Lab Guide to Quantitative Research Methods in Political Science, Public Policy & Public Administration

Joseph Ripberger, Cody Adams, Alex Davis, and Josie Davis

Contents

Copyright

Published by University of Oklahoma Libraries

Norman, Oklahoma

Download this book for free at [insert handle URL]

ISBN: X
DOI: X

Preface

This book is a companion to *Quantitative Research Methods for Political Science, Public Policy and Public Administration (With Applications in R): 4th Edition*, an open-source text book that is available here. It grew from our experiences teaching introductory and intermediate quantitative methods classes for graduate students in Political Science and Public Policy at the University of Oklahoma. We teach these courses using a format that pairs seminars on theory and statistics with exercises that focus on applications in `R`. We use the text book to motivate seminars and this book to guide exercises. The book is written in `R Markdown` and `bookdown`. While a "complete" copy of the book is available, we suggest that students and instructors use the "raw" `.Rmd` files for exercises. These materials are available in our `GitHub` repository here.

Currently, the labs use survey data from the *Meso-Scale Integrated Socio-geographic Network (M-SISNet)* to explain concepts and methods. The M-SISNet is a quarterly survey of approximately 1,500 households in Oklahoma that is conducted with support of the National Science Foundation (Grant No. IIA-1301789). One wave of M-SISNet data is available in our `GitHub` repository. Readers can learn more about the project and download the remaining waves here. We welcome students and instructors to explore and use these data, but we also encourage instructors to modify the book by incorporating data that is more relevant to the course they are teaching.

Acknowledgments

By intent, this book represents an open-ended group project that changes over time as new ideas and new instructors become involved in teaching graduate methods in the University of Oklahoma Political Science Department. Early ideas and materials came from Hank Jenkins-Smith, who began teaching these courses (in R) in 2010. Graduate assistants Matthew Nowlin, Tyler Hughes, and Aaron Fister were responsible for developing the labs for Hank's courses. Joseph Ripberger began teaching these courses in 2015. Wesley Wehde was responsible for updating the labs for his courses. After many years of informal development, Alexander Davis, Cody Adams, and Josie Davis took the labs and turned them in to this book. We thank everyone, especially our students, for helping us write and continue to improve this book.

Requirements

Each chapter is written in R Markdown, allowing readers to follow along and interact with examples via their favorite integrated development environment (e.g., RStudio), or to knit as a PDF. For more information on R Markdown, visit: https://rmarkdown.rstudio.com/.

Statistical analysis in R is possible via a plethora of publicly available packages. These chapters introduce various functions from select packages to expand upon the base functions of R, and therefore the following packages are required:

1. car
2. reshape2
3. descr
4. tidyverse
5. skimr
6. memisc
7. stargazer
8. MASS
9. pscl
10. broom
11. zeligverse
12. plotly
13. vcd

14. HistData
15. sfsmisc
16. interplot
17. sandwich
18. DAMisc

For convenience, executing the *setup.R* script or knitting this document within RStudio will install these packages if they are not already installed.

1 Introduction to R, RStudio, and R Markdown

In these labs and the corresponding textbook, we will use the `R` programming language to learn statistical concepts and analyze real-world data. Before we dive into the details, this lab will provide an introduction to the `R` language, RStudio, and R Markdown - as well as how the three interact.

To put it simply - `R` is the actual programming language, RStudio is a convenient interface in which to use it, and R Markdown is a specific type of file format designed to produce documents that include both code *and* text.

1.1 Introduction to R, the language

`R` itself is a language and environment for statistical computing and graphics, which operates as an integrated suite of software facilities for data manipulation, calculation, and graphical display. It includes:

- an effective data handling and storage facility,
- a suite of operators for calculations,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display, either on-screen or on hardcopy, and
- a well-developed, simple and effective programming language

`R` is a powerful and effective tool for computing, statistics and analysis, and producing data visualizations. However, many applications exist that can do these or similar things. `R` has a number of benefits that make it particularly useful, though.

1.1.1 Benefits of R

First, `R` is both **open source** and **free**. This allows you to take the tool and the skills you've learned with you wherever you go; you are not dependent on your employer to buy or have a license of a particular software. This is especially relevant as many other software with similar functionality often cost hundreds, if not thousands, of dollars for a single license.

The open source nature of `R` has also resulted in a robust set of users across a wide variety of disciplines who are constantly updating and revising the language. R therefore has some of the most up-to-date and innovative functionality and methods available to its users, should they know where to look.

1.1.2 Packages

Within `R`, these functions and tools are often implemented as packages. Packages allow advanced users of R to contribute statistical methods and computing tools to the general users of `R`. These packages are reviewed and vetted before being made available for public use; they are often frequently updated as well. In the first lab, we will install some basic packages that are frequently used throughout the course.

1.2 RSudio

RStudio is an integrated development environment (IDE) that makes `R` a bit more user-friendly. While it is not the only way to use `R`, it provides a helpful and intuitive interface for writing and editing code, as well as creating documents and reports. It is not, however, a requirement for using the `R` language.

Additionally, it is important to note that RStudio is an entirely separate piece of software - it will need to be downloaded separately from `R`.

1.2.1 Navigating RStudio

As you can see, the RStudio interface is primarily composed of 4 quadrants.

In the upper-left corner is the **source pane**. This is the primarily location where most of your work will take place. You will write and edit collections of code - or R Scripts - here. When working in the source pane, your code will not compile untl you tell it to run; this allows you the flexibility to work at your own pace, as well as to save your work.

In the lower-left corner is the **console**, or the command window. The console is the powerhouse of the software; this is where `R` actually evaluates code. While you can type code directly into the console and receive immediate results, it is advisable to stick to the source pane while you are learning how to use R. Running code in the source pane will automatically produce output in the console pane.

The upper-right quadrant contains the **Environment** and **History** tabs. The Environment tab displays a list of all the data objects that you havae defined in your current R session, as well as some basic details about the data (such as the number of observations and variables in each). The History tab contains an archive of all the commands you've run in the current session.

Finally, the lower-right quadrant holds a number of helpful navigation tabs. The "Files" tab displays your hard drive's own file directory for easy access. The "Plots" tab will show the plots and visualzations you have created in your current R session. The "Packages" tabs shows a list of all the packages currently installed, as well as an indication of whther or not that are loaded in the current session. The "Help" tab is, unsurprisingly, the help menu.

Until you are comfortable writing and executing code to analyze data, the RStudio interface can seem intimidating. Remember - since these are open source software, there are plenty of resources online to help as well. A "cheat sheet" for the RStudio IDE can be found here.

1.3 R Markdown

Markdown is a tool for converting plain text into formatted text. R Markdown utilizes the markdown syntax in order to combine formatted text with code in a single document.

Within RStudio, R Markdown is a specific type of file format for making dynamic documents. It allows you to simultanesouly 1. save and execute code, and 2. produce high quality documents that include both code and text

For the purposes of our labs, R Markdown allows us to include code chunks and the text that helps explain them in an easy-to-read manner. For your own use, R Markdown will allow you to write documents and reports that include traditional formatted text, as well as the data visualizations you make in class, and present them both together in a high quality professional document.

2 Basics of R

This chapter serves as a primer to `R` by introducing the basics. It is advised to follow the lab via the `.rmd` file within RStudio rather than solely the compiled PDF. This way, students can experiment with code in the "code blocks" provided. **Note:** In text R code is displayed in a `fixed-width` font.

2.1 R, as a Calculator

The first thing to know about R is that it is essentially a large calculator capable of performing arithmetic:

```
1 + 1
```

```
## [1] 2
```

```
8 * 8
```

```
## [1] 64
```

```
2 ^ 8 # exponent
```

```
## [1] 256
```

```
(5 + 2) ^ 4
```

```
## [1] 2401
```

```
5 + 2 ^ 4
```

```
## [1] 21
```

R also supports elementary and algebraic functions such as log and square root.

```
log(100)
```

```
## [1] 4.60517
```

```
sqrt(49)
```

```
## [1] 7
```

2.1.1 Order of Operations

R solves equations according to the order of operations, "PEMDAS":

1. Parentheses
2. Exponents
3. Multiplication
4. Division
5. Addition
6. Subtraction

Watch this video for a refresher on the order of operations: https://www.youtube.com/watch?v=94yAmf7GyHw

Try this! Using R, solve: (5 + 1) ^ 4 / (9 - 2) ^ 3

2.2 Objects

R is an "object oriented" programming language. Put simply, R uses objects to store attributes. Objects are created by assigning an attibute to them via the `<-` operation. You can always view the attribute of an object by typing the object name.

```
object1 <- 10 + 10
object1
```

```
## [1] 20
```

Try this! Create a new object below; you can name it almost anything!

R includes various functions for managing created objects. The `ls()` function lists all existing objects.

```
ls()
```

```
## [1] "ds"           "listObject"   "object.One"   "Object.One"
## [5] "object1"      "packages"     "pagebreak"    "pkg"
## [9] "vectorObject"
```

The `rm()` function removes existing objects.

```
rm(object1)
```

There is no strict convention for naming objects; however, there are best practices:

1. Avoid spaces; use *underscores*, periods, or CamelCase (or camelCase) for long object names
   - e.g., This_is_a_long_name, This.Is.A.Long.Name, thisIsALongName
2. Avoid names of existing functions or reserved R objects
   - e.g., `mean` or `sum`
3. Be descriptive but keep it short (less than 10 characters)
4. Avoid special characters
   - e.g., ? $ % ^ &
5. Numbers are fine, but names cannot begin with numbers.
   - e.g., `object1` is ok, but `1object` is not

Important: Object names are case sensitive. - e.g., `object.One` and `Object.One` refer to two separate objects

```
object.One <- 10 + 10
Object.One <- 5 + 5

object.One
```

```
## [1] 20
```

```
Object.One
```

```
## [1] 10
```

2.3 Functions

In addition to elementary and algebraic functions, R includes functions that simplify statistical analysis. For example, the `mean()` function calculates the mean.

Note: Sometimes `na.rm = TRUE` is necessary within the paranetheses to instruct R to ignore missing data.

```
mean(cars$dist)
```

```
## [1] 42.98
```

R comes with a vairety of "built in" data sets, such as the `cars` data set, which contains some information about about cars. This data set is used below to demonstrate and/or experiment with R functions.

A note about syntax: the dollar sign, `$`, is used to indicate the variable of interest relative to a data set. This is important in the case of multiple data sets that contain variables of the same name. In the previous code, R calculated the mean using the `dist` variable within the `cars` data set by specifying `cars$dist`

To ignore missing data when calculating the mean of `dist`, include the `na.rm = TRUE` argument within the paranetheses as follows.

```
mean(cars$dist, na.rm = TRUE)
```

```
## [1] 42.98
```

Note: The mean is exactly the same because there is no missing data in the `dist` variable.

2.3.1 Object Types

Object types are improtant in R and the type of an object is contingent on the attribute stored by the object. For example, an object storing characters (e.g., "blue") has a different type than an object storing a number (e.g., 1). Use of R functions is contingent on the type of objects. For example, functions like `mean()` work only for objects containing numbers. The R `str()` function describes the structure of objects and functions.

```
str(mean)
```

```
## Formal class 'standardGeneric' [package "methods"] with 8 slots
##   ..@ .Data     :function (x, ...)
##   ..@ generic   : chr "mean"
##   .. ..- attr(*, "package")= chr "base"
##   ..@ package   : chr "base"
##   ..@ group     : list()
##   ..@ valueClass: chr(0)
##   ..@ signature : chr "x"
##   ..@ default   :Formal class 'derivedDefaultMethod' [package "methods"] with 4 slots
##   .. .. ..@ .Data  :function (x, ...)
##   .. .. ..@ target :Formal class 'signature' [package "methods"] with 3 slots
##   .. .. .. .. ..@ .Data  : chr "ANY"
##   .. .. .. .. ..@ names  : chr "x"
##   .. .. .. .. ..@ package: chr "methods"
##   .. .. ..@ defined:Formal class 'signature' [package "methods"] with 3 slots
##   .. .. .. .. ..@ .Data  : chr "ANY"
##   .. .. .. .. ..@ names  : chr "x"
##   .. .. .. .. ..@ package: chr "methods"
##   .. .. ..@ generic: chr "mean"
##   .. .. .. ..- attr(*, "package")= chr "base"
##   ..@ skeleton  : language (new("derivedDefaultMethod", .Data = function (x, ...)  UseMethod("mean")
```

```
str(object.One)
```

```
##  num 20
```

```
str(cars)
```

```
## 'data.frame':    50 obs. of  2 variables:
##  $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

```
str(cars$dist)
```

```
##  num [1:50] 2 10 4 22 16 10 18 26 34 17 ...
```

The `str()` function described `mean` as a function, `object.One` as a numeric object, the `cars` data as a data frame, etc.

Previously, objects were introduced as a method of storing single attributes, either a specified value or the result of arithmetic. In addition, objects can contain a collection of data via a vector or list. In mathematics and physics, a vector is defined as a quantity of both direction and magnitude. In R, vectors are defined as a collection of data of the same type. The `c()` function creates a vector.

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

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

```
str(vectorObject)
```

```
##  num [1:3] 1 2 3
```

Further, a list is defined as a collection of multiple data types.

```
listObject <- list("your name", 1, F)
listObject
```

```
## [[1]]
## [1] "your name"
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] FALSE
```

```
str(listObject)
```

```
## List of 3
##  $ : chr "your name"
##  $ : num 1
##  $ : logi FALSE
```

Note: The structure of the list object consists of a character, a number, and a logic (True/False).

2.4 Packages

Packages expand R to include additional functions important to statistical analysis.

2.4.1 Installing Packages

Installing packages in R can be performed via `install.packages("packagename")`, whereby the name of the desired package must be within quotation marks.

Note: Occasionally a package may require dependencies for installation. The dependencies can be automatically installed along with the package by including the `dependencies = TRUE` argument within the `install.packages()` function.

Try this! Use this code to install the following packages:

1. `car`
2. `psych`
3. `memisc`
4. `Rcpp`

2.4.2 Loading Packages

After installation, packages must be loaded to use their functions within R. Packages are loaded via `library(packagename)`. **Note:** Unlike the `install.packages()` function, the `library()` function does not require quotation marks around the package name.

```
library(car)
library(psych)
library(memisc)
```

```
library(Rcpp)
library(rmarkdown)
library(knitr)
```

Note: The `memisc` package contains object/package conflicts with the `car` package for the `recode` object. A conflict occurs when two packages contain objects (e.g., functions) of the same name. A conflict will not prevent loading packages; however, use of a specific package's object requires an explicit call to the desired parent package. For example, to use the `recode()` function from `car`, the `car::recode(variable)` statement will explicitly call `recode()` from the `car` package. Vice versa, `memisc::recode()` will explicitly call `recode()` from the `memisc` package.

2.4.3 Updating Packages

Most packages are regularly updated. The `old.packages()` function compares installed packages to their latest versions online.

The `update.packages()` function updates out of date packages.

Note: Updating packages requires consent. The `ask = FALSE` argument will skip the additional consent step to save time.

```
# update.packages()
```

The `library()` function lists currently loaded packages.

```
library()
```

As previously demonstrated, occasionally conflicts exist between packages. The `conflicts()` fubction lists conflicts between loaded packages.

```
conflicts()
```

```
##   [1] "show"              "show"              "factorize"
##   [4] "show"              "traceplot"         "sim"
##   [7] "summary"           "Arith"             "as.array"
##  [10] "coerce"            "Compare"           "format"
##  [13] "initialize"        "Math"              "Math2"
##  [16] "print"             "rename"            "show"
##  [19] "style"             "summary"           "Summary"
##  [22] "negative.binomial" "select"            "%>%"
##  [25] "kable"             "%>%"               "%>%"
##  [28] "%>%"               "arrange"           "arrange_"
##  [31] "collect"           "contains"          "distinct"
##  [34] "distinct_"         "do"                "do_"
##  [37] "ends_with"         "everything"        "filter"
##  [40] "filter_"           "group_by"          "group_by_"
##  [43] "groups"            "last"              "matches"
##  [46] "mutate"            "mutate_"           "num_range"
##  [49] "one_of"            "recode"            "rename"
##  [52] "rename_"           "select"            "select_"
##  [55] "slice"             "slice_"            "starts_with"
##  [58] "summarise"         "summarise_"        "syms"
##  [61] "transmute"         "transmute_"        "ungroup"
##  [64] "%>%"               "reduce"            "%>%"
##  [67] "as_tibble"         "contains"          "ends_with"
##  [70] "everything"        "expand"            "last_col"
```

```
##  [73] "matches"           "num_range"         "one_of"
##  [76] "pack"              "starts_with"       "tibble"
##  [79] "tribble"           "unpack"            "add_row"
##  [82] "as_data_frame"     "as_tibble"         "data_frame"
##  [85] "data_frame_"       "frame_data"        "glimpse"
##  [88] "lst"               "lst_"              "tbl_sum"
##  [91] "tibble"            "tribble"           "trunc_mat"
##  [94] "type_sum"          "arrow"             "enexpr"
##  [97] "enexprs"           "enquo"             "enquos"
## [100] "ensym"             "ensyms"            "expr"
## [103] "last_plot"         "quo"               "quo_name"
## [106] "quos"              "stat"              "sym"
## [109] "syms"              "unit"              "vars"
## [112] "%+%"               "alpha"             "logit"
## [115] "rescale"           "sim"               "smiths"
## [118] "logit"             "recode"            "some"
## [121] "coef"              "coefficients"      "contr.sum"
## [124] "contr.treatment"   "contrasts"         "contrasts<-"
## [127] "cov2cor"           "df.residual"       "filter"
## [130] "fitted"            "lag"               "predict"
## [133] "residuals"         "toeplitz"          "update"
## [136] "vcov"              "image"             "layout"
## [139] "plot"              "head"              "prompt"
## [142] "tail"              "npk"               "Arith"
## [145] "cbind2"            "coerce"            "Compare"
## [148] "initialize"        "kronecker"         "Logic"
## [151] "Math"              "Math2"             "Ops"
## [154] "rbind2"            "show"              "Summary"
## [157] "%in%"              "all.equal"         "as.array"
## [160] "as.factor"         "as.matrix"         "as.ordered"
## [163] "body<-"            "chol"              "chol2inv"
## [166] "colMeans"          "colSums"           "crossprod"
## [169] "det"               "determinant"       "diag"
## [172] "diag<-"            "diff"              "drop"
## [175] "formals<-"         "format"            "intersect"
## [178] "isSymmetric"       "kronecker"         "labels"
## [181] "mean"              "merge"             "norm"
## [184] "Position"          "print"             "qr"
## [187] "qr.coef"           "qr.fitted"         "qr.Q"
## [190] "qr.qty"            "qr.qy"             "qr.R"
## [193] "qr.resid"          "rcond"             "row.names"
## [196] "rowMeans"          "rowSums"           "sample"
## [199] "setdiff"           "setequal"          "solve"
## [202] "subset"            "summary"           "t"
## [205] "tcrossprod"        "union"             "unique"
## [208] "unname"            "which"             "within"
## [211] "zapsmall"
```

The `detach()` function detaches packages and is an alternative method to resolve conflicts. Supplying the `unload = TRUE` argument within the `detach()` function will unload the package. For example, to resolve the `recode()` function conflict between `car` and `memisc`, the memisc package can be detached and unloaded as follows: `detach(package:memisc, unload = TRUE)`.

2.5 R Help

R includes a help function to assist with functions, accessible by including a `?` prior to the function name.

```
? mean
```

Note: The help documentation will display in the bottom right quadrant of RStudio. Alternatively, typing the function name into the help search bar will yield a similar result.

To search all of R documentation for help about a function, use `??`.

```
?? mean
```

Note: Google is a valuable tool for finding help. Large communities like StackExchange provide answers and explanations to common issues in R. At times, a particular problem may seem unique, but someone else has almost certainly had the same problem and the solution likely can be found online.

2.6 Setting a Working Directory

The working directory is the location where files are accessed and saved within a R session. Normally, the working directory is set at the beginning of every R file. The working directory should be set and the class data loaded at the beginning of each lab.

There are two methods of setting the working directory. First, the `setwd()` function can be used with the directory path. For example, `setwd("C:/Directory\_to\_folder/")`.

Note: Forward slashes are used in place of backward slashes for directory paths.

Second, within RStudio, the "Session" tab will allow you to set the working directory. The following steps provide guidance to the "Session" tab functionality:

1. Click the "Session" tab.
2. Select "Set Working Directory."
3. Select "Choose Directory."
4. Select the working directory.

The `getwd()` function returns the set working directory.

```
getwd()
```

```
## [1] "/Users/josephripberger/Documents/GitHub/qrmlabs"
```

2.7 Importing Your Data

R can read many different file types, including text files, Excel files, Google Sheet files, SPSS files, and Stata files. It can even read data sets directly from websites. The file type determines the function that is necessary import a data set. For example, CSV files use the function `read.csv` to import a dataset. Here is an example that uses this function:

```
ds <- read.csv("Class Data Set Factored.csv", header = TRUE)
```

This line of code saves the data set in `Class Data Set Factored.csv` to an object called ds (short for data set). The `header = TRUE` argument tells R that the first row in the data set provides column (or variable) names.

Note: This code assumes that `Class Data Set Factored.csv` is in the working directory. To check, use `list.files()`. If the file containing the data set is not in the working directory, provide the complete file path in the `read.csv` function, like this:

```
ds <- read_csv("https://github.com/ripberjt/qrmlabs/raw/master/Class%20Data%20Set%20Factored.csv",
header = TRUE)
```

3 Formatting, Describing, and Visualizing Data

This lab discusses the basics of formatting, describing, and visualizing data. The following packages are required for this lab:

1. car
2. reshape2
3. descr
4. psych
5. tidyverse
6. skimr

Reminder: Lab One introduced how to install packages via the `install.packages()` function.

Note: These packages should already be installed from the class initialization script.

The installed packages require loading at the beginning of a R session. Remember that this can be done via the `library()` command.

3.1 Factoring

The previous lab introduced several types of objects in R. For the purpose of this lab, objects are classified into two broad groups: factors and numerics.

Factors are nominal data that a label is applied to. - e.g., race, gender, party identification, etc.

Numerics are data consisting of numbers, which are ranked (ordinal and interval).

When data is read into R (e.g., by importing a .csv file), R automatically classifies the data by type. When data is recognized as non-numeric, R will classify it as a factor.

In the class dataset, the variable `f.party.2` is a variable consisting of factors that identifies the party affilation of the individuals who answered a survey. The `table()` function describes the variable by category.

```
table(ds$f.party.2)
```

```
##
##  Dem  Ind  Rep
##  869  356 1185
```

Attempting to take mean of the `f.party.2` variable inevitably fails.

```
mean(ds$f.party.2)
```

```
## Warning in mean.default(ds$f.party.2): argument is not numeric or logical:
## returning NA
```

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

Party affiliation is nominal data that cannot be described via mean. Consequently, R will error when attempting to calculate the mean or median for factor variables.

The `str()` function describes the structure of the `f.party.2` variable:

```
str(ds$f.party.2)
```

```
##  Factor w/ 3 levels "Dem","Ind","Rep": 3 1 3 NA 3 2 1 3 3 NA ...
```

The `f.party.2` variable is a factor with three levels: "Dem", "Ind", and "Rep."

With many data sets, data are initially coded in numbers and factored afterwards. For example, the `party` variable in the `ds` data set is numeric:

```
table(ds$party)
```

```
##
##    1    2    3    4
##  869 1185  356   91
```

Without a codebook to decipher the numeric values, statisticians are unable to explain what 1, 2, 3, or 4 represent within the `ds` data set. Factoring data remedies this issue. Factoring data in R serves two broad purposes:

1. Applies labels to data
2. Tells R to treat the data as categorical/nominal

At a very basic level, a variable can be factored without applying labels. At minimum, R will treat the data as categorical. This method is sufficient when a variable requires quick factoring. The basic syntax is to use the `factor()` function. **Note:** Best practice for factoring an existing variable is to create an additional variable within the same data set. Best practice then suggests to append the new variable with `f.` to indicate the variable is factored.

```
ds$f.gender <- factor(ds$gender)
```

Reminder: The `$` sign shows R which data set to draw the variable from, or to tell R what data set to assign the new factored variable to. The new factored variable can be described via the `table()` function.

```
table(ds$f.gender)
```

```
##
##    0    1
## 1520 1026
```

While this factored variable is split into two categories, it is not apparent which numbers represent male and female. Labels should be assigned to the numbers to clarify the relationship between the numbers and their meaning. When factoring a variable, R requires the number of levels within the variable and the corresponding labels. In the `f.gender` variable there are two levels, 0 and 1, that require women and men labels.

```
ds$f.gender <- factor(ds$gender, levels = c(0, 1),
                      labels = c("Women", "Men"))
table(ds$f.gender)
```

```
##
## Women   Men
##  1520  1026
```

The vector function, `c()`, tells R the levels and labels of the variable.

The `f.party` variable should also be factored, where 1 = Dem, 2 = Rep, 3 = Ind, and 4 = Other, and a table of the variable should be created.

```
ds$f.party <- factor(ds$party, levels = c(1, 2, 3, 4),
                     labels = c("Dem", "Rep", "Ind", "Other"))
table(ds$f.party)
```

```
##
##   Dem   Rep   Ind Other
##   869  1185   356    91
```

The structure of the `f.party` variable describes a factor with four levels.

```
str(ds$f.party)
```

```
##  Factor w/ 4 levels "Dem","Rep","Ind",..: 2 1 2 NA 2 3 1 2 2 4 ...
```

There are other types of data conversions as well. In most cases, the basic syntax is the same as the `factor()` function, except the function names are `numeric()` or `integer()`. **Note:** For most purposes, numeric and integer are the same.

3.1.1 Coerce Factoring

Sometimes the typical commands do not work and variables must be coerced. When a variable is coerced, R is instructed to treat the variable as if it were a different object type. The coercive functions are:

1. `as.factor()`
2. `as.numeric()`
3. `as.integer()`

Try it! Convert a factor variable into a numeric variable using the `numeric()` function.

This fails, so the coerce function should be employed.

```
ds$n.party <- as.numeric(ds$f.party)
```

Examine the new variable with the `table()` and `str()` functions.

```
table(ds$n.party)
```

```
##
##    1    2    3    4
##  869 1185  356   91
```

```
str(ds$n.party)
```

```
##  num [1:2547] 2 1 2 NA 2 3 1 2 2 4 ...
```

3.2 Recoding

In R, the `recode()` function recodes values within a variable. There many reasons to recode a variable, including:

1. To correct or change incorrect data.
2. To restructure data, making it easier for calculations.
3. To emphasize some intended effect.

For example, a recode is necessary to look at age groups instead of exact ages. If everyone in the survey reported their exact age (e.g., 54, 23, etc.) and age groups are necessary (e.g., 18-25, 26-35, etc.), the `recode()` function is useful.

To perform a recode, follow this basic syntax: `recoded.variable <- recode(old.variable, "recode commands")`

Performing a recode is best demonstrated via example. The best practice for recoding variables suggest creating a new variable with a `r.` prefix.

Within the class data set is an `ideol` variable describing ideology. Currently, the ideology variable goes from 1 to 7, with 1 being very liberal and 7 being very conservative. The ideology variable can be recoded from seven levels to three, liberal, moderate, and conservative, via the `recode()` function. Values from 1 to 2 will

recode as 1, 3 to 5 will recode as 2, and 6 to 7 will recode as 3. This `recode()` function is provided by the `car` package.

```
ds$r.ideol <- car::recode(ds$ideol, "1:2 = 1; 3:5 = 2; 6:7 = 3;
                          else = NA; NA = NA")
```

The `table()` function describes the result of recoding the `ideol` variable.

```
table(ds$r.ideol, useNA = "always")
```

```
##
##    1    2    3 <NA>
##  401 1084 1039   23
```

Note: `else = NA; NA = NA` is included at the end of the recode function. This instructs R to regard any other responses, whether missing data or data that for some reason is outside the original range, as NA, and to treat all existing NAs as NAs. Sometimes `-99 = -99` or `-99 = NA` requires inclusion in the function as well.

Note: In the recode function, all the recode arguments are required **inside one set of quotation marks**. Each argument is separated with a semicolon. R will generate an error message if commas are used.

Using colons to define ranges will save time. In the `recode()` function, using `3:5 = 1` instructs R to recode all values between 3 and 5 to a 2. This is the preferred method opposed to typing `3 = 2; 4 = 2; 5 = 2`.

There is no standard method to categorize or recode data. Let the research question, model design, and data determine the best approach to recoding. For example, in surveys that ask individuals to support something on a scale of 1 to 4, with 1 as very supportive and 4 as least supportive, perhaps recoding the higher value to indicate greater support is appropriate.

Now let's look at the race variable:

```
table(ds$race)
```

```
##
##    1    2    3    4    5    6    7
## 2227   79  119   10    2   76   23
```

In the `ds` data set, the `race` variable consists of codes: 1 for Caucasian, 2 for African-American, and 3 through 7 for a variety of other races (Native American, Asian, Pacific Islander, 2+ races, and Other).

Try it! Recode this variable to go from 7 levels to 3, where 1 is still Caucasian, 2 is African American, and 3 includes all others.

```
ds$r.race <- car::recode(ds$race, "1 = 1; 2 = 2; 3:7 = 3;
                         else = NA; NA = NA")
table(ds$r.race)
```

```
##
##    1    2    3
## 2227   79  230
```

3.2.1 Factoring and Recoding

Factoring a variable is generally easier subsequent to recoding. Given the `race` variable now consists of 3 levels instead of 7, factoring will add meaningful words in place of the values. The values 1, 2, and 3 can be factored as White, African-American, and Other, respectively.

```
ds$f.race.2 <- factor(ds$r.race, levels = c(1, 2, 3),
                      labels = c("White", "African-American", "Other"))
table(ds$f.race.2)
```

```
##
##            White African-American            Other
##             2227               79              230
```

The same can be done with the `ideol` variable. The values 1, 2, and 3 can be factored as Liberal, Moderate, and Conservative, respectively.

```
ds$f.ideol <- factor(ds$r.ideol, levels = c(1, 2, 3),
                     labels = c("Liberal", "Moderate", "Conservative"))
table(ds$f.ideol)
```

```
##
##      Liberal     Moderate Conservative
##          401         1084         1039
```

3.2.2 Creating a Dummy Variable

Factoring and recoding permit the creation of dummy variables. A dummy variable is a binary indicator (0 or 1) of some category, to test for an effect from a particular category. Dummy variables are prominent in political science, so it is imperative to understand how to create and use them.

A dummy variable will be created with the recoded `race` variable to indicate whether the respondent identified as African-American (1) or not (0). Recall that in the `r.race` variable, African-American is coded as 2.

```
ds$r.AfAm <- car::recode(ds$r.race, "2 = 1; else = 0; NA = NA")
table(ds$r.AfAm)
```

```
##
##    0    1
## 2468   79
```

The newly created dummy variable will now be factored to apply meaningful labels in place of the numbers.

```
ds$f.AfAm <- factor(ds$r.AfAm, levels = c(0, 1),
                    labels = c("Non African-American", "African-American"))
table(ds$f.AfAm)
```

```
##
## Non African-American     African-American
##                 2468                   79
```

3.3 Part III: Building and Sorting Your Data

In R, random data can be generated easily into objects and manipulated as desired. There are a variety of methods to accomplish this, the basics of which are explored below.

The `rnorm()` function generates `n` random values that fit a normal distribution, given a mean and standard deviation.

```
one <- rnorm(100, mean = 3)
two <- rnorm(100, mean = 7)
three <- rnorm(100, mean = 1)
```

The previous code created three objects consisting of 100 random values with different means. The three objects can be combined into a single column using the `cbind()` function. The `cbind()` function will combine the given objects sequentially in a column in the provided order.

```
four <- cbind(one, two, three)
```

Alternatively, the `rbind()` function can combine the three objects into a single row. Similar to the `cbind()` function, the `rbind()` function will combine the given objects sequentially in the provided order.

```
five <- rbind(one, two, three)
```

3.3.1 The Tidyverse

The `tidyverse` is a collection of packages and functions for exploring and visualizing data in R. Developed by Hadley Wickham, the `tidyverse` packages provide a succint and consistent method of data exploration and visualization. There are plenty of different methods of working with your data, but this class will employ the `tidyverse`, as it is considered to be intuitive and rather simple.

Let's begin with learning how to filter your data. Filtering allows you to create a subset of your data that meets specific criteria. For example, you could filter your data to examine only men, or only Republicans, etc. To filter data, use the `filter()` verb from the `dplyr` package. The `tidyverse` functions are best optimized by using the pipe operator, `%>%`, to pipe functions together. The pipe operator takes whatever is before it and sends it on to the next function. For example:

```
ds.men <- ds %>%
  filter(gender == 1)
```

The code chunk above creates a subset of the data that only includes men. The syntax can be read as "First take the data set, then filter it to include men only." We also assigned it to a new object, `ds.men`. The filter verb can include multiple specifications. You also do not have to always assign your filtered data to a new object. For example, suppose you wanted to examine women who are age 42. You can do so like this:

```
ds %>%
  filter(gender == 0, age == 42)
```

```
##     X.1     X      userid             wave_id agescreen age gender
## 1   202  2994  Ph-W202242 Wave 12 (Fall 2016)         1  42      0
## 2   483  7164  Ph-W211012 Wave 12 (Fall 2016)         1  42      0
## 3   514  7635  Ph-W302515 Wave 12 (Fall 2016)         1  42      0
## 4   651  9559    R2NM6835 Wave 12 (Fall 2016)         1  42      0
##    education adults children employment income footage ssn_precip ssn_tmp
## 1          6      2        3          1  86000    1400          1       3
## 2          5      3        1          4 120000    1100          1       3
## 3          6      2        1          4  90000    2800          2       3
## 4          5      2        0          4  95000    2100          2       3
##    drghtfreq drghtsev fldfreq fldsev avgtmp tmpextrm evntexp_wind
## 1          2       NA       1      1      2        2            0
## 2          2        2       2      1      3        3            0
## 3          2        1       2      1     NA       NA            0
## 4          3        2       2      2      2        2            1
##    evntexp_drght evntexp_rain evntexp_cold evntexp_snow evntexp_ice
## 1              0            0            0            0           0
## 2              0            0            0            0           0
## 3              0            0            0            0           0
## 4              0            0            0            0           0
##    evntexp_flood evntexp_torn evntexp_fire evntfreq_wind evntfreq_rain
```

```
## 1              0            0            1             2             2
## 2              0            0            0             2             1
## 3              0            0            0             2             2
## 4              0            0            0             2             2
##    evntfreq_cold evntfreq_snow evntfreq_ice evntfreq_torn evntfreq_fire
## 1              2             2            2             1             3
## 2              1             1            1             2             1
## 3              2             2            2             2             2
## 4              2             2            2             2             2
##    evntfutfreq_wind evntfutfreq_drght evntfutfreq_rain evntfutfreq_cold
## 1                 2                 3                1                 2
## 2                 2                 3                3                 1
## 3                 2                 2                2                 2
## 4                 2                 2                2                 2
##    evntfutfreq_snow evntfutfreq_ice evntfutfreq_flood evntfutfreq_torn
## 1                 2               2                 2                2
## 2                 1               1                 2                3
## 3                 2               2                 2                2
## 4                 2               2                 2               NA
##    evntfutfreq_fire wtr_bill_totcost wtr_bill_dk wtr_use wtr_short
## 1                NA              156           0       2         0
## 2                 2              200           0       2         0
## 3                 2                0           0       2         0
## 4                 2               86           0       2         0
##    mntain_lawn wtr_qual wtr_safe wtr_comm wtr_farm wtr_steps_flush
## 1            1        3        8        1        1               0
## 2            1        3        8        4        3               0
## 3            1        3        7        4        4               0
## 4            1        2        8        3        2               0
##    wtr_steps_flow wtr_steps_bath wtr_steps_lawn wtr_steps_car
## 1               0              0              1             0
## 2               1              1              1             0
## 3               0              0              0             0
## 4               0              0              0             0
##    wtr_steps_othr
## 1               0
## 2               1
## 3               0
## 4               0
##                                           wtr_steps_spec
## 1                                                   <NA>
## 2                          Use our sand point well to water
## 3                                                   <NA>
## 4                                                   <NA>
##    enrgy_sourc_elec enrgy_sourc_natgas enrgy_sourc_prop enrgy_sourc_wood
## 1                 1                  1                0               1
## 2                 1                  1                0               0
## 3                 1                  0                0               0
## 4                 1                  1                0               0
##    enrgy_sourc_othr enrgy_sourc_othr_spec elec_bill_totcost elec_bill_dk
## 1                 0                  <NA>              207            0
## 2                 0                  <NA>              100            0
## 3                 0                  <NA>              300            0
## 4                 0                  <NA>              136            0
```

```
##    elec_use elec_out gas_bill_totcost gas_bill_dk gas_use
## 1         1        0               60           0       1
## 2         2        0               50           0       1
## 3         2        1               NA          NA      NA
## 4         2        0               89           0       2
##    enrgy_steps_lghts enrgy_steps_ac enrgy_steps_heat enrgy_steps_savappl
## 1                  1              1                1                   0
## 2                  1              1                1                   0
## 3                  0              0                0                   0
## 4                  1              1                1                   0
##    enrgy_steps_unplug enrgy_steps_insul enrgy_steps_savdoor
## 1                   0                0                   0
## 2                   0                0                   0
## 3                   0                0                   0
## 4                   0                0                   0
##    enrgy_steps_othr                              enrgy_steps_othr_spec
## 1                 0                                               <NA>
## 2                 0                                               <NA>
## 3                 0                                               <NA>
## 4                 0                                               <NA>
##    party  party_spec lean ideol cncrn_secur cncrn_health cncrn_enrgy
## 1      4 Libertarian   NA     7          10            9           9
## 2      1        <NA>   NA     4           8            9           6
## 3      2        <NA>   NA     6           7            7           4
## 4      3        <NA>    1     2           8            8           4
##    cncrn_trns cncrn_tax cncrn_edu cncrn_econ glbwrm_ok time_outside
## 1           9         7        10         10         0            1
## 2           5        10        10         10         1            2
## 3           0         8         5          6         0            1
## 4           8         9         9          9         1            2
##    spent_outside cmpr_time_outside hlth_status nghbrhd_deal nghbrhd_close
## 1              2                 1           3            7             5
## 2              2                 3           5            4             4
## 3              3                 2           3            5             6
## 4              2                 1           4            5             3
##    nghbrhd_care nghbrhd_not nghbrhd_eyes nghbrhd_help
## 1             1           1            7            7
## 2             2           2            6            4
## 3             3           2            6            2
## 4             3           2            3            6
##            wthr_info_paper           wthr_info_web        wthr_info_govweb
## 1                    Never                   Never                   Never
## 2                     <NA>                    <NA>                    <NA>
## 3                    Never                   Never                   Never
## 4                    Never       About once a day                   Never
##            wthr_info_loctv         wthr_info_cabtv         wthr_info_radio
## 1     Several times a day Less than once per week     Several times a day
## 2     Several times a day                    <NA>     Several times a day
## 3        About once a day                   Never Less than once per week
## 4  Less than once per week Less than once per week Less than once per week
##              wthr_info_fam           wthr_info_soc wthr_info_othr
## 1        About once a day        About once a day           <NA>
## 2                    <NA>                    <NA>           <NA>
## 3  Less than once per week                   Never           <NA>
```

```
## 4      Several times a day     Several times a day          Never
##    wthr_info_othr_spec dot_answr click_bluedot begin_datetime end_datetime
## 1                <NA>        NA             1     1481587679   1481588604
## 2                <NA>        NA             1     1480998150   1481000446
## 3                <NA>         2             1     1481312462   1481755390
## 4                <NA>         1             0     1484166842   1484167687
##    is_statewide hispanic race race_spec evntexp_heat evntexp_ethqk
## 1             1        0    1      <NA>            0             1
## 2             1        0    1      <NA>            1             1
## 3             1        0    1      <NA>            0             0
## 4             1        0    1      <NA>            0             0
##    evntfreq_heat evntfreq_ethqk evntfutfreq_heat evntfutfreq_ethqk
## 1              2              3                2                 3
## 2              3              3                2                 3
## 3              2              3                2                 2
## 4              2              3                2                 3
##    wtr_steps_leaks wtr_steps_ldry enrgy_sourc_geo enrgy_steps_bulbs
## 1                0              0               0                1
## 2                0              0               0                1
## 3                0              0               0                0
## 4                0              0               0                0
##            wthr_info_phone mntain_flwr mntain_veg glbcc glbcc_cert
## 1   Several times per week           1          0     0          7
## 2                     <NA>           1          1     1          4
## 3  Less than once per week           0          0     0          6
## 4      Several times a day           1          0     1          8
##    glbcc_risk is_payne is_okcity evntexp_hail evntfreq_hail
## 1           3        0         0            0             2
## 2           4        0         0            0             1
## 3           5        0         0            0             2
## 4           7        0         0            0             2
##    evntfutfreq_hail wtr_freq cncrn_natres confirm_zip is_kiamichi
## 1                 2        2            4       73078           0
## 2                 3        4            6       74008           0
## 3                 2        1            4       73053           0
## 4                 2        2            6       73069           0
##    is_washita is_canadian                                    consent
## 1           0           0 Yes, I agree to participate in this study
## 2           0           0 Yes, I agree to participate in this study
## 3           0           0 Yes, I agree to participate in this study
## 4           0           0 Yes, I agree to participate in this study
##    home_lot ethqk_risk ethqk_cause ethqk_risk_mgmt ethqk_risk_rand
## 1         2          7           3               8      regulating
## 2         2          5           2               7      regulating
## 3         3          3           1               3        managing
## 4         2          7           2               7      regulating
##    ethqk_risk_fed_mgmt glbwrm_risk_rand glbwrm_risk_mgmt
## 1                    0         managing                0
## 2                    7         managing                4
## 3                    3         managing                3
## 4                    6         managing                7
##    glbwrm_risk_fed_mgmt know_fire purp_fire
## 1                     0         1         1
## 2                     9         1         1
```

```
## 3                    3         1         1
## 4                    8         1         1
##
## 1
## 2
## 3
## 4
##    useful_val_fire use_fire_rural nofire_notrain nofire_notnear
## 1                1             NA             NA             NA
## 2                1             NA             NA             NA
## 3                2              0              0              1
## 4                1             NA             NA             NA
##    nofire_othr                                      nofire_othr_spec
## 1           NA                                                 <NA>
## 2           NA                                                 <NA>
## 3            0                                                 <NA>
## 4           NA                                                 <NA>
##    urban_fire fire_implmnt okelec_foss okelec_nuc okelec_renew solar_prop
## 1          NA            1          50         25           25          0
## 2           0            1          70         10           20          0
## 3          NA           NA          76          0           24          0
## 4           1            1          25         25           50          0
##    heard_netmeter net_met_info ok_net_met vote vote_2016 vote_cand
## 1               0            1          4    1         1         2
## 2               0            1          1    1         1         1
## 3               0            1          4    1         1         0
## 4               0            1          3    1         1         1
##    vote_cand_spec vote_cand_spt f.gender f.party f.party.2 r.ind   f.ind
## 1            <NA>             3    Women   Other      <NA>     0 Non Ind
## 2            <NA>             2    Women     Dem       Dem     0 Non Ind
## 3            <NA>             4    Women     Rep       Rep     0 Non Ind
## 4            <NA>             2    Women     Ind       Ind     1     Ind
##    r.rep   f.rep r.dem   f.dem                f.edu f.race r.race f.race.2
## 1      0 Non Rep     0 Non Dem    Bachelor's Degree  White      1    White
## 2      0 Non Rep     1     Dem    2 year/Associates  White      1    White
## 3      1     Rep     0 Non Dem    Bachelor's Degree  White      1    White
## 4      0 Non Rep     0 Non Dem    2 year/Associates  White      1    White
##    r.white f.white r.AfricanAmer   f.AfricanAmer r.age f.age     f.age.2
## 1        1   White              0 Non AfricanAmer     3 41-50 (17.9,58.5]
## 2        1   White              0 Non AfricanAmer     3 41-50 (17.9,58.5]
## 3        1   White              0 Non AfricanAmer     3 41-50 (17.9,58.5]
## 4        1   White              0 Non AfricanAmer     3 41-50 (17.9,58.5]
##    r.ideology   f.ideology f.glbwrm_ok n.party r.ideol      f.ideol r.AfAm
## 1           3 Conservative          No       4       3 Conservative      0
## 2           2     Moderate         Yes       1       2     Moderate      0
## 3           3 Conservative          No       2       3 Conservative      0
## 4           1      Liberal         Yes       3       1      Liberal      0
##                  f.AfAm
## 1  Non African-American
## 2  Non African-American
## 3  Non African-American
## 4  Non African-American
##  [ reached getOption("max.print") -- omitted 14 rows ]
```

Similar to `filter()`, you can use the `select()` function from the `dplyr()` package to create a new data set that includes a few variables of interest.

Note: `select` is a very common verb in R, and therefore is often masked by other packages. If you encounter an error when using `select()`, include `dplyr::` before calling the function, so that R knows which version of `select()` to use. Follow it up by using `na.omit()` to remove missing observations:

```
ds.sub <- ds %>%
  select(gender, age, income, education, ideol, glbcc_cert) %>%
  na.omit()
```

The next verb is `arrange()`, which allows you to sort your data in ascending or descending order by a particular specification. Recall the previous code chunk where we filtered the data to include only women who are age 42. Adding another pipe operator, %>%, and `arrange()` will allow us to examine the data further. Use the `arrange()` function to sort the data by education:

```
ds.sub %>%
  filter(gender == 0, age == 42) %>%
  arrange(education)
```

```
##    gender age income education ideol glbcc_cert
## 1       0  42  75000         4     4          5
## 2       0  42 150000         4     6          6
## 3       0  42 120000         5     4          4
## 4       0  42  95000         5     2          8
## 5       0  42  95000         5     4          8
## 6       0  42  65000         5     6          7
## 7       0  42  80000         5     6          5
## 8       0  42 120000         5     4          4
## 9       0  42  86000         6     7          7
## 10      0  42  90000         6     6          6
## 11      0  42  36000         6     6          5
## 12      0  42 120000         6     2         10
## 13      0  42  90000         6     6          6
## 14      0  42  50000         6     6          6
## 15      0  42 115000         6     6          5
## 16      0  42  80000         6     2          7
## 17      0  42 155000         6     7          8
```

The `slice()` function selects a set amount of observations from the data. Use `slice()` to select the first 10 observations from the data:

```
ds.new <- ds.sub %>%
  slice(1:10)
```

The `mutate()` verb allows you to either change an existing variable or create a new one. For example, we can create a new variable that returns income in increments of 1000s:

```
ds.new %>%
  mutate(inc_100 = income / 1000)
```

```
##    gender age income education ideol glbcc_cert inc_100
## 1       1  50 160000         4     6          6     160
## 2       1  58  45000         4     3         10      45
## 3       0  52  40000         6     6          6      40
## 4       0  40  55000         4     4          0      55
## 5       0  44  33000         4     4          5      33
## 6       0  60  40000         6     2          5      40
```

```
## 7       1  67  50000         6     4          1      50
## 8       0  66  25000         6     7          8      25
## 9       0  44  70000         6     3          7      70
## 10      1  46  70000         4     7          9      70
```

Suppose you wanted to examine summary statistics for the data. Perhaps you needed to know the average income of everyone in the survey. The `summarize()` verb can be used to do this, and more:

```
ds.sub %>%
  summarize(mean_inc = mean(income))
```

```
##   mean_inc
## 1  70597.7
```

Combining the `summarize()` verb with the `filter()` verb allows you to summarize a more specific set of observations. To find the average income of men only, filter the data to include only men, then summarize:

```
ds.sub %>%
  filter(gender == 1) %>%
  summarize(mean_inc = mean(income))
```

```
##   mean_inc
## 1 80564.86
```

Summarize can also be used to return mulitple values of interest:

```
ds.sub %>%
  filter(gender == 1) %>%
  summarize(mean_inc = mean(income), med_inc = median(income), sd_inc = sd(income))
```

```
##   mean_inc med_inc   sd_inc
## 1 80564.86   67000 68869.39
```

The code chunk above returns the mean, median, and standard deviation of income for men in the data set.

Filtering for multiple categories can be tedious. Fortunately, there is a tidyverse function that will return desired information for each group in a variable. For example, we can get the mean, median, and standard deviation of the income variable for men and women by using the `group_by()` verb and indicating "gender":

```
ds.sub %>%
  group_by(gender) %>%
  summarize(mean_inc = mean(income), med_inc = median(income), sd_inc = sd(income))
```

```
## # A tibble: 2 x 4
##   gender mean_inc med_inc sd_inc
##    <int>    <dbl>   <dbl>  <dbl>
## 1      0   63598.   50000 51486.
## 2      1   80565.   67000 68869.
```

The `group_by` verb can be used to find lots of information. Perhaps you wanted to find the average level of belief that humans cause cliamte change by ideology level:

```
ds.sub %>%
  group_by(ideol) %>%
  summarize(mean_glbcc_cert = mean(glbcc_cert))
```

```
## # A tibble: 7 x 2
##   ideol mean_glbcc_cert
##   <int>           <dbl>
## 1     1            9.05
## 2     2            8.26
```

```
## 3     3            7.63
## 4     4            6.17
## 5     5            5.91
## 6     6            5.90
## 7     7            6.76
```

The ideology variable runs from 1 to 7, with 1 being very liberal and 7 being very conservative. It is clear that liberals, on average, are more certain that humans cause climate change. The `group_by()` verb can also be used to look at multiple groups at once. We can create the same table as above, but this time break it down by ideology and gender:

```
ds.sub %>%
  group_by(gender, ideol) %>%
  summarize(mean_glbcc_cert = mean(glbcc_cert))
```

```
## # A tibble: 14 x 3
## # Groups:   gender [2]
##    gender ideol mean_glbcc_cert
##     <int> <int>           <dbl>
##  1      0     1            9.22
##  2      0     2            8.15
##  3      0     3            7.42
##  4      0     4            5.95
##  5      0     5            5.81
##  6      0     6            5.73
##  7      0     7            6.52
##  8      1     1            8.80
##  9      1     2            8.46
## 10      1     3            7.85
## 11      1     4            6.69
## 12      1     5            6.04
## 13      1     6            6.09
## 14      1     7            7.06
```

3.3.2 Other Methods of Exploring Your Data

There are many functions in the tidyverse that can be used for data exploration. By loading the `tidyverse` package at the beginning of the lab, most of these functions should be readily available. The `glimpse()` verb returns an overall breakdown of what is contained in your data set:

```
ds.sub %>%
  glimpse()
```

```
## Observations: 2,276
## Variables: 6
## $ gender     <int> 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, ...
## $ age        <int> 50, 58, 52, 40, 44, 60, 67, 66, 44, 46, 39, 51, 25,...
## $ income     <dbl> 160000, 45000, 40000, 55000, 33000, 40000, 50000, 2...
## $ education  <int> 4, 4, 6, 4, 4, 6, 6, 6, 6, 4, 5, 5, 2, 6, 1, 6, 2, ...
## $ ideol      <int> 6, 3, 6, 4, 4, 2, 4, 7, 3, 7, 4, 6, 4, 3, 6, 2, 2, ...
## $ glbcc_cert <int> 6, 10, 6, 0, 5, 5, 1, 8, 7, 9, 6, 4, 6, 7, 7, 8, 10...
```

In the previous section we used `summarize()` to get breakdowns of ceratin variables. If you wanted to get diagnostics on all the variables in your data, use the `skim()` function from the `skimr` package:

```
skim_with(numeric = list(hist = NULL))
skim_with(integer = list(hist = NULL))
ds.sub %>%
  skim()
```

```
## Skim summary statistics
##  n obs: 2276
##  n variables: 6
##
## -- Variable type:integer -------------------------------------------------------------------------
##    variable missing complete    n  mean    sd p0   p25 p50 p75 p100
##         age       0     2276 2276 60.11 14.09 18 51.75  62  70   99
##   education       0     2276 2276  5.09  1.81  1  4      6   6    8
##      gender       0     2276 2276  0.41  0.49  0  0      0   1    1
##  glbcc_cert       0     2276 2276  6.63  2.71  0  5      7   9   10
##       ideol       0     2276 2276  4.64  1.74  1  4      5   6    7
##
## -- Variable type:numeric -------------------------------------------------------------------------
##  variable missing complete    n    mean       sd    p0   p25   p50   p75
##    income       0     2276 2276 70597.7 59850.38 10000 35000 59000 90000
##    p100
##  900000
```

This provides loads of useful incormation. As demonstrated in teh previous section, you can combine different tidyverse functions together in order to maximize efficiency. For example, combining `group_by()` and `skim()` provide breakdowns of all the variables by gender:

```
skim_with(numeric = list(hist = NULL))
skim_with(integer = list(hist = NULL))
ds.sub %>%
  group_by(gender) %>%
  skim()
```

```
## Skim summary statistics
##  n obs: 2276
##  n variables: 6
##  group variables: gender
##
## -- Variable type:integer -------------------------------------------------------------------------
##  gender   variable missing complete    n  mean    sd p0 p25 p50 p75 p100
##       0        age       0     1337 1337 59.73 14.66 18  50  61  71   94
##       0  education       0     1337 1337  4.98  1.77  1   4   6   6    8
##       0 glbcc_cert       0     1337 1337  6.48  2.71  0   5   7   9   10
##       0      ideol       0     1337 1337  4.54  1.73  1   4   5   6    7
##       1        age       0      939  939 60.64 13.22 22  53  62  70   99
##       1  education       0      939  939  5.26  1.86  1   4   6   7    8
##       1 glbcc_cert       0      939  939  6.84  2.69  0   5   7   9   10
##       1      ideol       0      939  939  4.79  1.74  1   4   5   6    7
##
## -- Variable type:numeric -------------------------------------------------------------------------
##  gender variable missing complete    n     mean       sd    p0   p25   p50
##       0   income       0     1337 1337 63597.58 51485.97 10000 30000 50000
##       1   income       0      939  939 80564.86 68869.39 10000 40000 67000
##     p75   p100
##   80000 750000
```

```
##  100000 900000
```

There are a variety of operators that can be used when working with your data. These Boolean operators assist with building data subsets in R:

1. < less than
2. <= less than or equal to
3. > greater than
4. >= greater than or equal to
5. == exactly equal to
6. ! = not equal to
7. ! not (example: !x - pronounced "not x")
8. | or (example: x | y - pronounced "x OR y")
9. & and (example: x & y - pronounced "x AND y")

3.4 Working with Nominal Data

Often times data is nominal. This is data that does not necessarily have a numeric value, but rather is categorized by a word or label (e.g., race, political party, etc.). The factored `party` variable is an example.

