable to recover the underlying pattern in the data.

A note on splines

If the data clearly do not display a linear trend, I personally automatically look to *splines* as a means to model the non-linearity in relationship. Splines are a method of constructing a