```
table(ds$f.party)
```

```
##
##   Dem   Rep   Ind Other
##   869  1185   356    91
```

If analyzing the data, the "Other" category does not explain much. In this case, recoding the Other responses as NA will exclude Other from summaries.

```
ds$f.party.2 <- car::recode(ds$f.party, "'Dem' = 'Dem'; 'Rep' = 'Rep'; 'Ind' = 'Ind';
                            'Other' = NA; else = NA; NA = NA")
table(ds$f.party.2)
```

```
##
##  Dem  Ind  Rep
##  869  356 1185
```

Note: When recoding a factored variable, label names must be in in apostrophe marks **within the quotation marks**.

3.4.1 Finding the Mode

When working with nominal data, there are some inapplicable statistics. For example, the mean for the factored political party variable does not exist. However, finding the mode for nominal data is useful. Recall from mathematics that the mode is the value, or in this case the label, that occurs the most often.

A simple way to find the mode is to use the `count` verb from the `tidyverse`. Use `count` to find the mode of the factored political party variable:

```
ds %>%
  count(f.party.2)
```

```
## Warning: Factor `f.party.2` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## # A tibble: 4 x 2
##   f.party.2     n
##   <fct>     <int>
```

```
## 1 Dem         869
## 2 Ind         356
## 3 Rep        1185
## 4 <NA>        137
```

Including `sort = TRUE` will tell R to sort the values in descending order, putting the modal value first:

```
ds %>%
  count(f.party.2, sort = TRUE)
```

```
## Warning: Factor `f.party.2` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## # A tibble: 4 x 2
##   f.party.2     n
##   <fct>     <int>
## 1 Rep        1185
## 2 Dem         869
## 3 Ind         356
## 4 <NA>        137
```

To get the percentage breakdown of the political parties, use the `count` verb along with the `mutate` verb. To get the percent of each category, you would take the n size of that category and divide it by the total n size!

```
ds %>%
  count(f.party.2) %>%
  mutate(percent = n / sum(n))
```

```
## Warning: Factor `f.party.2` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## # A tibble: 4 x 3
##   f.party.2     n percent
##   <fct>     <int>   <dbl>
## 1 Dem         869  0.341
## 2 Ind         356  0.140
## 3 Rep        1185  0.465
## 4 <NA>        137  0.0538
```

Perhaps you've noticed that the inclusion of NAs complicates the percentage numbers. NAs can be dropped by using the `drop_na()` function and specifying the variable of interest. Do that, and then construct the same table as above:

```
ds %>%
  drop_na(f.party.2) %>%
  count(f.party.2) %>%
  mutate(percent = n / sum(n))
```

```
## # A tibble: 3 x 3
##   f.party.2     n percent
##   <fct>     <int>   <dbl>
## 1 Dem         869   0.361
## 2 Ind         356   0.148
## 3 Rep        1185   0.492
```

3.4.2 Visualizing Nominal Data

R has nearly countless ways to visualize data. `ggplot2` will be the primary visualization method for these labs. There are many different visualization packages and sets of packages, but `ggplot2` provides a consistent set of visualization tools that shares a common language and syntax. A great introcution to ggplot2 can be found here: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

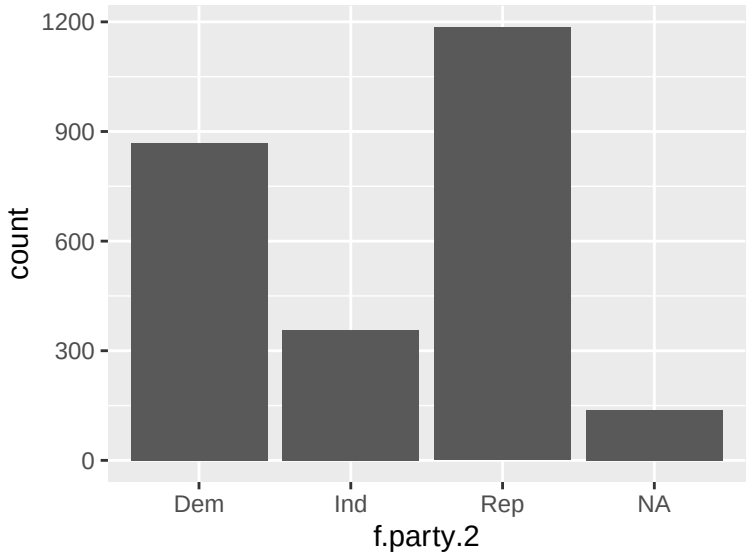
To make a visualization with `ggplot2`, first call it with the `ggplot()` function, then specify what type of visualization you want to make. The steps are:

1. Call ggplot
2. Identify the dataset (`data`)
3. Specify the aesthetic mapping (`aes`)
4. Choose what type of visualization by using the `geom_` function.

As a note, making a visualization with `ggplot2` requires multiple functions. Make sure to include a `+` sign after each function.

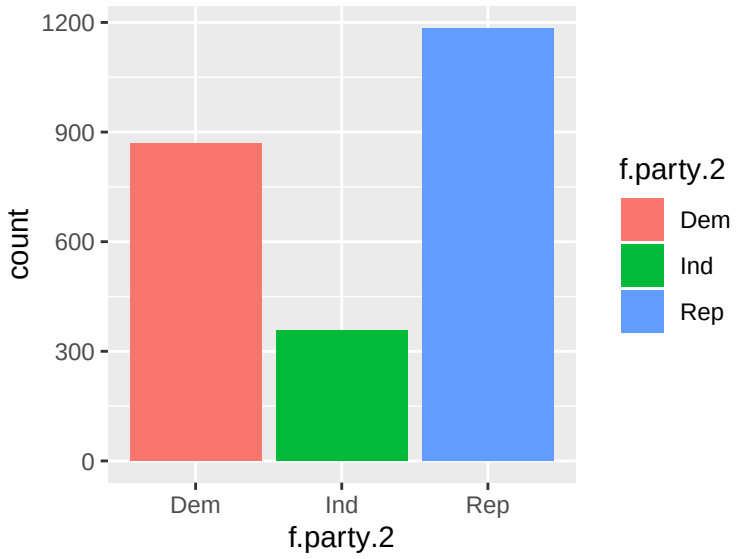
A good place to start is with a bar plot. To do this, use `geom_bar()`.

```
ggplot(ds, aes(f.party.2)) +
 geom_bar()
```



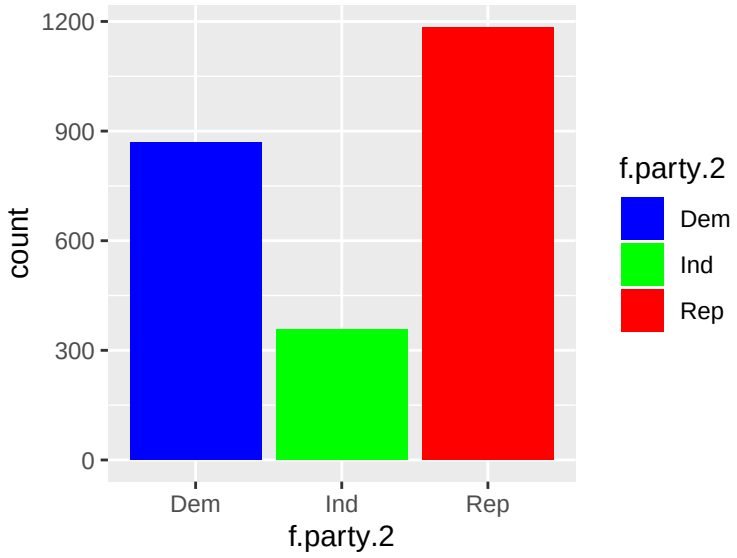
To create a visualization that only includes Democrats, Republicans, and Independents, use the `drop_na` function to filter out NAs, then pipe it to `ggplot2` Color can also be included by specifying `fill = f.party.2`. Note that inlcuding the `.` in place of the data set argument works when you pipe a dataset into `ggplot2`. Doing so tells R to use the data that is being piped into `ggplot2`.

```
ds %>%
  drop_na(f.party.2) %>%
  ggplot(., aes(f.party.2, fill = f.party.2)) +
  geom_bar()
```



`ggplot2` has a default color pallete that it assigns to groups based on the order they appear in the data. However, matching the color red with Democrats and the color blue with Republicans might not make a lot of intuitive sense. By using the `scale_fill_manual()` function, specific colors can be specified.

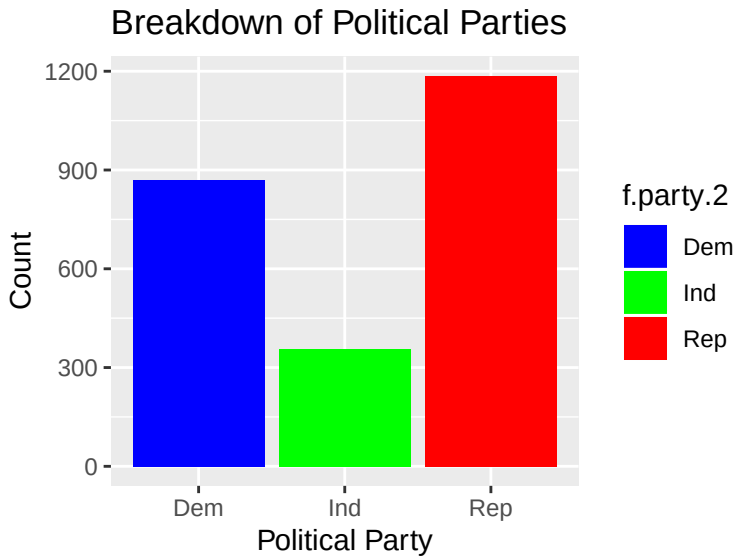
```
ds %>%
  drop_na(f.party.2) %>%
  ggplot(., aes(f.party.2, fill=f.party.2)) +
  geom_bar() +
  scale_fill_manual(values = c("blue", "green", "red"))
```



Construct one more visualization. This time, include a title by using `ggtitle()` and axis labels by using `xlab()` and `ylab()`.

```
ds %>%
  drop_na(f.party.2) %>%
  ggplot(., aes(f.party.2, fill = f.party.2)) +
  geom_bar() +
  scale_fill_manual(values = c("blue", "green", "red")) +
  ggtitle("Breakdown of Political Parties") +
```

```
  xlab("Political Party") +
  ylab("Count")
```



3.5 Working with Ordinal Data

Recall that ordinal data is data that is assigned numeric values, but on an ordered scale. An example of ordinal data is education level (e.g., some high school is higher than no high school, a high school diploma is higher than some high school, some college is higher than high school, etc).

The `count()` function describes the `education` variable. Note that the `table()` function is a good way to look at categorical and ordinal data as well, but using `count()` allows us to see the NAs and is consistent with tidyverse syntax.

```
ds %>%
  count(education)
```

```
## # A tibble: 9 x 2
##   education     n
##       <int> <int>
## 1         1    41
## 2         2   341
## 3         3   132
## 4         4   524
## 5         5   205
## 6         6   713
## 7         7   438
## 8         8   149
## 9        NA     4
```

The `education` variable contains 8 categories on an ordered scale. Factoring the `education` variable to apply labels will provide meaning to each value.

```
ds$f.education <- factor(ds$education, levels=c(1, 2, 3, 4, 5, 6, 7, 8),
                  labels=c("< HS", "HS/GED", "Vocational/Technical",
                           "Some College", "2 year/Associates", "Bachelor's Degree",
                           "Master's degree", "PhD/JD/MD"))
```

```
ds %>%
  count(f.education)
```

```
## Warning: Factor `f.education` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## # A tibble: 9 x 2
##   f.education              n
##   <fct>                <int>
## 1 < HS                    41
## 2 HS/GED                 341
## 3 Vocational/Technical   132
## 4 Some College           524
## 5 2 year/Associates      205
## 6 Bachelor's Degree      713
## 7 Master's degree        438
## 8 PhD/JD/MD              149
## 9 <NA>                     4
```

Including the `sort = TRUE` argument can return the modal education level.

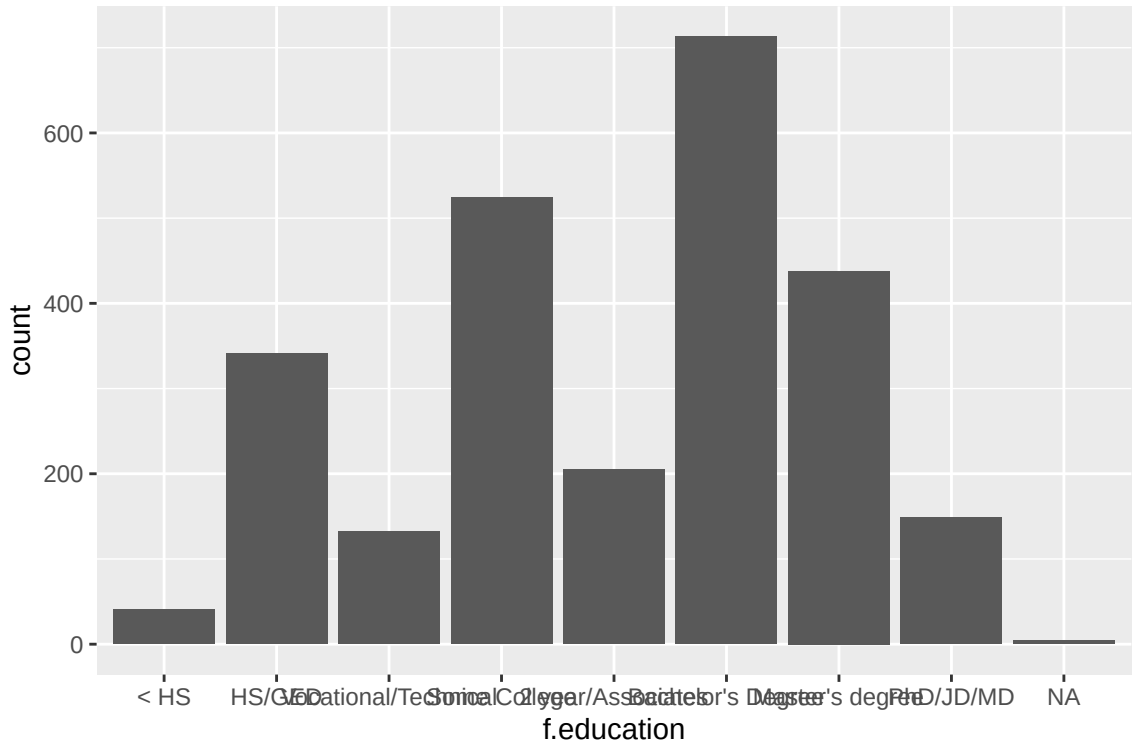
```
ds %>%
  count(f.education, sort = TRUE)
```

```
## Warning: Factor `f.education` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## # A tibble: 9 x 2
##   f.education              n
##   <fct>                <int>
## 1 Bachelor's Degree      713
## 2 Some College           524
## 3 Master's degree        438
## 4 HS/GED                 341
## 5 2 year/Associates      205
## 6 PhD/JD/MD              149
## 7 Vocational/Technical   132
## 8 < HS                    41
## 9 <NA>                     4
```

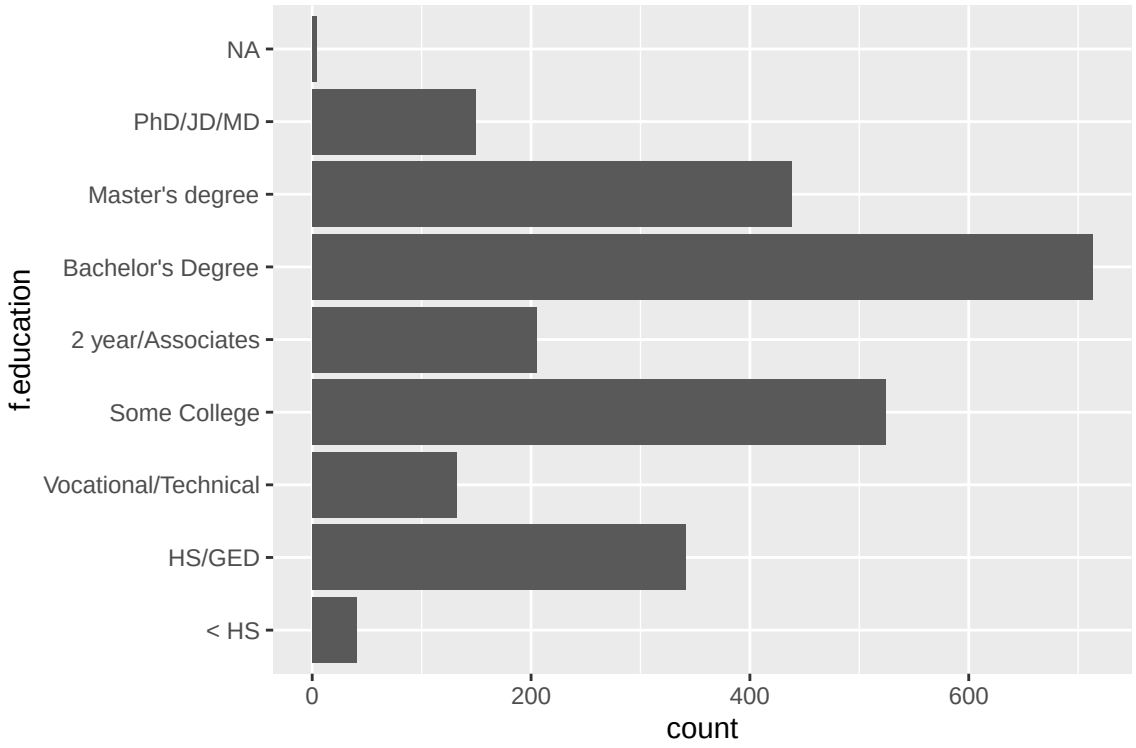
The `barplot()` function will visualize education levels.

```
ggplot(ds, aes(f.education)) +
  geom_bar()
```



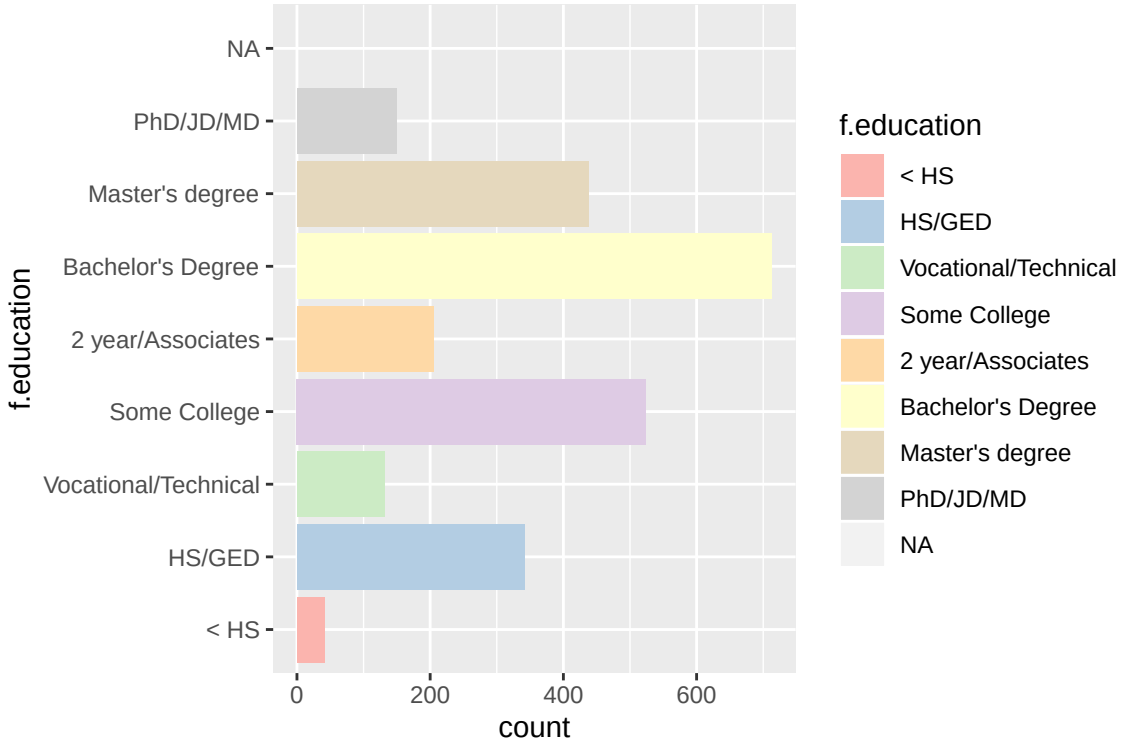
Notice the very muddled labels. One way to solve this is to flip the coordinates of the plot by using `coord_flip()`:

```
ggplot(ds, aes(f.education)) +
  geom_bar() +
  coord_flip()
```



More detail is available by adding colors. There are a few different methods of adding colors using ggplot. Include `fill = f.education` to indicate that the bars will be filled with color, and `scale_fill_manual()` to indiciate the colors:

```
ggplot(ds, aes(f.education, fill = f.education)) +
  geom_bar() +
  scale_fill_manual(values = c("#fbb4ae", "#b3cde3", "#ccebc5", "#decbe4",
                               "#fed9a6", "#ffffcc", "#e5d8bd", "#d3d3d3")) +
  coord_flip()
```



Note: The colors in the previous example are hexadecimal colors. Color schemes are available via the following website: http://colorbrewer2.org/

3.6 Working with Interval Data

Interval data is similar to ordinal data, but with interval data the difference between levels is meaningful. A good example is age, in which the difference between 23 and 24 is the same as the difference between 56 and 57, and so on.

The `psych` package provides a `describe()` function. The `describe()` function is useful to examine variables.

```
describe(ds$age)
```

```
##    vars    n  mean    sd median trimmed   mad min max range  skew kurtosis
## X1    1 2547 60.37 14.21     62   61.01 13.34  18  99    81 -0.39    -0.24
##      se
## X1 0.28
```

Find the modal value of `age`:

```
ds %>%
  count(age, sort = TRUE)
```

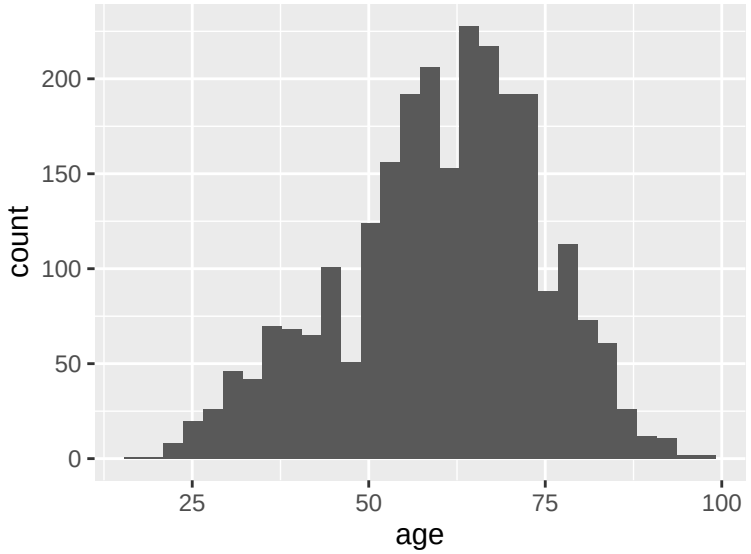
```
## # A tibble: 78 x 2
##      age     n
##    <int> <int>
##  1    65    88
##  2    62    86
##  3    60    79
##  4    68    78
##  5    69    78
##  6    64    71
##  7    67    71
##  8    72    71
##  9    57    70
## 10    63    69
## # ... with 68 more rows
```

The modal value is 65.

A histogram is the appropriate visualization for interval data. Histograms returns values on a continuous scale as opposed to individual values. To make a histogram, follow the same basic steps as before, but this time use `geom_histogram()` instead. ‘

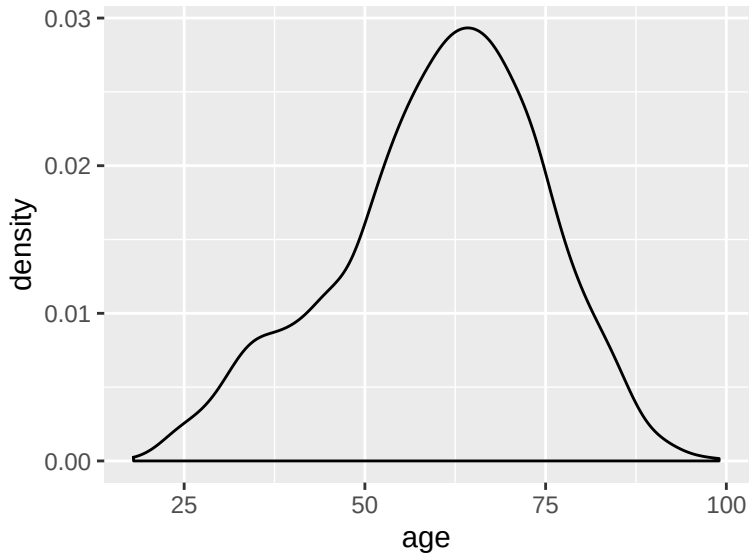
```
ggplot(ds, aes(age)) +
  geom_histogram()
```

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



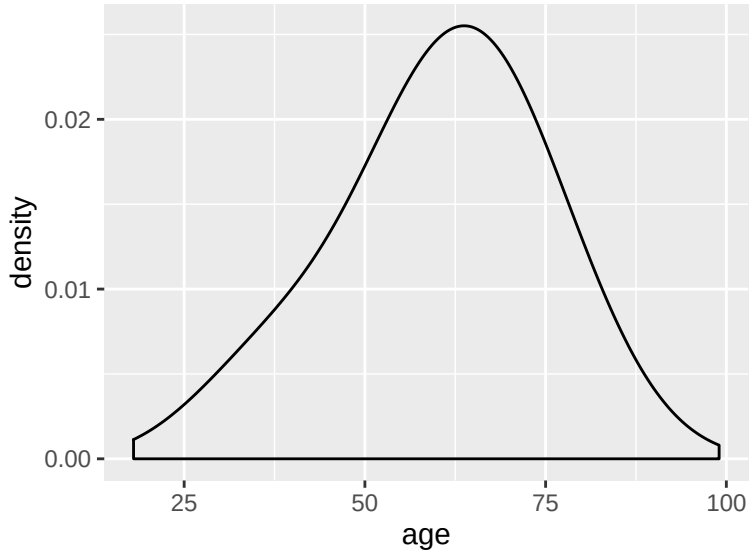
To look at the density distribution, use `geom_density()`

```
ggplot(ds, aes(age)) +
  geom_density()
```



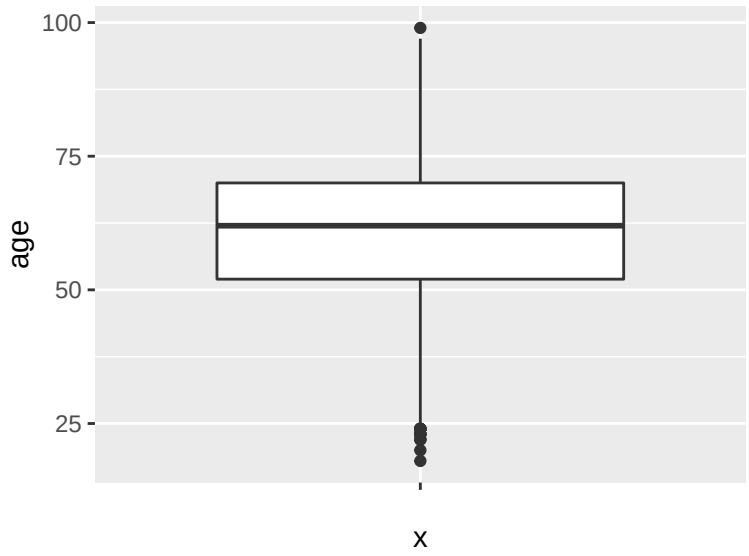
To adjust the bandwidth, use `adjust =` to specify a value.

```
ggplot(ds, aes(age)) +
  geom_density(adjust = 3)
```



Last, a box plot can be generated via the `boxplot()` function. Technically a box plot in `ggplot2` requires both an x and y value, so simply indiciate `""` for the x portion of the aesthetic.

```
ggplot(ds, aes(x = "", y = age)) +
  geom_boxplot()
```



4 Visualizing Data, Probability, the Normal Distribution, and Z Scores

This lab discusses the basics of visualizing data, probability, the normal distribution, and z scores. The following packages are required for this lab:

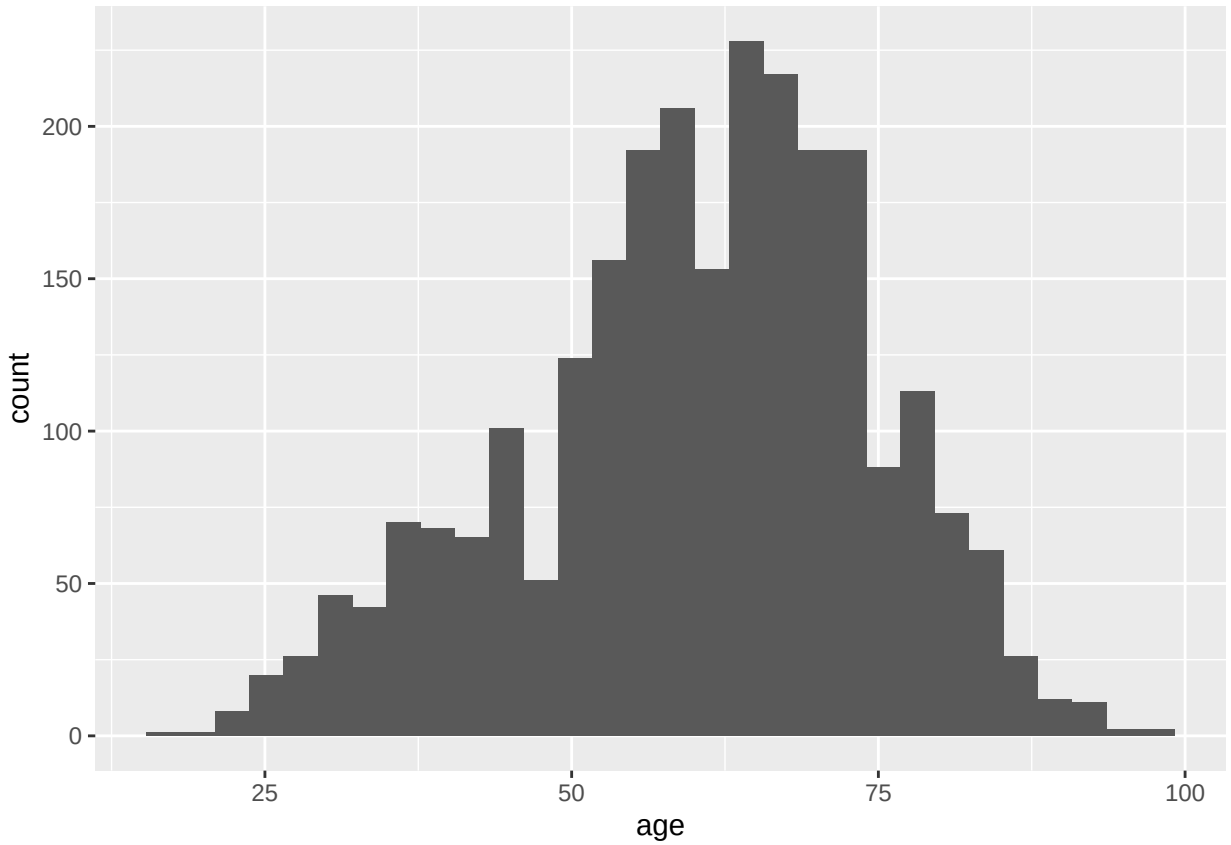
1. tidyverse
2. sfsmisc

4.1 Histograms and Density

Recall that histograms are used to visualize continuous data. Histograms are not used to visualize categorical data. Instead, a bar plot is advised for categorical data. The `geom_hist()` function creates histograms in R using ggplot visualizations. The following is an example of creating a histogram of the `age` variable within the `ds` data set.

```
ggplot(ds, aes(age)) +
  geom_histogram()
```

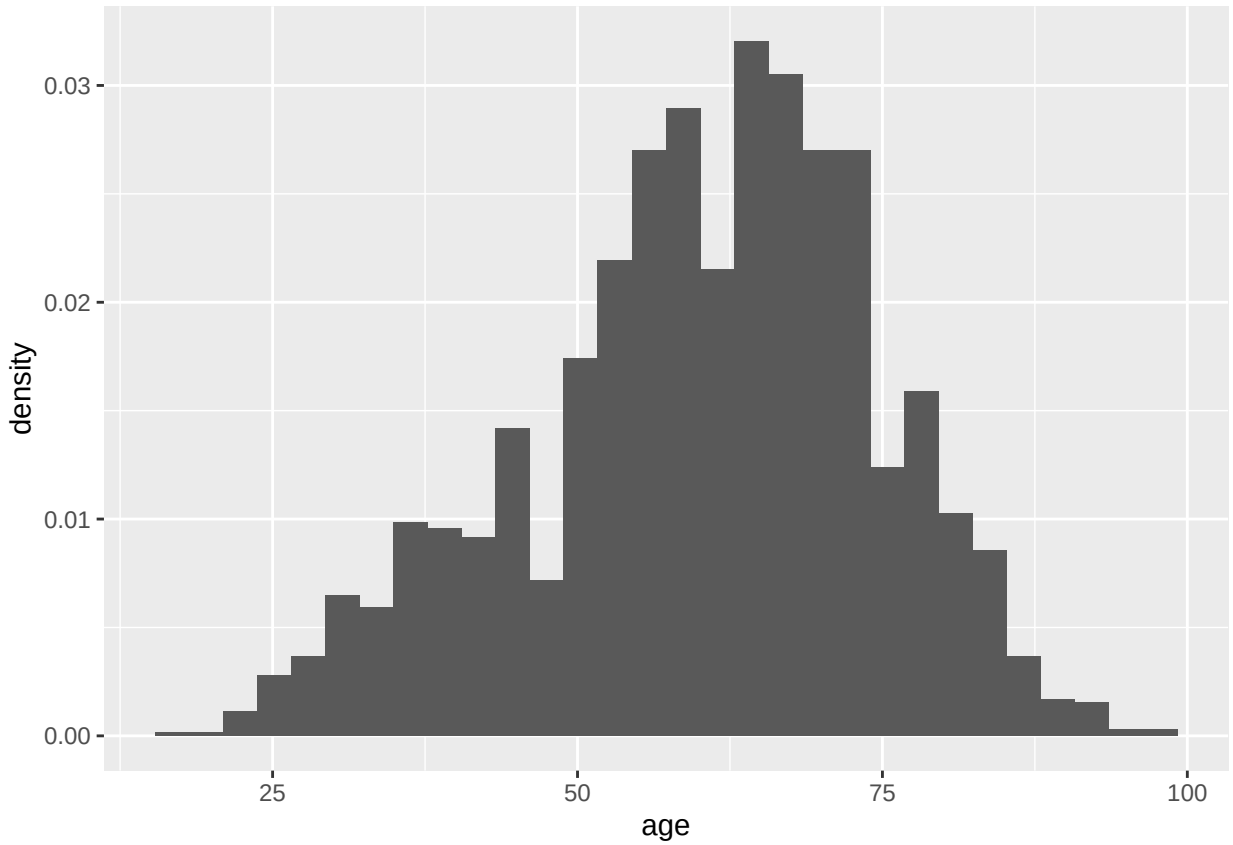
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The histogram displays the frequency of `age` for given bins. Alternatively, the density of `age` can be shown instead of frequency by making a slight change in the visualization. Use the mapping aesthetic inside the `geom_histogram()` function and setting x as the age variable and y as `..density..`

```
ggplot(ds, aes(x = age, y = ..density..)) +
  geom_histogram()
```

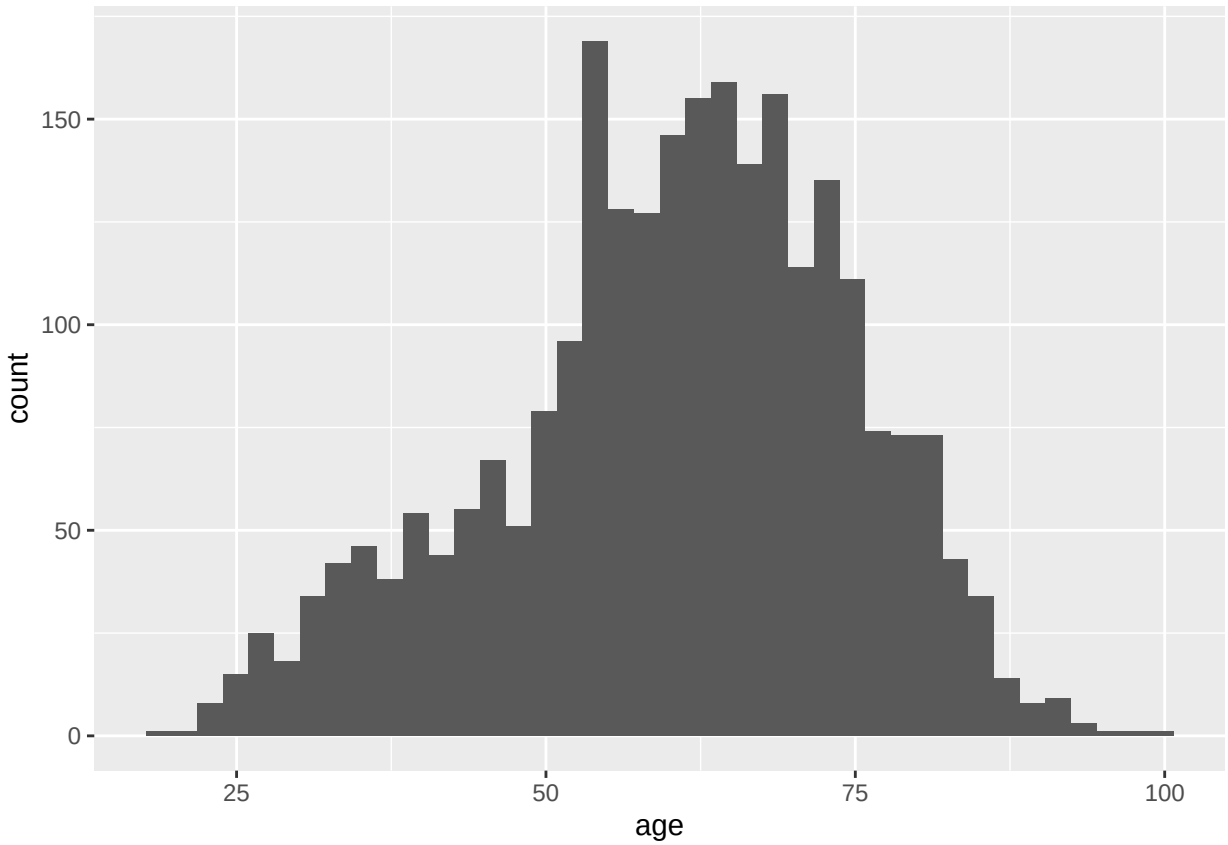
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The shape of the plot is the same for the frequency and density histograms; however, the y-axis measures in different units. The area associated with the largest y-axis value suggest that a higher percentage of respondents are likely to provide an age within the ages on the x-axis.

Data is organized into ranges, known as bins, to compose the x-axis. The number of bins is a potentially contentious topic; however, a good recommendation is to set the number of bins equal to $\sqrt{n}$, where n is the number of observations. To change the number of bins, use `bins=n` inside the `geom_histogram()` function. The square root of n for the current data set is a little over 50, but in this case 40 looks a little more appropriate.

```
ggplot(ds, aes(x = age)) +
  geom_histogram(bins = 40)
```

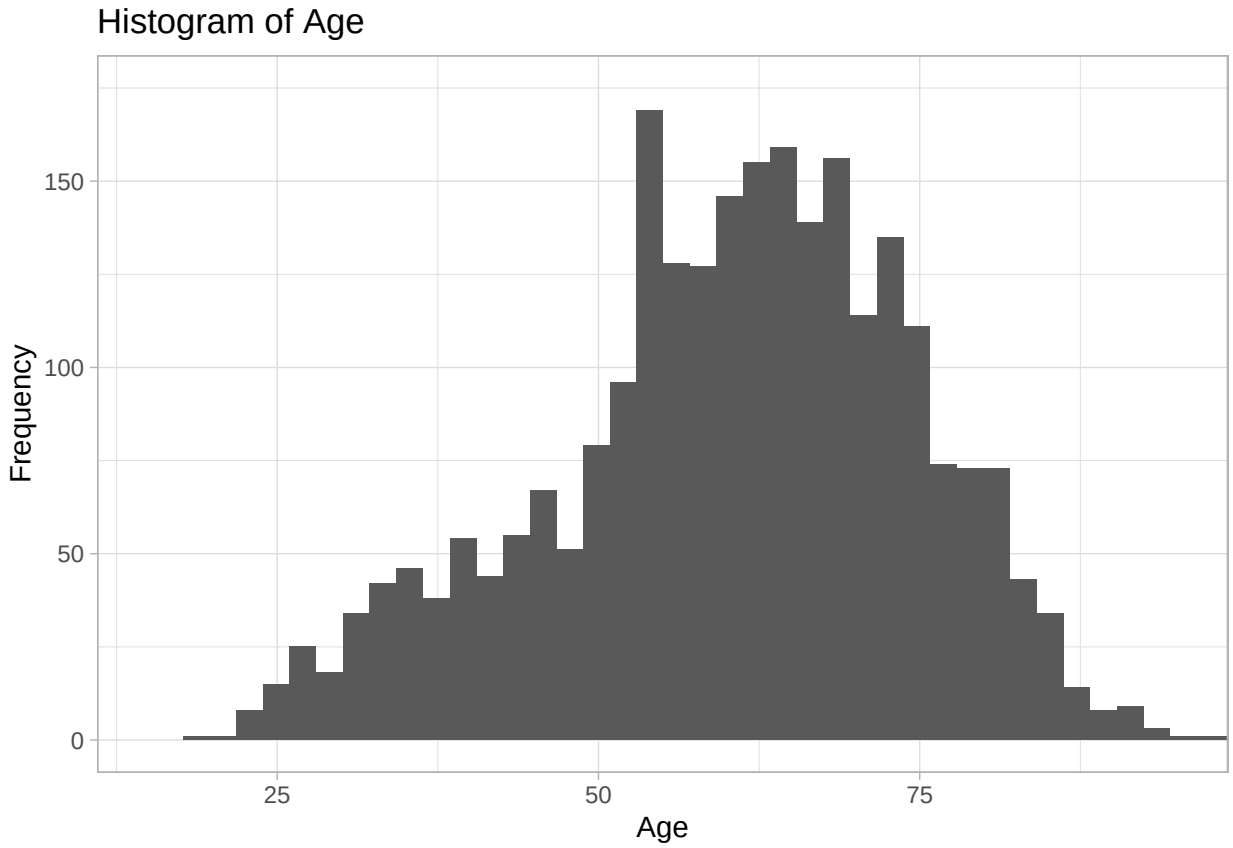


Using various functions along with the histogram function, the visualization is improved with more meaningful information. These functions can help:

1. xlab("X-Axis Label")
   - Sets the label for the x-axis
2. ylab("Y-Axis label")
   - Sets the label for the y-axis
3. ggtitle("Title")
   - Sets the histogram title
4. lims(x = c(min:max), y = c(min:max))
   - Sets the limits of the x and y axes.

The following is an excellent example of a histogram of the `age` data.

```
ggplot(ds, aes(age)) +
  geom_histogram(bins = 40) +
  xlab("Age") +
  ylab("Frequency") +
  ggtitle("Histogram of Age") +
  coord_cartesian(ylim = c(0, 175), xlim = c(15, 95)) +
  theme_light() # Sets the theme. There are a lot to choose from.
```

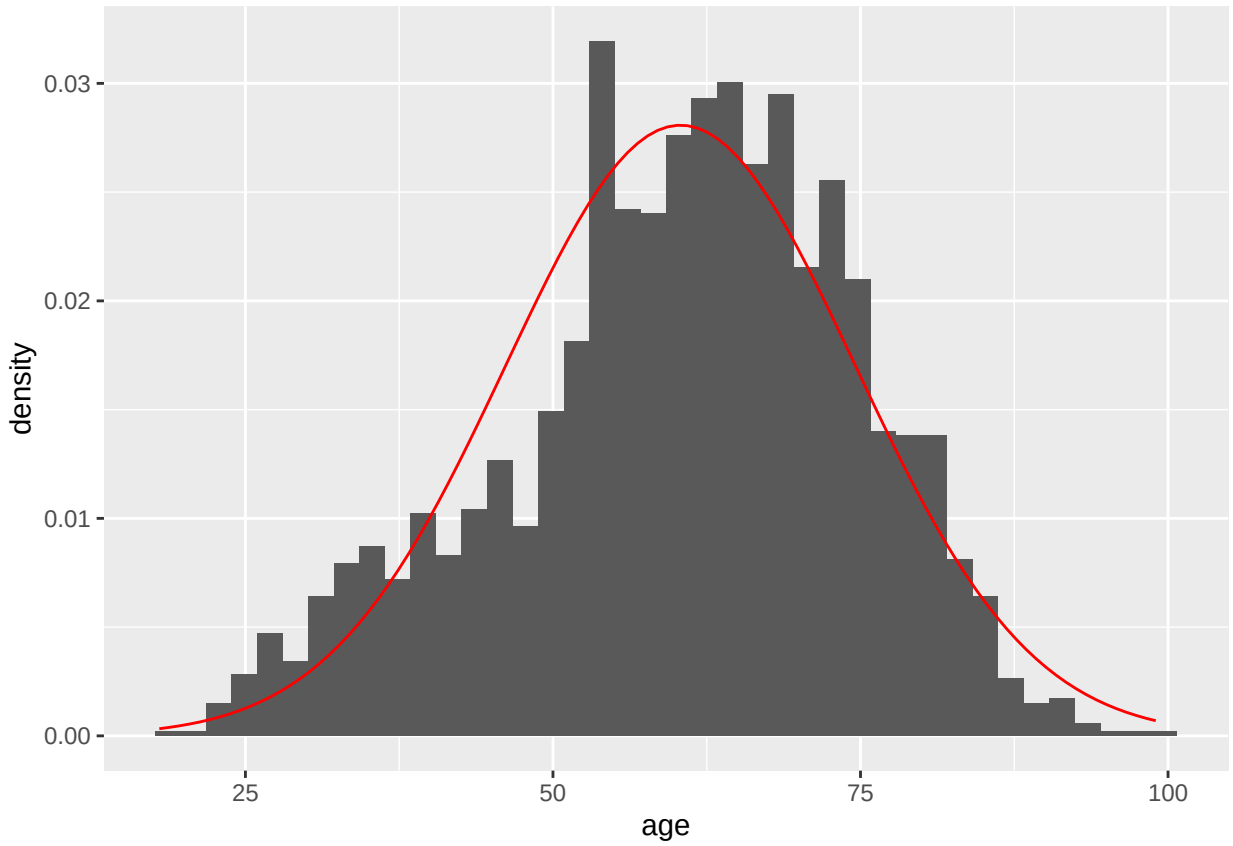


4.1.1 Normal Distribution and Histograms

Data approximated by the normal distribution can define probabilities. Using R, the normal distribution "bell curve" can be projected over a histogram.

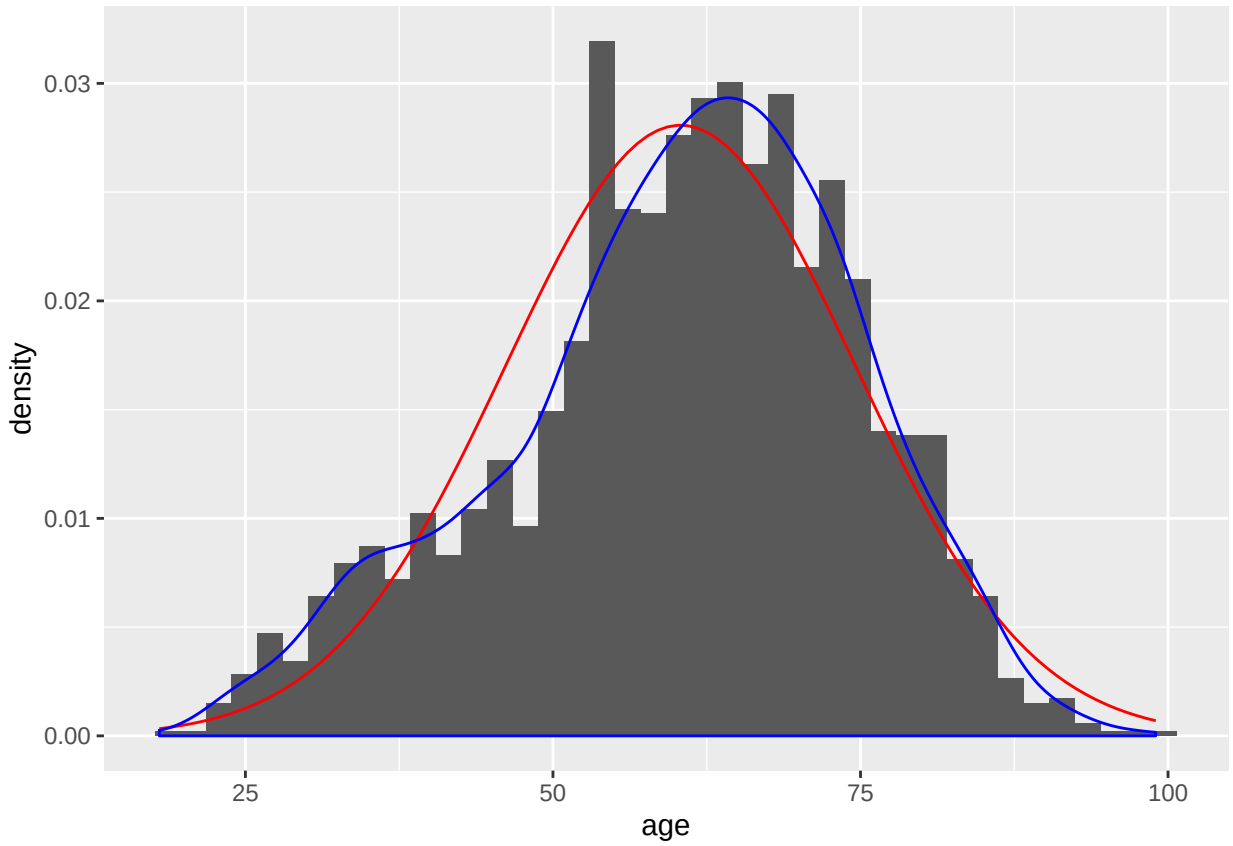
Given an identified mean and standard deviation, and a density histogram, the `stat_function()` function can project a normal distribution as follows. Specify `fun=dnorm`.

```
ggplot(ds) +
  geom_histogram(aes(x = age, y = ..density..), bins = 40) +
  stat_function(fun = dnorm, args = list(mean = mean(ds$age), sd = sd(ds$age)), color = "red")
```



Comparing the histogram plot to the normal distribution curve generated may prove difficult. The `geom_density()` function can draw a line using density data for *age* alongside the projected line of what the normal distribution would appear like given the mean and standard deviation. The two shapes can then be compared visually to interpret whether the *age* data can be approximated by the normal distribution.

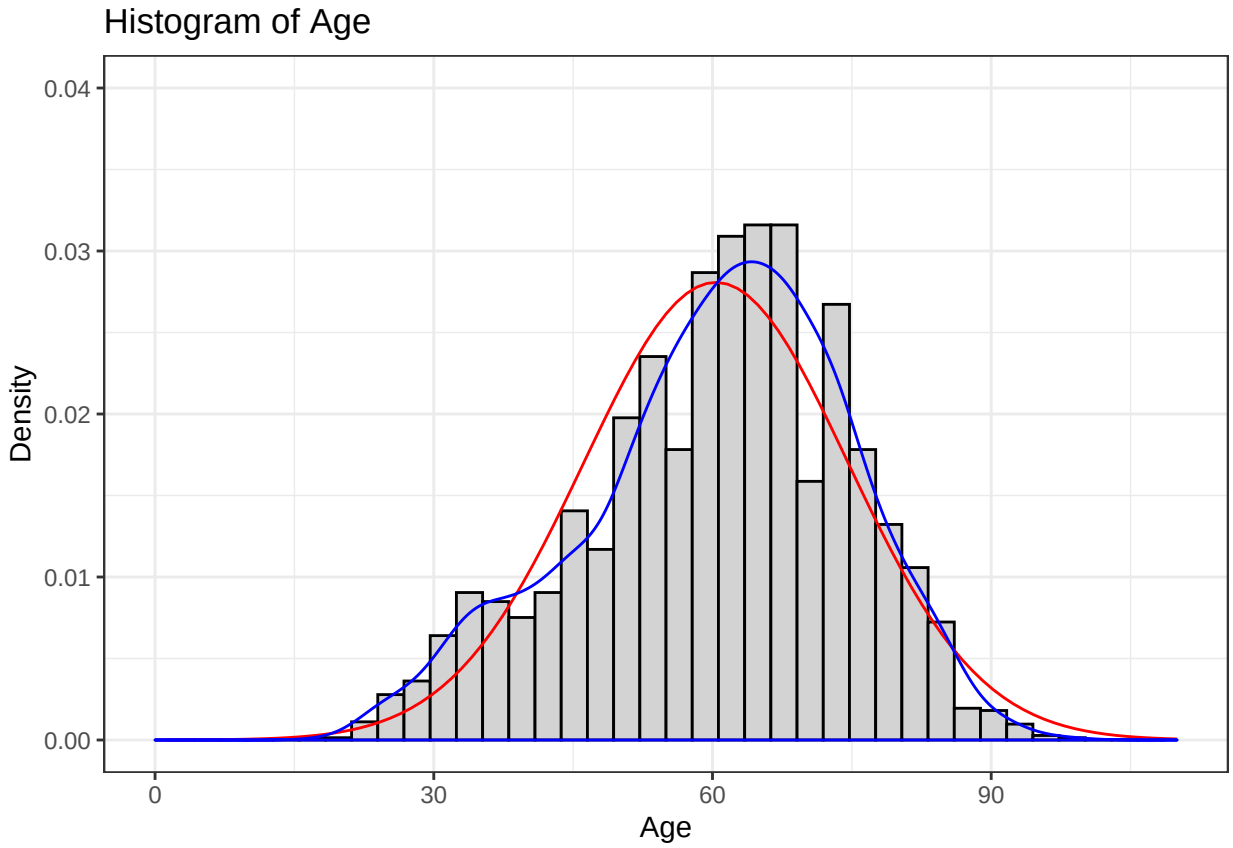
```
ggplot(ds) +
  geom_histogram(aes(x = age, y = ..density..), bins = 40) +
  stat_function(fun = dnorm, args = list(mean = mean(ds$age), sd = sd(ds$age)), color = "red") +
  geom_density(aes(x = age, y = ..density..), color = "blue")
```



The culmination of the histogram, curve, and density line is improved via the addition of limits and labels to the x-axis and y-axis, defining a number of bins, and a chart title. Including fill and outline colors for the histogram can also make it more readable:

```
ggplot(ds) +
  geom_histogram(aes(x = age, y = ..density..), bins = 40, fill = "#d3d3d3", color = "black") +
  stat_function(fun = dnorm, args = list(mean = mean(ds$age), sd = sd(ds$age)), color = "red") +
  geom_density(aes(x = age, y = ..density..), color = "blue") +
  ggtitle("Histogram of Age") +
  xlab("Age") +
  ylab("Density") +
  theme_bw() +
  lims(x = c(0, 110), y = c(0, 0.04))
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



4.2 Probability and Distributions

R supports a number of distributions; however, for the purpose of these labs we will focus primarily on the normal and binomial distributions. View the `help(Distributions)` documentation to explore the distributions supported by R.

help(Distributions)

The following R functions are applicable to the normal distribution:

1. `dnorm()`
2. `pnorm()`
3. `qnorm()`
4. `rnorm()`

The `dnorm()` function provides the height of a probability distribution function at a given x value: `dnorm(x, mean=$\mu$, standard deviation=$\sigma$)`. **Note:** The value returned by the `dnorm()` function is not the probability associated with the occurrence of the x value!

The default mean and standard deviation for the `dnorm()` function is 0 and 1, respectively. The following example finds the height of the probability distribution function at $x = 2$ with $\mu = 4$ and $\sigma = 1.5$.

```
dnorm(2, mean = 4, sd=1.5)
```

```
## [1] 0.10934
```

The `dnorm()` function used in conjunction with the `age` variable from `ds` data set can find the height of the probability distribution function. In the following example, the `dnorm()` function will find the height of the

probability distribution function for 65. Similar to previous examples, an argument exists to ignore NA and missing values.

```
dnorm(65, mean = mean(ds$age, na.rm = T), sd = sd(ds$age, na.rm = T))
```

```
## [1] 0.02662361
```

The `dnorm()` returns the height of the probability distribution function as 0.027. **Note:** This is a random value and, by itself, is not meaningful. The `dnorm()` function returns the relative likelihood, which can lead to determining a probability; however, to understand this value further requires an explanation of calculus.

For continuous data, the probability of a single value is small (near zero), so instead the approach should be to find the probability a value occurs within a specified range. The probability associated to a value occurring within a specified range is equal to the area of the probability distribution function between the two points. In calculus this is defined as finding the integral of the probability distribution function.

$$\int_{-x_1}^{x_2} f_X(t)\,dx$$

The above formula is the cumulative distribution function for two points, x_1 and x_2. In this case, x_1 is defined as the lower bound and x_2 is defined as the upper bound.

R includes the calculus function `integrate.xy()` to return the probability.

```
integrate.xy(density(ds$age)$x, density(ds$age)$y, 65, 66)
```

```
## [1] 0.02917993
```

The probability associated to an age between 65 and 66 in the `age` variable is .029 ($\approx 3\%$ chance).

Similarly, the `pnorm()` function calculates probabilities associated to a given x value. The default for the `pnorm()` function is the cumulative distribution function with a lower bound of $-\infty$ and an upper bound of x.

The following example calculates the probability associated to an age value between $-\infty$ and 5, given $\mu = 6$ and $\sigma = 6$.

```
pnorm(5, mean = 6, sd = 2, lower.tail = TRUE)
```

```
## [1] 0.3085375
```

The `pnorm()` calculates the probability of observing a value between $-\infty$ and 5 as 0.31. The following example uses the `pnorm()` function with the `ds` data set to find the probability that a respondent is 65 or less years old.

```
pnorm(65, mean = mean(ds$age, na.rm = TRUE), sd = sd(ds$age, na.rm = TRUE), lower.tail = TRUE)
```

```
## [1] 0.6277983
```

To calculate the probability associated to an age of 65 or greater, the `lower.tail = FALSE` argument will look at the upper tail (right side of the probability distribution function). This is equal to the difference between 1 and the lower tail probability previously calculated.

```
pnorm(65, mean = mean(ds$age, na.rm = TRUE), sd = sd(ds$age, na.rm = TRUE), lower.tail = FALSE)
```

```
## [1] 0.3722017
```

The `qnorm()` function is the inverse function of the `pnorm()` function. Given a probability, mean, and standard deviation, the `qnorm()` function will return an x value from the probability distribution function. The following example finds the upper bound x value of the probability distribution function associated to the probability, or area under the curve, of 0.3 given $\mu = 5$ and $\sigma = 1$.

```r
qnorm(0.30, mean = 5, sd = 1, lower.tail = TRUE)
```

```
## [1] 4.475599
```

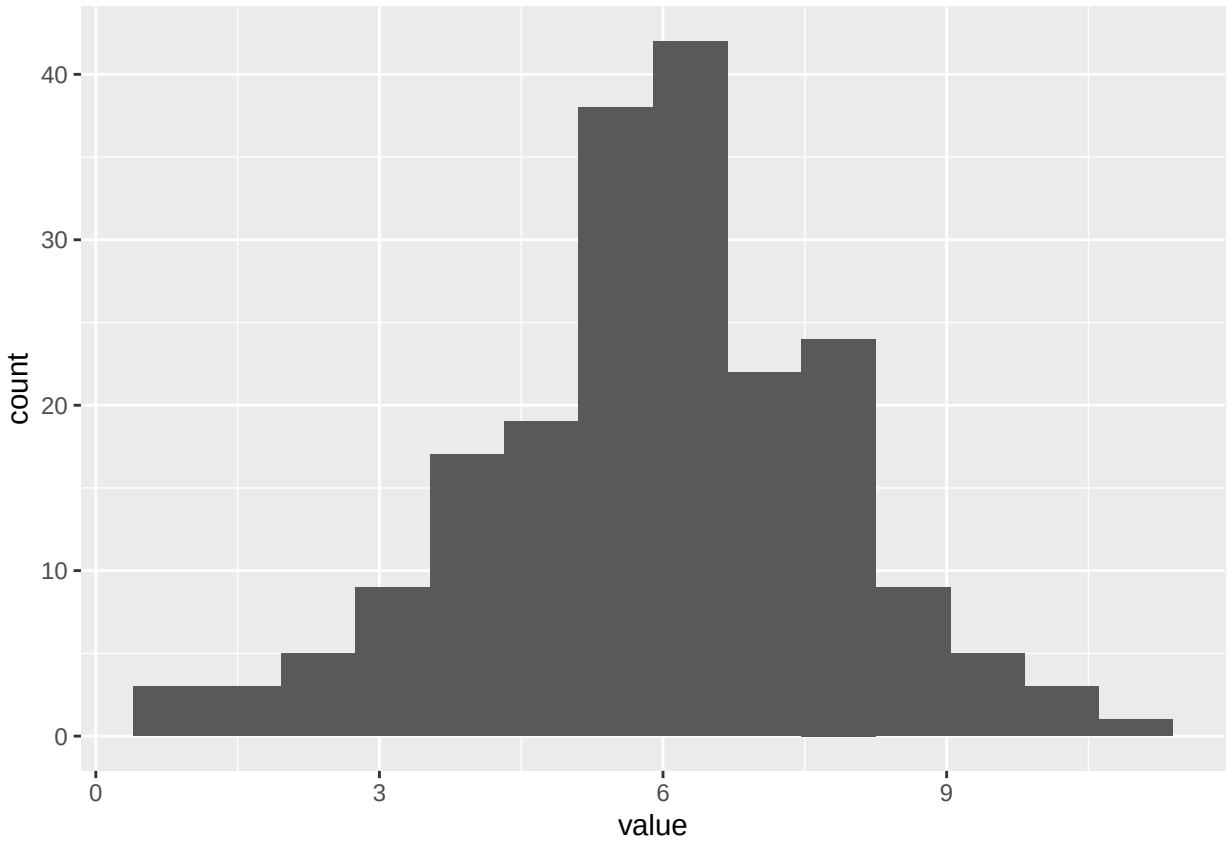
The following calculates the upper bound age from the `age` variable in the `ds` data set to demonstrate further associated to a 40% probability. That is, the `qnorm()` function calculates the age that 40% of respondents are equal or less to.

```r
qnorm(0.40, mean = mean(ds$age, na.rm = TRUE), sd = sd(ds$age, na.rm = TRUE), lower.tail = TRUE)
```

```
## [1] 56.7677
```

Lastly, the `rnorm()` function will generate random values that follow a normal distribution given a number of points (n), provided μ, and σ. The following example calculates 200 random values given $\mu = 6$ and $\sigma = 2$. The random values are stored to the `rvalues` object.

```r
rvalues <- data.frame(value = rnorm(200, mean= 6, sd = 2))
ggplot(rvalues, aes(x = value)) +
  geom_histogram(bins = 14)
```



Note: The discussed functions are relevant to the normal distribution functions provided by R. R includes similar functions for other distributions, with equivalent functionality.

4.3 Visualizing Normality

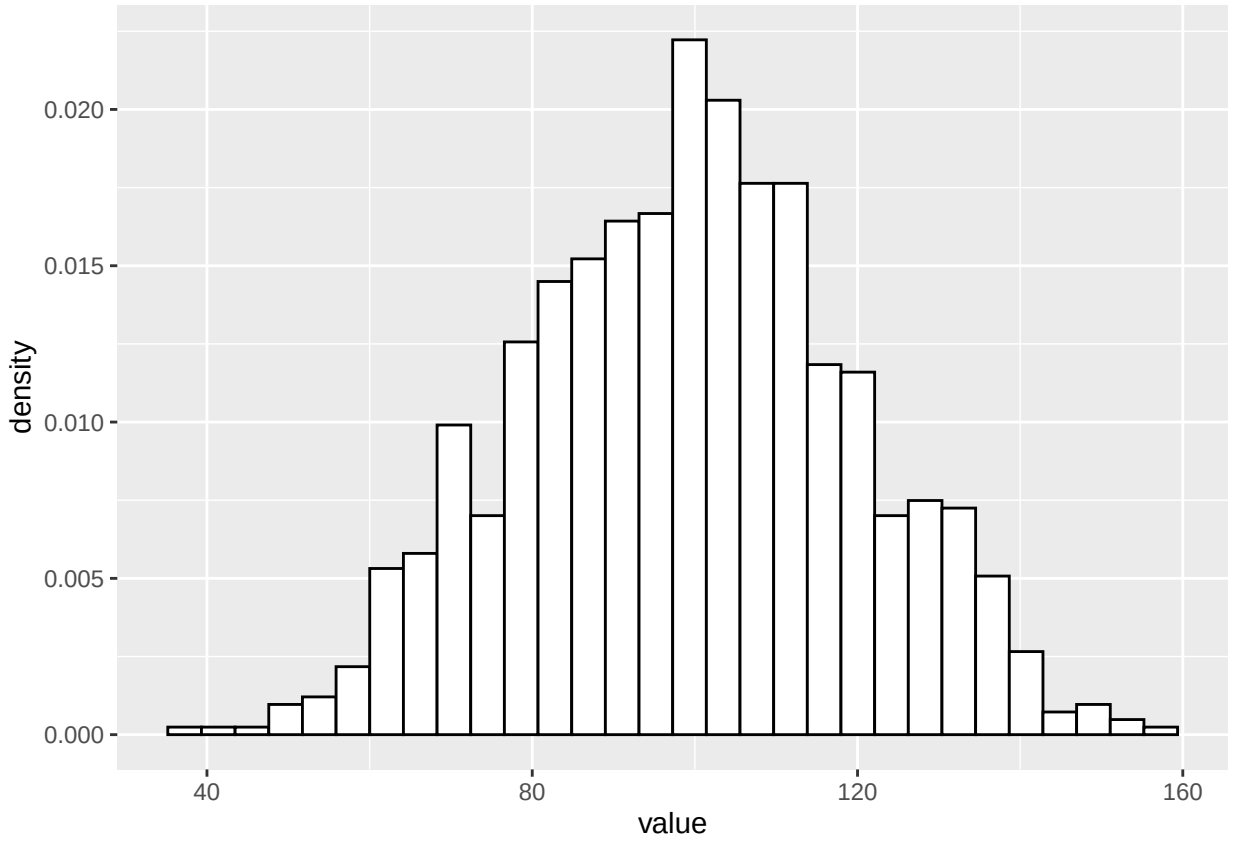
Thus far the normal distribution has been discussed without visualization. When graphed, data that follow a normal distribution resemble a bell shaped curve. To demonstrate, the following code employs the `rnorm()` function to generate 1000 random values with $\mu = 100$ and $\sigma = 20$, and assigns the values to an object named `random`.

```
random <- data.frame(value = rnorm(1000, mean = 100, sd = 20))
```

Inspecting a density histogram of the `random` object yields a bell shaped curve.

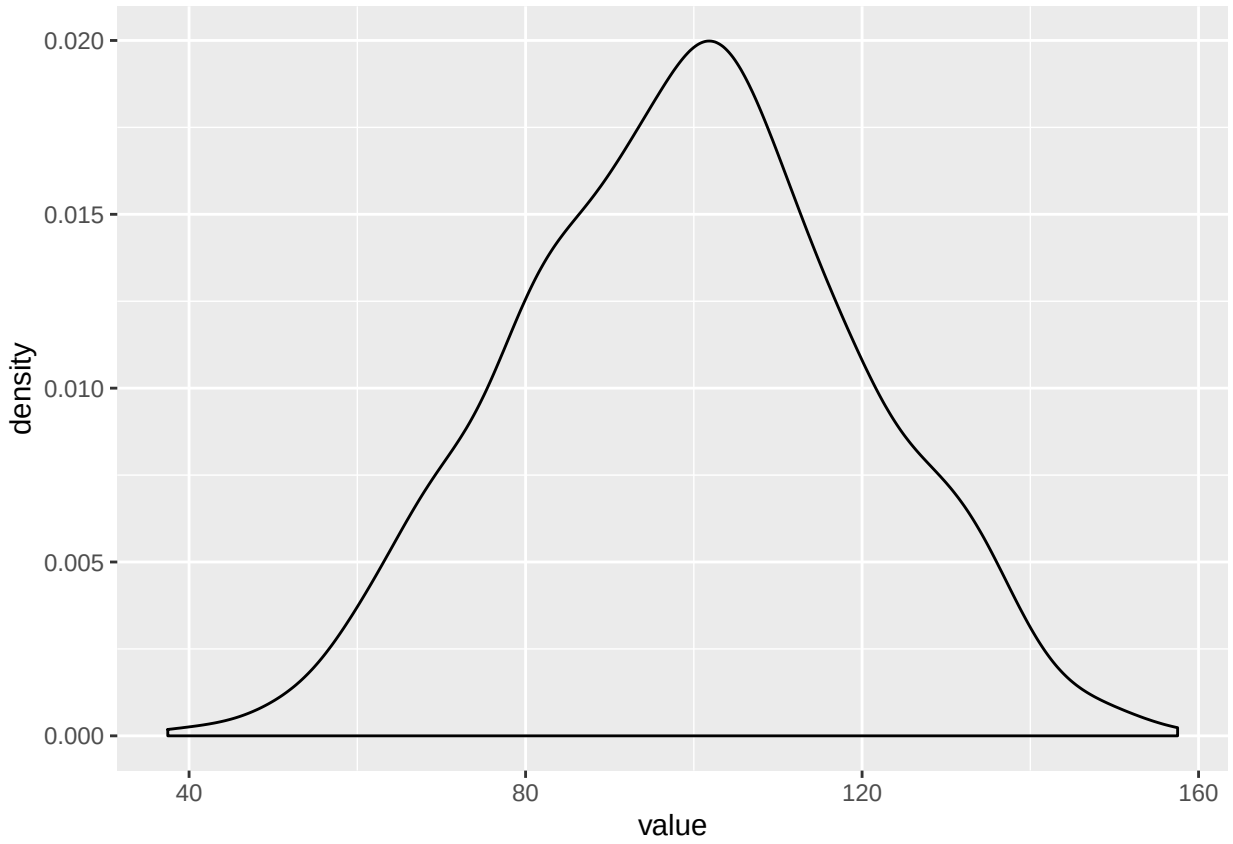
```
ggplot(random, aes(x = value, y = ..density..)) +
  geom_histogram(fill = "white", color = "black")
```

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



Let's start figuring out how to check if our data is normally distributed. There are many packages than will generate a density curve of your data and a projected normal distribution for comparison, but building all of the visualizations in ggplot provides both an intuitive and informative method of doing so. Start by creating a density plot of the randomly generated data.

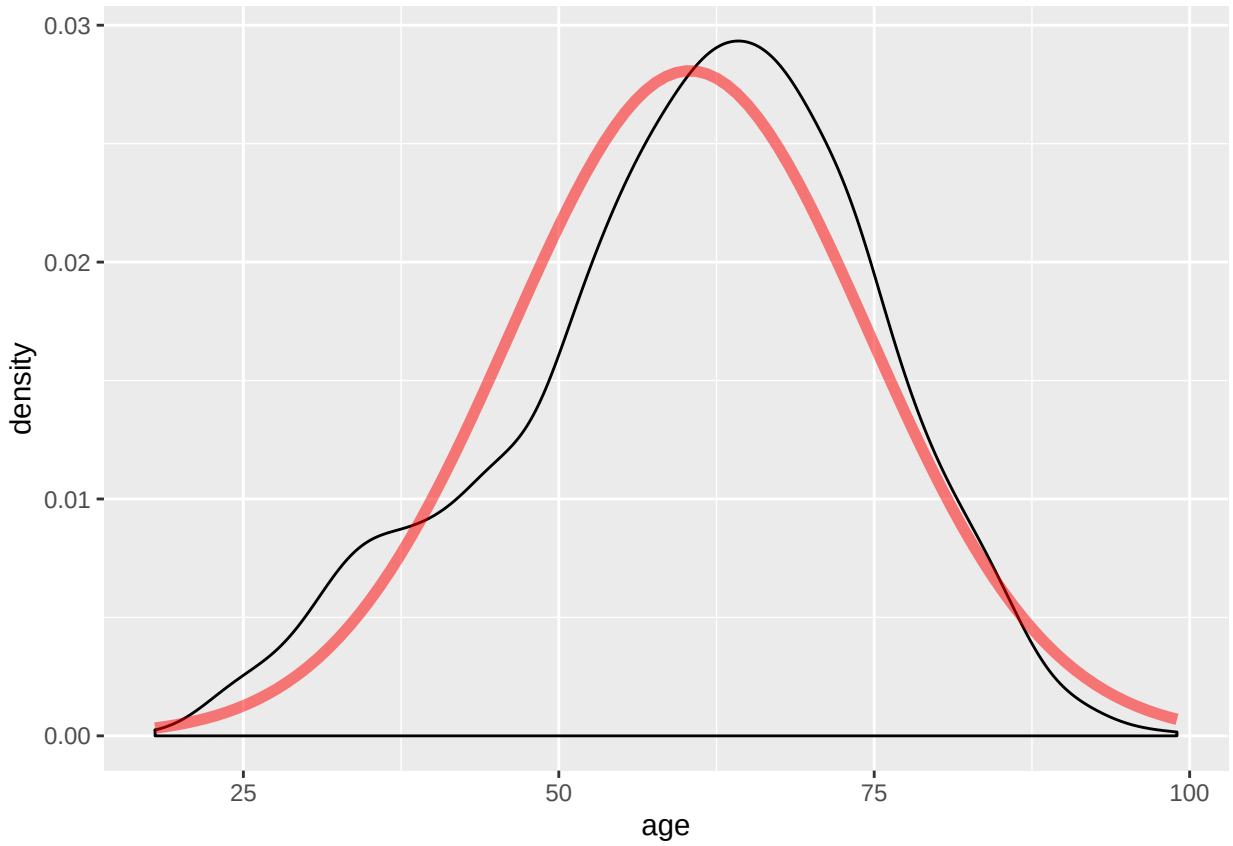
```
ggplot(random, aes(x = value)) +
  geom_density()
```



Given the `random` values consists of values generated by the `rnorm()` function, this distrubtion resembling the normal distribution is unsurprising.

The following code generates a density line for the `age` variable from the `ds` data set and a projected normal distribution given the mean and standaard deviation of the variable. Indicating `alpha=.5` will make the line slightly transparent.

```
ggplot(ds) +
  geom_density(aes(x = age, y = ..density..)) +
  stat_function(fun = dnorm, args = list(mean = mean(ds$age), sd = sd(ds$age)), color = "red", size = 2
```



The shape of the density line closely resembles a normal distribution; however, note the slight skew.

4.4 Z-Scores

Standardizing, or scaling, data provides conveniences in discussing data. For instance, discussing how many standard deviations a particular value occurs from the mean is more meaningful than purely the distance. Scaling data in terms of z-scores provides the number of standard deviations a value is from the mean.

The following example employs the `scale()` function to calculate z-score for each data point and assigns them to a newly created `z.age` variable in the `ds` data set. To do this, use the `mutate()` function, which is a tidyverse function that creates new variables or modifies existing variables. The `scale()` function is enclosed by the `c()` function to ensure the result is a 1 dimensional vector:

```
ds <- ds %>%
  mutate(z.age = c(scale(age)))
```

Using a filter approach, the following example finds the z-score associated to respondents younger than 19 years old. First filter the data with the preferred stipulations, then use the `select()` verb from the `dplyr()` package (part of the tidyverse) to view the results. All of this is connected using the pipe function, `%>%`.

```
ds %>%
  filter(age < 19) %>%
  dplyr::select(age, z.age)
```

```
##   age     z.age
## 1  18 -2.981749
```

The result shows that, within the `ds` data set, there is one respondent who is under 19 years old—their age was 18. The z-score for this respondent is -2.98, which is interpreted as this respondent's age is 2.98 standard

deviations below the mean. We can use the `pnorm()` function to calculate the percentile (probability) of this score within the sample.

```
pnorm(-2.98, mean = 0, sd = 1, lower.tail = FALSE)
```

```
## [1] 0.9985588
```

This respondent is younger than 99.9% of the respondents in the sample.

5 Foundations for Inference

This lab introduces the tools for the foundation of inference by examining the normal distribution, standard errors, confidence errors, and single sample t-tests. The following packages are required:

1. car
2. psych
3. sm
4. HistData
5. tidyverse

5.1 Testing for Normality

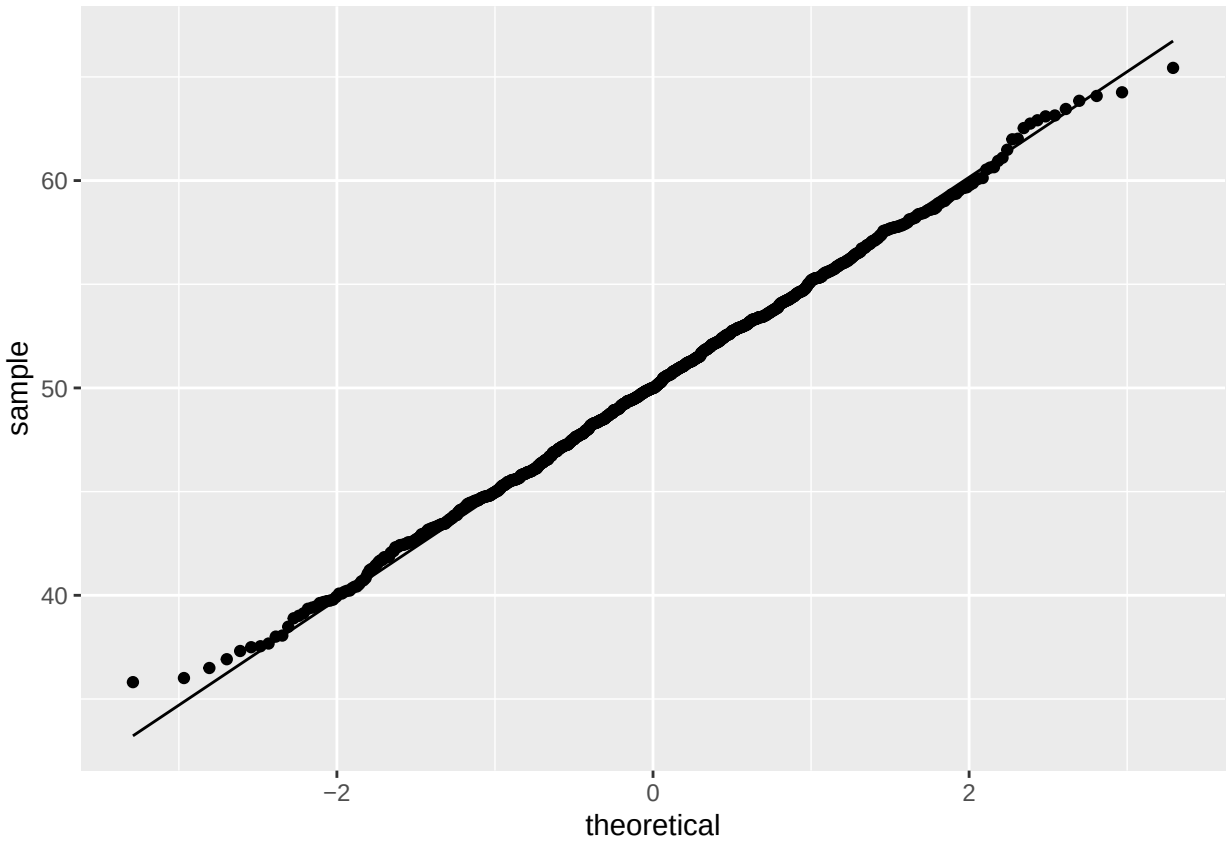
Lab three introduced the basics of normal distributions. Much of the statistical work done is built on the condition of data that follows a normal distribution. Inspecting data for normality is done via numerous methods.

Recall the `rnorm()` function to generate random values that follow a normal distribution given specified parameters. The following `random` object will consist of 1000 random values given $\mu = 50$ and $\sigma = 5$:

```
random <- rnorm(1000, mean = 50, sd = 5)
```

Visualizing data is an important first step to inspecting data for normality. The previous lab introduced density plots, box plots, and QQ plots as means of visualizing data and their relationships to a normal distribution. Recall that the QQ plot graphs the quantiles of data against quantiles of a normal distribution, given the μ and σ of the data.

```
ggplot(as.data.frame(random), aes(sample = random)) +
  stat_qq() +
  stat_qq_line()
```



The QQ plot of the `random` object demonstrates the data closely follows a normal distribution. This is expected given the `random` object was created via the `rnorm()` function. **Note:** Visualizing data for normality is an informal approach to inspecting for normality. Various empirical methods exist to test whether data follows a normal distribution, the most popular of which are:

1. Shapiro-Wilk test
2. Anderson-Darling test
3. Kolmogorov-Smirnov test
4. Pearson's Chi-Squared test

The Shapiro-Wilk test is the most popular method, as the test has been demonstrate as providing the most power for a given significance.

5.1.1 Shapiro-Wilk Test

The `shapiro.test()` function in R employs the Shapiro-Wilk test on data to test whether the data are normally distributed. Use of the Shapiro-Wilk test is contingent on univariate and continuous data. The hypotheses for the Shapiro-Wilk test are:

H_0: The data are normally distributed
H_1: The data are not normally distributed

Note: The Shapiro-Wilk test for normality is a statistical test that provides a p-value of the test statistic, `W`. This lab will focus on the p-value approach for statistical tests, using an α value of 0.05 as the desired significance level.

```
shapiro.test(random)
```

```
##
```

```
##  Shapiro-Wilk normality test
##
## data:  random
## W = 0.99888, p-value = 0.8073
```

The Shapiro-Wilk test p-value is greater than $\alpha = 0.05$, therefore failing to reject H_0 concluding the data are normally distributed. Again, this is expected given the `random` object was created via the `rnorm()` function.

5.1.2 Testing Normality

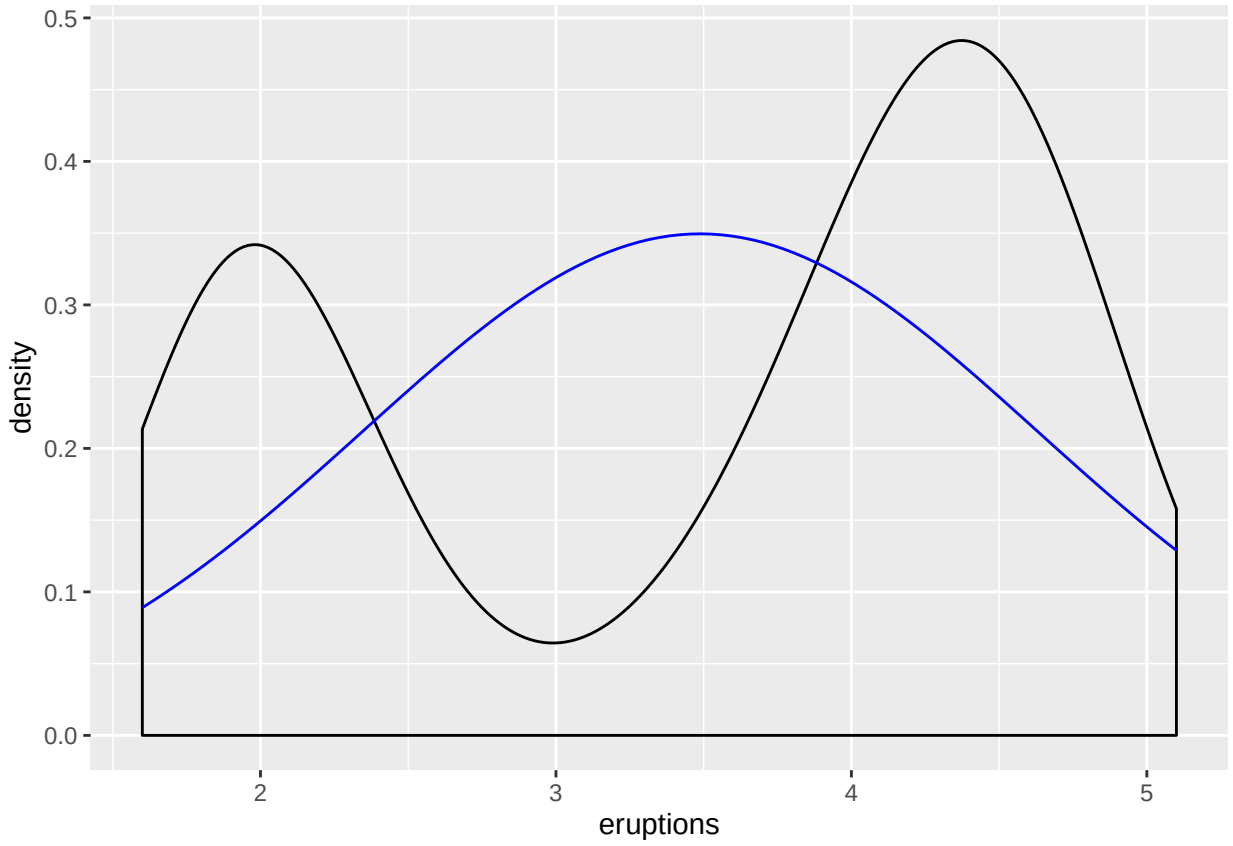
R provides data pertaining to Yellowstone National Park's Old Faithful geyser in the `faithful` object. Old Faithful eruptions are recorded as duration, in minutes, between events. First, the `describe()` function provides valuable information for the `eruptions` variable in the `faithful` object.

```
describe(faithful$eruptions)
```

```
##    vars   n mean   sd median trimmed  mad min max range  skew kurtosis
## X1    1 272 3.49 1.14      4    3.53 0.95 1.6 5.1   3.5 -0.41    -1.51
##      se
## X1 0.07
```

Comparing the `eruptions` data to a normal distribution, given μ and σ from the `eruptions` data, is available via the *geom_density()* and *stat_function()* functions.

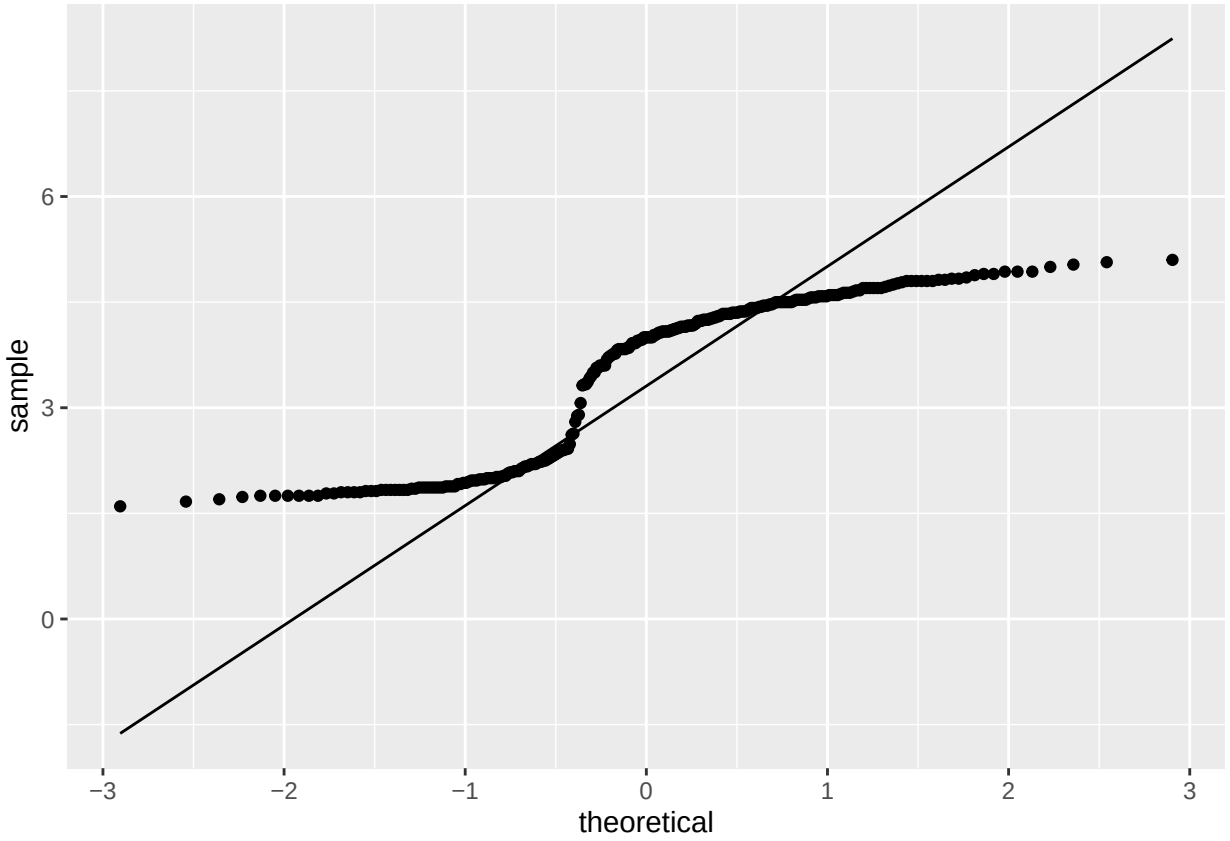
```
ggplot(faithful, aes(eruptions)) +
  geom_density() +
  stat_function(fun = dnorm, args = list(mean = mean(faithful$eruptions),
                                         sd = sd(faithful$eruptions)), color = "blue")
```



The black line represents the `eruptions` data, and the blue line represents the normal distribution given the μ and σ values of `eruptions`. **Note:** The `eruptions` data appears bimodal, and does not fit the normal distribution model given the parameters calculated via the `eruptions` data.

The `eruptions` data is further examined using QQ plots via the `qqPlot()` function:

```
ggplot(faithful, aes(sample = eruptions)) +
  stat_qq() +
  stat_qq_line()
```



Most of the points in the QQ plot fall outside the region defined by the dashed lines, further suggesting the `eruptions` data is likely not normally distributed. Lastly, a Shapiro-Wilk test can confirm whether the `eruptions` data is normally distributed:

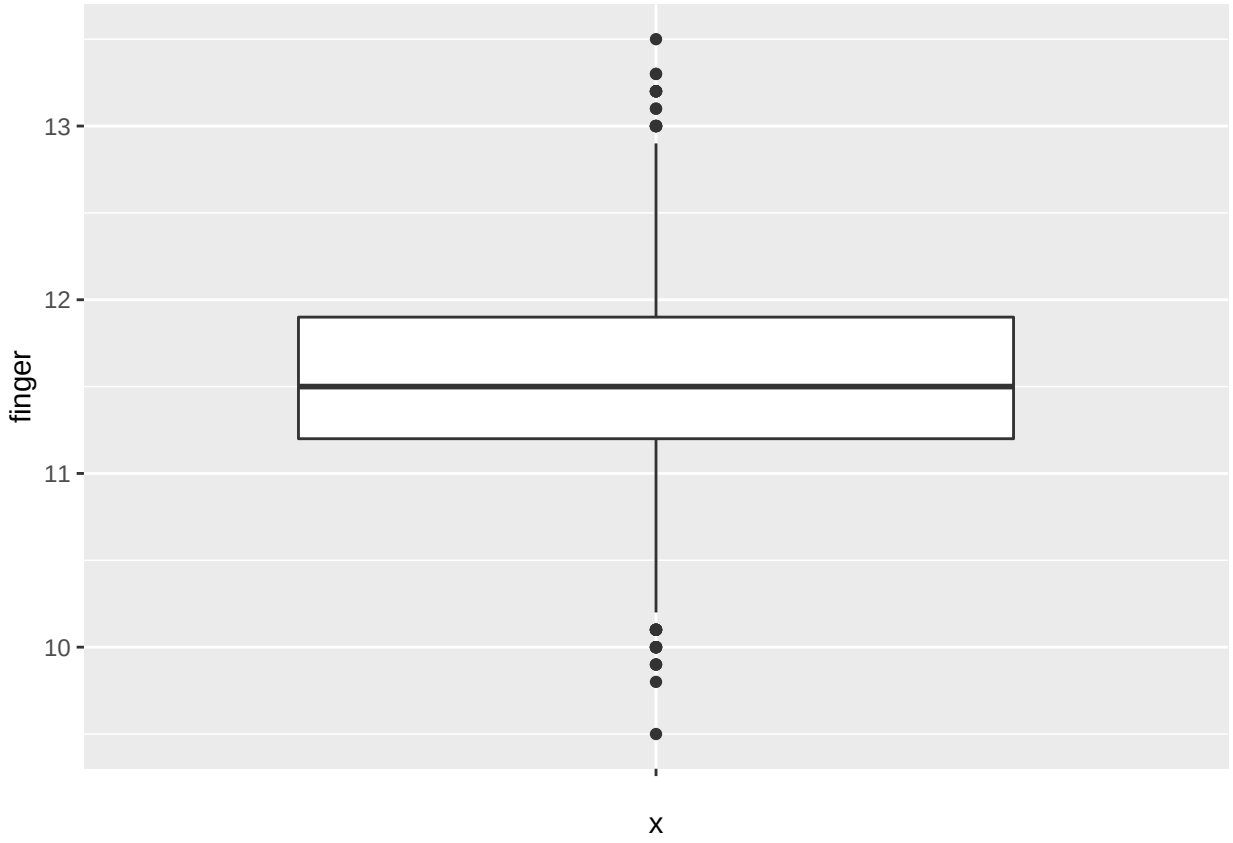
```
shapiro.test(faithful$eruptions)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  faithful$eruptions
## W = 0.84592, p-value = 0.0000000000000009036
```

The Shapiro-Wilk test p-value is less than $\alpha = 0.05$, leading to reject H_0: data are normally distributed. In conclusion, the `eruptions` data is not normally distributed. **Note:** The visual plots are likely enough to confirm the `eruptions` data are not normally distributed.

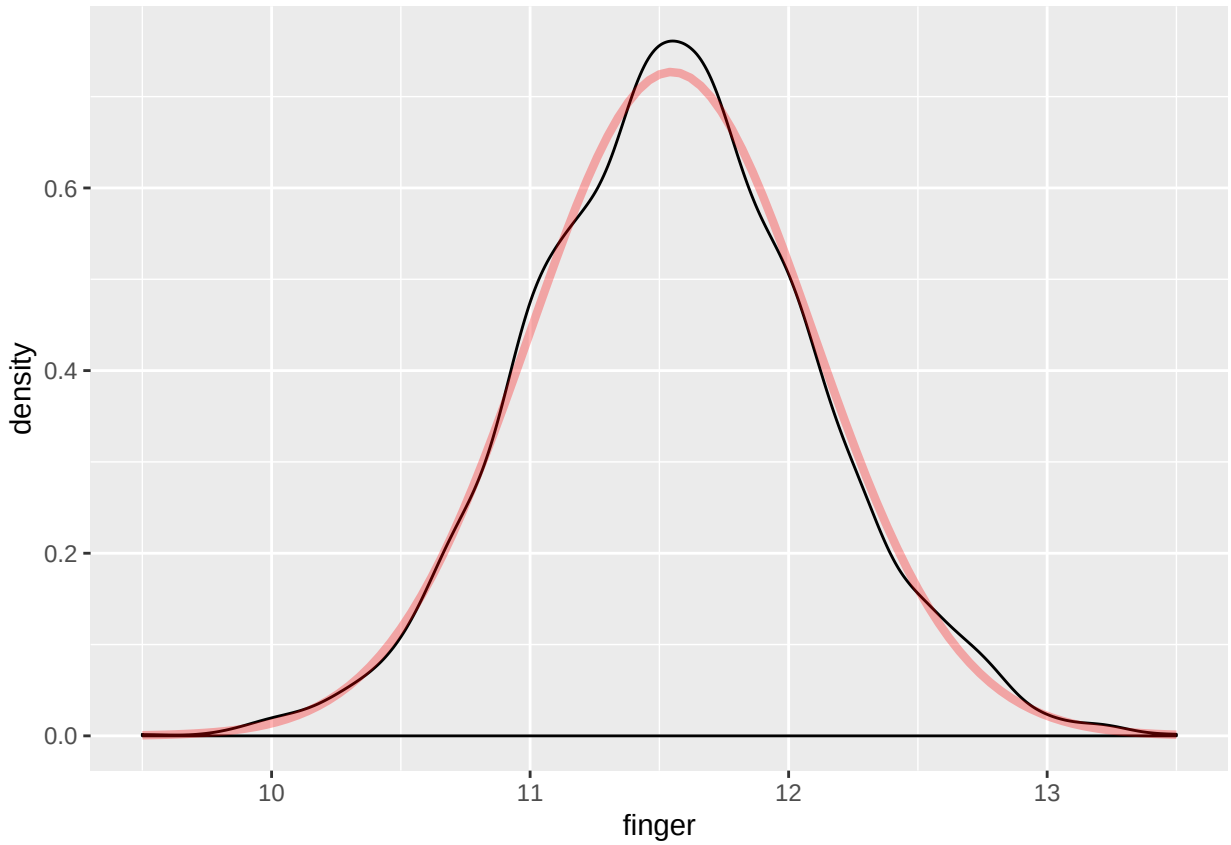
The `MacdonellDF` variable within the `HistData` package consists of finger length data. Again, visualizing the data for normality is performed via various methods. A box plot is generated for the `MacdonellDF` data:

```
ggplot(MacdonellDF, aes(x = "", y = finger)) +
  geom_boxplot()
```



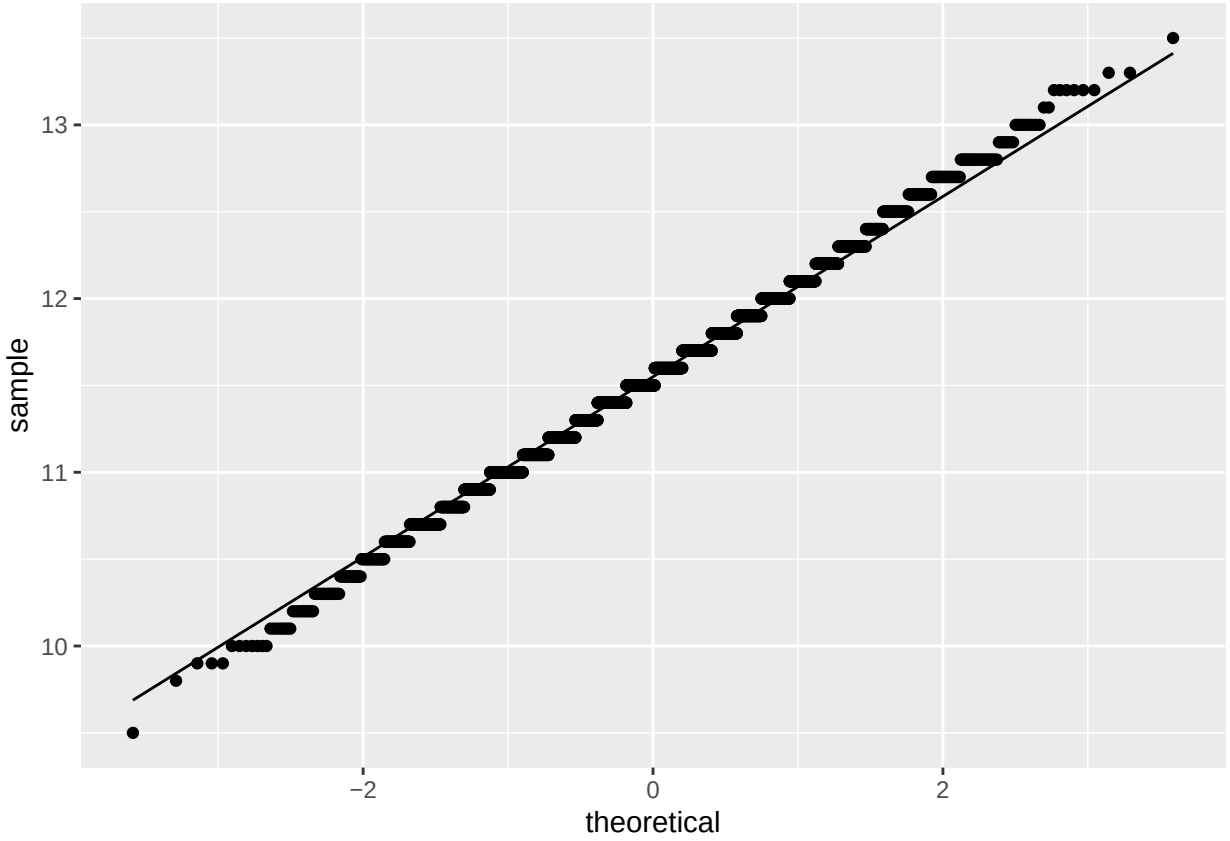
The box plot appears balanced, indicating normality. Generating a density plot should show the distribution of `MacdonellDF` variable as similar to the projected normal distribution given the μ and σ parameters of the `MacdonellDF` variable.

```
ggplot(MacdonellDF, aes(finger)) +
  geom_density() +
  stat_function(fun = "dnorm", args = list(mean = mean(MacdonellDF$finger),
                                           sd = sd(MacdonellDF$finger)),
                color = "red", size = 1.5,
                alpha = .3)
```



Additionally, a QQ plot:

```
ggplot(MacdonellDF, aes(sample = finger)) +
  stat_qq() +
  stat_qq_line()
```



The visualizations all suggest the data is normally distributed; however, a Shapiro-Wilk test is still useful to test for normality.

```
shapiro.test(MacdonellDF$finger)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  MacdonellDF$finger
## W = 0.99646, p-value = 0.000001666
```

Note: Despite the visualizations suggesting normality, the Shapiro-Wilk test p-value is less than $\alpha = 0.05$, resulting in rejecting the null hypothesis that the data are normally distributed. A closer examination of the QQ plot yields that most points exist outside the region, though at first glance nothing appears suspect. This merits a conversation on judgment for assessing whether the data could be treated as normally distributed.

5.2 Standard Errors

Recall that the standard error is the standard deviation of the sample distribution, calculated as the square root of the standard deviation divided by the square root of the sample size.

- Population: $\frac{\sigma}{\sqrt{n}}$
- Sample: $\frac{s}{\sqrt{n}}$

R does not provide a single purpose function to calculate the standard error. As a result, the following demonstrates the previous formula to calculate the standard error:

```
sd(ds$age, na.rm = T)/sqrt(length(ds$age) - sum(is.na(ds$age)))
```

```
## [1] 0.2815446
```

Alternatively, the `describe()` function includes the standard error statistic for a given variable, as follows:

```
describe(ds$age)
```

```
##    vars    n  mean    sd median trimmed   mad min max range  skew kurtosis
## X1    1 2547 60.37 14.21     62   61.01 13.34  18  99    81 -0.39    -0.24
##      se
## X1 0.28
```

Note: The previous two methods return the same result.

5.3 Confidence Intervals

The standard error is vital to inferential statistics. For example, the standard error is required to calculate confidence intervals. Confidence intervals employ standard error to assist with inferring information from a sample of a larger population, through inclusion of uncertainty.

Confidence intervals synthesize knowledge of standard error and z-scores. Recall that with the normal distribution, a z-score standardizes standard deviations values are from the mean. To calculate a level of confidence (90%, 95%, 99%), the z-score associated with those levels of confidence is necessary:

- 90% confidence = 1.645
- 95% confidence = 1.960
- 99% confidence = 2.576

The formula for confidence level is:

$$CI = \bar{x} \pm z\frac{s}{\sqrt{n}},$$

or

$$CI = \bar{x} \pm z * SE$$

Put simply, the confidence interval is the sample mean plus/minus the product of the z-score and standard error.

Using the `age` variable within the `ds` data set, the a 95% confidence interval is calculated as follows:

First calculate the mean:

```
mean(ds$age, na.rm = T)
```

```
## [1] 60.36749
```

Second, find the standard error:

```
describe(ds$age)$se # this is a way to pull the standard error out without having to look at everything
```

```
## [1] 0.2815446
```

Given a 95% confidence interval, the z-score is 1.96. The upper bound of the confidence interval is calculated as follows:

```
60.38 + 1.96 * 0.28
```

```
## [1] 60.9288
```

The lower bound of the confidence interval is calculated as follows:

```
60.38 - 1.96 * 0.28
```

```
## [1] 59.8312
```

With 95% confidence, the population mean of `age` is between 59.83 and 60.93 (rounded).

Increasing confidence will increase the interval of the confidence interval. For example, the z-score associated to 99% is 2.58, so the confidence interval for 99% is calculated as follows:

```
age.se <- describe(ds$age)$se
```

Note: The standard error of the `age` variable was stored as to the `age.se` object via specifying only to return the standard error from the `describe()` function. Using the `se` object simplifies the confidence interval calculation as exemplified:

```
mean(ds$age, na.rm = T) + 2.58 * age.se
```

```
## [1] 61.09388
```

```
mean(ds$age, na.rm = T) - 2.58 * age.se
```

```
## [1] 59.64111
```

With 99% confidence, the population mean of `age` is between 59.64 and 61.09. This interval is larger than the interval calculated for 95% confidence.

Alternatively, the `t.test()` function in R provides the 95% confidence interval for a given variable.

```
t.test(ds$age)
```

```
##
##  One Sample t-test
##
## data:  ds$age
## t = 214.42, df = 2546, p-value < 0.00000000000000022
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  59.81541 60.91957
## sample estimates:
## mean of x
##  60.36749
```

The results of *t.test()* is similar to the manual calculations performed previously. By default, the *t.test()* tests whether μ for the variable is equal to zero. The hypotheses for the *t.test()* function are:

- $H_0 : \bar{x} = \mu$
- $H_1 : \bar{x} \neq \mu$

Note the returned p-value of the previous `t.test()` performed is less than $\alpha = 0.05$; therefore, the null hypothesis is rejected. That is, given the sample data, with 95% confidence the μ is not zero.

To explain a different way, using the same variable, but with a defined μ of 50, the `t.test()` function can test whether μ is 50.

```
t.test(ds$age, mu = 50)
```

```
##
##  One Sample t-test
##
## data:  ds$age
## t = 36.824, df = 2546, p-value < 0.00000000000000022
```

```
## alternative hypothesis: true mean is not equal to 50
## 95 percent confidence interval:
##  59.81541 60.91957
## sample estimates:
## mean of x
##  60.36749
```

The same result occurs: the null hypothesis is rejected. Once again, the `t.test()` function is employed to test whether $\mu = 60$:

```
t.test(ds$age, mu = 60)
```

```
##
##  One Sample t-test
##
## data:  ds$age
## t = 1.3053, df = 2546, p-value = 0.1919
## alternative hypothesis: true mean is not equal to 60
## 95 percent confidence interval:
##  59.81541 60.91957
## sample estimates:
## mean of x
##  60.36749
```

The result differs, such that the null hypothesis is not rejected as the p-value is greater than $\alpha = 0.05$. That is, with 95% confidence, $\mu = 60$.

Note: The p-values can be calculated manually from z-scores to test population means using confidence intervals. This requires:

1. Calculate $\bar{x}$
2. Calculate confidence intervals (as previously shown)
3. Calculate the p-value associated with H_0

Objects are employed to simplify the calculation:

```
xbar <- mean(ds$age, na.rm = T)
a <- 60
s <- sd(ds$age, na.rm = T)
n <- 2547
```

The p-value for if the population mean $\neq 60$:

```
2 * (1 - pnorm(xbar, mean = a, sd = s/sqrt(n)))
```

```
## [1] 0.1918016
```

The p-value of 0.191 is greater than $\alpha = 0.05$; therefore, failing to reject H_0: $\mu = 60$.

5.4 More on Single Sample T-tests

The previous confidence interval section introduced a preliminary discussion of t-tests. The Student's t distribution is based on sample estimates, whereas the normal distribution is used when σ and μ are known.

Performing a t-test requires:

1. n
2. s
3. $\bar{x}$

4. t statistic

To demonstrate, the following tests whether the population mean is 58 for the given data by defining the above parameters:

```
nAge <- length(ds$age) - sum(is.na(ds$age))
sdAge <- sd(ds$age, na.rm = TRUE) # Standard deviation of age
seAge <- sdAge/(sqrt(nAge)) # Standard error of age
meanAge <- mean(ds$age, na.rm = TRUE)
```

To calculate the t statistic, the population mean is subtracted from the sample mean, and divided by the standard error as follows:

```
t_value <- (meanAge - 58)/seAge
t_value
```

```
## [1] 8.408938
```

To calculate the two-sided p value under the t distribution:

```
2 * (1 - pt(abs(t_value), df = nAge - 1))
```

```
## [1] 0
```

The returned p-value is less than $\alpha = 0.05$, rejecting H_0: $\mu = 60$. Put another way, H_1: $\mu \neq 60$.

```
t.test(ds$age, mu = 58)
```

```
##
##  One Sample t-test
##
## data:  ds$age
## t = 8.4089, df = 2546, p-value < 0.00000000000000022
## alternative hypothesis: true mean is not equal to 58
## 95 percent confidence interval:
##  59.81541 60.91957
## sample estimates:
## mean of x
##  60.36749
```

With 95% confidence, $\mu \neq 60$

6 Inference for Two Populations

This lab covers the basics of inference for two populations. We go through proportions and cross tabulations; the different types of two sample t-tests; difference in one and two tailed tests; and how to plot means and the differences between means. The following packages are required for this lab:

1. tidyverse
2. psych
3. car
4. vcd
5. reshape2
6. skimr

6.1 Proportions

To start with proportions, We will use the `glbcc` variable from the class dataset that contains respondents' opinions on global climate change. Specifically, it asks if they believe that greenhouse gases cause global temperatures to increase. We start with describing the data. A zero indicates "no" and 1 indicates "yes."

```
ds %>%
  count(glbcc)
```

```
## # A tibble: 2 x 2
##   glbcc     n
##   <int> <int>
## 1     0  1092
## 2     1  1455
```

Now we can describe the population proportions.

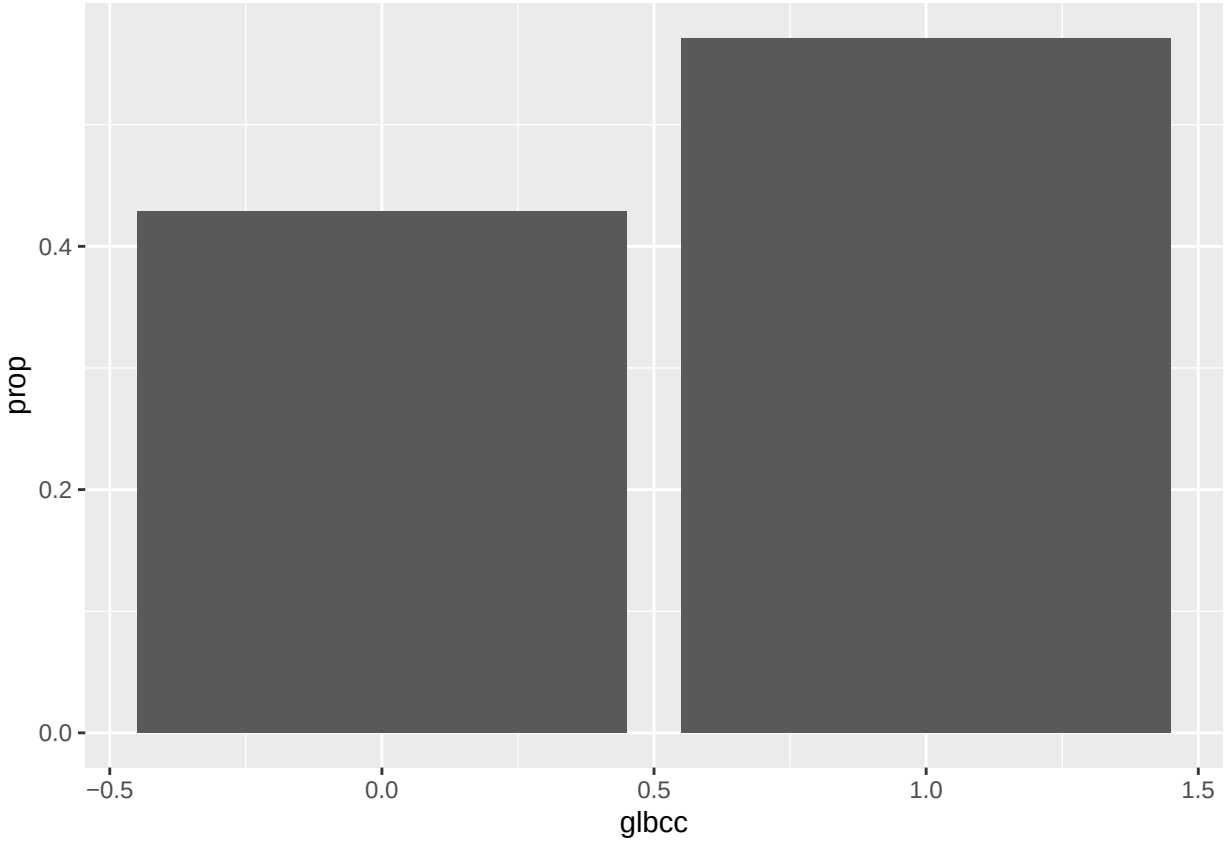
```
ds %>%
  count(glbcc) %>%
  mutate(prop = n / sum(n))
```

```
## # A tibble: 2 x 3
##   glbcc     n  prop
##   <int> <int> <dbl>
## 1     0  1092 0.429
## 2     1  1455 0.571
```

The above table describes the proportions of the population that believe humans cause climate change. Let's visualize the proportions.

Data frames are required for ggplot visualizations. Sometimes you have to construct the data frame manually, using the `data.frame()` function and creating vectors inside it with `c()`. However, when working with tidyverse functions, there are often shortcuts. Below we are able to visualize the proportion table by simply piping it directly to `ggplot2` and putting a `.` in place of the data set.

```
ds %>%
  count(glbcc) %>%
  mutate(prop = n / sum(n)) %>%
  ggplot(., aes(glbcc, prop)) +
  geom_bar(stat = "identity")
```



As we learned in the last lab, there are uncertainties associated to point estimates. We can include confidence intervals to get a range of values for which we are confident (at some level) the value actually falls between. To calculate confidence intervals we need to find the standard error of the proportionn and assign it to an object that we will name `se.gcc`.

```
se.gcc <-sqrt((0.5713 * (1 - 0.5713)/2547))
se.gcc
```

```
## [1] 0.009806056
```

Alternatively, we can use the `describe()` function to avoid rounding errors:

```
se.gcc <- describe(ds$glbcc)$se
se.gcc
```

```
## [1] 0.009808095
```

With the standard error object, we can calculate the confidence intervals. Here we will calculate the following: the upper bounds and lower bounds for the yes and no confidence intervals.

$$CI = \hat{p} \pm z\frac{s}{\sqrt{n}},$$

Where $\hat{p}$ is the sample proportion. Recall that 95% confidence corresponds to a 1.96 z-score.

The proportion table from earlier provides the values we will use:

```
ds %>%
  count(glbcc) %>%
  mutate(prop = n / sum(n))
```

```
## # A tibble: 2 x 3
##   glbcc     n  prop
##   <int> <int> <dbl>
## 1     0  1092 0.429
## 2     1  1455 0.571
```

First store the standard error as an object:

```
se.gcc <- se.gcc <- describe(ds$glbcc)$se
```

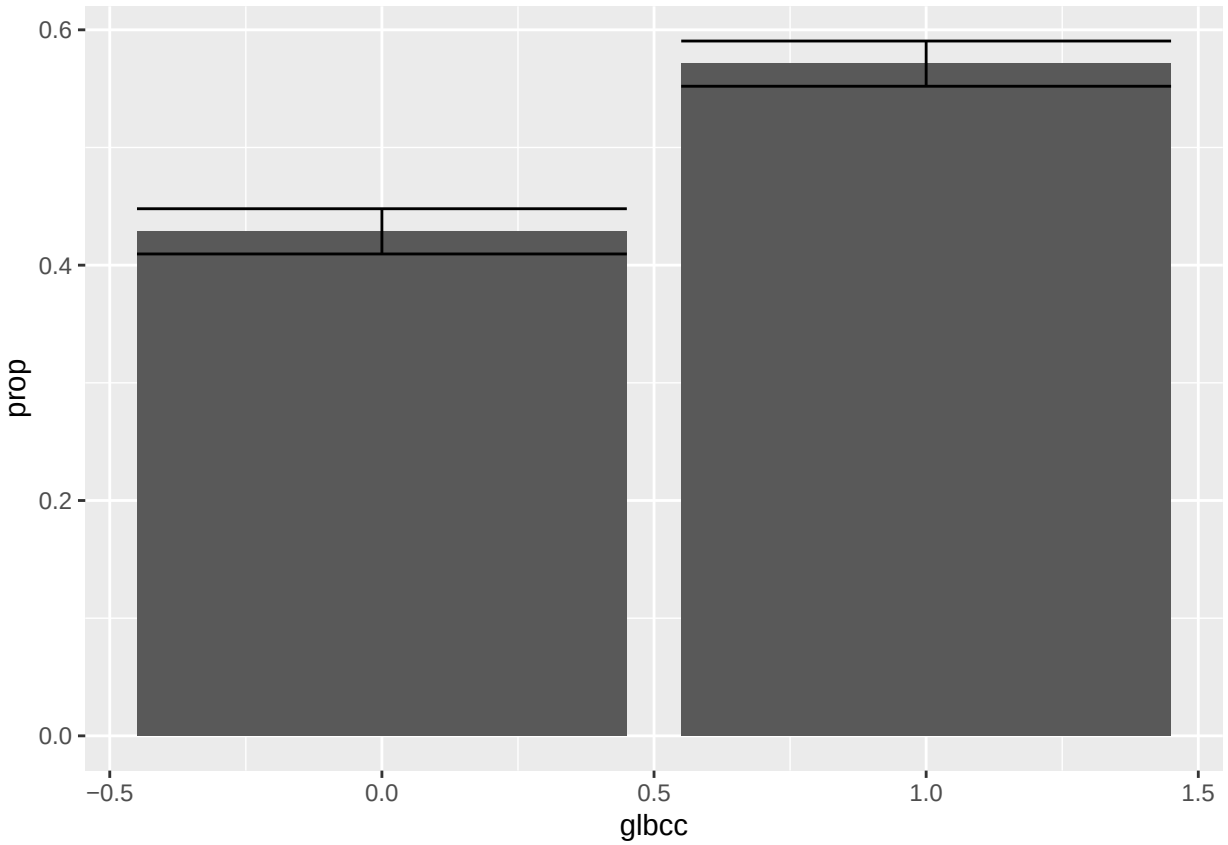
Recall that the `mutate()` verb can be used to create multiple variables at once. With this in mind, we can easily create a data frame that contains the measurements of interest, the standard error, and the lower and upper bounds of the confidence interval. For a 95% confidence interval, you take the proportion, plus or minus 1.96 muliplties by the standard error. Inside the `mutate()` function create three new variables: the standard error, lower, and upper:

```
new.ds <- ds %>%
  count(glbcc) %>%
  mutate(prop = n / sum(n),
         se.gcc = se.gcc,
         lower = prop - 1.96 * se.gcc,
         upper = prop + 1.96 * se.gcc)
new.ds
```

```
## # A tibble: 2 x 6
##   glbcc     n  prop  se.gcc lower upper
##   <int> <int> <dbl>   <dbl> <dbl> <dbl>
## 1     0  1092 0.429 0.00981 0.410 0.448
## 2     1  1455 0.571 0.00981 0.552 0.590
```

With the new data frame we can create a visualization that includes the confidence intervals. The code is similar to the previous `ggplot2` code used, but now with the addition of the `geom_errorbar()` function. The `geom_errorbar()` function requires arguments to use the lower and upper bound values for the confidence intervals:

```
ggplot(new.ds, aes(x = glbcc, y = prop)) +
  geom_bar(stat = "identity") +
  geom_errorbar(aes(ymin = lower, ymax = upper))
```



6.1.1 Two Populations

Research is often interested in differences between distinct populations. Using the `glbcc` variable, we will review responses differentiated by gender, using the `group_by()` function.

```
ds %>%
  group_by(f.gender) %>%
  count(glbcc) %>%
  drop_na() %>%
  mutate(prop = n / sum(n))
```

```
## Warning: Factor `f.gender` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.gender` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.gender` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## # A tibble: 4 x 4
## # Groups:   f.gender [2]
##   f.gender glbcc     n  prop
##   <fct>    <int> <int> <dbl>
## 1 Women        0   606 0.399
## 2 Women        1   914 0.601
## 3 Men          0   485 0.473
```

```
## 4 Men          1   541 0.527
```

To visually examine which gender has, on average, higher belief that greenshouse gases cause climate change, plotting the two mean levels of belief for the genders is a good place to start. To create a visualization, the mean level of `glbcc` needs to be found for each gender. In this case, the proportion of each gender believing in climate change is the same as the "mean" amount of men or women believing in climate change. The following code returns a new data frame that is just like the previous one, but filters the data to include only responses indicate a belief that greenhouse gases cause climage change.

```
ds %>%
  group_by(f.gender) %>%
  count(glbcc) %>%
  drop_na() %>%
  mutate(prop = n / sum(n)) %>%
  filter(glbcc == 1)
```

```
## Warning: Factor `f.gender` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.gender` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.gender` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## # A tibble: 2 x 4
## # Groups:   f.gender [2]
##   f.gender glbcc     n  prop
##   <fct>    <int> <int> <dbl>
## 1 Women        1   914 0.601
## 2 Men          1   541 0.527
```

Similarly to earlier in the lab, confidence intervals can also be calculated by finding the standard error. In this case, we need to find the standard error for men and women separately. We will use the `filter()` function to simplify the calculations:

```
male <- filter(ds, gender==1)
female <- filter(ds, gender==0)
m.se <- describe(male$glbcc)$se
f.se <- describe(female$glbcc)$se
```

Now that we have the standard errors for men and women, we can return to the previous data frame we constucted and add a column for the standard error. This is done using the tidyverse verb `add_column()`. Assign the new data frame to a new object, then print the data frame:

```
df2 <- ds %>%
  group_by(f.gender) %>%
  count(glbcc) %>%
  drop_na() %>%
  mutate(prop = n / sum(n)) %>%
  filter(glbcc == 1) %>%
  add_column(se = c(m.se, f.se))
```

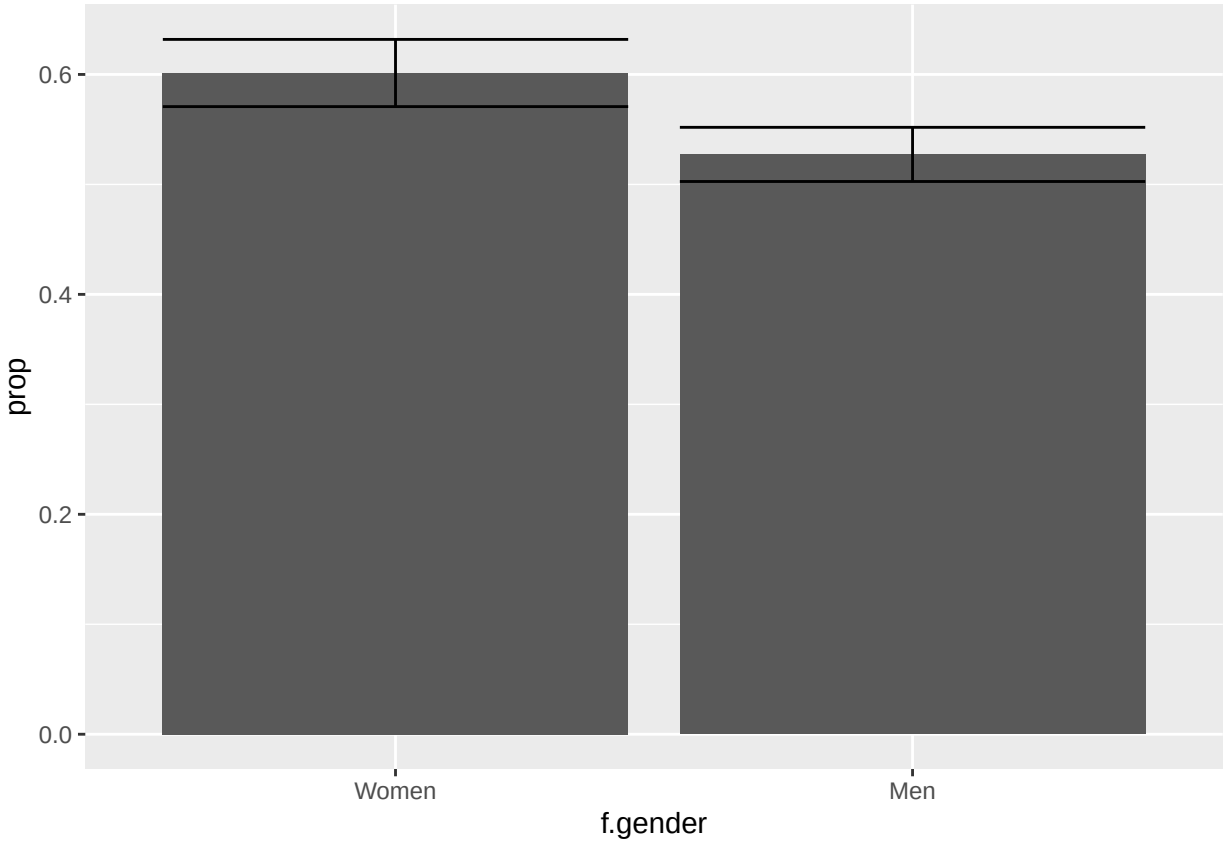
```
## Warning: Factor `f.gender` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.gender` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## Warning: Factor `f.gender` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

Now construct the visualization. Like the previous visualization, use `geom_errorbar()` to construct the confidence intervals. However, this time you will need to calculate them inside the `geom_errorbar()` function itself!

```
ggplot(df2, aes(f.gender, prop)) +
  geom_bar(stat = "identity") +
  geom_errorbar(aes(ymin = prop - 1.96 * se,
                    ymax = prop + 1.96 * se))
```



Suppose we wondered whether women believe humans cause climate change more than men: this visualization provides only a partial answer. By the "eye test," the visualization appears to show that women have a higher value than men, and furthermore the confidence intervals do not overlap; however, the eye test alone is insufficient. An empirical test is required.

To start, we formulate the following hypotheses:
H_0: there is no difference between genders
H_1: there is a difference between genders

We can use a two sample t-test to test these hypotheses. Using the different data sets created earlier for genders and the `glbcc` variable we will find the 95% confidence interval, p-value, and point estimate.

```
t.test(male$glbcc, female$glbcc)
```

```
##
##  Welch Two Sample t-test
##
```

```
## data:  male$glbcc and female$glbcc
## t = -3.6966, df = 2170.3, p-value = 0.0002239
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.11329550 -0.03475519
## sample estimates:
## mean of x mean of y
## 0.5272904 0.6013158
```

The t-test yields a p-value $< \alpha = 0.05$, thereby the null hypothesis is rejected to conclude there is a statistical significance in responses by gender. Further, the point estimate calculated as 0.074 informs us 7% more women than men believe humans cause climate change. **Note:** The confidence interval tells us that, with 95% confidence, the difference between women and men is between 3% and 11%. Judgment is required to determine whether gender difference is substantive.

6.2 Cross Tabulations

Another way to examine the difference of gender and beliefs about climate change is cross tabulation. Cross tabulations describe relationships between two variables. The basic building block of cross tabulations are tables, a skill acquired in previous labs.

For this section, we will use the `glbcc_risk` variable, that measures the level of risk respondents associate with climate change (on a scale of zero to ten). The range associated to this scale is simplified to zero to five using the `recode()` function:

```
ds$r.gccrsk <- car::recode(ds$glbcc_risk, "0:1=1; 2:3=2; 4:6=3; 7:8:=4; 9:10=5")
table(ds$r.gccrsk)
```

```
##
##   1   2   3   4   5
## 268 331 761 538 638
```

Next the variable is separated by gender using the `table()` function. The dependent variable, `r.gccrsk`, is specified followed by the independent variable, `f.gender`:

```
gcc.table <- table(ds$r.gccrsk, ds$f.gender)
gcc.table
```

```
##
##     Women Men
##   1   134 134
##   2   175 155
##   3   480 281
##   4   330 208
##   5   393 245
```

The `prop.table()` function describes the relationship by proportions. We convert the proportion to percentage of each response level by gender by including `margin=2`.

```
gcc.table %>% prop.table(margin = 2) * 100
```

```
##
##         Women       Men
##   1  8.862434 13.098729
##   2 11.574074 15.151515
##   3 31.746032 27.468231
##   4 21.825397 20.332356
```

```
##   5 25.992063 23.949169
```

There appears to be a difference in genders; however, as these differences are within a sample we cannot infer there is a difference in the population without an empirical test. We will use the Chi-Square test to empirically test whether there is a statistically significant difference in genders. The `chisq.test()` function performs the Chi-Square test given on a table:

```
chisq.test(gcc.table)
```

```
##
##  Pearson's Chi-squared test
##
## data:  gcc.table
## X-squared = 21.729, df = 4, p-value = 0.0002269
```

Note: The `summary()` function will provide additional information about this table, independent of a Chi-Square test.

```
summary(gcc.table)
```

```
## Number of cases in table: 2535
## Number of factors: 2
## Test for independence of all factors:
##  Chisq = 21.729, df = 4, p-value = 0.0002269
```

Given the Chi-Square test p-value $< \alpha = 0.05$, the null hypothesis is rejected such that there is a statistically significant difference between gender and perceived risk of climate change. Substantive difference is as important as statistical significance, and a variety of methods exist to test the strength of relationships. For the Chi-Square test, finding Cramer's Vs is the appropriate method. The `assocstats()` function will return a variety of coefficients and numbers, including Cramer's V.

```
assocstats(gcc.table)
```

```
##                     X^2 df   P(> X^2)
## Likelihood Ratio 21.494  4 0.00025270
## Pearson          21.729  4 0.00022695
##
## Phi-Coefficient   : NA
## Contingency Coeff.: 0.092
## Cramer's V        : 0.093
```

Cramer's V is a score ranging from 0 to 1, whereby 0 indicates a weak association and 1 indicates strong association. The Cramer's V score returned for the previous test is quite small, indicating a weak association.

Perhaps we are interested in gender as a control variable, with ideology as the independent variable. A new table for perceived risk of climate change, ideology (conservative, moderate, liberal), and gender is as follows:

```
gcc.table2 <- table(ds$r.gccrsk, ds$f.ideology, ds$f.gender)
gcc.table2
```

```
## , ,  = Women
##
##
##     Conservative Liberal Moderate
##   1          108       4       20
##   2          119       5       50
##   3          216      20      237
##   4           84      60      185
##   5           42     166      180
```

```
##
## , ,  = Men
##
##
##     Conservative Liberal Moderate
##   1          115       1       17
##   2          120       0       34
##   3          151      10      118
##   4           57      33      116
##   5           24      99      121
```

Again, a Chi-Square test and Cramer's V will provide insight. To separate the genders, [„1] is used for male and [„2] is used for female.

For male:

```
chisq.test(gcc.table2[,,1])
```

```
##
##  Pearson's Chi-squared test
##
## data:  gcc.table2[, , 1]
## X-squared = 485.92, df = 8, p-value < 0.00000000000000022
```

```
assocstats(gcc.table2[,,1])
```

```
##                     X^2 df P(> X^2)
## Likelihood Ratio 496.17  8        0
## Pearson          485.92  8        0
##
## Phi-Coefficient   : NA
## Contingency Coeff.: 0.495
## Cramer's V        : 0.403
```

For female:

```
chisq.test(gcc.table2[,,2])
```

```
##
##  Pearson's Chi-squared test
##
## data:  gcc.table2[, , 2]
## X-squared = 409.29, df = 8, p-value < 0.00000000000000022
```

```
assocstats(gcc.table2[,,2])
```

```
##                     X^2 df P(> X^2)
## Likelihood Ratio 440.55  8        0
## Pearson          409.29  8        0
##
## Phi-Coefficient   : NA
## Contingency Coeff.: 0.536
## Cramer's V        : 0.449
```

Both chi-square tests return significant results, and both Cramer's V scores are about .4. The interpretation of this Cramer's V score is clearly stronger than the previous score; however, the score itself is not judged as a strong association.

6.2.1 Other Coefficients

There are other coefficient scores that are appropriate to use in certain situations. The Phi coefficient is used with a 2x2 contingency table. The contingency coefficient, C, is used for square tables. Keep these different methods in mind when doing cross-tabulations and chi-square tests.

6.3 Independent t-tests

This section elaborates on t-tests. So far we have emphasized t-tests, as the Student's t distribution can be used when n < 30, and when σ is unknown. The rule of thumb is to use Student's t distribution in these two instances, and the normal distributions for all other cases; however, the Student's t distribution begins to approximate the normal distribution with high n sizes, so t-tests are applicable in most instances.

When using t-tests for two populations, you need to first decide if you should use an independent t-test or a paired t-test. An independent t-test is used when the two groups are independent from each other. A paired t-test is used for paired or connected groups. For the class data set the independent t-tests is appropriate. For an independent t-test, the variance between groups must be unequal.

Let's examine if there is a difference between the risk associated with climate change for Democrats and Republicans in our survey. In order to test only Democrats and Republicans, we need to recode our factored party variable to only include Democrats and Republicans:

```
ds$f.part <- car::recode(ds$f.party.2, "'Dem'='Dem'; 'Rep'='Rep'; else=NA")
ds %>%
  count(f.part) %>%
  drop_na()
```

```
## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## # A tibble: 2 x 2
##   f.part     n
##   <fct>  <int>
## 1 Dem      869
## 2 Rep     1185
```

We need to compare variances for Democrats and Republicans on their risk perception of climate change. First use `skim()` to take a look at some summary statistics by political party.

```
skim_with(numeric = list(hist = NULL))
skim_with(ingerger = list(hist = NULL))
```

```
## Adding type: ingerger
```

```
ds %>%
  group_by(f.part) %>%
  skim(glbcc_risk)
```

```
## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

```
## Skim summary statistics
##  n obs: 2547
##  n variables: 216
##  group variables: f.part
```

```
## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`

## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`

##
## -- Variable type:integer -------------------------------------------------------------------------

## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`

##  f.part   variable missing complete    n mean   sd p0 p25 p50 p75 p100
##     Dem glbcc_risk       4      865  869 7.71 2.43  0   6   8  10   10
##     Rep glbcc_risk       5     1180 1185 4.53 2.77  0   2   5   7   10
##    <NA> glbcc_risk       2      491  493 6.25 3.07  0   5   7   9   10

## Warning: Factor `f.part` contains implicit NA, consider using
## `forcats::fct_explicit_na`
```

The Levene test is used to test if the difference in variances is statistically significant. Similar to previously discussed hypotheses, the Levene tests the following hypotheses:

-H_0: variances are equal
-H_1: variances are not equal

```
leveneTest(ds$glbcc_risk, ds$f.part)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##         Df F value     Pr(>F)
## group    1  18.852 0.00001481 ***
##       2043
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Given the p-value $< \alpha = 0.05$, we reject the null hypothesis such that there is a statistically significant difference in variances. With this result we can perform an independent t-test for two populations. The framework for forming hypotheses is as follows:

-H_0: the two populations are indifferent; "there is no difference in perceived risk of climate change between Democrats and Republicans" -H_1: the two populations are different; "there is a difference in perceived risk of climate change between Democrats and Republicans"

```
t.test(ds$glbcc_risk ~ ds$f.part, var.equal = FALSE) # var.equal=FALSE is default
```

```
##
##  Welch Two Sample t-test
##
## data:  ds$glbcc_risk by ds$f.part
## t = 27.556, df = 1977.2, p-value < 0.00000000000000022
```

```
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  2.957765 3.411041
## sample estimates:
## mean in group Dem mean in group Rep
##          7.709827          4.525424
```

Given the p-value $< \alpha = 0.05$, we reject the null hypothesis such that there is a statistically significant difference in perceived risk of climate change between democrats and republicans.

The previous test is an example of a two-tailed test, which lacks directionality (greater than, less than). In contrast, a one-tailed t-test tests hypotheses that include direction. To perform a one-tailed t-test, we include the command alt="greater" inside the `t.test()` function:

```
t.test(ds$glbcc_risk ~ ds$f.part, alt="greater")
```

```
##
##  Welch Two Sample t-test
##
## data:  ds$glbcc_risk by ds$f.part
## t = 27.556, df = 1977.2, p-value < 0.00000000000000022
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  2.994229      Inf
## sample estimates:
## mean in group Dem mean in group Rep
##          7.709827          4.525424
```

6.3.1 Other Independent Sample Tests

Not all research leads to testing two populations. For example, consider whether a difference exists in perceived risk of climate change among Democrats, Republicans, and independents. Further, if there are statistically significant differences, how could ideologies be ranked? Unfortunately, performing pairwise t-tests on the possible combinations is inadequate, due to the "family-wise error rate" associated to the p-value. The `pairwise.t.test()` function employs the family-wise correction method (by default, Holm-Bonferroni method) to correct the p-value.

Like traditional independent sample tests involving two samples, the variances must be unequal:

```
leveneTest(ds$glbcc_risk ~ ds$f.party.2)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##         Df F value      Pr(>F)
## group    2  13.683 0.000001234 ***
##       2396
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Given that the p-value $< \alpha = 0.05$, we reject the null hypothesis such that there is a difference in variances. The `pairwise.t.test()` function can perform the pairwise test, using the `f.party_2` variable that was factored for Democrats, Republicans, and independents:

```
pairwise.t.test(ds$glbcc_risk, ds$f.party.2, pool.sd=FALSE)
```

```
##
##  Pairwise comparisons using t tests with non-pooled SD
##
```

```
## data:  ds$glbcc_risk and ds$f.party.2
##
##     Dem                  Ind
## Ind 0.0000000000013      -
## Rep < 0.0000000000000002 < 0.0000000000000002
##
## P value adjustment method: holm
```

The output of the `pairwise.t.test()` function is read as a table, whereby each value is associated to the pairwise comparison.

An alternative, multiple comparisons test, is the Tukey Honest Significance Difference test ("Tukey's test"), that is commonly used with Analysis of Variance (ANOVA). In short, Tukey's test is used to find means that are significantly difference between multiple samples. Tukey's test requires using the `aov()` function to perform ANOVA, as follows:

```
aov(ds$glbcc_risk ~ ds$f.party.2) %>% TukeyHSD()
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = ds$glbcc_risk ~ ds$f.party.2)
##
## $`ds$f.party.2`
##              diff       lwr        upr p adj
## Ind-Dem -1.294572 -1.691945 -0.8971998     0
## Rep-Dem -3.184403 -3.466308 -2.9024976     0
## Rep-Ind -1.889831 -2.271490 -1.5081713     0
```

The p-value $< \alpha = 0.05$ between each pair.

6.4 Paired t-test

Paired t-tests are appropriate for data that are not independent. Consider the need to compare data from the same population for different points in time, such as semester exams for the same students in a class.

The teacher wants to examine if there is a difference in the students' performances between exams one and two. The following hypothetical data frame could represent the students and their performance:

```
Student <- c("st1", "st2", "st3", "st4", "st5", "st6", "st7", "st8", "st9", "st10")
Exam1 <- c(99, 98, 67, 68, 70, 71, 72, 88, 75, 83)
Exam2 <- c(94, 93, 62, 63, 65, 66, 67, 83, 70, 76)
exam.ds <- data.frame(Student, Exam1, Exam2)
exam.ds
```

```
##    Student Exam1 Exam2
## 1      st1    99    94
## 2      st2    98    93
## 3      st3    67    62
## 4      st4    68    63
## 5      st5    70    65
## 6      st6    71    66
## 7      st7    72    67
## 8      st8    88    83
## 9      st9    75    70
## 10    st10    83    76
```

The `exam.ds` data frame consists of three vectors: the student, exam one, and exam two.

First check the variance between the two groups:

```
var.test(exam.ds$Exam1, exam.ds$Exam2)
```

```
##
##  F test to compare two variances
##
## data:  exam.ds$Exam1 and exam.ds$Exam2
## F = 1.0091, num df = 9, denom df = 9, p-value = 0.9895
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.2506424 4.0625692
## sample estimates:
## ratio of variances
##           1.009085
```

Given the p-value $> \alpha = 0.05$, the null hypothesis is not rejected such that there is no statistically significant difference in variances.

Next, we define our hypotheses for the paired test:
-H_0: there is no difference between exams one and two for students
-H_1: there is a difference between exams one and two for students

```
t.test(exam.ds$Exam1, exam.ds$Exam2, paired = TRUE, var.equal = TRUE)
```

```
##
##  Paired t-test
##
## data:  exam.ds$Exam1 and exam.ds$Exam2
## t = 26, df = 9, p-value = 0.0000000008884
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  4.747569 5.652431
## sample estimates:
## mean of the differences
##                     5.2
```

Given the p-value $< \alpha = 0.05$, the null hypothesis is rejected such that there is a statistically significant difference in means. Next, to determine directionality, the `alt="greater"` argument is included in the `t.test()` function. Additionally, the hypotheses are modified as follows: H_0: there is no difference between exams one and two for students
H_1: exams one performance is greater than exam two performance for students

```
t.test(exam.ds$Exam1, exam.ds$Exam2, paired = TRUE, var.equal = TRUE, alt = "greater")
```

```
##
##  Paired t-test
##
## data:  exam.ds$Exam1 and exam.ds$Exam2
## t = 26, df = 9, p-value = 0.0000000004442
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  4.833377      Inf
## sample estimates:
## mean of the differences
##                     5.2
```

Given the p-value $< \alpha = 0.05$, the null hypothesis is rejected.

6.5 Visualizing Differences in Means

Visualization of data is essential to interpreting and communicating difference in means. To demonstrate, we will use `ggplot2` to visualize the difference in exam scores used with the paired t-tests.

First, we consider the difference of means with confidence intervals. We start by calculating the means for each exam:

```
exam1 <- mean(exam.ds$Exam1, na.rm = T)
exam2 <- mean(exam.ds$Exam2, na.rm = T)
```

Then, the standard errors:

```
exam1.se <- describe(exam.ds$Exam1)$se
exam2.se <- describe(exam.ds$Exam2)$se
```

Lastly, the bounds for the confidence intervals. The 1.96 z-score is used for 95% confidence:

```
exam1up <- exam1 + exam1.se*1.96
exam1low <- exam1 - exam1.se*1.96
exam2up <- exam2 + exam2.se*1.96
exam2low <- exam2 - exam2.se*1.96
```

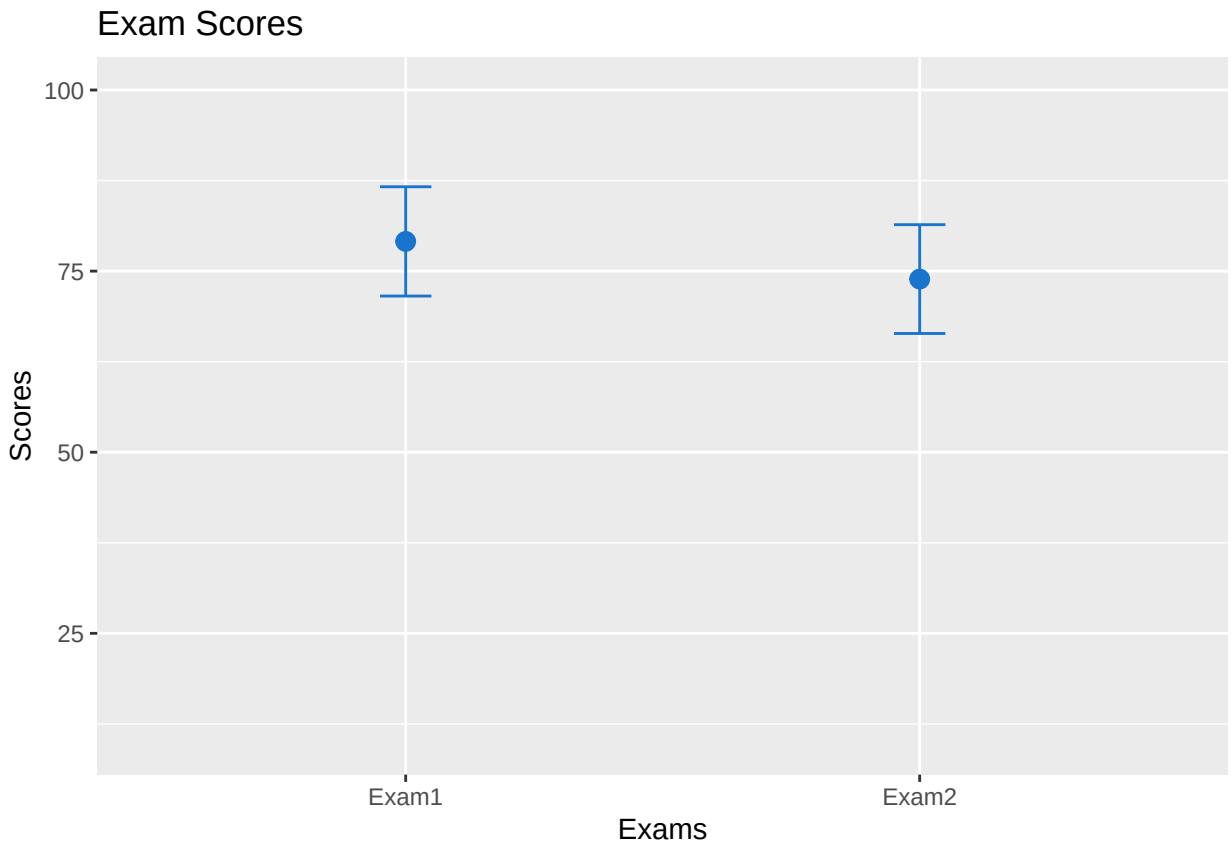
Next, a data frame is constructed with the previous information in respective vectors:

```
test <- c("Exam1", "Exam2")
scores <- c(exam1, exam2)
upper <- c(exam1up, exam2up)
lower <- c(exam1low, exam2low)
examdf <- data.frame(test, scores, upper, lower)
examdf
```

```
##    test scores    upper    lower
## 1 Exam1   79.1 86.64282 71.55718
## 2 Exam2   73.9 81.40879 66.39121
```

Now, for the visualization. Instead of a bar plot, we will create a graph consisting of a point for the means for each exam, and the confidence intervals. In additional, we will add color and labels. These arguments are available for other data sets and visualizations using `ggplot2`:

```
ggplot(examdf, aes(x = test, y = scores)) +
  geom_point(size = 3, shape = 19, col = "dodgerblue3") +
  geom_errorbar(aes(ymin = lower, ymax = upper), width = .1, col = "dodgerblue3") +
  ylim(10, 100) +
  ggtitle("Exam Scores") +
  xlab("Exams") +
  ylab("Scores")
```



Following the basic steps above will help you make attractive visualizations for different needs.

Suppose the teacher wanted a visualization that plots each students' test scores between the two exams for easy comparison. A grouped bar plot is a simple approach to accomplish this.

This requires a different type of data frame. So far the data has been "wide" data, but we need "long" data. The `melt()` function found in the `dplyr` package converts data from wide to long format. Include "Exam1" and "Exam2" as the measured variables.

```
exam.m <- melt(exam.ds, measure.vars = c("Exam1", "Exam2"))
```

The new data frame pairs the exams with the students and the respective score. This is what is meant by "long" data.

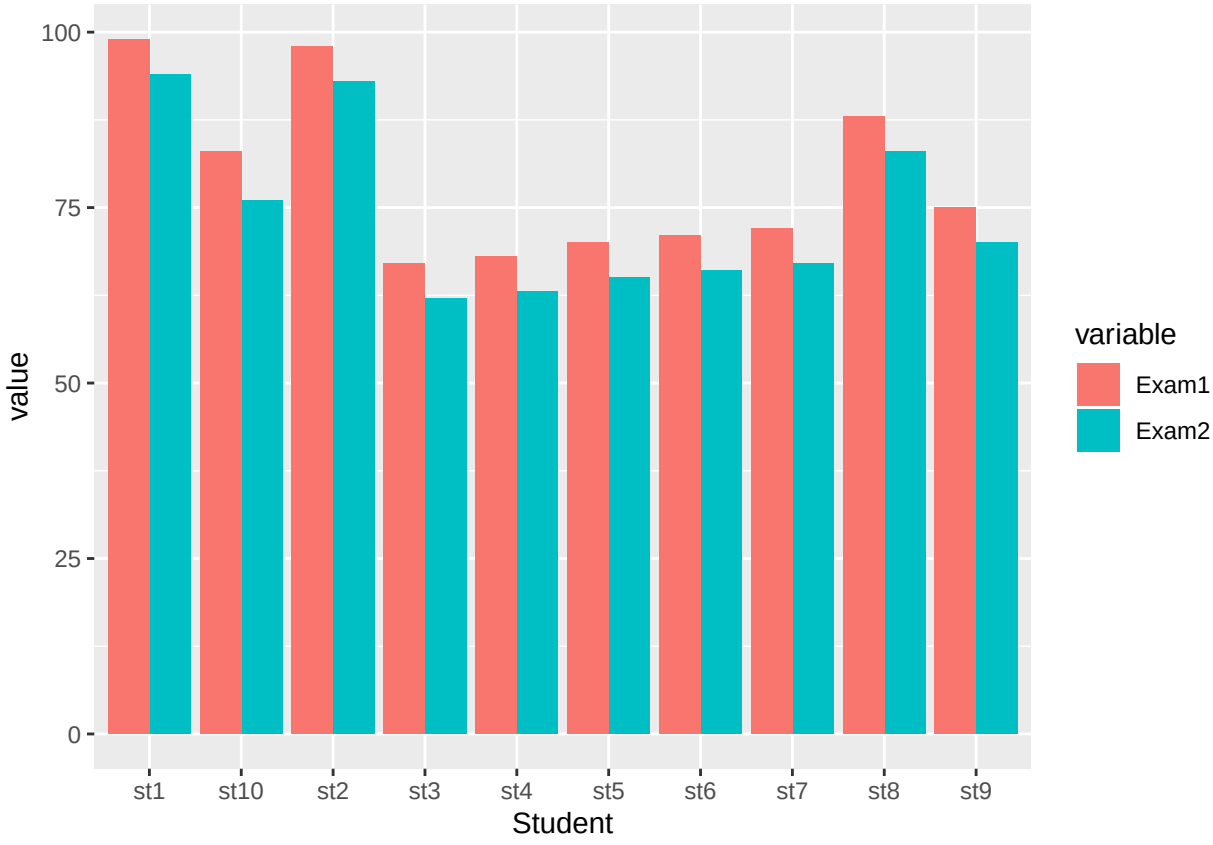
```
exam.m
```

```
##    Student variable value
## 1      st1    Exam1    99
## 2      st2    Exam1    98
## 3      st3    Exam1    67
## 4      st4    Exam1    68
## 5      st5    Exam1    70
## 6      st6    Exam1    71
## 7      st7    Exam1    72
## 8      st8    Exam1    88
## 9      st9    Exam1    75
## 10    st10    Exam1    83
## 11     st1    Exam2    94
## 12     st2    Exam2    93
## 13     st3    Exam2    62
```

```
## 14     st4    Exam2    63
## 15     st5    Exam2    65
## 16     st6    Exam2    66
## 17     st7    Exam2    67
## 18     st8    Exam2    83
## 19     st9    Exam2    70
## 20    st10    Exam2    76
```

The grouped bar plot is constructed similar to a bar plot, except that the visualization is based on the exam and uses the `position.dodge()` function to place bars near each other and the `fill=Exam` argument to specify what the groups

```
ggplot(exam.m, aes(x=Student, y=value, fill=variable)) +
  geom_bar(stat="identity", position = position_dodge())
```



7 Covariance and Correlation

Research is not always interested in the contrast of two variables, but oftentimes the relationship of two variables. For instance, what is the relationship between climate science and ideology? The following packages are required for this lab:

1. tidyverse
2. psych
3. car
4. vcd

7.1 Covariance

Covariance is the measure of change in one variable associated to change in another variable. That is, the measure of how two random variables vary together.

Calculating covariance of two variables of a known population is trivial, as the product of variation of two variables. However, for samples, covariance is calculated via the following formula:

$$cov(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$

Where `x` and `y` are two random variables, `i` is each observation (row), and `n` is the sample size.

7.1.1 Covariance by Hand

To find covariance by hand, let's construct a hypothetical data set with variables x and y. We only give each variable three values so that the calculation will be shorter.

```
x <- c(25, 27, 29)
y <- c(5, 15, 9)
```

First we calculate the difference of each value and the mean for the variables:

```
xdev <- x - mean(x)
ydev <- y - mean(y)
```

Next we find the product of the above differences:

```
xdev_ydev <- xdev * ydev
xdev_ydev
```

```
## [1]  9.333333  0.000000 -1.333333
```

We complete the numerator by finding the sum of the products:

```
sum_xdev_ydev <- sum(xdev_ydev)
```

Lastly, we complete the covariance calculation by dividing the numerator by the denominator. The denominator is calculated as one less than the sample size:

```
cov_xy <- sum_xdev_ydev / (3 - 1)
cov_xy
```

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

7.1.2 Covariance in R

By using the `cov()` function, R calculates the covariance. We demonstrate the `cov()` function by confirming the previous section's calculations:

```
cov(x, y)
```

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

7.1.3 Covariance in Class Data Set

To demonstrate further we will calculate covariance for various pairs of variables within the class data set. First, suppose we are interested in the relationship between certainty that humans cause climate change (`glbcc_cert`) and the perceived risk of climate change (`glbcc_risk`). That is, is there a relationship between respondents' certainty that humans cause climate change and their perceived risk of climate change? Our hypothesis could be that individuals that are more certain that climate change is a consequence of humans are likely more concerned about the associated risk.

```
cov(ds$glbcc_cert, ds$glbcc_risk, use = "complete.obs")
```

```
## [1] 3.092168
```

Note: The `cov()` function requires `use="complete.obs"` to remove NA entries.

The calculated covariance of certainty and risk perception is positive, indicating the two variables are positively related. As one variable changes, the other variable will change in the same direction with a magnitude of 3.09. Increased certainty that humans cause climate change increases the perceived risk of climate change.

Suppose we are also interested in the relationship of income and perceived risk of climate change:

```
cov(ds$income, ds$glbcc_risk, use = "complete.obs")
```

```
## [1] -10826.91
```

The calculated covariance of income and perceived risk of climate change is negative, indicating the two variables are negatively related. As one variable changes, the other variable will change in the opposite direction with a magnitude of -10826.91.

An important follow-up question is: which variable is more strongly associated to perceived risk, certainty or income? We cannot compare magnitudes to the difference in scales and units. Income is measured on a scale inclusive of higher numbers compared to certainty. To compare strengths of association we need to standardize covariance.

7.2 Correlation

Correlation standardizes covariance on a scale of negative one to one, whereby the magnitude from zero indicates strength of relationship. Similar to covariance, a positive and negative value reflects the respective relationship. The formula for correlation is the following:

$$r_{xy} = \frac{cov(x, y)}{s_x s_y}$$

Where x and y are two random variables.

7.2.1 Correlation by Hand

To find covariance by hand, let's use the data set we constructed earlier to calculate covariance.

Recall we calculated covariance manually via the following steps:

```
x <- c(25, 27, 29)
y <- c(5, 15, 9)
xdev <- x - mean(x)
ydev <- y - mean(y)
xdev_ydev <- xdev * ydev
sum_xdev_ydev <- sum(xdev_ydev)
```

```
cov_xy <- (1 / (3 - 1)) * sum_xdev_ydev
cov_xy
```

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

For calculating correlation we need the product of the standard deviations of variables `x` and `y` for the denominator:

```
stnd.dev <- sd(x)*sd(y)
```

Now we find the quotient of the covariance numerator and standard deviations denominator:

```
cov_xy/stnd.dev
```

```
## [1] 0.3973597
```

The relationship is positive; however, the correlation coefficient is ≈ 0.40.

Returning to the class data set, we can find the correlation coefficient for certainty humans cause climate change and perceived risk of climate change from the class data set:

```
numerator <- cov(ds$glbcc_cert, ds$glbcc_risk, use = "complete.obs")
denominator <- sd(ds$glbcc_cert, na.rm = T) * sd(ds$glbcc_risk, na.rm = T)
numerator / denominator
```

```
## [1] 0.3699554
```

The correlation coefficient is ≈ 0.37.

Now let's find the correlation coefficient for ideology and perceived risk from climate change.

First we will create a subset from the class data set `ds` for the variables of interest, absent of missing observations. **Note:** This subset includes additional variables, `f.gender` and `f.party.2` for use later in this lab.

```
ds.sub <- ds %>%
  dplyr::select("glbcc_risk", "ideol", "f.gender", "f.party.2") %>%
  na.omit()
```

The perceived risk and ideology variables are assigned to `x` and `y` variables within the `ds.sub` object, as to follow the formula.

```
ds.sub$x <- ds.sub$glbcc_risk
ds.sub$y <- ds.sub$ideol
```

The `x` and `y` variables are then used to find the covariance, similar to the steps demonstrated earlier:

```
xbar <- mean(ds.sub$x)
ybar <- mean(ds.sub$y)

x.m.xbar <- ds.sub$x - xbar
y.m.ybar <- ds.sub$y - ybar

n <- length(ds.sub$x)
n
```

```
## [1] 2388
```

```
cov.xy <- (sum(x.m.xbar * y.m.ybar)) / (n - 1)
cov.xy
```

```
## [1] -3.134752
```

Next, we find the correlation coefficient using the covariance as the numerator and the product of both variable standard deviations as the denominator:

```
sd.x <- sqrt((sum(x.m.xbar^2)) / (n - 1))
sd.x
```

```
## [1] 3.058544
```

```
sd.y <- sqrt((sum(y.m.ybar^2)) / (n - 1))
sd.y
```

```
## [1] 1.74181
```

```
sd.xy <- sd.x*sd.y

cov.xy/sd.xy
```

```
## [1] -0.5884202
```

The correlation coefficient for the variables is $\approx$ -0.59. The manual calculation is confirmed using the `cor()` function in R:

```
cor(ds.sub$glbcc_risk, ds.sub$ideol)
```

```
## [1] -0.5884202
```

Note: To calculate the correlation coefficient manually in one line of code:

```
sum((x-mean(x))*(y-mean(y))) /(sqrt(sum((x-mean(x))^2))*sqrt(sum((y-mean(y))^2)))
```

7.2.2 Correlation Tests

The previous section demonstrated the `cor()` function to confirm the manual calculation of the correlation coefficient. Using this function we can find the correlation coefficient of the income and perceived risk variables:

```
cor(ds$income, ds$glbcc_risk, use = "complete.obs")
```

```
## [1] -0.05904785
```

The correlation coefficient of -0.06 informs us of two things: the relationship is negative and very weak. **Note:** The correlation coefficient is drawn from observations within a sample, and therefore is a random value. That is, if we were to collect multiple samples we would calculate different correlation coefficients for each sample collected. This leads to a new hypothesis: is there a relationship between income and perceived risk? To test this we employ Pearson's product-moment correlation available via the `cor.test()` function. Our testing hypotheses are:

-H_0: the true correlation coefficient is zero
-H_1: the true correlation coefficient is not zero

```
cor.test(ds$income, ds$glbcc_risk, use="complete.obs")
```

```
##
##  Pearson's product-moment correlation
##
## data:  ds$income and ds$glbcc_risk
## t = -2.83, df = 2289, p-value = 0.004695
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.09975880 -0.01813952
```

```
## sample estimates:
##         cor
## -0.05904785
```

The correlation test yields a p-value $< \alpha = 0.05$, thereby the null hypothesis is rejected such that the true correlation coefficient is not zero.

Note: Despite rejecting the null hypothesis, our random correlation value is still quite small at -0.06. This indicates a very weak, or non-substance, relationship between income and perceived risk.

To demonstrate a potentially substantive relationship we look at ideology and perceived risk of climate change using the `cor.test()` function:

```
cor.test(ds$ideol, ds$glbcc_risk)
```

```
##
##  Pearson's product-moment correlation
##
## data:  ds$ideol and ds$glbcc_risk
## t = -36.633, df = 2511, p-value < 0.00000000000000022
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.6150780 -0.5640865
## sample estimates:
##        cor
## -0.5901706
```

The default correlation test method for the `cor.test()` function is the Pearson test. The Spearman test is required for ordinal data via the `method="spearman"` argument within the `cor.test()` function. For calculation correlation with ordinal data.

7.2.3 Correlation Across Groups

Suppose you are interested in correlations across multiple variables. The `cor()` function examines the correlation between each variable pair for an entire data set. As such, if you are interested in only the correlation for select variables then use the `select()` function and select your variablesof interest, then pipe them into the `cor()` function. Include `drop_na()`. To demonstrate using variables from the class data set:

```
ds %>%
  dplyr::select(glbcc_risk, ideol, income, age) %>%
  drop_na() %>%
  cor()
```

```
##             glbcc_risk       ideol      income         age
## glbcc_risk  1.00000000 -0.59999737 -0.06079785 -0.06799270
## ideol      -0.59999737  1.00000000  0.03901982  0.08608048
## income     -0.06079785  0.03901982  1.00000000 -0.11761722
## age        -0.06799270  0.08608048 -0.11761722  1.00000000
```

This resulting matrix provides the correlation coefficients for two variables at a time. The correlation coefficients are defined by the intercept of the row and columns corresponding to each variable.

Additionally, recalling that each correlation coefficient itself is a random value, a test is required for inference. The `corr.test()` provided by the `psych` package is an alternative to the `cor.test()` function previously used. The `corr.test()` function supports examining variable pairs simultaneously within a given data set. Use the `print()` function with the `short=FALSE` argument to view the complete test with confidence intervals and p-values.

```
ds %>%
  dplyr::select(glbcc_risk, ideol, income, age) %>%
  drop_na() %>%
  corr.test %>%
  print(short = FALSE)
```

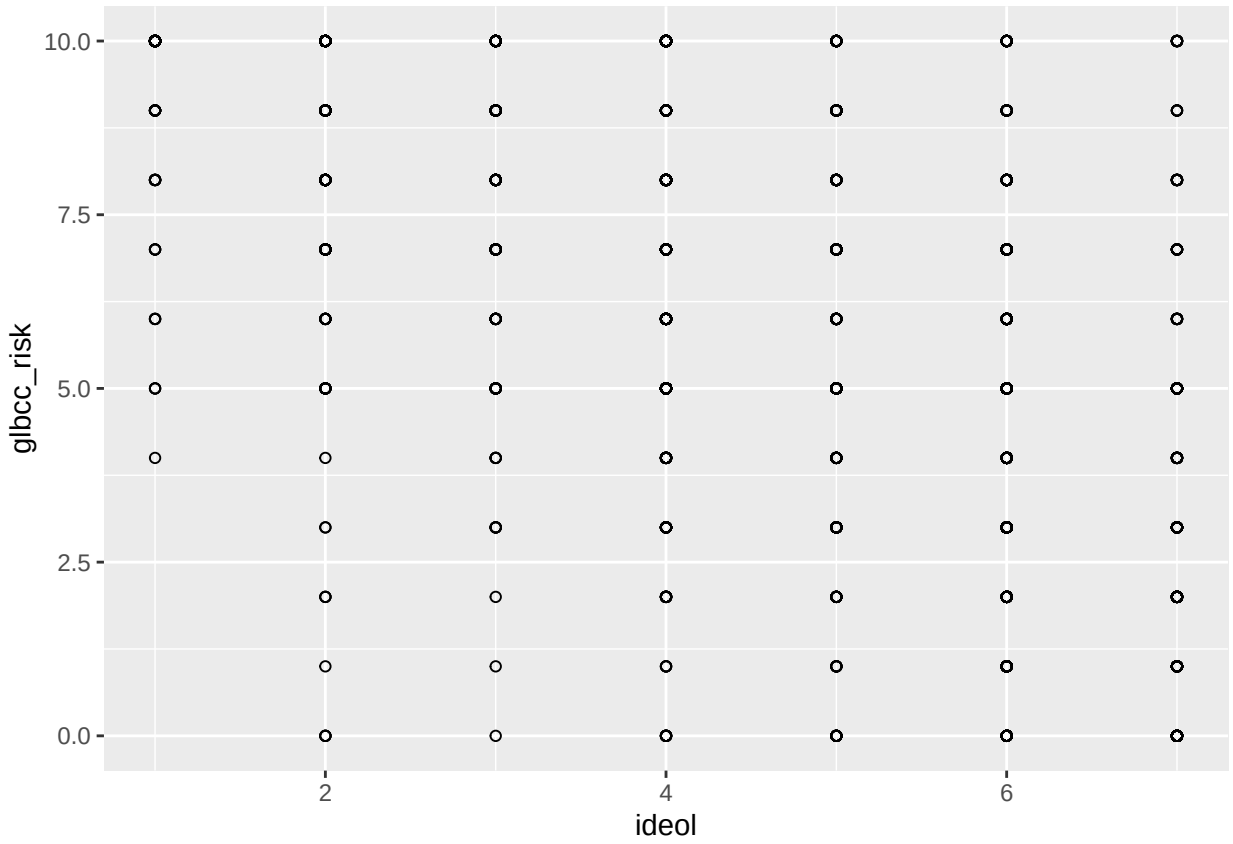
```
## Call:corr.test(x = .)
## Correlation matrix
##            glbcc_risk ideol income   age
## glbcc_risk       1.00 -0.60  -0.06 -0.07
## ideol           -0.60  1.00   0.04  0.09
## income          -0.06  0.04   1.00 -0.12
## age             -0.07  0.09  -0.12  1.00
## Sample Size
## [1] 2275
## Probability values (Entries above the diagonal are adjusted for multiple tests.)
##            glbcc_risk ideol income age
## glbcc_risk          0  0.00   0.01   0
## ideol               0  0.00   0.06   0
## income              0  0.06   0.00   0
## age                 0  0.00   0.00   0
##
##  Confidence intervals based upon normal theory.  To get bootstrapped values, try cor.ci
##             raw.lower raw.r raw.upper raw.p lower.adj upper.adj
## glbc_-ideol     -0.63 -0.60     -0.57  0.00     -0.63     -0.56
## glbc_-incom     -0.10 -0.06     -0.02  0.00     -0.11     -0.01
## glbc_-age       -0.11 -0.07     -0.03  0.00     -0.12     -0.02
## ideol-incom      0.00  0.04      0.08  0.06      0.00      0.08
## ideol-age        0.05  0.09      0.13  0.00      0.03      0.14
## incom-age       -0.16 -0.12     -0.08  0.00     -0.17     -0.06
```

7.3 Visualizing Correlation

The simplest form to visualize correlation is a scatter plot with a trend line.

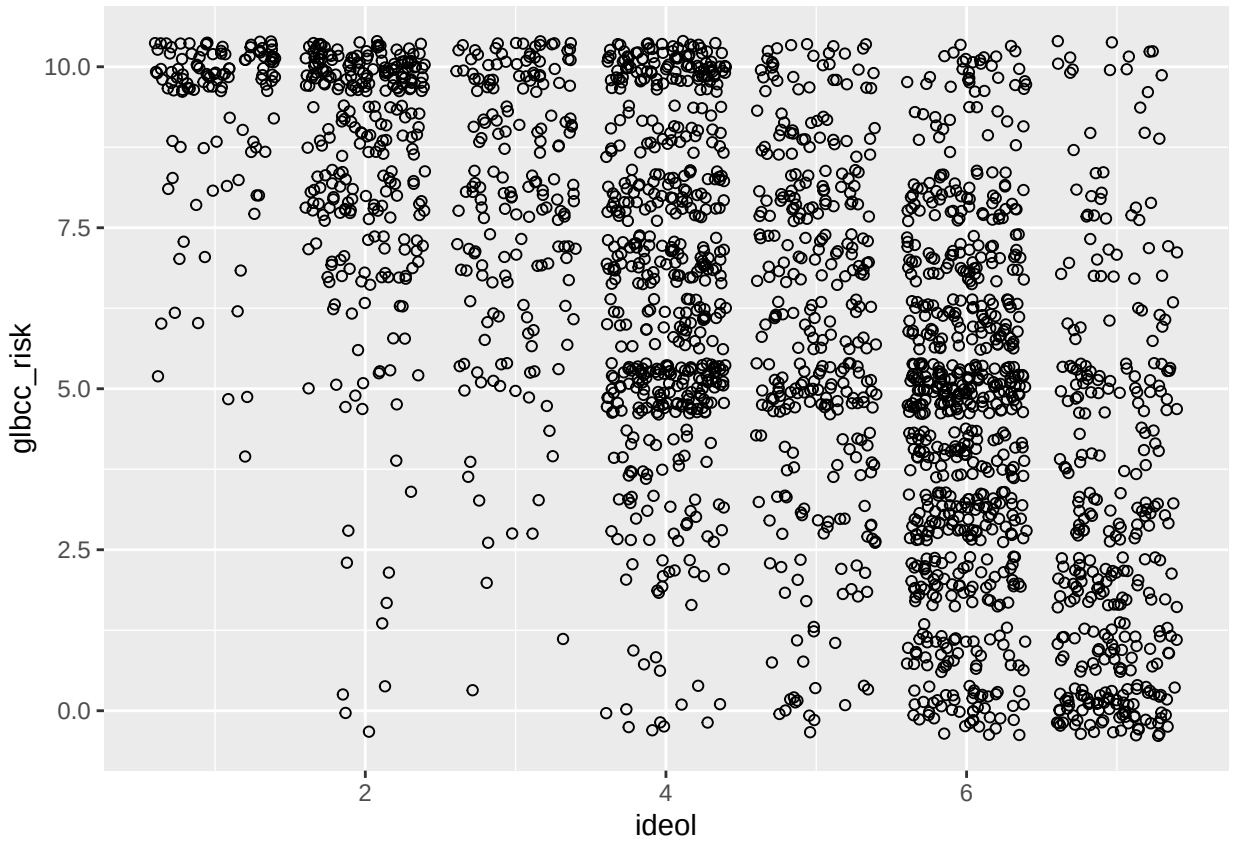
We will review the relationship between ideology and perceived risk from climate change. Build the basic visualization by using `ggplot()` and the `geom_point` functions, with ideology on the x axis and perceived risk about climate change on the y axis. Use the `ds.sub` dataset, because we removed the missing values.

```
ggplot(ds.sub, aes(x = ideol, y = glbcc_risk)) +
  geom_point(shape = 1)
```



Notice how this doesn't really make sense? This is because there are thousands of observations being placed on a discrete set of values. To get a better idea of the relationship, we can tell R to jitter the points. "Jittering" provides a tiny bit of white noise and variance to the values, so that we can see where there is high overlap of observations. This provides a better picture of the relationship. This time use the `geom_jitter` function instead of the `geom_point` function:

```
ggplot(ds.sub, aes(x = ideol, y = glbcc_risk)) +
  geom_jitter(shape = 1)
```



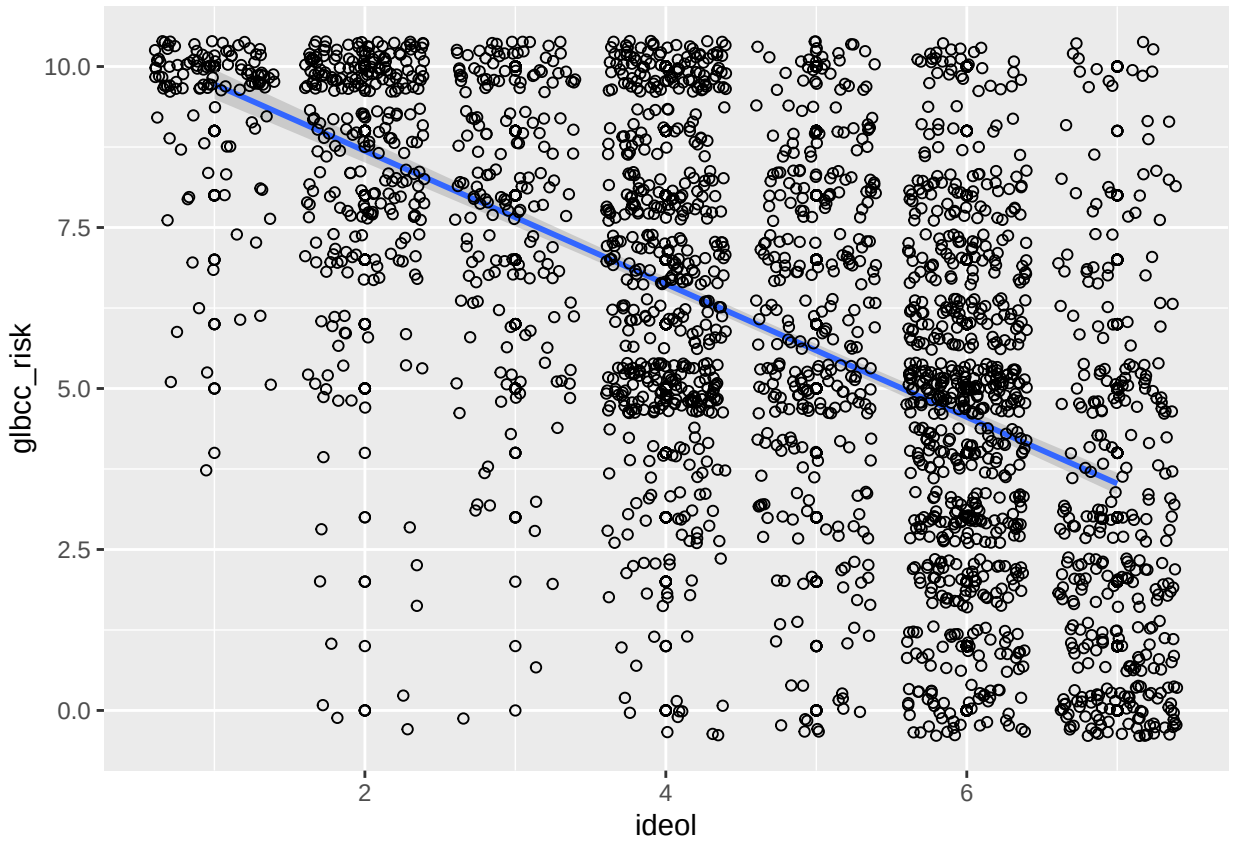
Notice the apparent negative correlation. Verify this by checking the correlation:

```
ds %>%
  dplyr::select(ideol, glbcc_risk) %>%
  cor(use = "complete.obs")
```

```
##                  ideol glbcc_risk
## ideol       1.0000000 -0.5901706
## glbcc_risk -0.5901706  1.0000000
```

A trend line to the scatter plot helps to interpret directionality of a given variable pair. The next lab will further introduce trend lines, but for now we introduce it via the `geom_smooth()` function by including the `method=lm` argument. We also need to include `geom_point()` for each point.

```
ggplot(ds.sub, aes(x = ideol, y = glbcc_risk)) +
  geom_point(shape = 1) +
  geom_smooth(method = lm) +
  geom_jitter(shape = 1)
```



The ideology points can be differentiated by other variables, such as gender, to examine potential difference among gender by defining the `color` argument as follows:

```
ggplot(ds.sub, aes(x = ideol, y = glbcc_risk, color = f.gender)) +
  geom_point(shape = 1) +
  geom_smooth(method = lm) +
  geom_jitter(shape = 1)
```

7.3.1 Another Example: Political Party

Let's look at this relationship broken down by political party. Perhaps you wanted to see if the relationship looks different for Republicans and Democrats. We can first subset the data for Republicans and Democrats:

```
ds.dem <- filter(ds.sub, f.party.2 == "Dem")
ds.rep <- filter(ds.sub, f.party.2 == "Rep")
```

Next we investigate the correlation between ideology and perceived risk of climate change for Republicans and Democrats separately:

```
cor.test(ds.rep$ideol, ds.rep$glbcc_risk)
```

```
##
##  Pearson's product-moment correlation
##
## data:  ds.rep$ideol and ds.rep$glbcc_risk
## t = -14.352, df = 1171, p-value < 0.00000000000000022
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.4343892 -0.3369909
## sample estimates:
##        cor
## -0.3867682
```

```
cor.test(ds.dem$ideol, ds.dem$glbcc_risk)
```

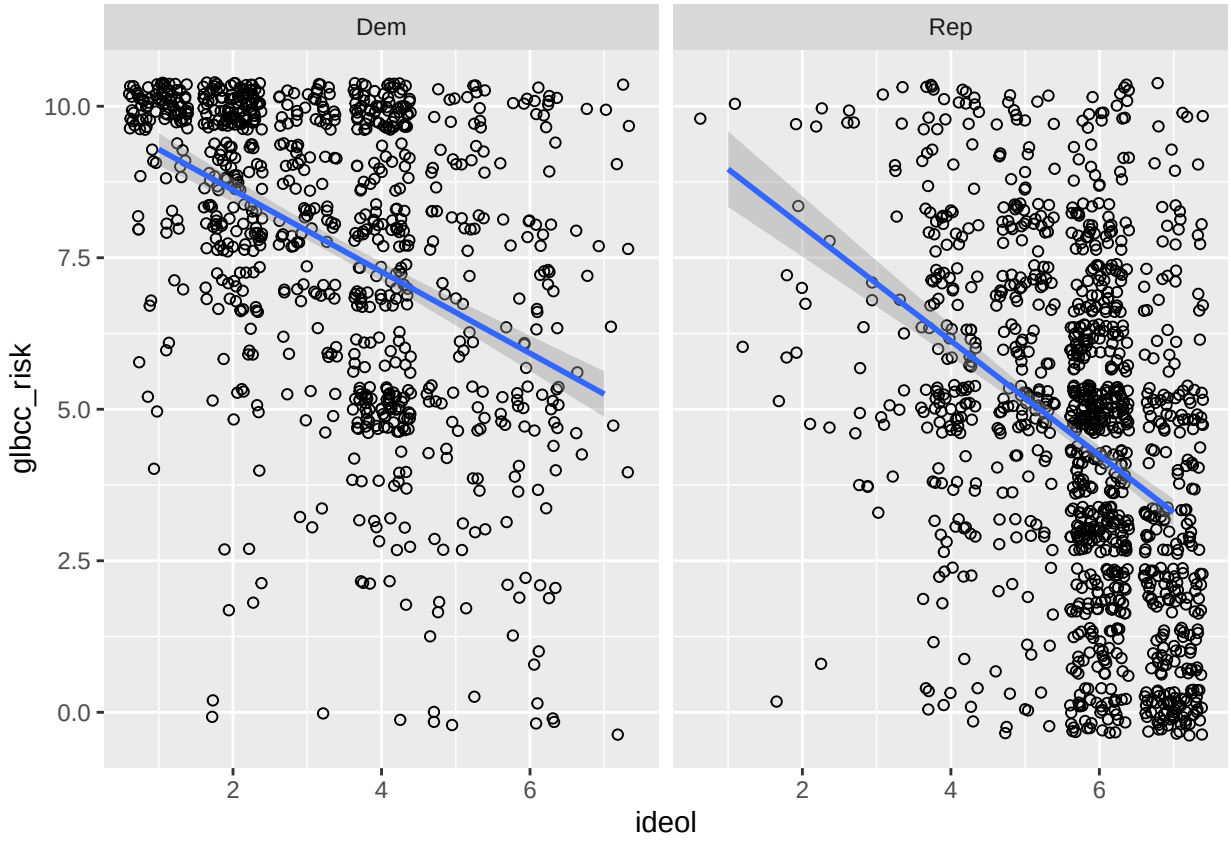
```
##
```

```
##  Pearson's product-moment correlation
##
## data:  ds.dem$ideol and ds.dem$glbcc_risk
## t = -13.977, df = 859, p-value < 0.00000000000000022
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.4833657 -0.3744104
## sample estimates:
##       cor
## -0.4304548
```

The correlation coefficient is slightly more negative for Democrats.

To create a visualization that compares the two parties and the relationship between climate change risk and ideology, a few simple lines of code can get the job done. First filter the data to include Democrats and Republicans, select the variables of interest, drop NAs, pipe it all into `ggplot2`, then use `facet_wrap()` to create two visualizations, one for each party:

```
ds %>%
  filter(f.party.2 == "Dem" | f.party.2 == "Rep") %>%
  dplyr::select(ideol, glbcc_risk, f.party.2) %>%
  na.omit() %>%
  ggplot(., aes(ideol, glbcc_risk)) +
  geom_jitter(shape = 1) +
  geom_smooth(method = lm) +
  facet_wrap(~ f.party.2, scales = "fixed")
```



7.3.2 One More Visualization

We cap this lab off with creating one last visualization. First create a new subset of our data exclusive of all missing observations. Include variables for climate change risk and age.

```
sub.ds <- filter(ds) %>%
  dplyr::select("glbcc_risk", "age") %>%
  na.omit()
```

First we look at our age variable.

```
describe(sub.ds$age)
```

```
##    vars    n  mean   sd median trimmed   mad min max range  skew kurtosis
## X1    1 2536 60.37 14.2     62   61.02 13.34  18  99    81 -0.39    -0.24
##      se
## X1 0.28
```

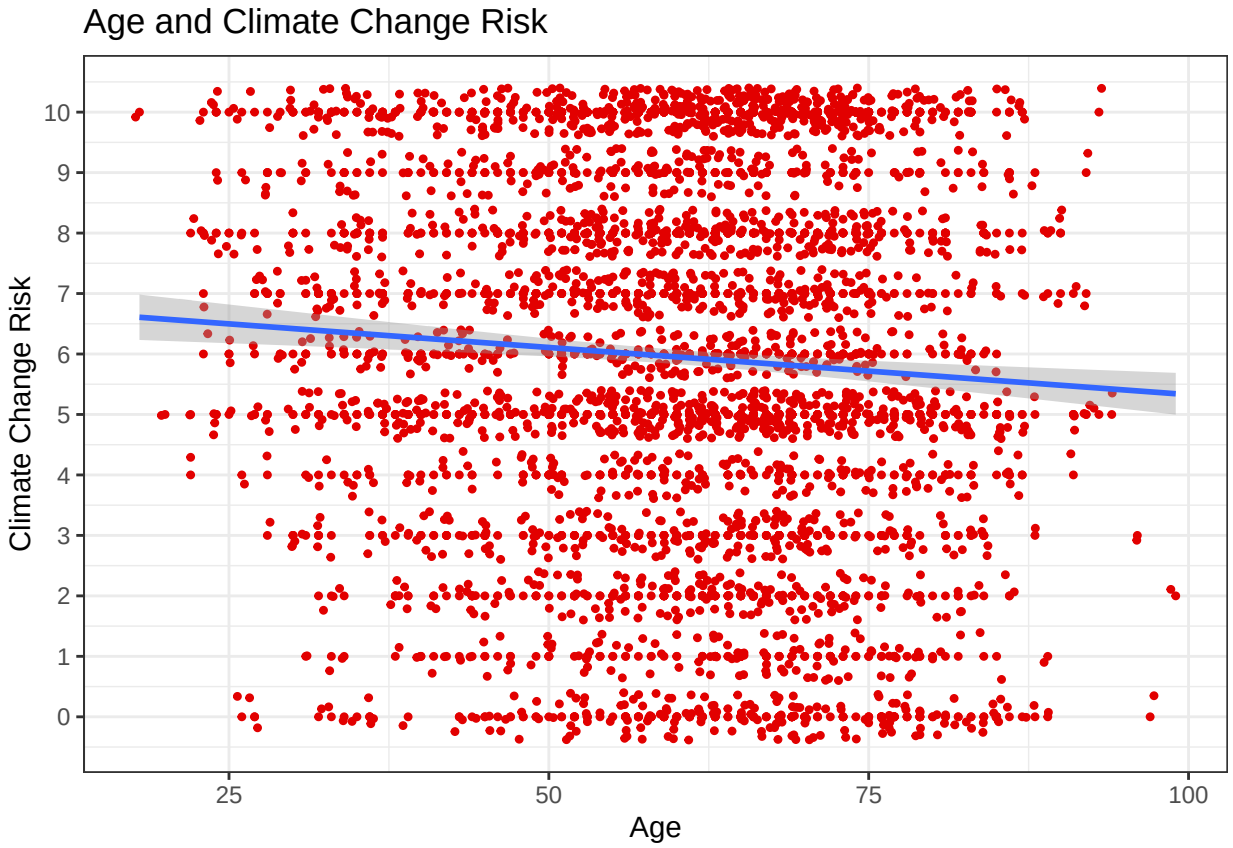
Now find the correlation:

```
cor.test(sub.ds$glbcc_risk, sub.ds$age)
```

```
##
##  Pearson's product-moment correlation
##
## data:  sub.ds$glbcc_risk and sub.ds$age
## t = -3.6446, df = 2534, p-value = 0.0002732
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.11082410 -0.03338258
## sample estimates:
##         cor
## -0.07221218
```

There is a slight negative correlation. We can interpret this as indicating that younger people are slightly more concerned about climate change. Now we construct the visualization:

```
ggplot(sub.ds, aes(y = glbcc_risk, x = age)) +
  geom_point(shape = 20, color = "#e20000") +
  geom_jitter(shape = 20, color = "#e20000") +
  geom_smooth(method = lm) +
  xlab("Age") +
  ylab("Climate Change Risk") +
  ggtitle("Age and Climate Change Risk") +
  scale_y_continuous(breaks = c(0:10),
                     labels = c("0","1","2", "3", "4", "5", "6", "7", "8", "9", "10")) +
  theme_bw()
```



8 Bivariate Linear Regression

This lab will cover the basics of bivariate linear regression, introducing via manual calculations and R functions. The following packages are required for this lab:

1. tidyverse
2. psych
3. car
4. memisc
5. stargazer
6. reshape2

8.1 Bivariate Linear Regression by Hand

The goal of bivariate linear regression is to estimate a line (slope and intercept) that minimizes the error term (residual). The bivariate linear regression model is as follows:

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

Where, `y` is the dependent variable, `i` is the unit of analysis, α is the y-intercept, β is the slope, `x` is the independent variable, and ε is the error term (residuals).

When working with samples, we develop an estimated model:

$$\hat{y} = \hat{\alpha} + \hat{\beta}x$$

Where, the hat implies the coefficients are estimates from data.

This lab focuses on the ordinary least squares method to find $\hat{\alpha}$ and $\hat{\beta}$ such that if given a value of `x` you can return an estimate of y ($\hat{y}$).

To demonstrate, we will explore the relationship between ideology and concern for natural resources. The first step is to subset the variables of interest absent of NA values:

```
ds %>%
  dplyr::select("ideol", "cncrn_natres") %>%
  na.omit() -> ds.sub
```

Reviewing the variables is an important first step:

```
describe(ds.sub$ideol)
```

```
##    vars    n mean   sd median trimmed  mad min max range  skew kurtosis
## X1    1 2508 4.65 1.73      5    4.75 1.48   1   7     6 -0.45     -0.8
##      se
## X1 0.03
```

```
describe(ds.sub$cncrn_natres)
```

```
##    vars    n mean   sd median trimmed  mad min max range  skew kurtosis
## X1    1 2508 7.57 2.22      8    7.82 2.97   0  10    10 -0.88     0.38
##      se
## X1 0.04
```

The ideology variable ranges from 1 "very liberal" to 7 "very conservative." The *cncrn_natres* variable measures concern for natural resources ranging from 0 "not concerned" to 10 "extremely concerned." The dependent and independent variables are defined as:

DV: Ideology
IV: Concern for natural resources

We need to find $\hat{\alpha}$ and $\hat{\beta}$ to develop the estimated bivariate regression model. Our first step is to find $\hat{\beta}$. Recall the formula for $\hat{\beta}$ is:

$$\hat{\beta} = \frac{cov(x, y)}{var(x)}$$

First create `x` and `y` variables to make the calculations simpler, with `x` being the independent variable and `y` being the dependent variable:

```
ds.sub$x <- ds.sub$ideol
ds.sub$y <- ds.sub$cncrn_natres
```

Next we calculate the cov(x,y) for the numerator and var(x) for the denominator:

```
r <- cor(ds.sub$x, ds.sub$y)
cov.xy <- cov(ds.sub$x, ds.sub$y, use = "complete.obs")
var.x <- var(ds.sub$x, na.rm = T)
```

Finally, to calculate $\hat{\beta}$:

```
beta.hat <- cov.xy / var.x
```

Now that we have $\hat{\beta}$, we can calculate $\hat{\alpha}$ Recall the formula for calculating $\hat{\alpha}$:

$$\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$$

Calculate $\bar{y}$ and $\bar{x}$ from the sample data:

```
ybar <- mean(ds.sub$y, na.rm = T)
xbar <- mean(ds.sub$x, na.rm = T)
```

Use $\bar{y}$, $\bar{x}$, and $\hat{\beta}$ to calculate $\hat{\alpha}$:

```
alpha.hat <- ybar - beta.hat * xbar
```

Take a look at our calculated $\hat{\alpha}$ and $\hat{\beta}$:

```
alpha.hat
```

```
## [1] 8.815973
```

```
beta.hat
```

```
## [1] -0.2687128
```

Now create some predicted values of y, concern for natural resources, based on our estimated regression model:

```
yhat <- alpha.hat + beta.hat * ds.sub$x
head(yhat) # Returns the first 5 values
```

```
## [1] 7.203696 8.009834 7.203696 7.741122 7.741122 8.278547
```

To draw inference using the estimated regression model we need to understand how `x` helps us understand `y`. More formally, we explore this relationship by evaluating the residuals associated to the coefficients $\hat{\alpha}$ and $\hat{\beta}$. We are interested in whether the coefficients are statistically different than zero.

-H_0: $\beta = 0$
-H_1: $\beta \neq 0$

Put simply, if the coefficient $\hat{\beta}$ is zero, than that is to say that no value of x helps us understand y ($0 * x = 0$).

If we assume the residuals follow a normal distribution, then we can calculate the t-statistic to examine the coefficient's statistical significance.

Recall that residuals are the differences in the observed values of the data and the predicted values of the estimated regression model. We can calculate residuals by subtracting the $\hat{y}$ values we just calculated from the `y` values:

```
res <- ds.sub$y - yhat
```

Now that we have the residuals, we need to calculate the residual standard error, which measures the spread of observations around the regression line we just calculated. We also need to know the residual standard error so that we can find the standard errors for the regression coefficients, which are then used to calculate the t scores of the coefficients. To calculate the residual standard error we need to find:

1. The residual sum of squares
2. The degrees of freedom for our model

```
res.sqr <- res^2
RSS <- sum(res.sqr, na.rm=T)
```

Now we need to calculate the degrees of freedom. In this case, we have the intercept and the ideology variable, so we subtract two. Since we subset our data to remove all missing observations, we know the n size for x and y are the same:

```
df <- length(ds.sub$y) - 2
df
```

```
## [1] 2506
```

With the residual sum of squares and the degrees of freedom, we have what we need to find the residual standard error:

```
RSE <- sqrt(RSS / df)
RSE
```

```
## [1] 2.167453
```

With the residual standard error, we can now start to calculate the standard errors for our coefficients: $\hat{\alpha}$ and $\hat{\beta}$ (ideology). To calculate these, we need to find the total sum of squares of the independent variable, x:

```
TSSx <- sum((ds.sub$x - xbar)^2)
TSSx
```

```
## [1] 7520.929
```

Now that we have the total sum of squares for the independent variable, we can find the standard errors of our coefficients:

```
SEB <- RSE / sqrt(TSSx)
SEB
```

```
## [1] 0.02499274
```

For the standard error of $\hat{\alpha}$, the calculation is different:

```
SEA <- RSE * sqrt((1 / 2508)+(xbar^2 / TSSx))
SEA
```

```
## [1] 0.1240024
```

With the standard errors calculated, we can now find the corresponding t-statistics:

```
t.B <- beta.hat / SEB
t.B
```

```
## [1] -10.75163
```

```
t.A <- alpha.hat / SEA
t.A
```

```
## [1] 71.0952
```

These t-statistics tell us how many standard deviations the coefficients are away from 0.

8.1.1 Calculating Goodness of Fit

Now we turn to R^2, the measure how well the estimated regression model explains the variability of the data. To find R^2, you need to know:

1. The residual sum of squares
2. The total sum of squares
3. The explained sum of squares.

We already have the residual sum of squares. We found it by taking the sum of the residuals squared. Earlier we calculated the total sum of squares for our independent variable, x, but now we need to find the total sum of squares for our dependent variable, y.

```
TSS <- sum((ds.sub$y - ybar)^2)
TSS
```

```
## [1] 12315.88
```

Now that we have the residual sum of squares and the total sum of squares, we can find the explained sum of squares. The explained sum of squares tells us how much of the variance of the dependent variable is accounted for by our model.

```
ESS <- TSS - RSS
ESS
```

```
## [1] 543.0606
```

R^2 is found by dividing the explained sum of squares by the total sum of squares:

```
r.sqr <- ESS / TSS
r.sqr
```

```
## [1] 0.04409434
```

4% of the variability of the data is explained by the estimated regression model.

8.1.1.1 Adjusted R Squared

There is a slightly more accurate measure of model fit, though, known as adjusted R squared. Adjusted R squared addresses some problems that are inherent in the R squared calculation, like the realtiy that R squared tends to increase as you add more predictors to your model, even if it's more due to chance than actual predicting power. Adjusted R squared addresses this issue by penalizing the model for an increased number of predictors. Use this formula to find adjusted R squared

$$1 - \frac{(1 - R^2)(n - 1)}{n - k - 1}$$

where k is the number of predictors in our model, not including the intercept(A). We can actually find this pretty simply. Recall that we are working with an n size of 2508. Our k value is 1, becuase we only have one predictor in the model (ideology). Find the adjusted R squared value:

```
adj.r2 <- 1-(((1 - r.sqr) * (2508 - 1)) / (2508 - 1 - 1))
adj.r2
```

```
## [1] 0.04371289
```

8.1.2 Checking Our Work

To check our work in the previous steps we will employ the native R functions for linear models: `lm()`. We provide the observed y and x values from data as an argument, separated by a tilda in the following format: `lm(y ~ x)`

To demonstrate with the `cncrn_natres` and `ideol` variables:

```
model <- lm(ds.sub$cncrn_natres ~ ds.sub$ideol)
```

Further, the `summary()` function will provide results of our model, including t-statistics and R^2 values:

```
summary(model)
```

```
##
## Call:
## lm(formula = ds.sub$cncrn_natres ~ ds.sub$ideol)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.2785 -1.2785  0.3558  1.7215  3.0650
##
## Coefficients:
##              Estimate Std. Error t value            Pr(>|t|)
## (Intercept)   8.81597    0.12400   71.09 <0.0000000000000002 ***
## ds.sub$ideol -0.26871    0.02499  -10.75 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.167 on 2506 degrees of freedom
## Multiple R-squared:  0.04409,    Adjusted R-squared:  0.04371
## F-statistic: 115.6 on 1 and 2506 DF,  p-value: < 0.00000000000000022
```

Let's compare to our coefficients, residual standard error, coefficient standard errors, t scores, r squared, and adjusted r squared:

```
stats <- data.frame(name = c("Intercept", "Beta", "RSE", "IntSE", "BetaSE", "IntT",
                             "BetaT", "Rsqr", "AdjRsqr"),
                    values = c(alpha.hat, beta.hat, RSE, SEA, SEB, t.A, t.B,
                             r.sqr, adj.r2))
stats
```

```
##        name       values
## 1 Intercept   8.81597293
## 2      Beta  -0.26871281
## 3       RSE   2.16745310
## 4     IntSE   0.12400237
## 5    BetaSE   0.02499274
## 6      IntT  71.09519785
## 7     BetaT -10.75163278
## 8      Rsqr   0.04409434
## 9   AdjRsqr   0.04371289
```

8.2 Bivariate Regression in R

Suppose you are interested in the relationship between ideology and opinions about how much of Oklahoma's electricity should come from renewable sources. The class data set includes these variables as `ideol` and `okelec_renew`. A new subset data set is created with these variables, absent missing observations, to develop an estimated regression model:

```
sub.ds <- ds %>%
  dplyr::select("okelec_renew", "ideol") %>%
  na.omit()
```

The `okelec_renew` variable is new to us, so we should examine its structure:

```
str(sub.ds$okelec_renew)
```

```
##  Factor w/ 63 levels "0","1","10","100",..: 20 42 32 42 54 42 20 13 53 4 ...
```

The variable appears to be a factor, which needs to be coerced as a numeric type. Recall the `as.numeric()` function:

```
sub.ds %>%
  mutate(renew = as.numeric(okelec_renew)) %>%
  drop_na() -> sub.ds
```

Reassess the structure to ensure it is numeric:

```
str(sub.ds$renew)
```

```
##  num [1:2524] 20 42 32 42 54 42 20 13 53 4 ...
```

With the variable now numeric, examine it using the `describe()` function:

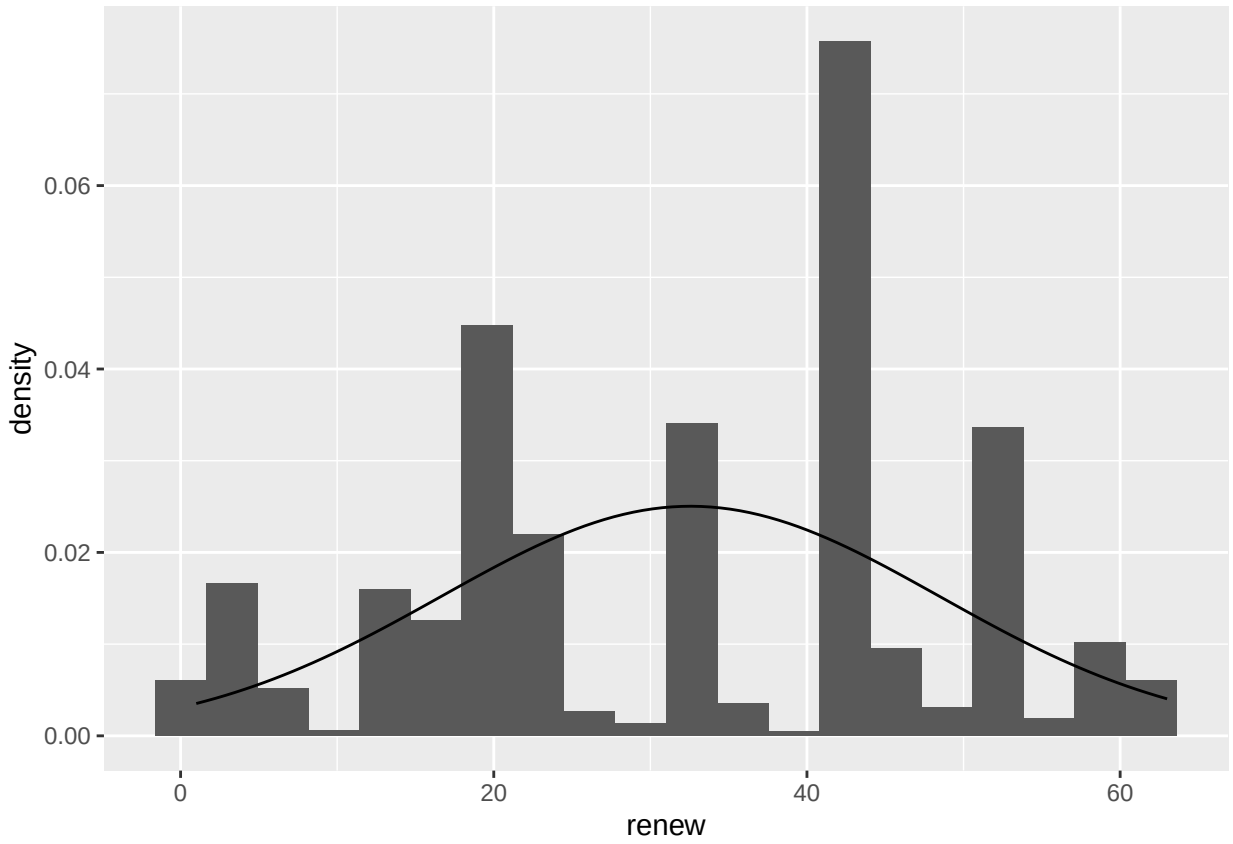
```
describe(sub.ds$renew)
```

```
##    vars    n  mean    sd median trimmed   mad min max range  skew kurtosis
## X1    1 2524 32.56 15.94     34   33.03 20.76   1  63    62 -0.18    -0.96
##      se
## X1 0.32
```

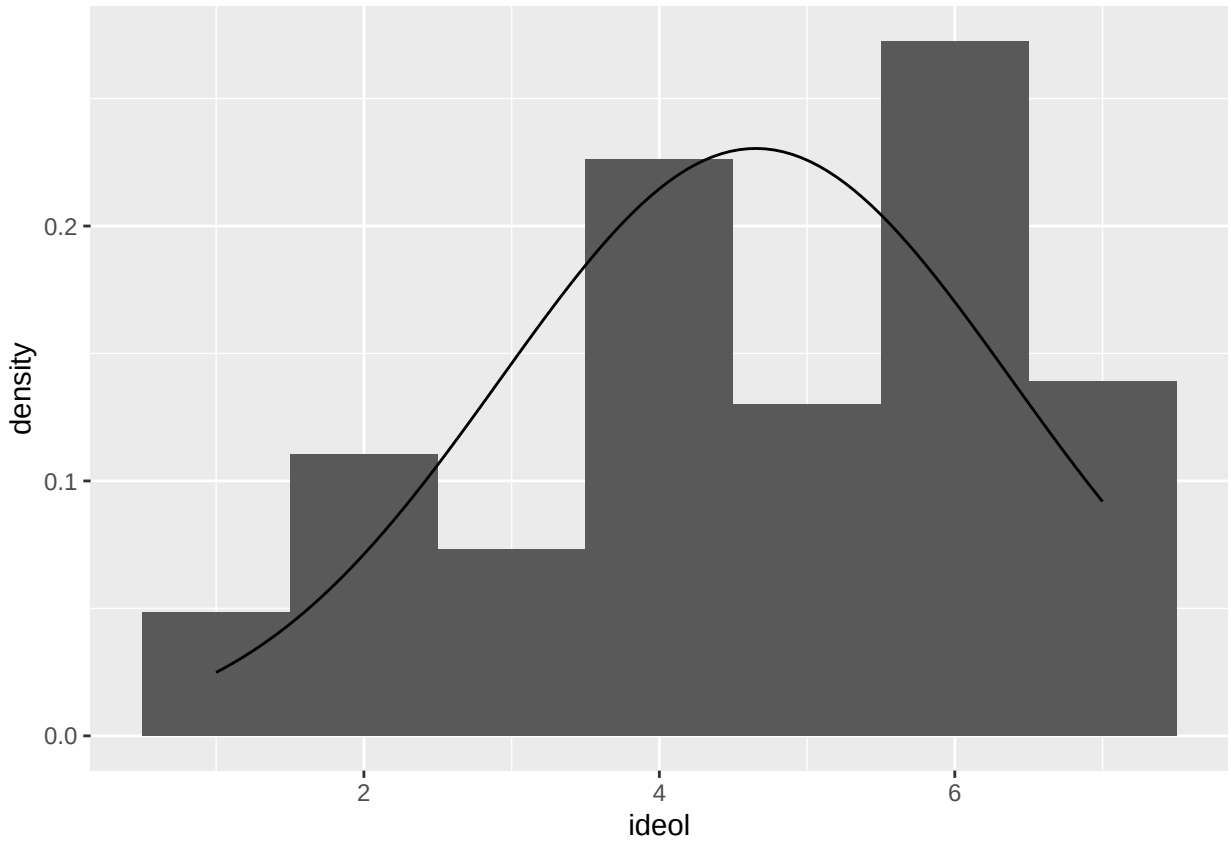
Before proceeding, we should formalize a hypothesis. Perhaps we hypothesize that conservatives want a lower percentage of renewable energy than liberals. Using renewable energy as a function of ideology, we further specify our hypothesis as a more conservative ideology corresponds to a decrease preference for renewable energy. The null hypothesis is there is no difference in preference among ideologies.

First we examine the normality of both variables. Create a histogram and overlay it with a normal distribution curve, with the correct mean and standard deviation.:

```
ggplot(sub.ds, aes(renew)) +
  geom_histogram(aes(y= ..density.. ), bins = 20) +
  stat_function(fun = dnorm, args = list(mean = mean(sub.ds$renew),
                                         sd = sd(sub.ds$renew)))
```




```
ggplot(sub.ds, aes(ideol)) +
  geom_histogram(aes(y = ..density..), bins = 7) +
  stat_function(fun = dnorm, args = list(mean = mean(sub.ds$ideol), sd = sd(sub.ds$ideol)))
```



Now construct the model:

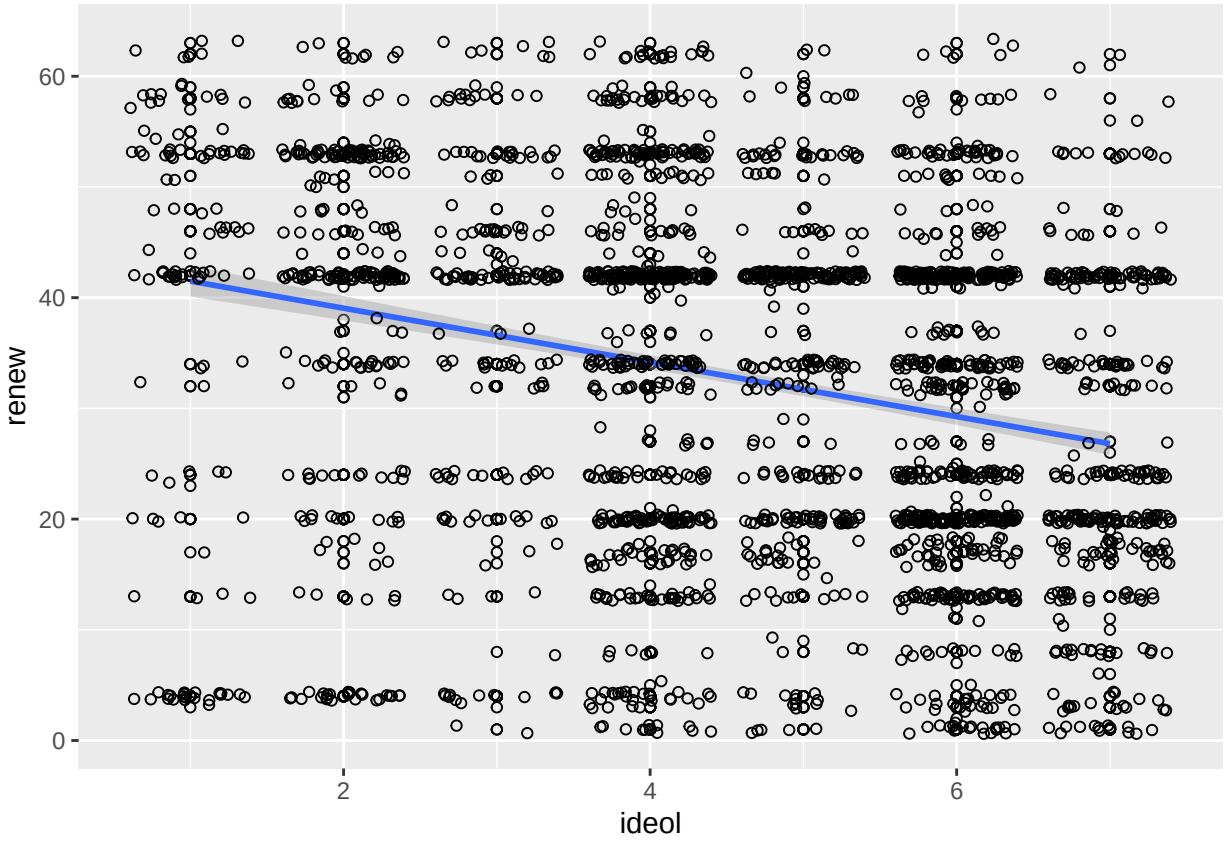
```
model1 <- lm(sub.ds$renew ~ sub.ds$ideol)
summary(model1)
```

```
##
## Call:
## lm(formula = sub.ds$renew ~ sub.ds$ideol)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -38.50 -11.26   2.74  12.74  35.19
##
## Coefficients:
##              Estimate Std. Error t value            Pr(>|t|)
## (Intercept)   43.9430     0.8772   50.10 <0.0000000000000002 ***
## sub.ds$ideol  -2.4472     0.1767  -13.85 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.36 on 2522 degrees of freedom
## Multiple R-squared:  0.07069,    Adjusted R-squared:  0.07032
## F-statistic: 191.8 on 1 and 2522 DF,  p-value: < 0.00000000000000022
```

Based on the results, ideology helps us understand preference for renewable energy. Further examination of the coefficient for `ideol` yields an estimate value of -2.45. This is interpreted as a -2.45 unit decrease in renewable energy preference for each unit increase in ideology. That is, an increase on the ideology scale (1 "liberal" to 7 "conservative") results in a reduction in preference for renewable energy.

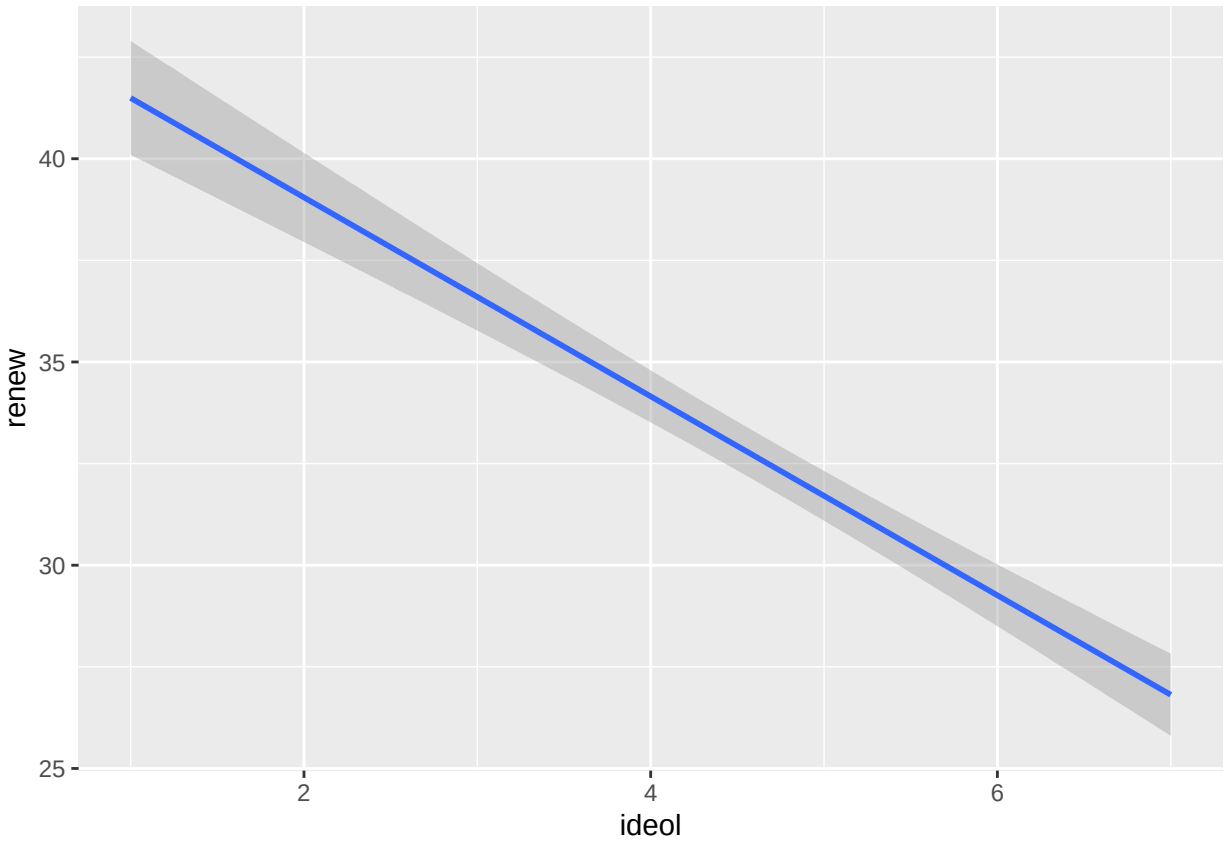
We should always visualize a relationship that we're trying to convey. Start by constructing a scatter plot and adding a regression line:

```
ggplot(sub.ds, aes(x = ideol, y = renew)) +
  geom_point(shape = 1) +
  geom_smooth(method = lm) +
  geom_jitter(shape = 1)
```



Sometimes it is more beneficial when visualizing the relationship to plot the regression line without first showing the scatter plot. To do this with ggplot2, simply do not include the `geom_point()` or `geom_jitter()` functions in the visualization.

```
ggplot(sub.ds, aes(x = ideol, y = renew)) +
  geom_smooth(method = lm)
```



8.3 The Residuals

Recall that when using Ordinary Least Squares regression, there are three assumptions made about the error terms:

1. Errors have identical distributions
2. Errors are independent of X and other error terms
3. Errors are normally distributed

To look at the residual values for the estimated regression model, use the `names()` function:

```
names(model)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

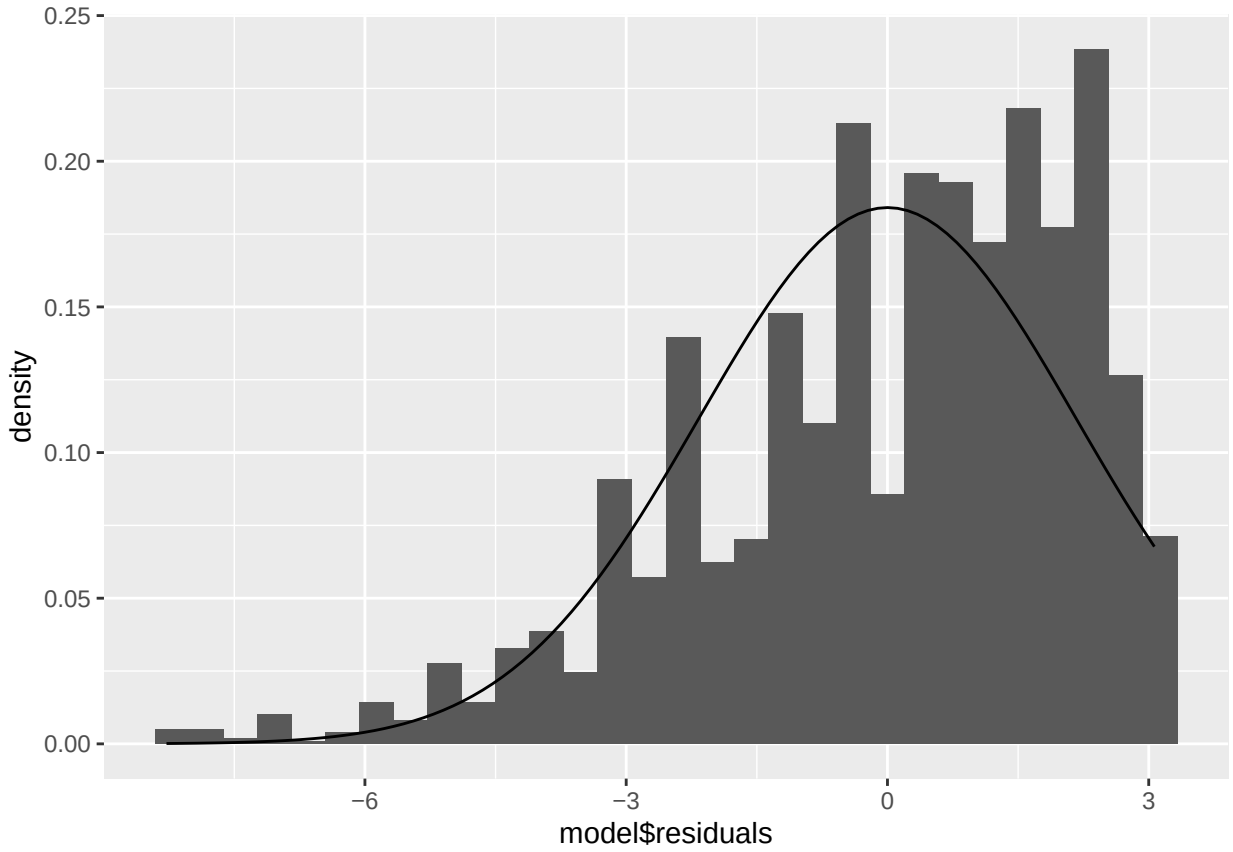
In R we can examine the distribution of the residuals relatively simply. First assign the residuals to an object:

```
resid <- model$residuals
```

Now plot a histogram of the residuals, adding a normal density curve with the mean and standard deviation of our residuals:

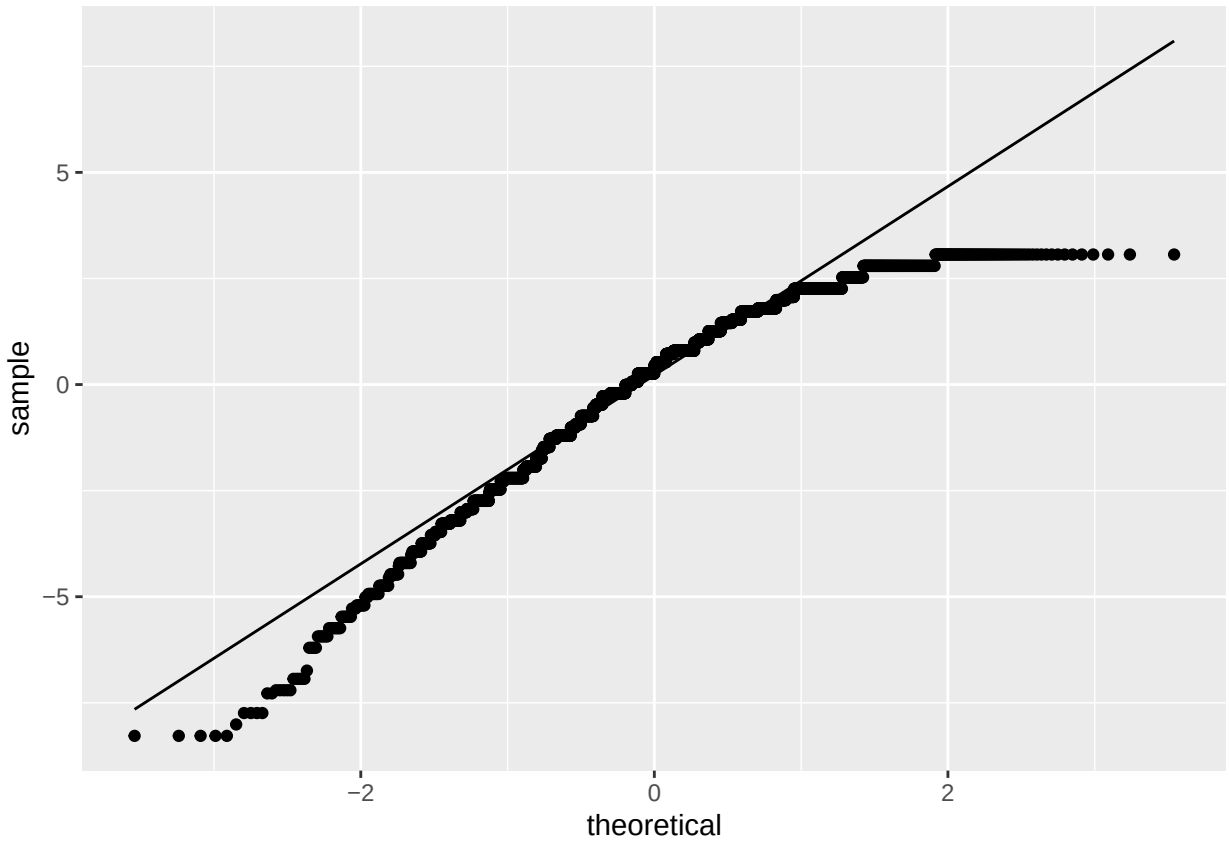
```
ggplot(model, aes(model$residuals)) +
  geom_histogram(aes(y =  ..density..)) +
   stat_function(fun = dnorm, args = list(mean = mean(model$residuals), sd = sd(model$residuals)))
```

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



We also look at a QQ plot of the residuals:

```
ggplot(model, aes(sample = model$residuals)) +
  stat_qq() +
  stat_qq_line()
```



8.4 Comparing Models

Suppose you wanted to create multiple bivariate models and contrast them. This is possible using the `mtable()` function from the `memisc` package. To demonstrate, we create three models looking at the relationship between an independent variable and concern about global climate change. The three independent variables will be ideology, certainty that humans cause climate change, and age. We start by creating a subset of our data and removing missing observations:

```
sub <- ds %>%
  dplyr::select("glbcc_risk", "glbcc_cert", "age", "ideol") %>%
  na.omit()
model1 <- lm(sub$glbcc_risk ~ sub$ideol)
model2 <- lm(sub$glbcc_risk ~ sub$glbcc_cert)
model3 <- lm(sub$glbcc_risk ~ sub$age)
```

Using the `mtable()` function, we can create regression tables that compare all three of the models:

```
mtable(model1, model2, model3)
```

```
##
## Calls:
## model1: lm(formula = sub$glbcc_risk ~ sub$ideol)
## model2: lm(formula = sub$glbcc_risk ~ sub$glbcc_cert)
## model3: lm(formula = sub$glbcc_risk ~ sub$age)
##
## ============================================================
##                       model1        model2        model3
```

```
## ------------------------------------------------------------
##   (Intercept)        10.821***      3.171***      6.927***
##                      (0.142)       (0.150)       (0.267)
##   sub$ideol          -1.048***
##                      (0.029)
##   sub$glbcc_cert                    0.419***
##                                     (0.021)
##   sub$age                                        -0.016***
##                                                  (0.004)
## ------------------------------------------------------------
##   R-squared           0.349         0.137         0.006
##   adj. R-squared      0.349         0.137         0.005
##   sigma               2.479         2.854         3.064
##   F                1344.256       397.859        14.232
##   p                   0.000         0.000         0.000
##   Log-likelihood  -5834.480     -6188.233     -6365.911
##   Deviance        15399.025     20417.722     23525.687
##   AIC             11674.961     12382.465     12737.823
##   BIC             11692.442     12399.947     12755.304
##   N                2508          2508          2508
## ============================================================
```

Alternatively, you can use the `stargazer()` function to create tables for your models. The `stargazer()` function is different in that you can specify a table created with text or with LaTex code that you can subsequently paste into a LaTex document. We'll create a text table, but if you wanted to create a Latex table, you would use the `type=latex` argument.

```
stargazer(model1, model2, model3, type="text", style="apsr")
```

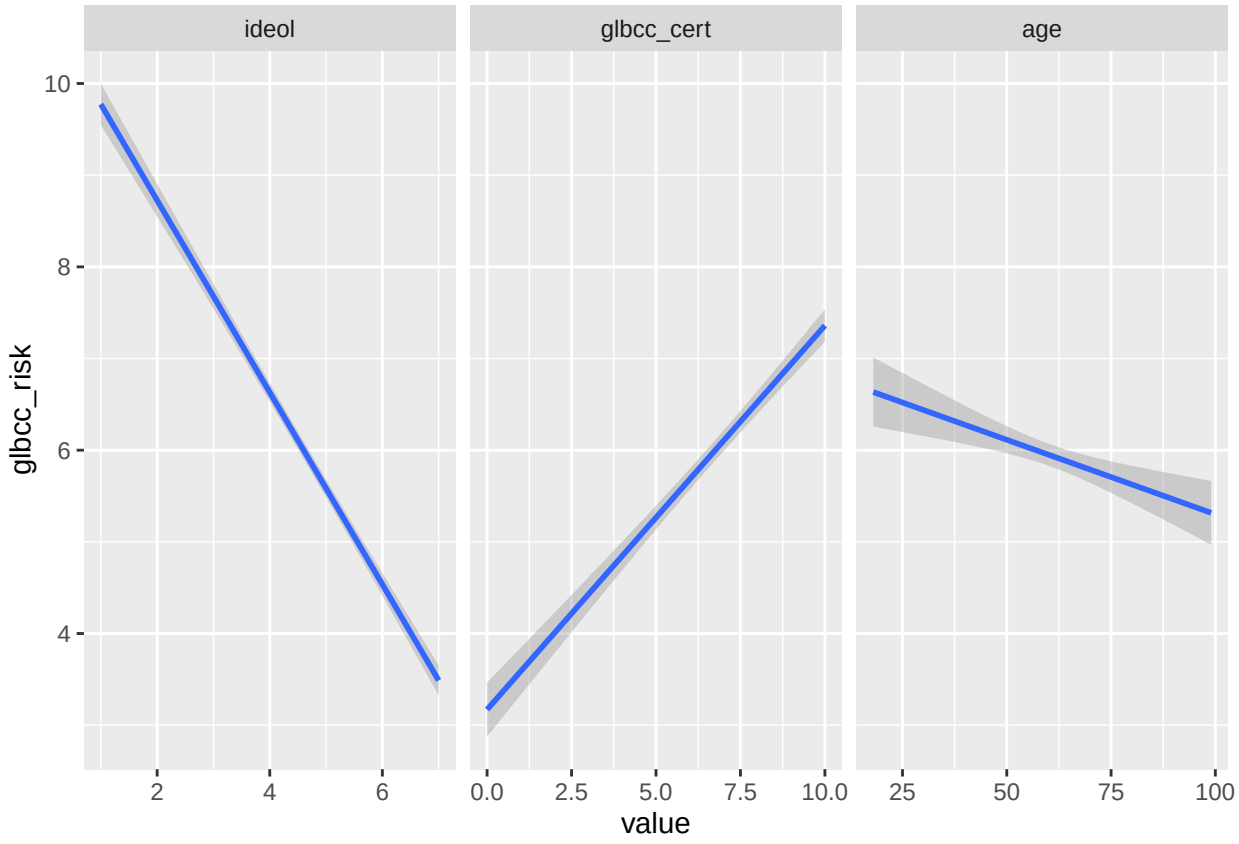
```
##
## ---------------------------------------------------------------------
##                                          glbcc_risk
##                                    (1)         (2)         (3)
## ---------------------------------------------------------------------
## ideol                            -1.048***
##                                   (0.029)
## glbcc_cert                                    0.419***
##                                               (0.021)
## age                                                      -0.016***
##                                                           (0.004)
## Constant                         10.821***    3.171***  6.927***
##                                   (0.142)     (0.150)    (0.267)
## N                                  2,508       2,508      2,508
## R2                                 0.349       0.137      0.006
## Adjusted R2                        0.349       0.137      0.005
## Residual Std. Error (df = 2506)    2.479       2.854      3.064
## F Statistic (df = 1; 2506)      1,344.256*** 397.859*** 14.232***
## ---------------------------------------------------------------------
## *p < .1; **p < .05; ***p < .01
```

8.4.1 Visualizing Multiple Models

The three models can be visualized together by melting the data set into long form, with the three IVs as measure variables, then using `ggplot2` and facet wrapping by independent variable. Including `scales`

= "free_x" inside the facet_wrap() function will make the visualization so that each plot has its own independent x axis but is on the same fixed y axis.

```
melt(sub, measure.vars = c("ideol", "glbcc_cert", "age"),
             variable.name = c("IV")) %>%
  ggplot(., aes(value, glbcc_risk)) +
  geom_smooth(method = lm) +
  facet_wrap(~ IV, scales = "free_x")
```



8.5 Hypothesis Testing

Let's do one more example of how we would hypothesis test with bivariate regression. In the class data set there is a variable, wtr_comm, that asks the respondent if they think the supplies of water in their region will be adequate to meet their community's needs over the next 25 years. In essence, this measures concern about water supply for the community.

Start with a research question: What is the relationship between concern for water supply and concern about climate change?

Now build a theory: We could reasonably theorize that individuals who are more concerned about water supply are also likely more concerned about climate change. There is likely a link in their head between climate change and a shortened water supply.

Built on this theory, we can specify a hypothesis that individuals more concerned about climate change will be more concerned about water supply for their community. The null hypothesis is that there is no relationship between climate change concern and water concern.

We create a subset of the dataset that includes out two variables and remove missing observations:

```
new.ds <- ds %>%
  dplyr::select("wtr_comm", "glbcc_risk") %>%
  na.omit()
```

Now examine the variables:

```
describe(new.ds$wtr_comm)
```

```
##    vars    n mean   sd median trimmed  mad min max range  skew kurtosis
## X1    1 2524 3.21 1.03      3    3.24 1.48   1   5     4 -0.34    -0.68
##      se
## X1 0.02
```

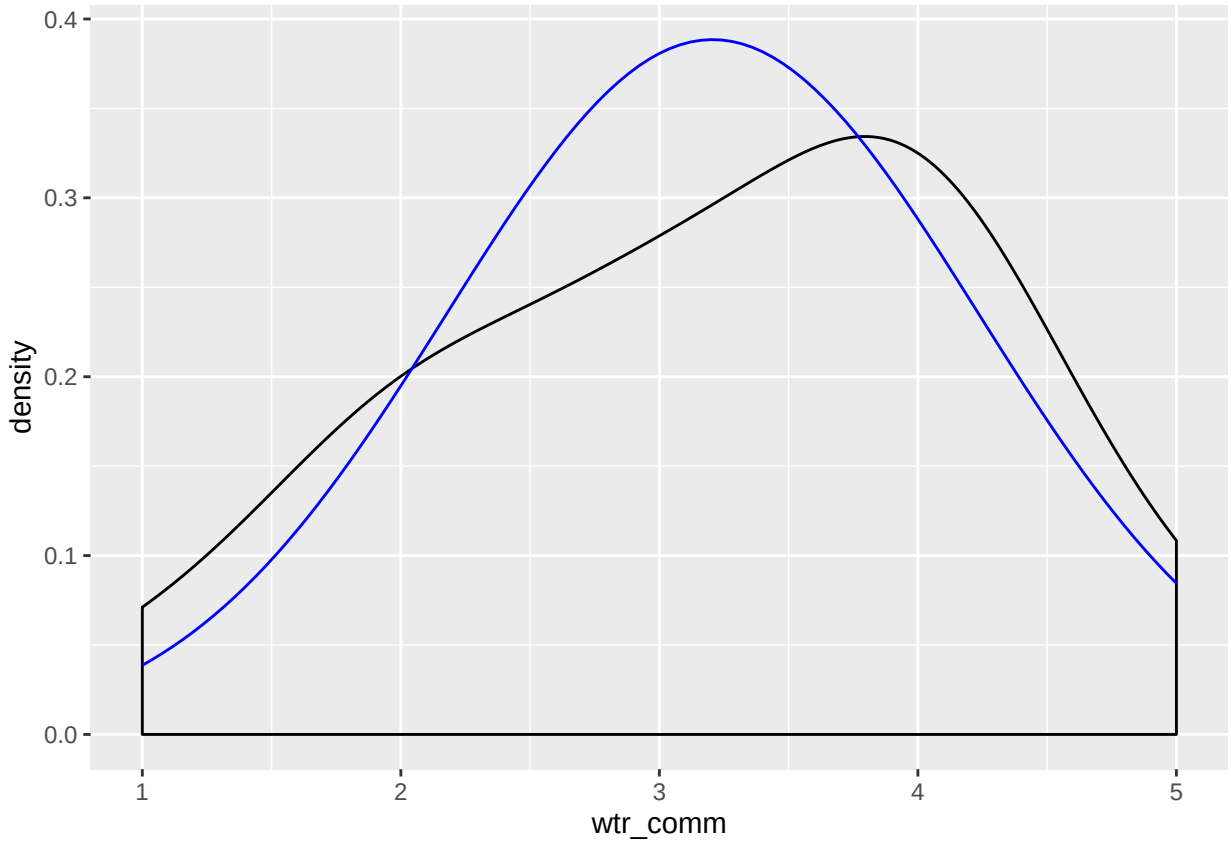
```
describe(new.ds$glbcc_risk)
```

```
##    vars    n mean   sd median trimmed  mad min max range  skew kurtosis
## X1    1 2524 5.95 3.07      6    6.14 2.97   0  10    10 -0.32    -0.93
##      se
## X1 0.06
```

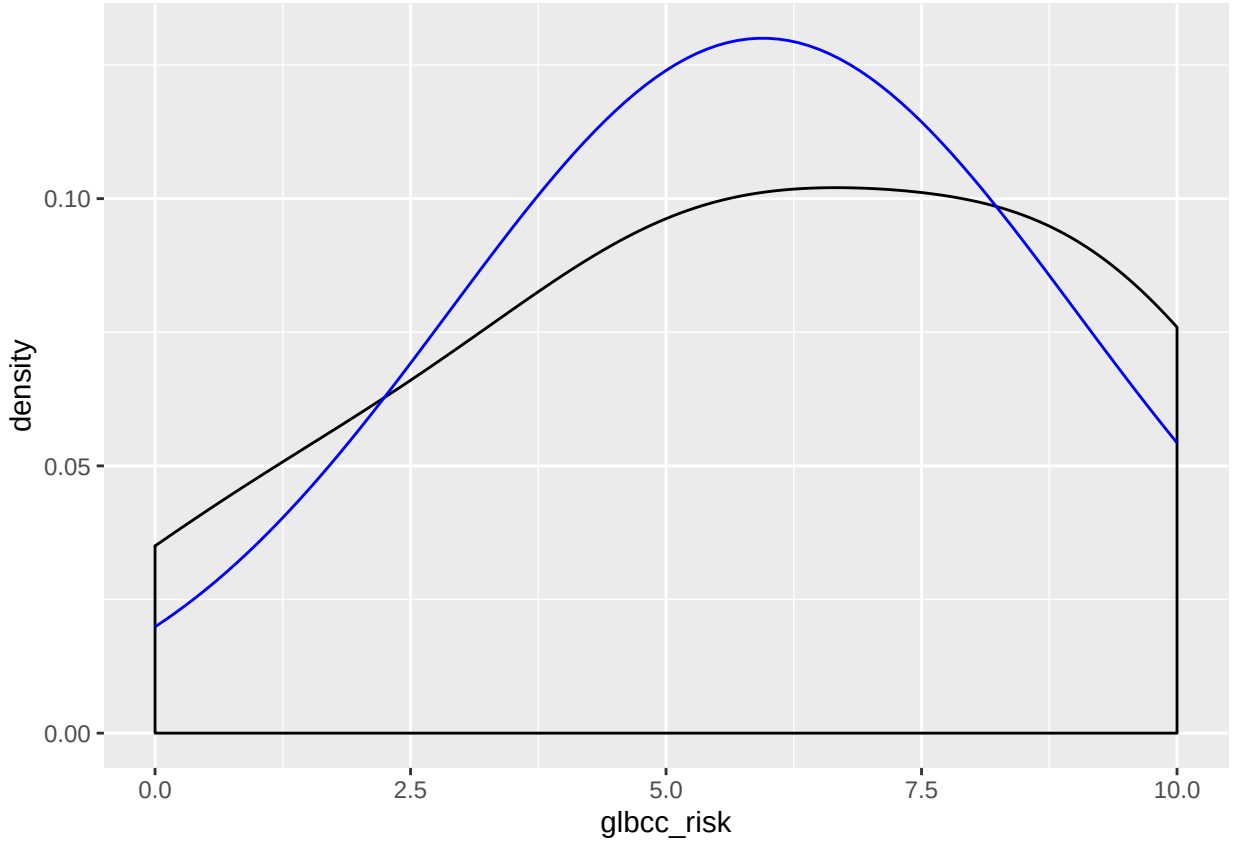
Note that the climate change risk variable goes from 0 to 10 and the water supply concern variable ranges from 1 to 5, with 1 being definitely no (the supplies of water are NOT enough) and 5 being definitely yes.

Now visualize the normality of the variables:

```
ggplot(new.ds, aes(wtr_comm)) +
  geom_density(adjust = 3) +
  stat_function(fun = dnorm, args = list(mean = mean(new.ds$wtr_comm), sd = sd(new.ds$wtr_comm)), color
```



```
ggplot(new.ds, aes(glbcc_risk)) +
  geom_density(adjust = 3) +
  stat_function(fun = dnorm, args = list(mean = mean(new.ds$glbcc_risk), sd = sd(new.ds$glbcc_risk)), co
```



Next, create the model. Recall that the dependent variable is concern for water supply and the independent variable is climate change risk.

```
lm1 <- lm(new.ds$wtr_comm ~ new.ds$glbcc_risk)
```

Now examine and interpret the results:

```
summary(lm1)
```

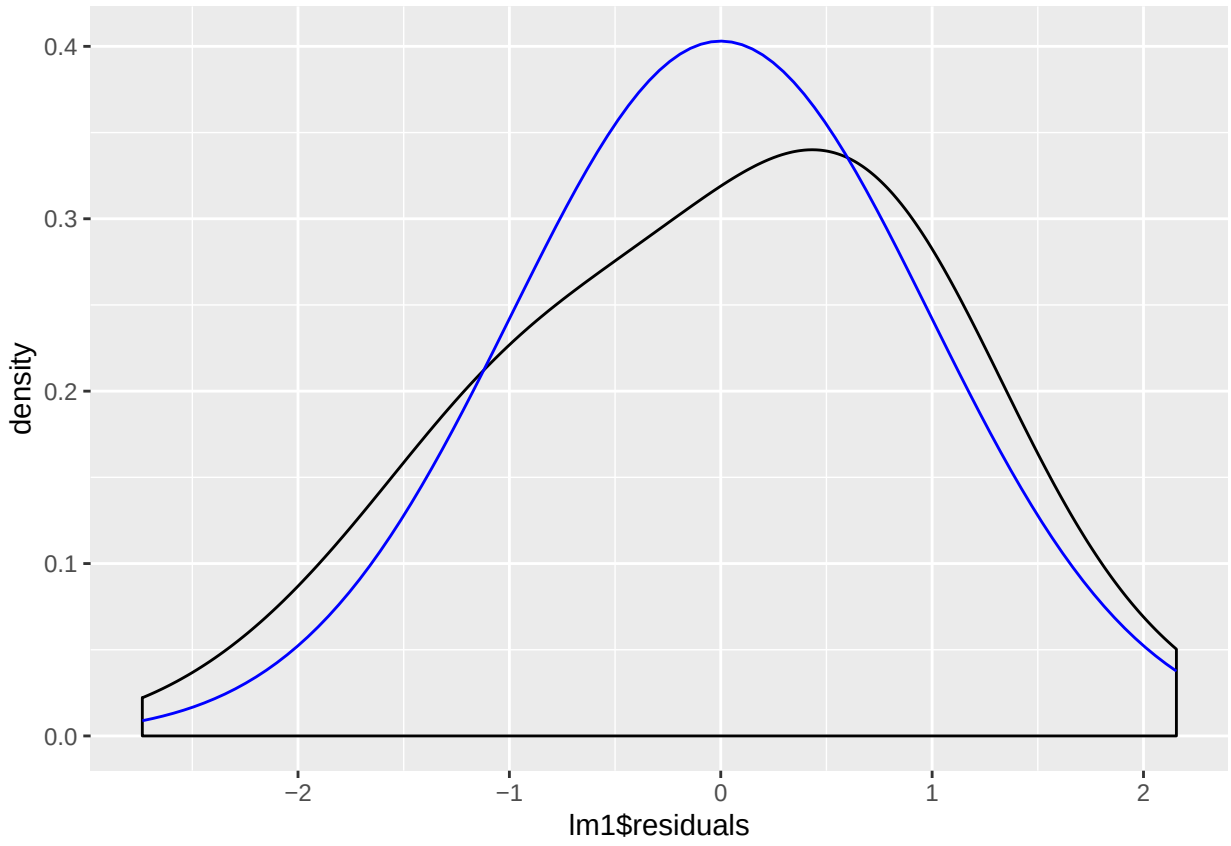
```
##
## Call:
## lm(formula = new.ds$wtr_comm ~ new.ds$glbcc_risk)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7366 -0.8442  0.1557  0.7096  2.1557
##
## Coefficients:
##                    Estimate Std. Error t value            Pr(>|t|)
## (Intercept)        3.736575   0.042974   86.95 <0.0000000000000002 ***
## new.ds$glbcc_risk -0.089233   0.006423  -13.89 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.99 on 2522 degrees of freedom
## Multiple R-squared:  0.07109,    Adjusted R-squared:  0.07072
## F-statistic:   193 on 1 and 2522 DF,  p-value: < 0.00000000000000022
```

The independent variable coefficient is about -.09, with a corresponding p-value ≈ 0. This is interpreted as a one unit change in climate change risk corresponds with a -0.09 unit change in water supply concern. Remember that the water supply variable goes from 1 (there is definitely not enough water) to 5 (there definitely is enough water). These findings suggest that an individual more concerned about climate change is also more concerned about water supply.

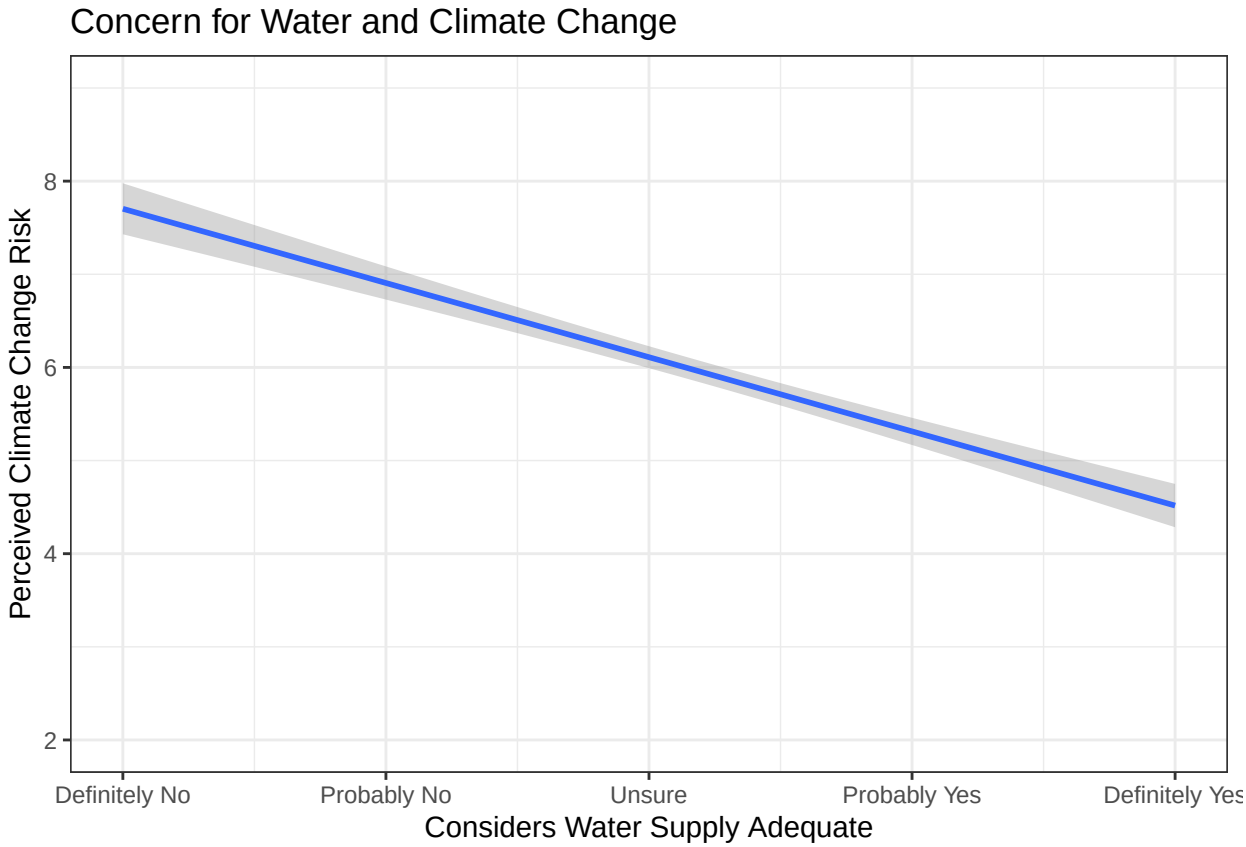
We examine the normality of the residuals:

```
ggplot(lm1, aes(lm1$residuals)) +
  geom_density(adjust = 3) +
  stat_function(fun = dnorm, args = list(mean = mean(lm1$residuals), sd = sd(lm1$residuals)), color = "
```



Now build a good visualization that would be worthy of a paper:

```
ggplot(new.ds, aes(wtr_comm, glbcc_risk)) +
  geom_smooth(method = lm) +
  coord_cartesian(ylim = c(2, 9), xlim = c(1, 5)) +
  ggtitle("Concern for Water and Climate Change") +
  xlab("Considers Water Supply Adequate") +
  ylab("Perceived Climate Change Risk") +
  scale_x_continuous(breaks=c(1, 2 ,3 ,4 ,5), labels=c("Definitely No",
                                                       "Probably No", "Unsure",
                                                       "Probably Yes", "Definitely Yes")) +
  theme_bw()
```



Our findings support that individuals less concerned about climate change are also less concerned about their community's water supply.

9 Multivariable Linear Regression

This lab covers the basics of multivariable linear regression. We begin by reviewing linear algebra to perform ordinary least squares (OLS) regression in matrix form. Then we will cover an introduction to multiple linear regression and visualizations with R. The following packages are required for this lab:

1. tidyverse
2. psych
3. car
4. stargazer
5. reshape2
6. broom
7. skimr

9.1 Calculating Least-Squared Estimates

The previous lab introduced the estimated bivariate linear regression model as follows:

$$\hat{y} = \hat{\alpha} + \hat{\beta}x$$

Where $\hat{\alpha}$ and $\hat{\beta}$ are solved via the following formulas:

$$\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$$

$$\hat{\beta} = \frac{cov(x, y)}{var(x)}$$

In this lab we use matrix algebra to calculate the least-squared estimates. This proves useful for multivariable linear regression models where the methods introduced for bivariate regression models become more complex and computationally cumbersome to express as equations.

9.1.1 Matrix Algebra

For multivariable regression analysis, the formulas for calculating coefficients are more easily found using matrix algebra. In this section we cover the basics of matrix algebra relevant to calculating the coefficients for an estimated linear regression model. A matrix is a rectangular array of numbers organized in rows and columns and each number within a matrix is an element. A matrix is defined as consisting of `m` rows and `n` columns. **Note:** If m = n, the matrix is referred to as a square matrix.

The following is a general form of a matrix:

$$x_{m \times n} = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \dots & x_{mn} \end{bmatrix}$$

There are a number of operations in matrix algebra; however, this review focuses on those pertinent to solutions of the least-squared equations in multiple regression:

9.1.1.1 Transpose of a matrix

The transpose of a matrix **A**, denoted as **A'**, is obtained by interchanging the rows and columns of the ***A*** matrix:

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 5 & 7 & 6 \end{bmatrix}, A' = \begin{bmatrix} 1 & 5 \\ 2 & 7 \\ 4 & 5 \end{bmatrix}$$

Note: The product of a matrix and its transpose is a square matrix.

The `matrix()` function in R will create a matrix object consisting of provided values in a defined number of rows and columns. The above matrix, `A`, is created as follows:

```
A <- matrix(c(1, 2, 4, 5, 7, 6), 2, 3)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    6
```

Further, the `t()` function will transpose a given matrix:

```
Aprime <- t(A)
Aprime
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    4    5
## [3,]    7    6
```

9.1.1.2 Row-column multiplication

Matrix multiplication is done by summing the products of the row elements in the first matrix by the column elements in the second matrix in corresponding position. This is shown in the following example:

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 5 & 7 & 6 \end{bmatrix} \times A' = \begin{bmatrix} 1 & 5 \\ 2 & 7 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 1*1+2*2+4*4 & 1*5+2*7+4*6 \\ 5*1+7*2+6*4 & 5*5+7*7+6*6 \end{bmatrix} = \begin{bmatrix} 21 & 43 \\ 43 & 110 \end{bmatrix}$$

Similarly, the product of matrices can be calculated in R using the %*% operator:

```
AxAprime <- A %*% Aprime
AxAprime
```

```
##      [,1] [,2]
## [1,]   66   64
## [2,]   64   65
```

Note: Not all matrices can be multiplied. The number of columns in the first matrix must equal the number of rows in the second matrix. Further, unlike ordinary algebra multiplication, matrix multiplication is NOT commutative (order of operands matter). In the previous example we were able to find the product of A and A', because the number of columns in A (3) is equal to the number of rows in A' (3). Suppose we wanted the product of A' and B, where B is defined as the following matrix:

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 6 & 4 \\ 4 & 5 & 6 \end{bmatrix}$$

We are unable to find the product of A'B because the number of columns in A' (2) does not equal the number of rows in B (3). We are able to find the product of BA' as follows:

$$A' \times B = \begin{bmatrix} 1 & 2 & 4 \\ 5 & 7 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 6 & 4 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 21 & 34 & 35 \\ 42 & 82 & 79 \end{bmatrix}$$

Or, in R:

```
B <- matrix(c(1, 2, 3, 2, 6, 4, 4, 5, 6), 2, 3)
```

```
## Warning in matrix(c(1, 2, 3, 2, 6, 4, 4, 5, 6), 2, 3): data length [9] is
## not a sub-multiple or multiple of the number of rows [2]
```

```
BxAprime <- B %*% Aprime
BxAprime
```

```
##      [,1] [,2]
## [1,]   55   53
## [2,]   38   38
```

In ordinary algebra, 1 is the identity element, whereby any number multiplied by 1 returns that number:

$$5 \times 1 = 5$$

Similarly, there exists an identity element in matrix algebra, denoted by the symbol `I`. The identity matrix is a square matrix with a 1 in the same pattern, regardless of size:

$$I_{1\times1} = \begin{bmatrix}1\end{bmatrix}, I_{2\times2} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}, I_{3\times3} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

The following demonstrates the identity property for matrices:

```
I <- matrix(c(1, 0, 0, 0, 1, 0, 0, 0, 1),3, 3)
AI <- A %*% I
AI
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    6
```

```
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    6
```

Lastly, the reciprocal of A is known as the inverse matrix, denoted as A^{-1}. In ordinary algebra, the product of a number and its reciprocal is 1. In matrix algebra, the product of a matrix and its inverse is the identity matrix. $AA^{-1} = A^{-1}A = I$ **Note:** Inverse matrices only exist for square matrices, and not all square matrices possess inverses. The inverse of a matrix can be found via the `solve()` function as follows:

```
A.inv <- solve(AxAprime)
A.inv
```

```
##            [,1]       [,2]
## [1,]  0.3350515 -0.3298969
## [2,] -0.3298969  0.3402062
```

The following R example demonstrates `I` as the product of AA^{-1}:

```
Aident <- A.inv %*%  A
Aident
```

```
##            [,1]       [,2]       [,3]
## [1,] -0.3247423 -0.3092784  0.3659794
## [2,]  0.3505155  0.3814433 -0.2680412
```

9.1.2 Representing System of Linear Equations as Matrices

The previous lab introduced the estimated bivariate linear regression model as follows:

$$\hat{y}_i = \hat{\alpha} + \hat{\beta}x_i$$

Where $\hat{\alpha}$ anf $\hat{\beta}$ could be solved via the following formulas:

$$\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$$

$$\hat{\beta} = \frac{cov(x, y)}{var(x)}$$

In this lab we demonstrate how to use matrix algebra to calculate the least-squared estimates. For bivariate linear regression, the above formulas and matrix algebra will produce the same result; however, for multivariable linear regression the underlying methods that develop the above formulas become computationally complex such that matrix algebra is the preferred method to calculate the coefficients.

Using bivariate regression we explored hypotheses related to how preference for renewable energy is a function of ideology. Sometimes the dependent variable (`y`) is not easily explainable via a single independent variable (`x`), but rather multiple independent variables ($x_0, x_1, ..., x_n$). We can modify the bivariate regression model to append the additional variables of interest, as follows:

Note: The α intercept coefficient is replaced with β_0.

$$y_i = \beta_0 + \sum_{j=1}^{n} \beta_j x_{ij} + \varepsilon_i$$

Where, n is the number of independent variables, β_0 is the regression intercept, and β_j is the slope for the j^{th} variable. The above model is the general form of a regression model, and as such, will work for bivariate and multivariable models. If $n = 1$, the model is exactly the same as the model stated in the textbook and previous lab.

Using this model we can imagine collected data as a system of linear equations. To demonstrate, suppose we collected the following data:

```
ex.ds <- data.frame(x = c(1, 2, 3, 4, 5),
                    y = c(1, 1, 2, 2, 4))
ex.ds
```

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

Using the linear regression model above, we can state the data as a system of linear equations:

$$1 = \beta_0 + \beta_1 * 1$$

$$1 = \beta_0 + \beta_1 * 2$$

$$2 = \beta_0 + \beta_1 * 3$$

$$2 = \beta_0 + \beta_1 * 4$$

$$4 = \beta_0 + \beta_1 * 5$$

Given we are working with two variables, `x` and `y`, our `n` is 1, so we are working with the bivariate linear regression model. Now, we can use ordinary algebra to solve for β_0 and β_1. To do so, we will solve for one

variable, then solve for the other. Given our system consists of 5 linear equations, the ordinary algebra approach is not practical.

This is where matrix algebra is useful. The general regression model can be expressed in the following matrix form, where we capitalize variables to represent matrices:

$$Y = X\beta_1 + \varepsilon$$

The system of linear equations can be converted into the following matrices:

$$Y = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 4 \end{bmatrix}, X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix}, and \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

Note: The first column of the X matrix is always 1.

9.1.3 OLS Regression and Matrices

With data expressed in matrix form, we then use matrix algebra to calculate the least-squared estimates. The least-squared estimates formula is:

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

Using the matrices formed from the system of linear equations, we demonstrate calculating the least-squared estimates using R:

```
Y <- matrix(c(1, 1, 2, 2, 4), 5, 1)
Y
```

```
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    2
## [4,]    2
## [5,]    4
```

```
X <- matrix(c(1, 1, 1, 1, 1, 1, 2, 3, 4, 5), 5, 2)
X
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    2
## [3,]    1    3
## [4,]    1    4
## [5,]    1    5
```

Now we need to find X' with the `t()` function:

```
Xprime <- t(X)
Xprime
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    1    1    1    1
## [2,]    1    2    3    4    5
```

Next calculate (X'X):

```
XprimeX <- Xprime %*% X
XprimeX
```

```
##      [,1] [,2]
## [1,]    5   15
## [2,]   15   55
```

Now find the inverse of (X'X):

```
XprimeXinv <- solve(XprimeX)
XprimeXinv
```

```
##      [,1] [,2]
## [1,]  1.1 -0.3
## [2,] -0.3  0.1
```

Multiply $(X'X)^{-1}$ by X':

```
XprimeXinvXprime <- XprimeXinv %*% Xprime
XprimeXinvXprime
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  0.8  0.5  0.2 -0.1 -0.4
## [2,] -0.2 -0.1  0.0  0.1  0.2
```

Now multiply by Y to find $\hat{\beta}$:

```
b <- XprimeXinvXprime %*% Y
b
```

```
##      [,1]
## [1,] -0.1
## [2,]  0.7
```

We read this matrix as β_0 is the first position and β_1 is the following position. **Note:** We could just as easily used the method introduced in the last lab. The following R code demonstrates the equivalence:

```
df <- data.frame(x = c(1, 2, 3, 4, 5), y = c(1, 1, 2, 2, 4))
covar <- cov(df$x, df$y)
vari <- var(df$x)
bhat <- covar / vari
xbar <- mean(df$x)
ybar <- mean(df$y)
alpha <- ybar - bhat * xbar

alpha
```

```
## [1] -0.1
```

```
bhat
```

```
## [1] 0.7
```

Note: In the previous code block, `alpha` is β_0 and `bhat` is β_1.

With the coefficients calculated, our estimated linear regression model is:

$$\hat{y} = -0.10 + 0.70x$$

Let's check our work using the `lm()` function:

```
ols <- lm(Y ~ 0 + X)
ols
```

```
##
## Call:
## lm(formula = Y ~ 0 + X)
##
## Coefficients:
##   X1    X2
## -0.1   0.7
```

```
b
```

```
##      [,1]
## [1,] -0.1
## [2,]  0.7
```

The previous example demonstrated calculating least-squared estimates for a bivariate regression model. Next is an example using matrix algebra to calculate the least-squared estimates for a multivariable linear regression model.

Suppose we have the following data set:

```
mv.df <- data.frame(y = c(1, 1, 2, 2, 4),
                    x1 = c(1, 2, 3, 4, 5),
                    x2 = c(1, 2, 2, 4, 3))
mv.df
```

```
##   y x1 x2
## 1 1  1  1
## 2 1  2  2
## 3 2  3  2
## 4 2  4  4
## 5 4  5  3
```

The system of linear equations is:

$1 = \beta_0 + \beta_1 * 1 + \beta_2 * 1$ $1 = \beta_0 + \beta_1 * 2 + \beta_2 * 2$ $2 = \beta_0 + \beta_1 * 3 + \beta_2 * 2$ $2 = \beta_0 + \beta_1 * 4 + \beta_2 * 4$ $4 = \beta_0 + \beta_1 * 5 + \beta_2 * 3$

Converted to matrix form:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 3 & 2 \\ 1 & 4 & 4 \\ 1 & 5 & 3 \end{bmatrix} \times \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 4 \end{bmatrix}$$

We will use R to find the least-squared coefficients. Note that calculating Bhat in R has been reduced to a single line:

```
Y <- matrix(c(1, 2, 2, 4, 4), 5, 1)
X <- matrix(c(1, 1, 1, 1, 1, 1, 2, 3, 4, 5, 1, 2, 2, 4, 3), 5, 3)
Bhat <- (solve((t(X) %*% X))) %*% (t(X) %*% Y)
Bhat
```

```
##        [,1]
## [1,] -0.175
## [2,]  0.425
```

```
## [3,]  0.625
```

Again, we check our work using the `lm()` function:

```
ols <- lm(Y ~ 0 + X)
ols
```

```
##
## Call:
## lm(formula = Y ~ 0 + X)
##
## Coefficients:
##     X1      X2      X3
## -0.175   0.425   0.625
```

9.2 Multiple Regression in R

The R syntax for multiple linear regression is similar to what we used for bivariate regression: add the independent variables to the `lm()` function. Construct a model that looks at climate change certainty as the dependent variable with age and ideology as the independent variables:

```
sub.ds <- ds %>%
  dplyr::select("glbcc_cert", "ideol", "age") %>%
  na.omit()
model1 <- lm(sub.ds$glbcc_cert ~ sub.ds$ideol + sub.ds$age)
```

Now look at the model:

```
summary(model1)
```

```
##
## Call:
## lm(formula = sub.ds$glbcc_cert ~ sub.ds$ideol + sub.ds$age)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.1037 -1.5439  0.3458  1.9603  4.3913
##
## Coefficients:
##               Estimate Std. Error t value            Pr(>|t|)
## (Intercept)   8.650915   0.258926  33.411 <0.0000000000000002 ***
## sub.ds$ideol -0.392264   0.030372 -12.915 <0.0000000000000002 ***
## sub.ds$age   -0.003368   0.003705  -0.909               0.363
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.629 on 2515 degrees of freedom
## Multiple R-squared:  0.06372,    Adjusted R-squared:  0.06297
## F-statistic: 85.58 on 2 and 2515 DF,  p-value: < 0.00000000000000022
```

Before interpreting these results, we need to review partial effects. Chapter 12 of the textbook discusses partial effects in great detail. Essentially, multivariable regression "controls" for the effects of other dependent variables when reporting the effect of one particular variable. This is not accidental. Explore this for our model. First construct a bivariate regression model for each of independent variable in the multivariable regression model:

```
model2 <- lm(sub.ds$glbcc_cert ~ sub.ds$ideol)
model3 <- lm(sub.ds$glbcc_cert ~ sub.ds$age)
stargazer(model2, model3, single.row = TRUE, type = "text")
```

```
##
## ===============================================================
##                                   Dependent variable:
##                               ----------------------------------
##                                         glbcc_cert
##                                   (1)              (2)
## ---------------------------------------------------------------
## ideol                          -0.395*** (0.030)
## age                                                -0.008** (0.004)
## Constant                       8.459*** (0.150)  7.087*** (0.236)
## ---------------------------------------------------------------
## Observations                        2,518            2,518
## R2                                  0.063            0.002
## Adjusted R2                         0.063            0.001
## Residual Std. Error (df = 2516)     2.629            2.714
## F Statistic (df = 1; 2516)        170.341***         4.077**
## ===============================================================
## Note:                                *p<0.1; **p<0.05; ***p<0.01
```

Notice the ideology coefficient in the bivariate model is larger than the ideology coefficient in the multivariable regression model. The same is also true for the age variable. This is because the bivariate model reports the total effects of X on Y, but the multivariable regression model reports the effects of X on Y when "controlling" for the other independent variables. Look at the multivariable regression model again:

```
summary(model1)
```

```
##
## Call:
## lm(formula = sub.ds$glbcc_cert ~ sub.ds$ideol + sub.ds$age)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.1037 -1.5439  0.3458  1.9603  4.3913
##
## Coefficients:
##               Estimate Std. Error t value            Pr(>|t|)
## (Intercept)   8.650915   0.258926  33.411 <0.0000000000000002 ***
## sub.ds$ideol -0.392264   0.030372 -12.915 <0.0000000000000002 ***
## sub.ds$age   -0.003368   0.003705  -0.909               0.363
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.629 on 2515 degrees of freedom
## Multiple R-squared:  0.06372,    Adjusted R-squared:  0.06297
## F-statistic: 85.58 on 2 and 2515 DF,  p-value: < 0.00000000000000022
```

To interpret these results: a one unit increase in ideology corresponds with a -0.392 unit decrease in climate change certainty with all other variables held constant at their means. Further, given the p-value $< \alpha = 0.05$, the change in climate change certainty is statistically significant. Further, assessing the p-value of the age coefficient yields that the partial effect of age on climate change certainty is not statistically significant. You may have noticed a difference in the p-value for the age variable's coefficient between the bivariate and

multivariable regression models. This is due to the reduction in error in the model by adding the ideology variable. This is a good example of why, in most cases, multivariable regression provides a clearer picture of relationships. Only looking at age and climate change risk, we could potentially conclude that there is a statistically significant relationship; however, when appending ideology to model, we find that ideology is the more likely cause of change in climate change certainty.

9.3 Hypothesis Testing with Multivariable Regression

Perhaps we want to explore the relationship between opinions about fossil fuels and ideology. The class data set includes a question asking respondents what percentage of Oklahoma's electricity should come from fossil fuels. It is reasonable to posit that more conservative individuals will want a higher percentage of the state's electricity to come from fossil fuels. For this test, we include other independent variables: age, income, and education. We establish a hypothesis that the more conservative a respondent is, the more electricity they want to come from fossil fuels, all other variables held constant. The null hypothesis is that there is no difference in preferred percentage of electricity coming from fossil fuels by ideology. First we look at our variables:

```
str(ds$okelec_foss)
```

```
##  Factor w/ 63 levels "0","10","100",..: 51 28 44 3 28 13 35 51 57 16 ...
```

Notice that R reads the fossil fuels variable as a factor. We change this to a numeric variable. We coerce it and create a new variable:

```
ds$foss <- as.numeric(ds$okelec_foss)
```

Now let's look at all the variables. First we create a subset of the data and remove missing observations, then use the `skim()` function:

```
ds.sub <- ds %>%
  dplyr::select("income", "education", "ideol", "foss", "age") %>%
  na.omit()

ds.sub %>%
  skim()
```

```
## Skim summary statistics
##  n obs: 2283
##  n variables: 5
##
## -- Variable type:integer ---------------------------------------------------------------------
##   variable missing complete    n  mean    sd p0  p25 p50 p75 p100
##        age       0     2283 2283 60.11 14.08 18 51.5  62  70   99
##  education       0     2283 2283  5.09  1.81  1  4     6   6    8
##      ideol       0     2283 2283  4.64  1.74  1  4     5   6    7
##
## -- Variable type:numeric ---------------------------------------------------------------------
##  variable missing complete    n     mean       sd    p0   p25   p50   p75
##      foss       0     2283 2283    31.48    16.19     1    16    35    44
##    income       0     2283 2283 70622.59 59882.75 10000 35000 59000 90000
##    p100
##      63
##  900000
```

Now construct the model:

```
model <- lm(foss ~ income + education + age + ideol, data = ds.sub)
```

Note the different syntax in constructing this model. Instead of writing `dataset$variable` every time, you can write the variable names, and at the end of the model include `data=dataset`.

Now examine the model:

```
summary(model)
```

```
##
## Call:
## lm(formula = foss ~ income + education + age + ideol, data = ds.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -41.917 -10.845   1.464  11.665  39.861
##
## Coefficients:
##                 Estimate   Std. Error t value             Pr(>|t|)
## (Intercept)  5.828128065  1.895228915   3.075              0.00213 **
## income       0.000014020  0.000005537   2.532              0.01140 *
## education   -0.190966563  0.183425325  -1.041              0.29793
## age          0.188847307  0.022485238   8.399 < 0.0000000000000002 ***
## ideol        3.075532222  0.183457245  16.764 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.95 on 2278 degrees of freedom
## Multiple R-squared:  0.1491, Adjusted R-squared:  0.1476
## F-statistic: 99.81 on 4 and 2278 DF,  p-value: < 0.00000000000000022
```

To interpret these results, we start with the intercept. In a regression model, the intercept is the expected mean of our dependent variable when our independent variables are 0. In this case the expected mean is 5.83. In some models this has meaning; however, given none of our independent variables adopt a zero value, this provides minimal value to interpretation. Now examine the variable that the hypothesis is concerned with: ideology. We see there is a statistically significant coefficient of 3.07. We interpret this by saying that a one unit increase in ideology (from liberal to conservative) corresponds with a 3.07 unit increase in preferred percentage of electricity coming from fossil fuels, all other variables held constant. There are also statistically significant relationships for age and income, suggesting an increase in those correspond with an increase in the dependent variable as well. The adjusted R squared value of .15 suggests that our model accounts for 15 percent of the variability in the dependent variable.

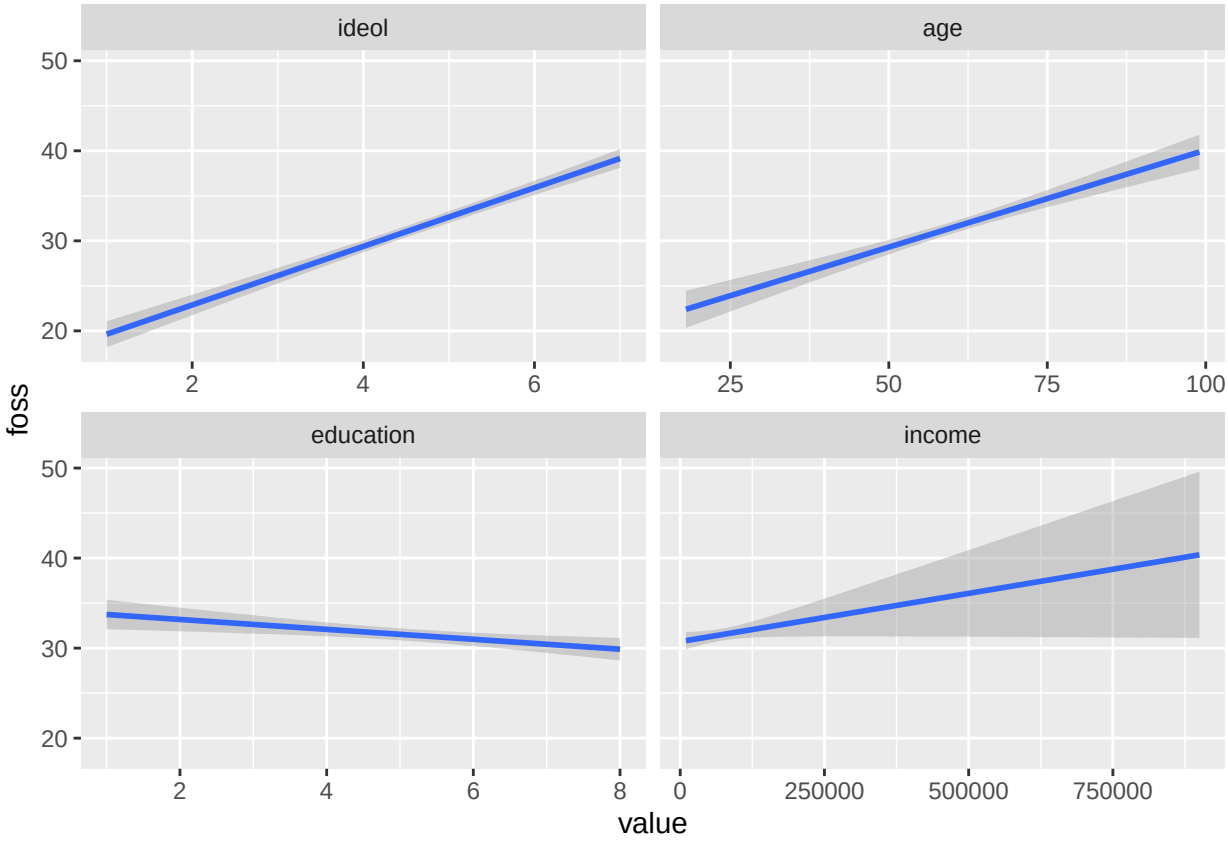
Next we need to visualize the model. Visualizing multiple linear regression is not as simple as visualizing bivariate relationships. There is no intuitive way for us to visualize, in one graphic, how all of these variables look while all others are held constant simultaneously. For a relationship with one dependent variable and two independent variables, we can make a three dimensional scatter plot and regression plane; however, these still don't provide a very intuitive visualization.

9.3.1 Visualizing Multivariable Linear Regression

The best way to visualize multiple linear regression is to create a visualization for each independent variable while holding the other independent variables constant. Doing this allows us to see how each relationship between the DV and IV looks. Constructing a quick and dirty visualization for each IV in `ggplot2` is simiular to the methods used for bivariate linear regression. We will go into greater detail in the next section when we cover predictions with OLS, but making a quick visualization is rather simple. The `augment()` function from

the `broom` package is very useful for this. The function transforms the data created from an OLS model into a tidyverse data frame format. Use `augment()`, then melt the data into long form, and create a `ggplot2` visualization for each IV by using `facet_wrap()`.

```
model %>%
  augment() %>%
  melt(measure.vars = c("ideol", "age", "education", "income"), variable.name = c("IV")) %>%
  ggplot(., aes(value, foss)) +
  geom_smooth(method = "lm") +
  facet_wrap(~IV, scales = "free_x")
```



Now we can see the general relationship between each independent variable and the dependent variable. The `geom_smooth()` function defaults to showing 95% confidence intervals. You can disable the confidence intervals with the `se=FALSE` argument, but that is not recommended. Notice the very large confidence interval in the income visualization, especially at the higher income levels. This happens because there are fewer observations with very high incomes. A smaller sample size leads to a larger standard error, and in turn larger confidence intervals.

9.4 Predicting with OLS Regression

Regression analysis is performed not only to explain relationships, but also to predict. In R, we can use the model we built to predict values of the dependent variable based on the values of the independent variables. Doing so is rather simple. Recall that our model attempts to explain how much of the state's electricity a respondent thinks should come from fossil fuels as a function of their ideology, age, education, and income.

To start predicting, we need to identify what values we want to assign to our independent variables. Perhaps you wanted to know the preferred percentage of energy coming from fossil fuels for an individual with a

bachelor's degree, an income of 45000, 40 years old, and a moderate (3) ideology. First we need to know the beta coefficients for each variable:

```
coef(model)
```

```
##    (Intercept)         income      education            age          ideol
##  5.82812806450  0.00001401976 -0.19096656321  0.18884730710  3.07553222165
```

Now recall the scalar formula for multiple linear regression:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_3 x_3$$

Therefore for our model, the formula would be

$$\hat{y} = \beta_{intercept} + \beta_{income} + \beta_{educ} + \beta_{age} + \beta_{ideol}$$

Note: For a bachelor's degree, the value of the education variable is 6.

```
(5.83 + (.000014 * (45000)) + (-.19 * (6)) + (.19 * (40)) + (3.07 * (3)))
```

```
## [1] 22.13
```

Based on the calculation, a predicted result is 22% of the state's electricity should come from fossil fuels. There is a more precise way to do this calculation, as is often the case. Using the `augment()` function from the `broom` package, we can tell R to return predicted values based on specifications of variable values.

```
model %>%
  augment(newdata = data.frame(ideol = 3, income = 45000, education = 6, age = 40))
```

```
## # A tibble: 1 x 6
##   ideol income education   age .fitted .se.fit
##   <dbl>  <dbl>     <dbl> <dbl>   <dbl>   <dbl>
## 1     3  45000         6    40    22.1   0.648
```

Note: The original estimate is a little off due to rounding. Using `augment()` returns a data frame in tidy format. The .fitted value is the predicted value of interest. You can use the `predict()` function in a similar way, but doing so returns information in vector, not data frame, format.

The `augment()` function can return multiple values at a time. We can tell R to predict values for a sequence of values, like the range of ideology values. Sequencing one variable while holding the others constant is very common in OLS analysis. The method we will most often use will be holding all other IVs constant at their means.

```
model %>%
  augment(newdata = data.frame(ideol = 1:7, income = mean(ds.sub$income),
                               education = mean(ds.sub$education),
                               age = mean(ds.sub$age)))
```

```
## # A tibble: 7 x 6
##   ideol income education   age .fitted .se.fit
##   <int>  <dbl>     <dbl> <dbl>   <dbl>   <dbl>
## 1     1 70623.      5.09  60.1    20.3   0.738
## 2     2 70623.      5.09  60.1    23.3   0.577
## 3     3 70623.      5.09  60.1    26.4   0.435
## 4     4 70623.      5.09  60.1    29.5   0.334
## 5     5 70623.      5.09  60.1    32.6   0.320
## 6     6 70623.      5.09  60.1    35.7   0.400
## 7     7 70623.      5.09  60.1    38.7   0.534
```

Recall that the earlier hypothesis stated that the more conservative a respondent, the more electricity they will prefer comes from fossil fuels. We can safely reject the null hypothesis, given the clear relationship and overall evidence that there is a positive relationship. To conlcude, let's build a solid, paper-worthy visualization of the relationship between ideology and opinions on fossil fuels from our model.

Note:Until now, we have used `geom_smooth()` to create regression lines. However, the superior method of creating regression lines is to generate predictions outside of ggplot and use `geom_line()` to plot the line and `geom_ribbom()` to plot the confidence intervals. This is because `geom_smooth()` does not let you set the specific IV values. However, the `augment()` (or `predict()`!) function tells R to predict values of the DV based on specific values of the IV. To create a good regression line and confidence interval using the `augment()` function, instruct R to hold all DVs at their means except ideology. Sequence ideology from 1 to 7, and include `se.fit=TRUE`, then assign the fitted values to an object:

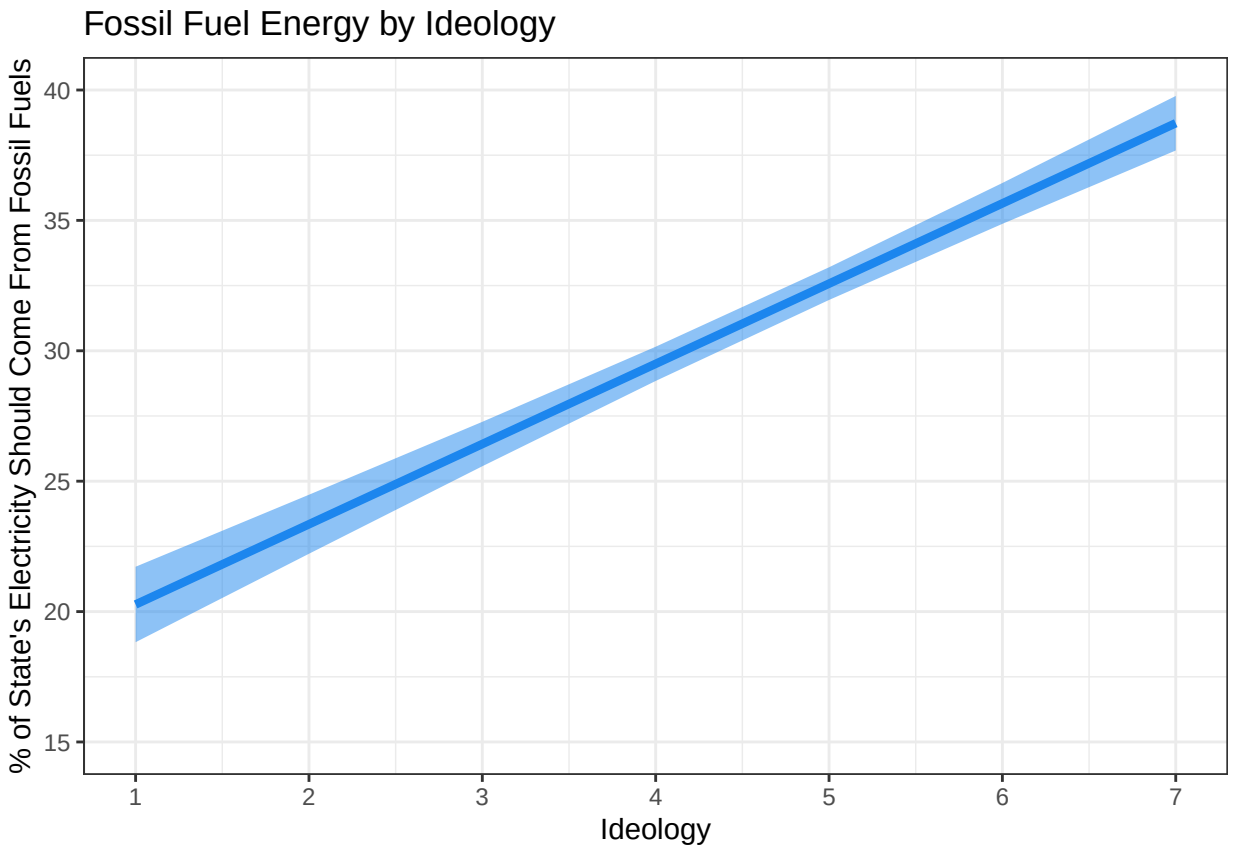
The next step is to calculate the confidence interval. This is rather simple! Using the `augment()` function in tandem with the `mutate()` function, we can create a tidy data frame that provides us with all the information needed to construct a visualization with a confidence interval. Assign the data frame to an object called `fit.df`.

```
model %>%
  augment(newdata = data.frame(ideol = 1:7, income = mean(ds.sub$income),
                               education = mean(ds.sub$education),
                               age = mean(ds.sub$age))) %>%
  mutate(upper = .fitted + 1.96 * .se.fit,
         lower = .fitted - 1.96 * .se.fit) -> fit.df
fit.df
```

```
## # A tibble: 7 x 8
##   ideol income education   age .fitted .se.fit upper lower
##   <int>  <dbl>     <dbl> <dbl>   <dbl>   <dbl> <dbl> <dbl>
## 1     1 70623.      5.09  60.1    20.3   0.738  21.7  18.8
## 2     2 70623.      5.09  60.1    23.3   0.577  24.5  22.2
## 3     3 70623.      5.09  60.1    26.4   0.435  27.3  25.6
## 4     4 70623.      5.09  60.1    29.5   0.334  30.2  28.8
## 5     5 70623.      5.09  60.1    32.6   0.320  33.2  31.9
## 6     6 70623.      5.09  60.1    35.7   0.400  36.4  34.9
## 7     7 70623.      5.09  60.1    38.7   0.534  39.8  37.7
```

Now build the visualization:

```
ggplot(fit.df, aes(ideol, .fitted)) +
  geom_line(size=1.5, color = "dodgerblue2") +
  geom_ribbon(aes(ymax = upper, ymin = lower), alpha = .5, fill = "dodgerblue2") +
  ggtitle("Fossil Fuel Energy by Ideology") +
  ylab("% of State's Electricity Should Come From Fossil Fuels") +
  xlab("Ideology") +
  scale_x_continuous(breaks=c(1:7), labels = c("1", "2", "3", "4", "5", "6", "7")) +
  coord_cartesian(ylim = c(15, 40), xlim = c(1, 7)) +
  theme_bw()
```



10 Categorical Explanatory Variables, Dummy Variables, and Interactions

This lab focuses on ways in which we use and understand categorical independent variables. So far the independent variables we have worked with have been interval or ordinal data. When working with categorical data, there are different approaches and techniques of interpretation. The following packages are required for this lab:

1. tidyverse
2. psych
3. stargazer
4. interplot
5. car
6. reshape2
7. broom

10.1 Dummy Variables

We often have situations in the social sciences that require constructing models to include qualitative variables. To facilitate this, we employ dichotomous dummy variables to make the model function via 0s and 1s. When using dichotomous dummy variables for catergorical data, the presense of the category of interest receives a value of 1 and in its absence the value is 0.

To demonstrate dummy variables in models we will look to the class data set. The gender variable is coded as a 0 for women and 1 for men. This makes it a dummy variable for men, with women as the referent group. If

we wanted to construct a model that looked at how certainty of climate change varied by ideology, education, income, age, and gender, our model would look like this:

$$Y_i = \alpha + \beta_{ideol} + \beta_{educ} + \beta_{inc} + \beta_{age} + \beta_{gend} + \epsilon_i$$

Where `B_gend` is a binary indicator of gender, 0 for female and 1 for male. This means that when gender is female, gender equals 0.

Pull the data, omit missing variables, and look at the gender variable we are going to use:

```
ds.sub <- ds %>% dplyr::select("ideol", "education", "income",
                                        "age", "gender", "f.gender",
                                        "glbcc_cert", "f.party", "glbcc_risk",
                                        "glbwrm_risk_fed_mgmt") %>%
  na.omit()
```

```
table(ds.sub$f.gender)
```

```
##
## Women   Men
##  1268   905
```

Note: The factored gender variable lists men as 0 and women as 1. If you look at a table of the non-factored version, it shows the opposite. This is because R reads factored variables in alphabetical order. If we wanted to change the order of the factored variable:

```
ds.sub$f.gender <- factor(ds.sub$gender, levels = c(0, 1), labels = c("Women", "Men"))
ds.sub %>%
  count(f.gender, gender)
```

```
## # A tibble: 2 x 3
##   f.gender gender     n
##   <fct>     <int> <int>
## 1 Women         0  1268
## 2 Men           1   905
```

When working with a binary categorical explanatory variable (like the gender variable), you can use the numeric version of the variable. However, when working with categorical variables with more than two categories, it is often easier to use the factored version of the variable, for reasons we will discuss shortly. We will use the factored gender variable in our model:

```
lm1 <- lm(glbcc_cert ~ ideol + education + income + age + f.gender, data = ds.sub)
summary(lm1)
```

```
##
## Call:
## lm(formula = glbcc_cert ~ ideol + education + income + age +
##     f.gender, data = ds.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.0644 -1.5134  0.3312  1.9218  4.9935
##
## Coefficients:
##                  Estimate    Std. Error t value             Pr(>|t|)
## (Intercept)  7.7467825284  0.3338336414  23.206 < 0.0000000000000002 ***
## ideol       -0.4053788823  0.0324305925 -12.500 < 0.0000000000000002 ***
## education    0.1205104649  0.0324417726   3.715             0.000209 ***
```

```
## income       0.0000007186  0.0000009753   0.737             0.461338
## age         -0.0006379876  0.0039721277  -0.161             0.872411
## f.genderMen  0.4082557132  0.1133996694   3.600             0.000325 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.569 on 2167 degrees of freedom
## Multiple R-squared:  0.08614,    Adjusted R-squared:  0.08403
## F-statistic: 40.85 on 5 and 2167 DF,  p-value: < 0.00000000000000022
```

Relying on our understanding from previous labs, we know that ideology and education have an effect on someone's certainty of climate change; however, now we want to look at the role gender plays. We used the factored version of gender in the model, so we need to interpret the results as such. The summary table says "f.genderMen", which means the variable is a dummy variable for men, with the referent category being women. The coefficient is interpreted as: the difference in the dependent variable from the referent category to the dummy category. In this case, the coefficient is statistically significant. To interpret it, we say that men are on average .408 units more convinced of climate change, on a scale of 0 to 10, all else held constant. The rest of the coefficients are interpreted as they have been in the past. But now you likely have a more accurate model, since you are "controlling" for gender.

Visualizing the model should likely this more clear. If we want to visualize the relationship between ideology and climate change risk in our model **by gender**, we go about it in a similar way to previous visualizations:

1. Generate fitted values and standard errors for each ideology level and gender via the `augment()` function.
2. Use the `full_join()` function to join the data frames for men and women together.
3. Calculate the upper and lower bounds of the confidence interval using `mutate()`
4. Visualize

```
lm1 %>%
  augment(newdata = data.frame(f.gender = "Women",
                               ideol = 1:7,
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income),
                               age = mean(ds.sub$age))) -> fit.w
lm1 %>%
  augment(newdata = data.frame(f.gender = "Men",
                               ideol = 1:7,
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income),
                               age = mean(ds.sub$age))) -> fit.m
```

Now join them:

```
fit.df <- full_join(fit.w, fit.m)
```

```
## Joining, by = c("f.gender", "ideol", "education", "income", "age", ".fitted", ".se.fit")
```

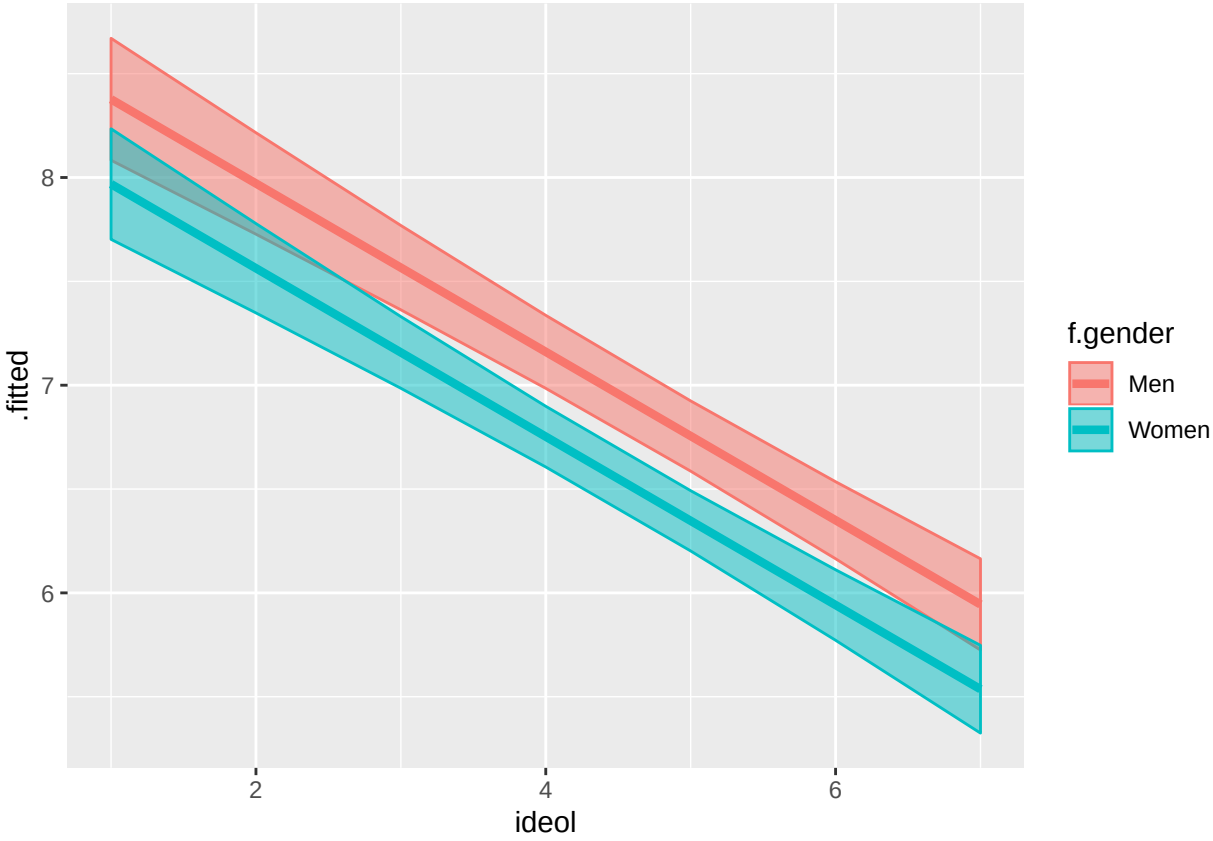
```
## Warning: Column `f.gender` joining factors with different levels, coercing
## to character vector
```

Now add confidence intervals:

```
fit.df <- fit.df %>%
  mutate(up = .fitted + 1.96 * .se.fit,
         low = .fitted - 1.96 * .se.fit)
```

Now visualize! To separate men and women, you can use `group=DummyVariable`, to separate the two groups. Unfortunately, this does not give a way to distinguish between them. You can also use `color=DummyVariable` to separate the groups, assign colors, and include a legend.

```
ggplot(fit.df, aes(ideol, .fitted, color = f.gender)) +
  geom_line(size=1.5) +
  geom_ribbon(aes(ymin = low, ymax = up, fill = f.gender), alpha = .5)
```



You can think of the effect of dummy variables as a change in the value of the intercept. In this case our dummy variable for men is about 0.41, and you will notice that the line for men looks about that much above the women line.

10.1.1 Multiple Dummy Variables

Sometimes multiple dummy variables are necessary in models. This is the case when you need to include categorical variables with greater than two options, such as ideology (e.g., Republican, Democrat, Independent, Other). When working with these categorical variables, you need to select a referent group. Sometimes this decision is driven by the theory and or by convenience. R will automatically select a referent group if nothing is supplied. When using categorical variables with multiple options, the model will consist of multiple dummy variables for each of the groups (minus the referent group). You will always have one less dummy variable than the number of options. For example, for Republican, Democrat, Independent, and Other as the options, with Republican as the referent group, you will have 3 dummy variables.

Let's look at an example using political party as a dummy variable. Start by looking at a table of the factored party variable:

```
table(ds.sub$f.party)
```

```
##
##   Dem   Rep   Ind Other
##   746  1042   308    77
```

Note: Democrat is listed first, therefore it is the referent category. Therefore, in a model, there would exist coefficients and dummy variables for each of the political parties sans Democrat. In other words, R reads ideology as a factored variable and treats every party option as an independent dummy variable with Democrats as the referent category. Let's create a model based on the model we used earlier, but include the factored party variable as an independent variable. Due to potential multicollinearity issues, we will omit the ideology variable from the model. To make calculations simpler, we're going to use the non-factored version of gender. Since its a binary group, this will not change any of the coefficients:

```
lm2 <- lm(glbcc_cert ~ f.party  +education + income + age + gender, data = ds.sub)
summary(lm2)
```

```
##
## Call:
## lm(formula = glbcc_cert ~ f.party + education + income + age +
##     gender, data = ds.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.9870 -1.5441  0.3358  2.0151  4.6757
##
## Coefficients:
##                    Estimate    Std. Error t value             Pr(>|t|)
## (Intercept)   6.7403467930  0.3216406712  20.956 < 0.0000000000000002 ***
## f.partyRep   -1.3639756955  0.1250310842 -10.909 < 0.0000000000000002 ***
## f.partyInd   -0.7178788385  0.1778603443  -4.036           0.00005621 ***
## f.partyOther -0.2752990638  0.3116689227  -0.883             0.377169
## education     0.1583974332  0.0323714456   4.893           0.00000107 ***
## income        0.0000005345  0.0000009842   0.543             0.587139
## age          -0.0052742660  0.0040513327  -1.302             0.193103
## gender        0.3818814312  0.1144371993   3.337             0.000861 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.589 on 2165 degrees of freedom
## Multiple R-squared:  0.07255,    Adjusted R-squared:  0.06955
## F-statistic:  24.2 on 7 and 2165 DF,  p-value: < 0.00000000000000022
```

We can see our model suggests that Independents and Republicans are, on average, less certain about climate change. The coefficient for Other is not significant, which makes sense given Other could indicate a panoply of political parties spanning the ideological spectrum.

Now we will visualize this model. Dummy variables are similar to performing t-tests, but with statistical controls. First we predict values based on party affiliation using the `augment()` function for R to return predicted climate change certainty values based on political party, along with associated standard errors, holding each other variable constant. Assign the newly created object to a data frame and print the data frame:

```
lm2 %>%
  augment(newdata = data.frame(f.party = c("Dem", "Ind", "Other", "Rep"),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income),
                               age = mean(ds.sub$age),
                               gender = mean(ds.sub$gender))) -> fit2.df
fit2.df
```

```
## # A tibble: 4 x 7
##   f.party education income   age gender .fitted .se.fit
```

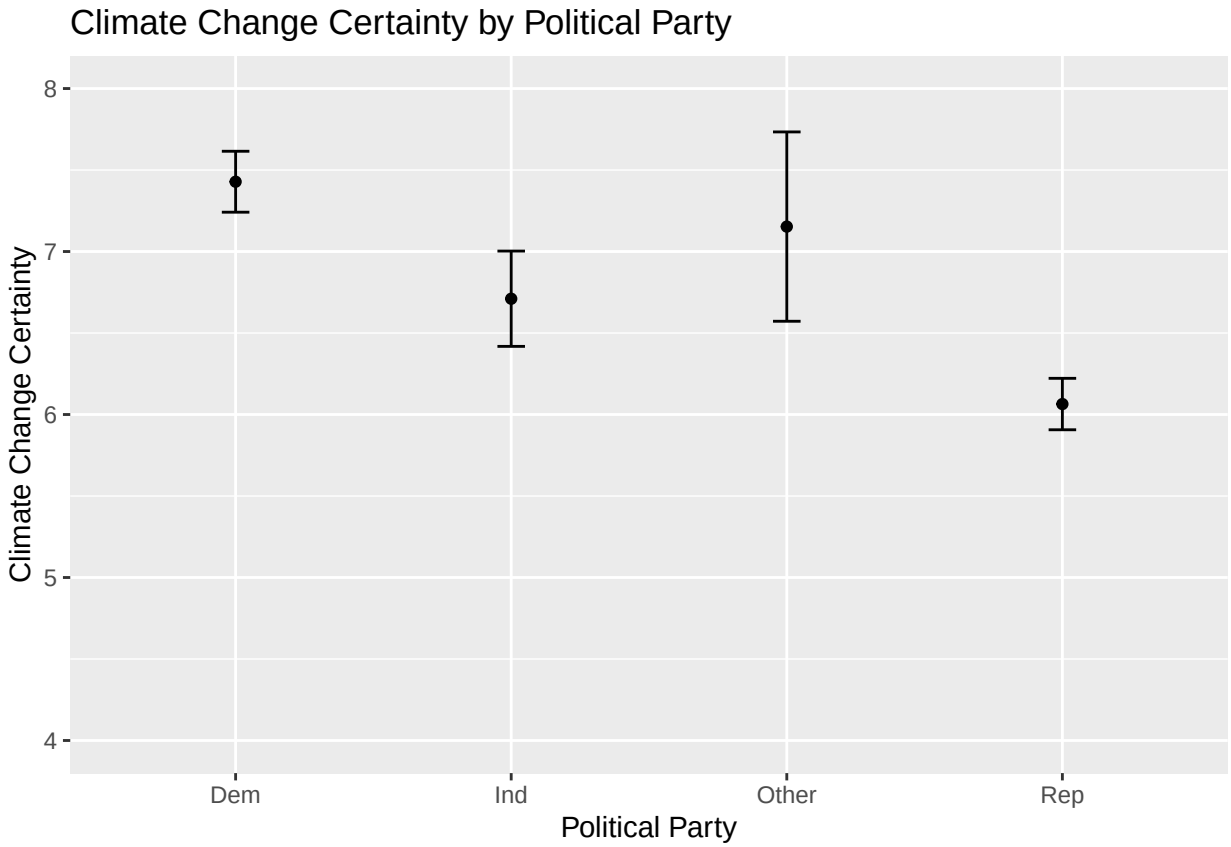
```
##   <fct>       <dbl>  <dbl> <dbl>  <dbl>   <dbl>   <dbl>
## 1 Dem          5.10 71036.  60.0  0.416    7.43  0.0954
## 2 Ind          5.10 71036.  60.0  0.416    6.71  0.149
## 3 Other        5.10 71036.  60.0  0.416    7.15  0.296
## 4 Rep          5.10 71036.  60.0  0.416    6.06  0.0805
```

We will also calculate confidence intervals using the `mutate()` function. Remember, a t score of 1.96 is associated with 95% confidence intervals:

```
fit2.df %>%
   mutate(upper = .fitted + 1.96 * .se.fit,
          lower = .fitted - 1.96 * .se.fit) -> fit2.df
```

With the data frame constructed, next build the visualization. Our x-axis is party and the y-axis is climate change certainty. We'll use `geom_point()` and `geom_errorbar()` to build the point estimates and confidence intervals:

```
ggplot(fit2.df, aes(x = f.party, y = .fitted)) +
  geom_point() +
  geom_errorbar(aes(ymin = lower, ymax = upper), width = 0.1) +
  ylim(4, 8) +
  ggtitle("Climate Change Certainty by Political Party") +
  ylab("Climate Change Certainty") +
  xlab("Political Party")
```



Perhaps you noticed that our model suggests that gender also plays a role. Next we are going to breakdown the relationship by party and gender simultaneously, by creating different predictions for each gender within each party. We create a new model including the factored gender variable:

```
lm3 <- lm(glbcc_cert ~ f.party  + education
          + income + age + f.gender, data = ds.sub)
summary(lm3)
```

```
##
## Call:
## lm(formula = glbcc_cert ~ f.party + education + income + age +
##     f.gender, data = ds.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.9870 -1.5441  0.3358  2.0151  4.6757
##
## Coefficients:
##                    Estimate    Std. Error t value             Pr(>|t|)
## (Intercept)   6.7403467930  0.3216406712  20.956 < 0.0000000000000002 ***
## f.partyRep   -1.3639756955  0.1250310842 -10.909 < 0.0000000000000002 ***
## f.partyInd   -0.7178788385  0.1778603443  -4.036           0.00005621 ***
## f.partyOther -0.2752990638  0.3116689227  -0.883             0.377169
## education     0.1583974332  0.0323714456   4.893           0.00000107 ***
## income        0.0000005345  0.0000009842   0.543             0.587139
## age          -0.0052742660  0.0040513327  -1.302             0.193103
## f.genderMen   0.3818814312  0.1144371993   3.337             0.000861 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.589 on 2165 degrees of freedom
## Multiple R-squared:  0.07255,    Adjusted R-squared:  0.06955
## F-statistic:  24.2 on 7 and 2165 DF,  p-value: < 0.00000000000000022
```

The difference between the models is that the factored gender variable is used, which does not change any of the results. Now we follow a similar process in constructing the graphic, except we predict different values for men and women, and build data frames separately before combining them. Use `augment()` twice, once for men and once for women, then use `full_join()` to combine them into one data frame. From there we can calculate the confidence intervals just like we always do:

```
lm3 %>%
  augment(newdata = data.frame(f.party = c("Dem", "Ind", "Other", "Rep"),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income),
                               age = mean(ds.sub$age),
                               f.gender = "Men")) -> fit3.m
lm3 %>%
  augment(newdata = data.frame(f.party = c("Dem", "Ind", "Other", "Rep"),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income),
                               age = mean(ds.sub$age),
                               f.gender = "Women")) -> fit3.w

fit3.df <- full_join(fit3.m, fit3.w)
```

```
## Joining, by = c("f.party", "education", "income", "age", "f.gender", ".fitted", ".se.fit")
```

```
## Warning: Column `f.gender` joining factors with different levels, coercing
## to character vector
```

Now the confidence intervals:

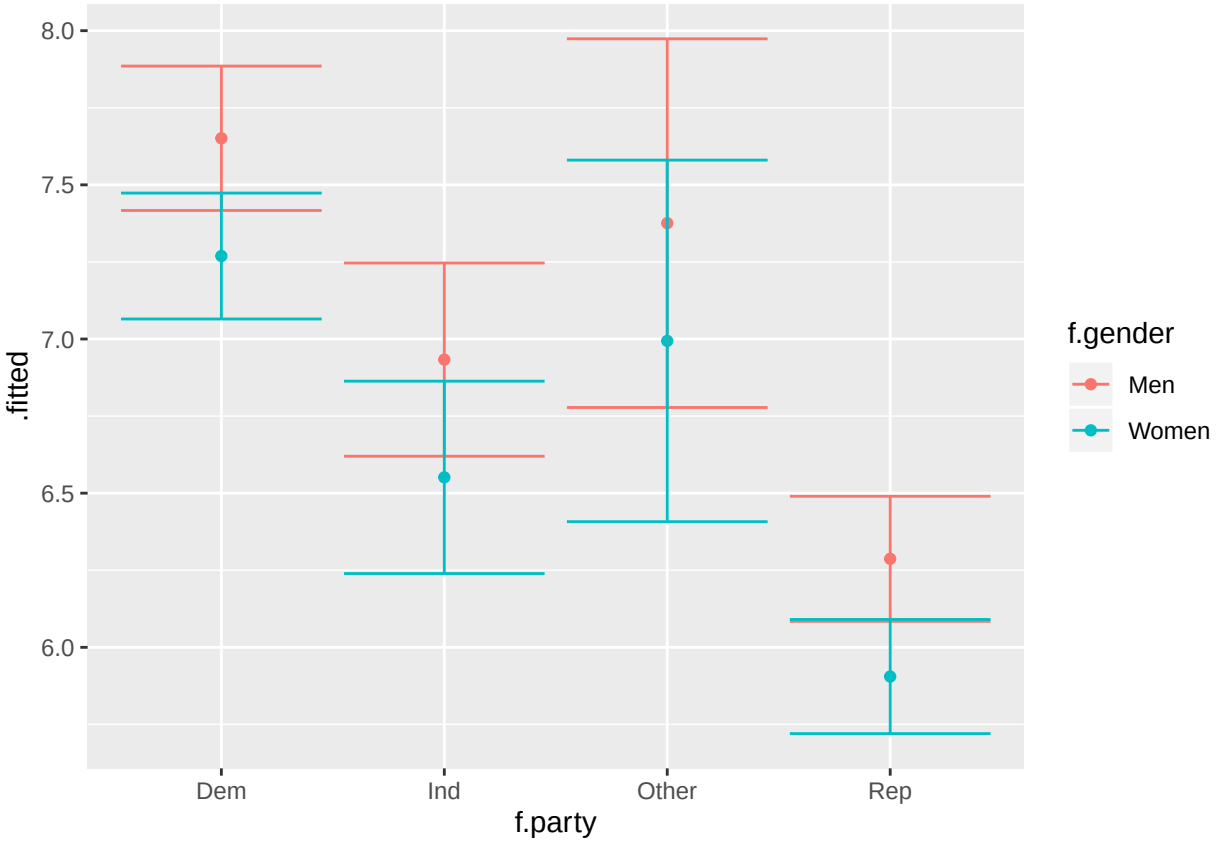
```
fit3.df %>%
  mutate(up = .fitted + 1.96 * .se.fit,
         low = .fitted - 1.96 * .se.fit) -> fit3.df
fit3.df
```

```
## # A tibble: 8 x 9
##   f.party education income   age f.gender .fitted .se.fit    up   low
##   <fct>       <dbl>  <dbl> <dbl> <chr>      <dbl>   <dbl> <dbl> <dbl>
## 1 Dem          5.10 71036.  60.0 Men         7.65  0.119   7.89  7.42
## 2 Ind          5.10 71036.  60.0 Men         6.93  0.160   7.25  6.62
## 3 Other        5.10 71036.  60.0 Men         7.38  0.305   7.97  6.78
## 4 Rep          5.10 71036.  60.0 Men         6.29  0.103   6.49  6.08
## 5 Dem          5.10 71036.  60.0 Women       7.27  0.104   7.47  7.06
## 6 Ind          5.10 71036.  60.0 Women       6.55  0.159   6.86  6.24
## 7 Other        5.10 71036.  60.0 Women       6.99  0.299   7.58  6.41
## 8 Rep          5.10 71036.  60.0 Women       5.91  0.0945  6.09  5.72
```

Build the visualization. Creating a grouped bar plot will allow us to see each party broken down by gender. To create a grouped bar plot, include `position = position_dodge()` in the `geom_bar()` and `geom_errorbar()` functions.

```
ggplot(fit3.df, aes(f.party, .fitted, color = f.gender)) +
  geom_point(stat = "identity", position = position_dodge())  +
  geom_errorbar(aes(ymin = low, ymax = up))
```

```
## Warning: Width not defined. Set with `position_dodge(width = ?)`
```



It appears that there might be a difference in certainty of climate change between the political parties by

gender. If we were to test how political beliefs vary as a function of gender and are related to opinions about climate change certainty, we would need to explore interaction terms.

10.2 Interactions

Interactions occur when the effect of one x is dependent on the value of another within a model. Previously, the value at any point of x was the same across all levels of another in predicting y. To demonstrate an interaction effect we will explore the interaction of gender and ideology on climate change certainty. We include the other predictors and specify this model:

$$y_i = \beta_0 + \beta_1 * (ideol) + \beta_2 * (gender) + \beta_3 * (ideol * gend) + \beta_4 * (educ) + \beta_5 * (inc) + \beta_6 * (age) + \varepsilon_i$$

where gender is a binary indicator of men (1) or women (0). To specify this model in R:

```
lm4 <- lm(glbcc_cert ~ ideol * gender + education + income + age , data = ds.sub)
summary(lm4)
```

```
##
## Call:
## lm(formula = glbcc_cert ~ ideol * gender + education + income +
##     age, data = ds.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.9951 -1.5174  0.3393  1.9300  5.0661
##
## Coefficients:
##                    Estimate    Std. Error t value             Pr(>|t|)
## (Intercept)   7.8894159475  0.3564206870  22.135 < 0.0000000000000002 ***
## ideol        -0.4365023001  0.0423636086 -10.304 < 0.0000000000000002 ***
## gender        0.0634842747  0.3225564070   0.197             0.843990
## education     0.1197405909  0.0324465076   3.690             0.000229 ***
## income        0.0000006970  0.0000009754   0.715             0.474940
## age          -0.0005748755  0.0039722341  -0.145             0.884943
## ideol:gender  0.0737647297  0.0646069235   1.142             0.253686
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.569 on 2166 degrees of freedom
## Multiple R-squared:  0.08669,    Adjusted R-squared:  0.08416
## F-statistic: 34.26 on 6 and 2166 DF,  p-value: < 0.00000000000000022
```

Note: The formula includes an ideology and gender interaction but does not specify the variables individually. R interprets the interaction and includes the separate variable terms for you. To interpret the results, notice that the `ideol:gender` interaction coefficient is not statistically significant.

Let's review a new model looking at climate change risk instead of certainty. The independent variables, and interaction, remain the same:

```
lm5 <- lm(glbcc_risk ~ ideol * f.gender + education + income + age , data = ds.sub)
summary(lm5)
```

```
##
## Call:
```

```
## lm(formula = glbcc_risk ~ ideol * f.gender + education + income +
##     age, data = ds.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.7769 -1.7061  0.1519  1.4476  6.9480
##
## Coefficients:
##                         Estimate    Std. Error t value             Pr(>|t|)
## (Intercept)       10.5560045389  0.3360457021  31.412 < 0.0000000000000002
## ideol             -0.9650733858  0.0399418696 -24.162 < 0.0000000000000002
## f.genderMen        0.5460952632  0.3041172923   1.796              0.07269
## education          0.0717394671  0.0305916851   2.345              0.01911
## income            -0.0000023174  0.0000009197  -2.520              0.01182
## age               -0.0029354676  0.0037451591  -0.784              0.43324
## ideol:f.genderMen -0.1677538955  0.0609136332  -2.754              0.00594
##
## (Intercept)       ***
## ideol             ***
## f.genderMen       .
## education         *
## income            *
## age
## ideol:f.genderMen **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.422 on 2166 degrees of freedom
## Multiple R-squared:  0.3687, Adjusted R-squared:  0.3669
## F-statistic: 210.8 on 6 and 2166 DF,  p-value: < 0.00000000000000022
```

As would be expected, ideology, education and income also exert statistically significant influence. Further, the interaction of ideology and gender is also statistically significant. To interpret these results, we would say that there is an interaction (ideology affects perceived climate change risk as a function of gender). We also know that the slope of the lines is negative. Often times the most intuitive way to understand interactions is to make predictions and visualize them.

Visualizing an interaction effect when the interaction term is binary is rather simple. There are two possible lines, when z=0 and when z=1, in this case when the gender is female or male. This makes the visualizing process similar to the first visualization, with the dummy variable:

```
lm5 %>%
  augment(newdata = data.frame(ideol = 1:7,
                               f.gender = "Men",
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income),
                               age = mean(ds.sub$age))) -> fit5.m

lm5 %>%
  augment(newdata = data.frame(ideol = 1:7,
                               f.gender = "Women",
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income),
                               age = mean(ds.sub$age))) -> fit5.w
```

```
full_join(fit5.m, fit5.w) %>%
  mutate(upper = .fitted + 1.96 * .se.fit,
         lower = .fitted - 1.96 * .se.fit) -> fit5.df
```

```
## Joining, by = c("ideol", "f.gender", "education", "income", "age", ".fitted", ".se.fit")
```

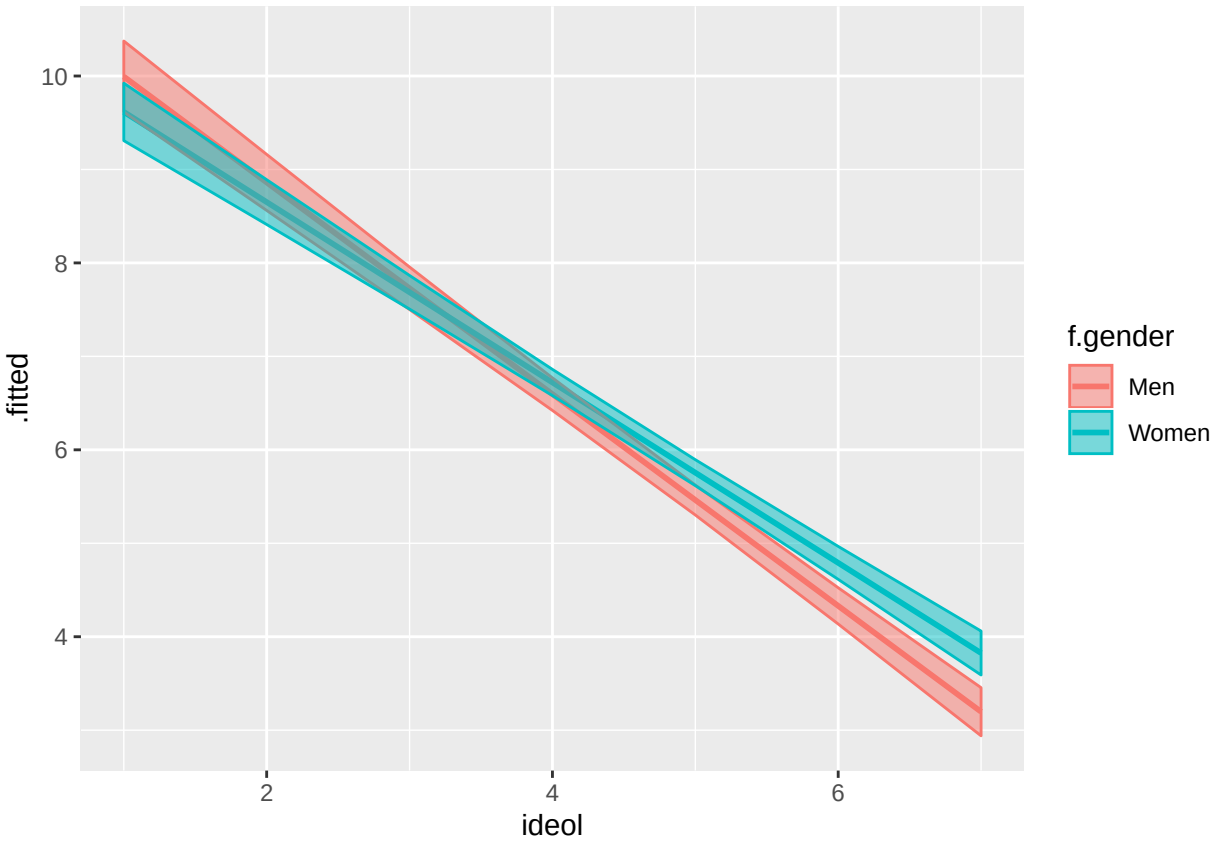
```
## Warning: Column `f.gender` joining factors with different levels, coercing
## to character vector
```

```
fit5.df
```

```
## # A tibble: 14 x 9
##    ideol f.gender education income   age .fitted .se.fit upper lower
##    <int> <chr>        <dbl>  <dbl> <dbl>   <dbl>   <dbl> <dbl> <dbl>
##  1     1 Men           5.10 71036.  60.0    9.99  0.194  10.4   9.61
##  2     2 Men           5.10 71036.  60.0    8.86  0.153   9.16  8.56
##  3     3 Men           5.10 71036.  60.0    7.73  0.116   7.96  7.50
##  4     4 Men           5.10 71036.  60.0    6.60  0.0889  6.77  6.42
##  5     5 Men           5.10 71036.  60.0    5.46  0.0817  5.62  5.30
##  6     6 Men           5.10 71036.  60.0    4.33  0.0990  4.52  4.14
##  7     7 Men           5.10 71036.  60.0    3.20  0.131   3.45  2.94
##  8     1 Women         5.10 71036.  60.0    9.62  0.157   9.92  9.31
##  9     2 Women         5.10 71036.  60.0    8.65  0.122   8.89  8.41
## 10     3 Women         5.10 71036.  60.0    7.69  0.0918  7.87  7.51
## 11     4 Women         5.10 71036.  60.0    6.72  0.0716  6.86  6.58
## 12     5 Women         5.10 71036.  60.0    5.76  0.0708  5.89  5.62
## 13     6 Women         5.10 71036.  60.0    4.79  0.0900  4.97  4.61
## 14     7 Women         5.10 71036.  60.0    3.83  0.120   4.06  3.59
```

Now the visualization:

```
ggplot(fit5.df, aes(ideol, .fitted, color = f.gender)) +
  geom_line(size = 1) +
  geom_ribbon(aes(ymin = lower, ymax = upper, fill = f.gender), alpha = .5)
```



Notice how the slopes are different for men and women. The slope is steeper for men, suggesting that there is more of an interaction for men.

The difference between the first predicted value and the last is called the "first difference." Find the first differences for men and women:

```
fit5.m$.fitted[1] - fit5.m$.fitted[7]
```

```
## [1] 6.796964
```

```
fit5.w$.fitted[1] - fit5.w$.fitted[7]
```

```
## [1] 5.79044
```

We can tell that the first difference is larger for men. They have a higher first value and a lower last value.

10.2.1 Interactions with Two Non-binary Variables

Theory and hypotheses often dictate the need to include an interaction between two variables when neither are binary. This makes the interpretation of interaction coefficients difficult, but nonetheless the process is still the same. Suppose you want to explore people's attitudes about the role of federal government in climate change management. We can theorize that two primary predictors of these attitudes are ideology and climate change risk. Conservatives tend to oppose federal government intervention and someone more concerned about climate change should likely support the attitudes about the role of federal government and perceived risk of climate change will be different among liberals and conservatives. We could further theorize that the relationship between federal climate change management and climate change risk will be positive regardless of group, with individuals perceiving greater risk from climate change supporting more government management, but that relationship will be weaker for conservatives. We will specify the following hypothesis:

The relationship between perceived climate change risk and support for federal government management of climate change will be positive, but conditional on ideology. The relationship will be more pronounced for liberals and less pronounced for conservatives.

First take a look at the federal climate change management variable:

```
describe(ds.sub$glbwrm_risk_fed_mgmt)
```

```
##    vars    n mean   sd median trimmed  mad min max range  skew kurtosis
## X1    1 2173 5.86 3.49      6    6.07 4.45   0  10    10 -0.33    -1.23
##      se
## X1 0.07
```

We see that it is an ordinal variable ranging from 0 (not involved) to 10 (very involved).

Now we should specify the model, including appropriate controls:

```
lm6 <- lm(glbwrm_risk_fed_mgmt ~ ideol * glbcc_risk + education + gender + income
          + age, data = ds.sub)
summary(lm6)
```

```
##
## Call:
## lm(formula = glbwrm_risk_fed_mgmt ~ ideol * glbcc_risk + education +
##     gender + income + age, data = ds.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.1434 -0.8689  0.0685  0.8649  9.4746
##
## Coefficients:
##                       Estimate     Std. Error t value
## (Intercept)       4.11750754060  0.43761946604   9.409
## ideol            -0.57049022374  0.06717613178  -8.492
## glbcc_risk        0.68141095439  0.04716364727  14.448
## education        -0.05192566006  0.02368444974  -2.192
## gender           -0.20829016774  0.08263826809  -2.521
## income            0.00000009089  0.00000071014   0.128
## age              -0.00025067243  0.00289196846  -0.087
## ideol:glbcc_risk  0.02685643594  0.00848796975   3.164
##                             Pr(>|t|)
## (Intercept)      < 0.0000000000000002 ***
## ideol            < 0.0000000000000002 ***
## glbcc_risk       < 0.0000000000000002 ***
## education                    0.02846 *
## gender                       0.01179 *
## income                       0.89817
## age                          0.93093
## ideol:glbcc_risk             0.00158 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.868 on 2165 degrees of freedom
## Multiple R-squared:  0.714,  Adjusted R-squared:  0.7131
## F-statistic: 772.2 on 7 and 2165 DF,  p-value: < 0.00000000000000022
```

Right from the start we see that ideology and climate change risk both play significant roles. These coefficients

are both statistically significant and substantive. A one unit change in either of the variables corresponds with more than a half point change in opinions about federal climate change management. Education and income also play roles, and notice that the interaction is significant. There is not a lot of intuitive interpretation we can gather from the coefficient alone; however, we see that it is very small, .026, and will likely not alter the slopes much. The best way to understand an interaction of two non-binary variables is to make predictions and visualize. We start with predictions.

First predict values of y for liberals:

```
lm6 %>%
  augment(newdata = data.frame(ideol = (1), gender = mean(ds.sub$gender),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income), age = mean(ds.sub$age),
                               glbcc_risk = seq(1, 10, 1))) -> lib
```

Now conservatives:

```
lm6 %>%
  augment(newdata = data.frame(ideol = (7), gender = mean(ds.sub$gender),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income), age = mean(ds.sub$age),
                               glbcc_risk = seq(1, 10, 1))) -> con
```

Now compare, starting with liberals:

Liberals:

```
lib$.fitted
```

```
##  [1]  3.895254  4.603522  5.311789  6.020056  6.728324  7.436591  8.144859
##  [8]  8.853126  9.561393 10.269661
```

Conservatives:

```
con$.fitted
```

```
##  [1] 0.6334515 1.5028575 2.3722635 3.2416695 4.1110755 4.9804815 5.8498875
##  [8] 6.7192935 7.5886995 8.4581056
```

Find the first difference of each:

Liberals:

```
lib$.fitted[10] - lib$.fitted[1]
```

```
## [1] 6.374407
```

Conservatives:

```
con$.fitted[10] - con$.fitted[1]
```

```
## [1] 7.824654
```

There is a greater first difference for conservatives. Combined with a significant interaction coefficient that is positive, we can start to see that perhaps the slopes of the lines are steeper for conservatives. Let's build a visualization that includes a prediction line for every ideology level. To do so we need to:

1. Generate predictions for every ideology score.
2. Put those predictions in a data frame.
3. Visualize each individual line on a single plot.

It might look like a lot of code, but the process is rather simple!

Start with an ideology of 1 and then go to 7:

```
lm6 %>%
  augment(newdata = data.frame(ideol = 1, gender = mean(ds.sub$gender),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income), age = mean(ds.sub$age),
                               glbcc_risk = seq(1, 10, 1))) -> id1
lm6 %>%
  augment(newdata = data.frame(ideol = 2, gender = mean(ds.sub$gender),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income), age = mean(ds.sub$age),
                               glbcc_risk = seq(1, 10, 1))) -> id2
lm6 %>%
  augment(newdata = data.frame(ideol = 3, gender = mean(ds.sub$gender),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income), age = mean(ds.sub$age),
                               glbcc_risk = seq(1, 10, 1))) -> id3
lm6 %>%
  augment(newdata = data.frame(ideol = 4, gender = mean(ds.sub$gender),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income), age = mean(ds.sub$age),
                               glbcc_risk = seq(1, 10, 1))) -> id4
lm6 %>%
  augment(newdata = data.frame(ideol = 5, gender = mean(ds.sub$gender),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income), age = mean(ds.sub$age),
                               glbcc_risk = seq(1, 10, 1))) -> id5
lm6 %>%
  augment(newdata = data.frame(ideol = 6, gender = mean(ds.sub$gender),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income), age = mean(ds.sub$age),
                               glbcc_risk = seq(1, 10, 1))) -> id6
lm6 %>%
  augment(newdata = data.frame(ideol = 7, gender = mean(ds.sub$gender),
                               education = mean(ds.sub$education),
                               income = mean(ds.sub$income), age = mean(ds.sub$age),
                               glbcc_risk = seq(1, 10, 1))) -> id7
```

Now put all the data frames into one data frame. The `full_join()` function only merges two data sets at a time, so we are going to join them step-by-step. Recall that when piping one function to the next, a `.` can be used to stand in for the previous data frame, therefore we can build one data frame like this:

```
full_join(id1, id2) %>%
  full_join(., id3) %>%
  full_join(., id4) %>%
  full_join(., id5) %>%
  full_join(., id6) %>%
  full_join(., id7) -> fit6.df
```

```
## Joining, by = c("ideol", "gender", "education", "income", "age", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "gender", "education", "income", "age", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "gender", "education", "income", "age", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "gender", "education", "income", "age", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "gender", "education", "income", "age", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "gender", "education", "income", "age", "glbcc_risk", ".fitted", ".se.fit")
```

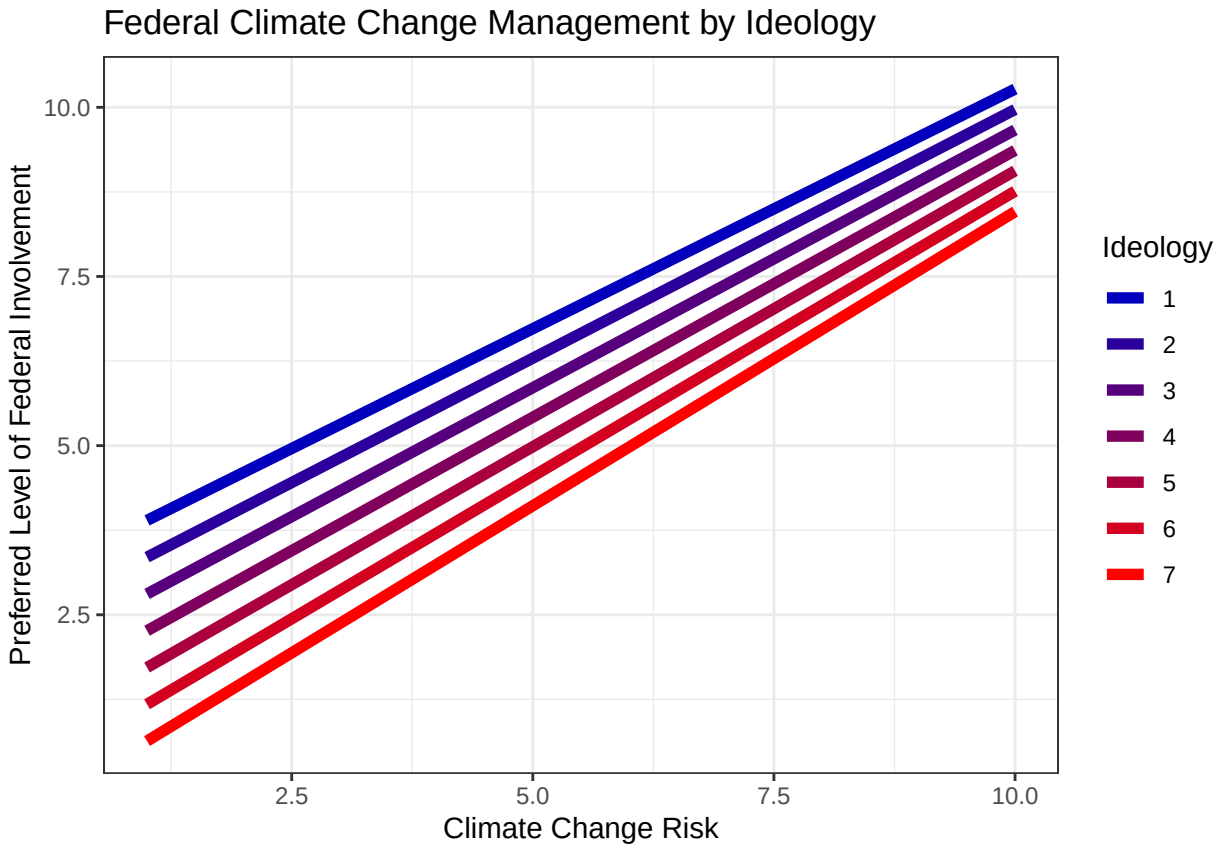
Next we are going to create a color scale. This will help us interpret the visualization because we are

visualizing multiple lines. Create a scale with 7 values, one for each ideology score, that goes from blue (liberal) to red (conservative):

```
col_scale<-colorRampPalette(c("#0200bd50","#FF000050"))(7)
```

Now build the visualization. Let's make this a complete visualization, including axis labels and a title. Because of the numeric nature of the ideology variable, `ggplot2` will try and read it as a numeric set of values. However, we are creating a different line for each level of ideology, so we need to instruct `ggplot2` to treat ideology as a factor. Use `as.factor()` to do so.

```
ggplot(fit6.df, aes(glbcc_risk, .fitted, color = as.factor(ideol))) +
  geom_line(size = 2) +
  scale_color_manual(values = c(col_scale[1], col_scale[2], col_scale[3],
                        col_scale[4], col_scale[5], col_scale[6], col_scale[7]),
                     labels = c("1", "2", "3", "4", "5", "6", "7"),
                     name = "Ideology") +
  ggtitle("Federal Climate Change Management by Ideology") +
  xlab("Climate Change Risk") +
  ylab("Preferred Level of Federal Involvement") +
  theme_bw()
```



Consider everything we've found so far. The positive interaction coefficient, the larger first difference for conservatives, and now this visualization. It is quite clear that the relationship between climate change risk and preferred level of federal climate change management appears to be stronger for conservatives. The slopes of the lines become steeper as the ideology score increases. This was not what we hypothesized, and therefore we cannot reject the null hypothesis.

10.3 Releveling Variables

As mentioned earlier, R can sometimes re-order a variable. This is the case for factored variables, when R automatically reads them alphabetically. Sometimes you need to relevel a variable by necessity and othertimes by preference. Let's look at the factored party variable:

```
table(ds$f.party)
```

```
##
##   Dem   Rep   Ind Other
##   869  1185   356    91
```

Notice the variable is in alphabetical order. If we included this variable in our model, R would read the Democrat category as the referent group. Perhaps you wanted Republicans to be the referent group. There are a couple ways to do this. The first way would be to refactor the numeric version of the variable. First look at the unfactored version:

```
table(ds$party)
```

```
##
##    1    2    3    4
##  869 1185  356   91
```

We compare this to the factored version and extrapolate that the 2 value indicates Republicans. So when factoring the variable we will list 2 first:

```
ds$f.party2 <- factor(ds$party, levels = c(2,1,3,4), labels = c("Rep", "Dem", "Ind", "Other"))
table(ds$f.party2)
```

```
##
##   Rep   Dem   Ind Other
##  1185   869   356    91
```

This is one way to go about it. This way has its merits, such as wanting to re-order every level of the variable, not just the referent group. But there is another way, one that changes the referent group one: using the `relevel()` function, and indicating the referent group with the `ref=` argument. Let's do so, and make Republicans the referent group. Make sure R reads the `f.party` variable as a factor!

```
ds$f.party3 <- relevel(as.factor(ds$f.party), ref = "Rep")
table(ds$f.party3)
```

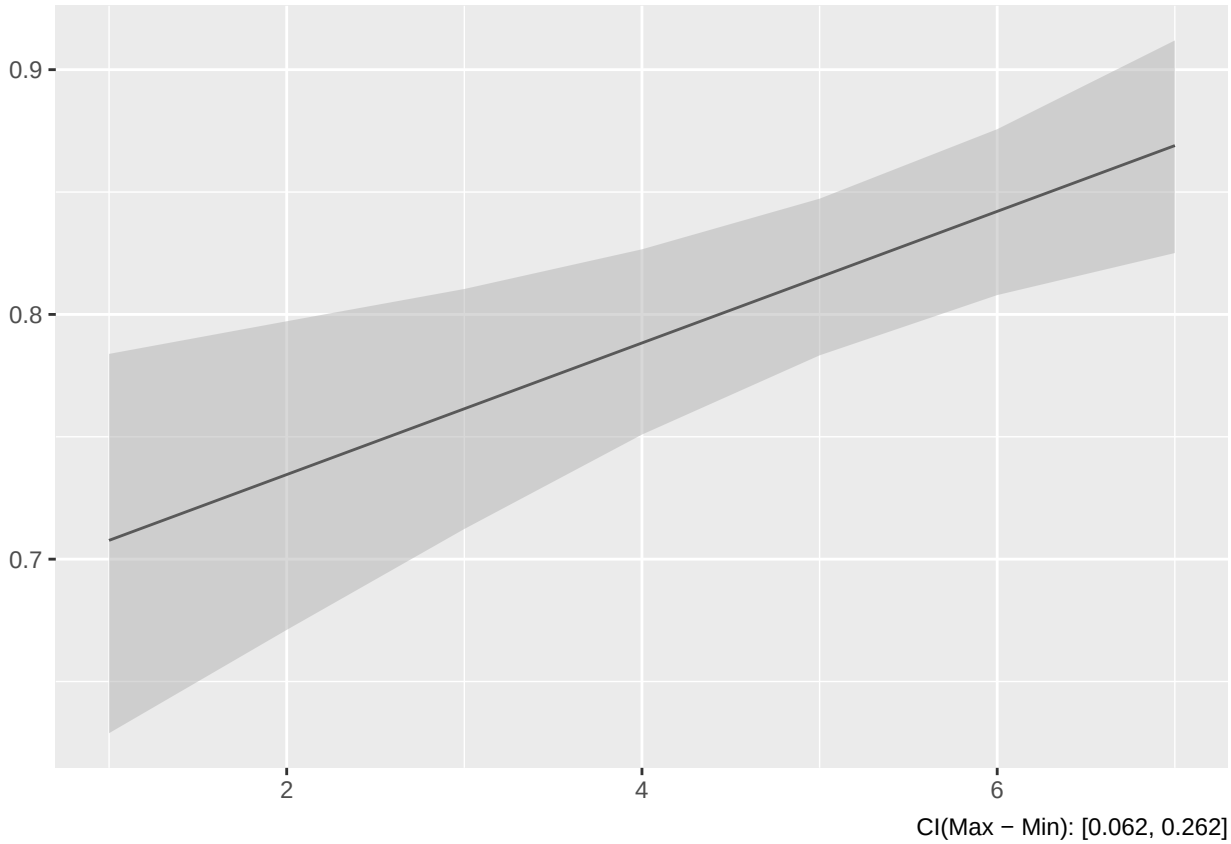
```
##
##   Rep   Dem   Ind Other
##  1185   869   356    91
```

10.4 Interaction Plots

Our exploration of interactions plotted estimated Y values as X varies as a function of Z. Specifically, we looked at the relationship between climate change management and climate change risk as a function of ideology. We plotted prediction lines for every level of ideology. However, there is another way we can explore interaction effects. Using the `interplot()` function, we calculate and visualize estimates of the coefficient of an independent variable in an interaction. Meaning, instead of looking at predicted values of a dependent variable, we are looking at the estimated effect of an independent variable on a dependent variable in a model that includes a two-way interaction.

To create an interaction plot for our earlier model we need to specify the two variables in the interaction. The first variable we specify is the variable of interest, the one for which we want estimated coefficients. The second is the other variable in the interaction.

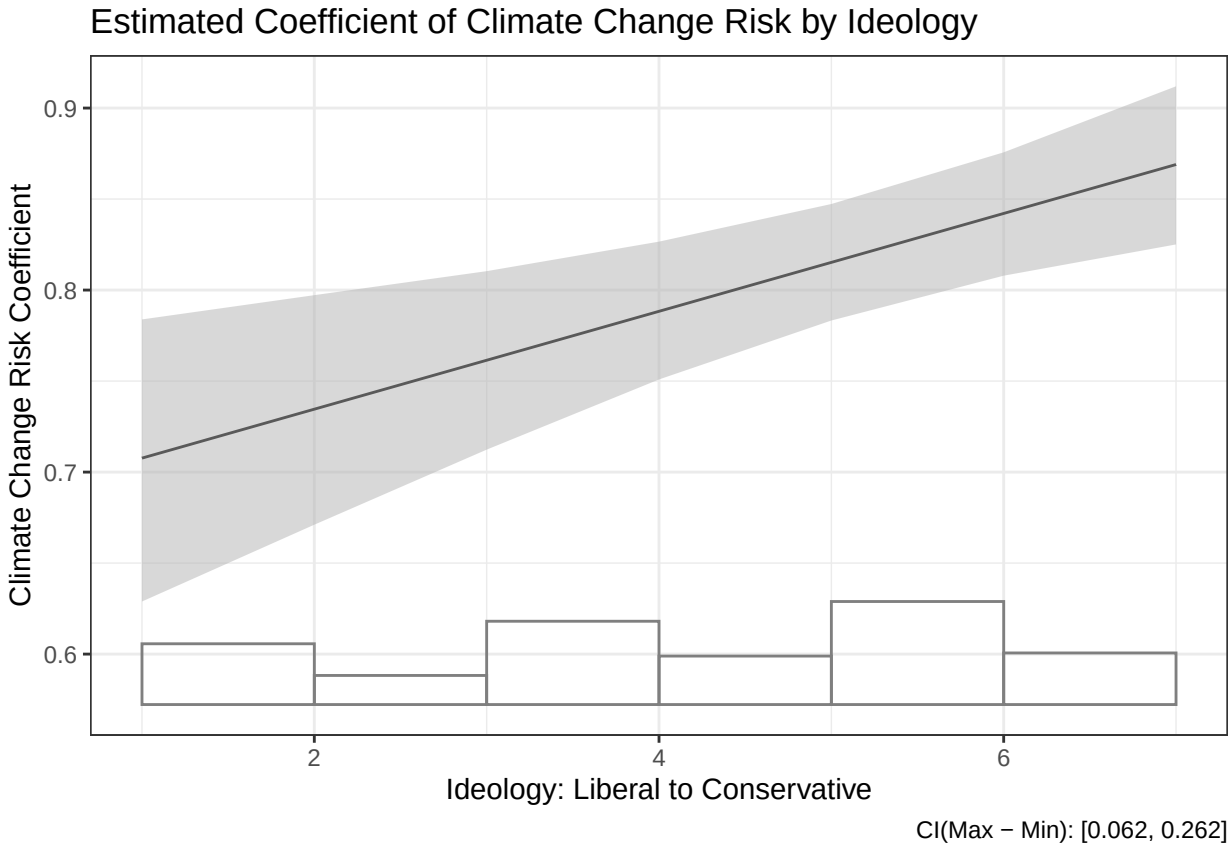
```
interplot(lm6, var1="glbcc_risk", var2 = "ideol")
```



This plot graphs the estimated coefficient of climate change risk by ideology score. The estimated effect of climate change risk on federal climate change management appears stronger for more conservative individuals.

Instead of plotting individual estimates, we could plot a line that also visualizes the relationship. To do this we would specify `hist=T`, which plots a histogram on the bottom and a line for the estimated coefficients. Let's add titles on this plot as well:

```
interplot(m=lm6, var1="glbcc_risk", var2="ideol", hist=T) +
  ggtitle("Estimated Coefficient of Climate Change Risk by Ideology") +
  theme_bw() +
  xlab("Ideology: Liberal to Conservative") +
  ylab("Climate Change Risk Coefficient")
```



11 Non-linearity, Non-normality, and Multicollinearity

This lab focuses on issues that arise with non-linearity, non-normality, and multicollinearity. We begin with non-linearity. The following packages are required for this lab:

1. tidyverse
2. psych
3. car
4. stargazer
5. reshape2
6. skimr
7. broom

11.1 Non-linearity

11.1.1 Exploring Non-linearity

A critical assumption of OLS is that the relationship between variables is linear in their functional form, therefore it is imperative to inspect whether a model consist of non-linear relationships between variables of interest. As the book describes, regression diagnostics is largely an art. To demonstrate, suppose you want to examine the relationship between ideology and candidate support in the 2016 presidential election. The survey associated with the class data set asked respondents to indicate on a scale of 1 to 5 the level of support they have for the candidate they voted for in the 2016 presidential election. We will start by exploring data in preparation for a linear regression model:

```
sub <- ds %>%
  dplyr::select("vote_cand_spt", "ideol","education",
                                    "income", "age", "gender", "f.gender",
                                    "glbcc_cert", "f.party", "glbcc_risk",
                                    "cncrn_econ") %>%
  na.omit()
```

Now examine the candidate support variable:

```
psych::describe(sub$vote_cand_spt)
```

```
##    vars    n mean   sd median trimmed  mad min max range  skew kurtosis
## X1    1 1980 3.56 1.15      4    3.65 1.48   1   5     4 -0.45    -0.52
##      se
## X1 0.03
```

Now a table:

```
table(sub$vote_cand_spt)
```

```
##
##   1   2   3   4   5
## 117 209 600 560 494
```

We can see that the modal value is 3, a response indicating moderate support for the candidate the respondent voted for. We also observe a negative skew indicating more responses in the higher values.

Next build the linear regression model:

```
model <- lm(vote_cand_spt ~ ideol + education + income + age + gender, data = sub)
summary(model)
```

```
##
## Call:
## lm(formula = vote_cand_spt ~ ideol + education + income + age +
##     gender, data = sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8895 -0.6401  0.2183  0.8652  1.8780
##
## Coefficients:
##                  Estimate    Std. Error t value             Pr(>|t|)
## (Intercept)  3.2245955848  0.1628686111  19.799 < 0.0000000000000002 ***
## ideol        0.0215255996  0.0148191732   1.453              0.14651
## education   -0.0462020061  0.0153075522  -3.018              0.00257 **
## income      -0.0000002871  0.0000004401  -0.652              0.51420
## age          0.0085719712  0.0018812530   4.557           0.00000552 ***
## gender      -0.0601777317  0.0525893860  -1.144              0.25264
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.134 on 1974 degrees of freedom
## Multiple R-squared:  0.02201,    Adjusted R-squared:  0.01953
## F-statistic: 8.884 on 5 and 1974 DF,  p-value: 0.00000002359
```

Based on the p-values for the model variables, the ideology variable does not appear to help us understand what candidate the responded supported; however, we should examine the linearity of the variables. One way

to do this is to plot the residuals by the values of the independent variables. Residual relationships should constitute a straight line with the residuals spread around a line at zero. To build this visualization we need to:

1. Use `augment()` to predict values and create a data frame containing fitted values and residuals
2. Melt the data into long form, sorted by independent variables.
3. Visualize the relationship between the residuals and IVs simultaneously using `facet_wrap()`.

First the residuals and fitted values. Use `head()` to look at the first five observations in the data frame:

```
model %>%
  augment() -> m.df
head(m.df)
```

```
## # A tibble: 6 x 14
##   .rownames vote_cand_spt ideol education income   age gender .fitted
##   <chr>             <int> <int>     <int>  <dbl> <int>  <int>   <dbl>
## 1 1                     5     6         4 160000    50      1    3.49
## 2 2                     2     3         4  45000    58      1    3.53
## 3 3                     4     6         6  40000    52      0    3.51
## 4 5                     4     4         4  55000    40      0    3.45
## 5 7                     3     2         6  40000    60      0    3.49
## 6 8                     4     4         6  50000    67      1    3.53
## # ... with 6 more variables: .se.fit <dbl>, .resid <dbl>, .hat <dbl>,
## #   .sigma <dbl>, .cooksd <dbl>, .std.resid <dbl>
```

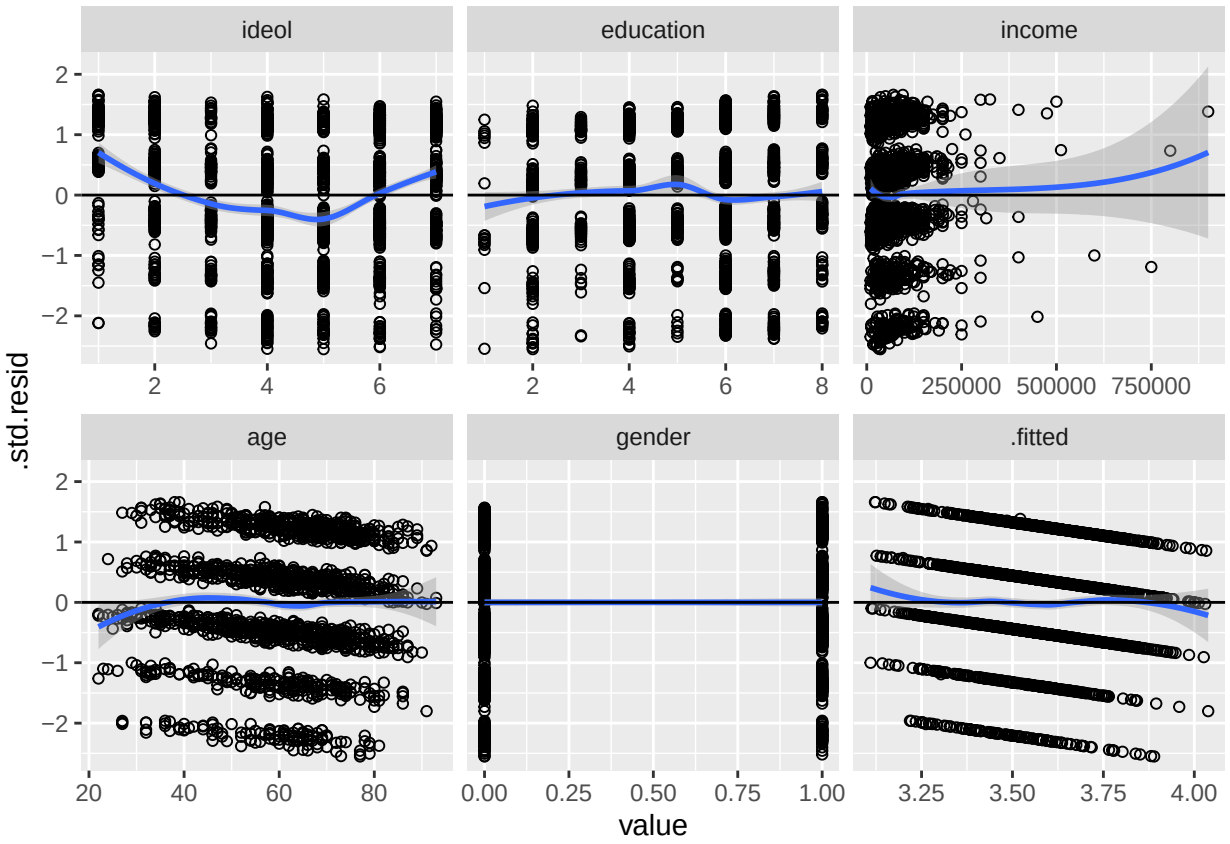
Now we need to melt the data into rows with unique id-variable combinations.

```
m.df %>%
melt(measure.vars = c("ideol", "education", "income", "age",
                                      "gender", ".fitted")) -> m.df
head(m.df)
```

```
##   .rownames vote_cand_spt    .se.fit     .resid        .hat   .sigma
## 1         1             5 0.06292522  1.5085812 0.003078242 1.133935
## 2         2             2 0.05491383 -1.5284394 0.002344319 1.133922
## 3         3             4 0.04930230  0.4892061 0.001889677 1.134391
## 4         5             4 0.05455346  0.5470242 0.002313651 1.134378
## 5         7             3 0.05215890 -0.4932672 0.002114999 1.134390
## 6         8             4 0.04636595  0.4667270 0.001671289 1.134396
##         .cooksd .std.resid variable value
## 1 0.00091331373  1.3321858    ideol     6
## 2 0.00071294152 -1.3492255    ideol     3
## 3 0.00005881882  0.4317469    ideol     6
## 4 0.00009012074  0.4828766    ideol     4
## 5 0.00006696004 -0.4353802    ideol     2
## 6 0.00004732953  0.4118630    ideol     4
```

The next step is to plot the residuals by the values of the independent variables. We're going to use `geom_smooth()` to create a loess line that approximates the spread of our data, and we will use `geom_hline()` to plot a horizontal line at 0. Then we will use `facet_wrap()` to tell R to create different plots for each independent variable of the model:

```
ggplot(m.df, aes(value, .std.resid)) +
  geom_point(shape = 1) +
  geom_smooth(aes(value, .std.resid), method = "loess") +
  geom_hline(yintercept = 0) +
  facet_wrap(~variable, scales = "free_x")
```



We can see there are some potential non-linear relationships; for instance, the ideology variable. The next step is to consider adding an exponent the ideology variable. **Note:** We want to avoid over fitting our model to the data. Therefore it is important to have an understanding of why you are including an exponent.

For example, when thinking about how ideology might influence candidate support, you could imagine this might be an instance when a quadratic model would be appropriate. Think about it: it wouldn't make sense to theorize that the more conservative an individual, the more enthusiastic they were for the candidate they voted for (regardless of who the candidate was), but it **does** make sense to theorize that the more ideologically extreme an individual is (very liberal or very conservative) the more they supported the candidate they voted for. Perhaps the moderates voting in the 2016 election felt left our or alienated by the polarized political environment, and therefore might have had less support for the candidate they voted for.

With this in mind, let's build a new model to include ideology as a squared term. **Note:** The syntax to use a polynomial is: `poly(var,# of powers)`. The `poly` function creates an independent variable for each of the powers as required to create orthogonal power terms. Let's construct the model:

```
pol <- lm(vote_cand_spt ~ poly(ideol,2) + education + income + age + gender, data = sub)
summary(pol)
```

```
##
## Call:
## lm(formula = vote_cand_spt ~ poly(ideol, 2) + education + income +
##     age + gender, data = sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2697 -0.6101  0.0023  0.7914  2.2046
##
## Coefficients:
```

```
##                       Estimate    Std. Error t value             Pr(>|t|)
## (Intercept)      3.4716343110  0.1404083490  24.725 < 0.0000000000000002
## poly(ideol, 2)1  1.6130128925  1.1096206250   1.454                0.146
## poly(ideol, 2)2 14.8590944999  1.0888870633  13.646 < 0.0000000000000002
## education       -0.0586338098  0.0146646212  -3.998            0.0000661
## income          -0.0000001029  0.0000004210  -0.245                0.807
## age              0.0071042635  0.0018019703   3.942            0.0000835
## gender          -0.0760264761  0.0502966844  -1.512                0.131
##
## (Intercept)     ***
## poly(ideol, 2)1
## poly(ideol, 2)2 ***
## education       ***
## income
## age             ***
## gender
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.084 on 1973 degrees of freedom
## Multiple R-squared:  0.1064, Adjusted R-squared:  0.1036
## F-statistic: 39.13 on 6 and 1973 DF,  p-value: < 0.00000000000000022
```

We see that our squared ideology term is statistically significant, but the coefficient may not provide us an intuitive interpretation. Before visualizing the model, it may be helpful to compare the new model to the previous, via the `anova()` function, to compare their fit to the data:

```
anova(model, pol)
```

```
## Analysis of Variance Table
##
## Model 1: vote_cand_spt ~ ideol + education + income + age + gender
## Model 2: vote_cand_spt ~ poly(ideol, 2) + education + income + age + gender
##   Res.Df    RSS Df Sum of Sq      F               Pr(>F)
## 1   1974 2539.2
## 2   1973 2320.2  1    218.99 186.22 < 0.00000000000000022 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The significant p-value for the second line implies the second model is a better fit. We can also compare the adjusted R^2 values:

```
stargazer(model, pol, single.row = TRUE, type = "text")
```

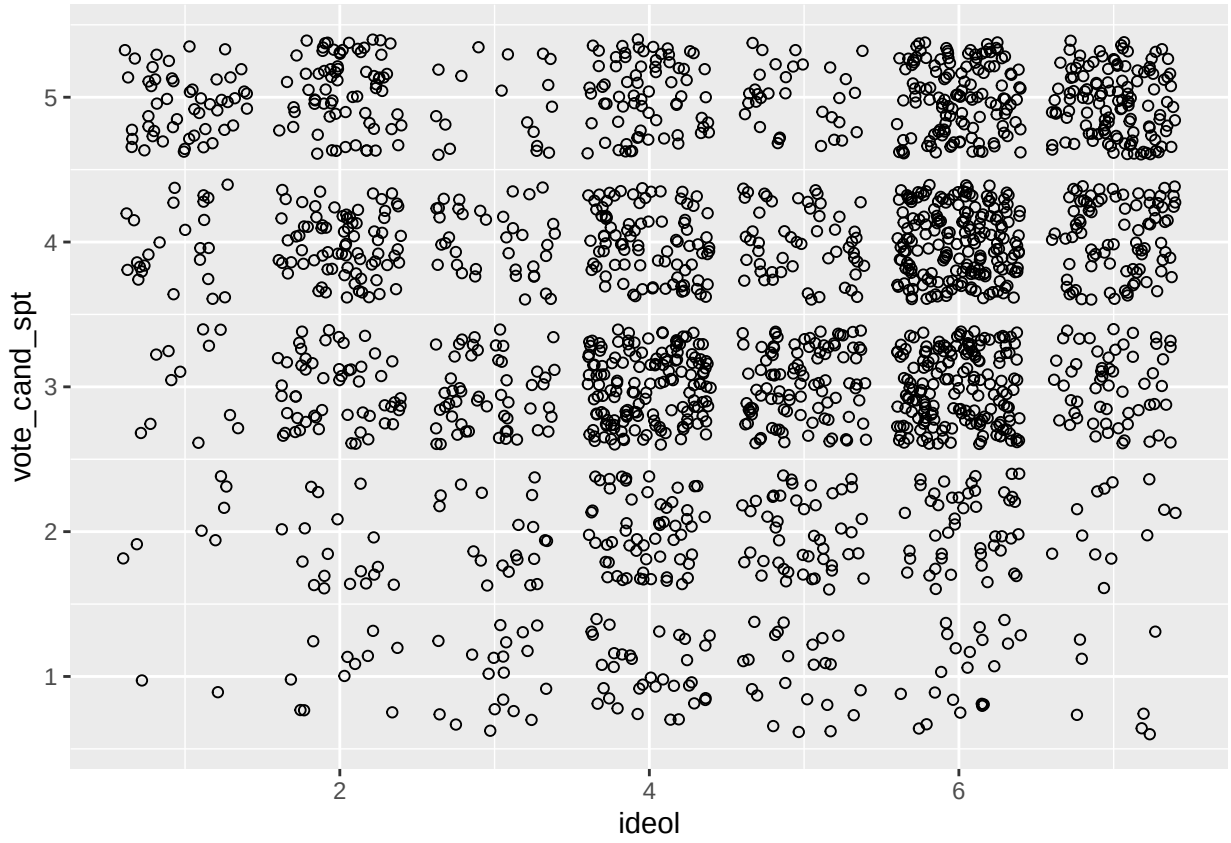
```
##
## ===================================================================
##                                  Dependent variable:
##                     ----------------------------------------------
##                                       vote_cand_spt
##                                (1)                     (2)
## -------------------------------------------------------------------
## ideol                     0.022 (0.015)
## poly(ideol, 2)1                                    1.613 (1.110)
## poly(ideol, 2)2                                  14.859*** (1.089)
## education              -0.046*** (0.015)         -0.059*** (0.015)
## income                -0.00000 (0.00000)        -0.00000 (0.00000)
```

```
## age                     0.009*** (0.002)         0.007*** (0.002)
## gender                   -0.060 (0.053)           -0.076 (0.050)
## Constant                3.225*** (0.163)         3.472*** (0.140)
## ------------------------------------------------------------------
## Observations                 1,980                    1,980
## R2                           0.022                    0.106
## Adjusted R2                  0.020                    0.104
## Residual Std. Error    1.134 (df = 1974)       1.084 (df = 1973)
## F Statistic         8.884*** (df = 5; 1974) 39.134*** (df = 6; 1973)
## ==================================================================
## Note:                                   *p<0.1; **p<0.05; ***p<0.01
```

The coefficients for the individual variables in the quadratic model are statistically significant, suggesting they help us understand our dependent variable. Let's move onto the visualization. Since ideology is our variable of interest, we will visualize the relationship between ideology and candidate support while holding the other variables constant at their means.

Start by looking at a scatter plot. To assist we need to jitter the data because ideology is a discrete variable:

```
ggplot(sub, aes(ideol, vote_cand_spt)) +
  geom_jitter(shape = 1)
```



Now add the regression line and confidence interval. To do this we will:

1. Create predicted values and standard error values of candidate support using the `augment()` function, while holding all other values constant at their mean.
2. Generate upper and lower limits of the confidence interval using `mutate()`
3. Visualize.

First: predict. We are going to sequence ideology from 1 to 7 by 0.1 instead of by 1 to produce a smoother

line:

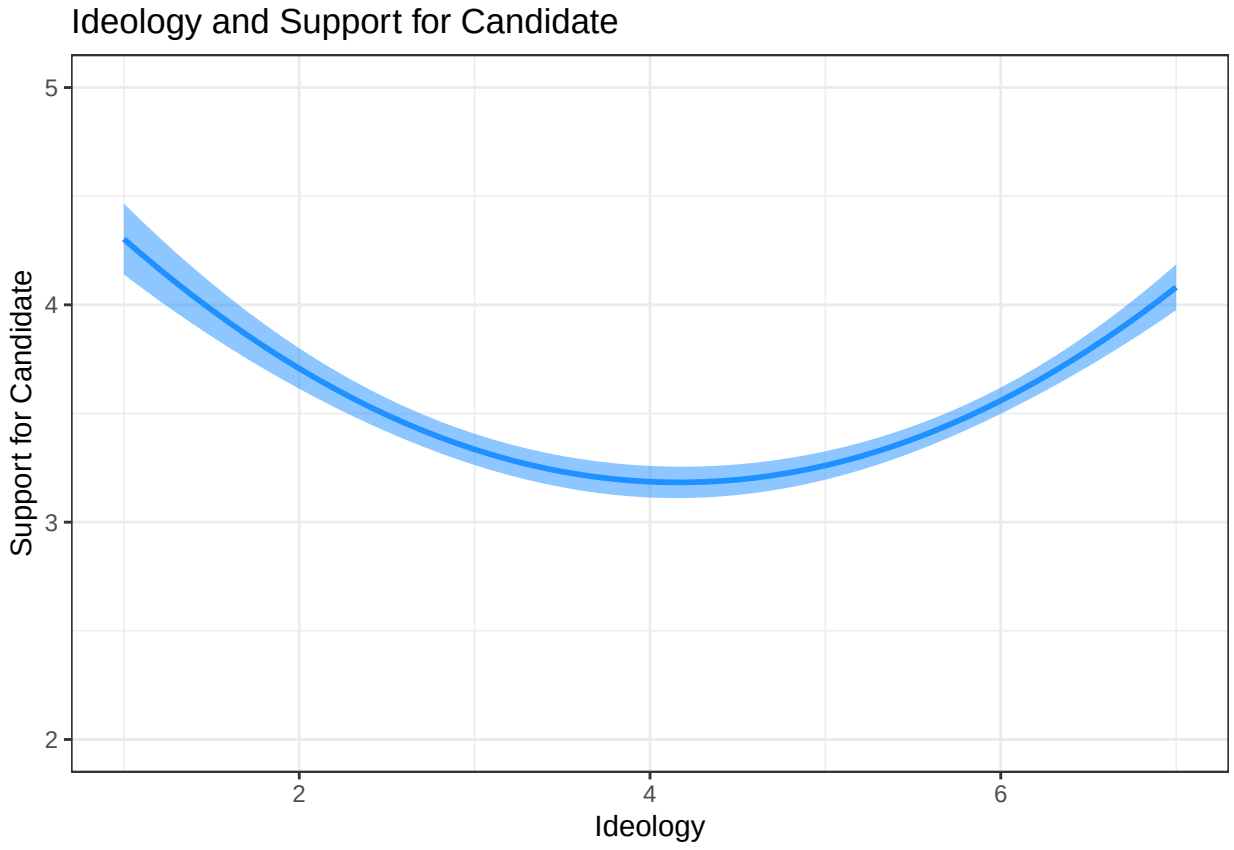
```
pol %>%
  augment(newdata = data.frame(ideol = seq(1,7,.1),
                               education = mean(sub$education),
                               income = mean(sub$income),
                               age = mean(sub$age),
                               gender= mean(sub$gender))) -> pol.df
```

Now create the upper and lower limits of the confidence interval:

```
pol.df %>%
  mutate(upper = .fitted + 1.96 * .se.fit,
         lower = .fitted - 1.96 * .se.fit) -> pol.df
```

The next step is to visualize. Use `geom_line()` to create the line, and `geom_ribbon()` to create the confidence interval. For practice, let's label the axes and add a title.

```
ggplot(pol.df, aes(ideol, .fitted)) +
  geom_line(size = 1, color = "dodgerblue") +
  geom_ribbon(aes(ymin = lower, ymax = upper), fill = "dodgerblue", alpha = .5) +
  coord_cartesian(ylim = c(2:5), xlim = c(1:7)) +
  xlab("Ideology") +
  ylab("Support for Candidate") +
  ggtitle("Ideology and Support for Candidate") +
  theme_bw()
```



This visualization shows the relationship specified in our theory: that the relationship between ideology and candidate support is quadratic, where more ideologically extreme individuals have more support for the

candidate they voted for than moderates.

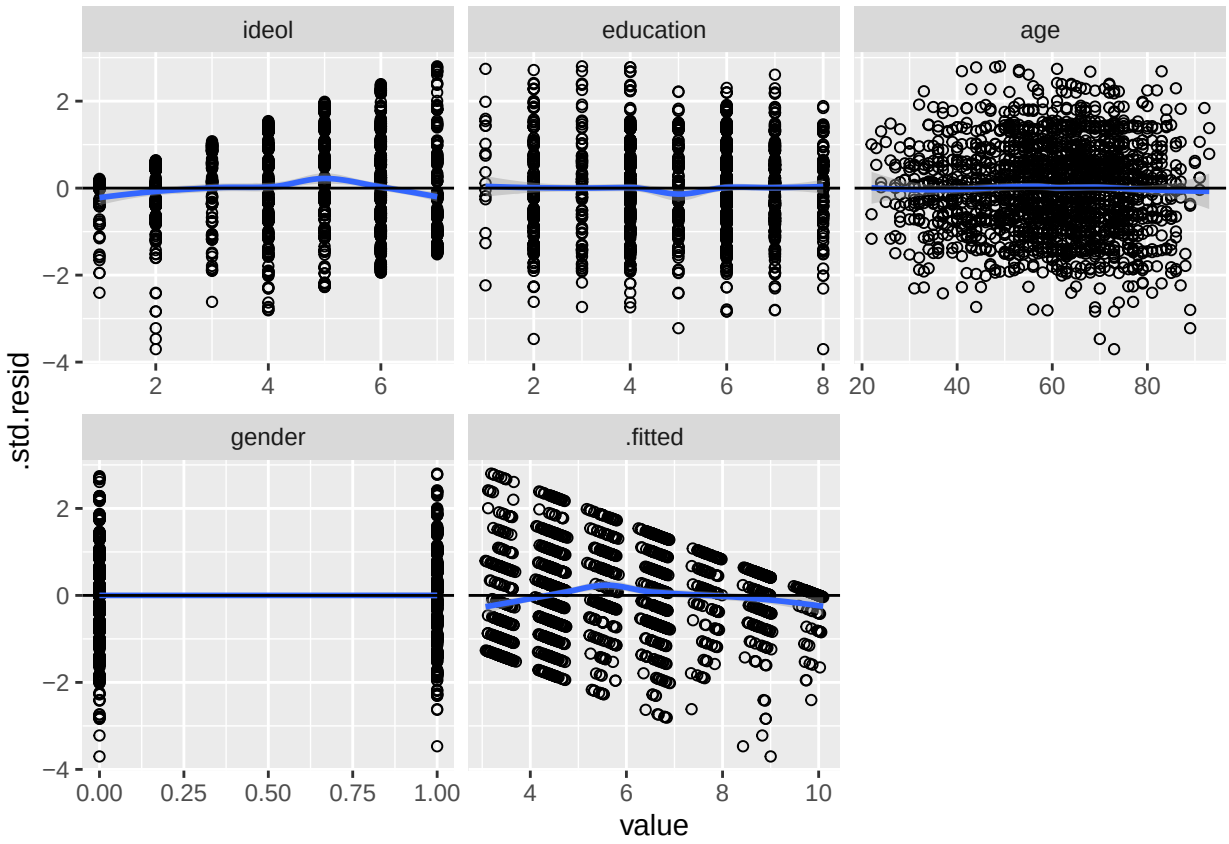
Let's use an example where a cubed exponent would be appropriate. A very common model we've used has been exploring the relationship between ideology and climate change risk. Perhaps the relationship between the two is not best described linearly. Let's first make a model we've looked at many times before:

```
lm1 <- lm(glbcc_risk ~ age + gender + ideol + education, data = sub)
summary(lm1)
```

```
##
## Call:
## lm(formula = glbcc_risk ~ age + gender + ideol + education, data = sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.0014 -1.6496  0.1836  1.4960  6.8115
##
## Coefficients:
##              Estimate Std. Error t value            Pr(>|t|)
## (Intercept) 10.788953   0.348172  30.987 <0.0000000000000002 ***
## age         -0.001105   0.004013  -0.275               0.7832
## gender      -0.264746   0.111988  -2.364               0.0182 *
## ideol       -1.062645   0.031745 -33.474 <0.0000000000000002 ***
## education    0.052300   0.031618   1.654               0.0983 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.436 on 1975 degrees of freedom
## Multiple R-squared:  0.3796, Adjusted R-squared:  0.3784
## F-statistic: 302.1 on 4 and 1975 DF,  p-value: < 0.00000000000000022
```

Now plot the residuals by the independent variables:

```
lm1 %>%
  augment() %>%
  melt(measure.vars = c("ideol", "education", "age",
                        "gender", ".fitted")) %>%
ggplot(., aes(value, .std.resid)) +
  geom_point(shape = 1) +
  geom_smooth(aes(value, .std.resid), method = "loess") +
  geom_hline(yintercept = 0) +
  facet_wrap(~variable, scales = "free_x")
```



Looking at the ideology variable, we can see that it is likely not linear. It looks more like the loess line moves above and below the line at zero. This may imply that a cube term might be appropriate. Build a model that cubes ideology:

```
cubed <- lm(formula = glbcc_risk ~ age + gender + poly(ideol, 3)  + education, data = sub)
summary(cubed)
```

```
##
## Call:
## lm(formula = glbcc_risk ~ age + gender + poly(ideol, 3) + education,
##     data = sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.8042 -1.7562  0.2032  1.5035  7.3212
##
## Coefficients:
##                       Estimate   Std. Error t value             Pr(>|t|)
## (Intercept)       5.71039034   0.31056558  18.387 < 0.0000000000000002 ***
## age              -0.00006485   0.00399923  -0.016               0.9871
## gender           -0.25936550   0.11136612  -2.329               0.0200 *
## poly(ideol, 3)1 -83.16325764   2.47050241 -33.662 < 0.0000000000000002 ***
## poly(ideol, 3)2 -12.07796545   2.42949486  -4.971          0.000000722 ***
## poly(ideol, 3)3  -3.53484785   2.42400684  -1.458               0.1449
## education         0.06113881   0.03146973   1.943               0.0522 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 2.421 on 1973 degrees of freedom
## Multiple R-squared:  0.3879, Adjusted R-squared:  0.3861
## F-statistic: 208.4 on 6 and 1973 DF,  p-value: < 0.00000000000000022
```

Now compare the two models:

```
stargazer(lm1, cubed, single.row = TRUE, type = "text")
```

```
##
## ===============================================================================
##                                      Dependent variable:
##                      ---------------------------------------------------------
##                                            glbcc_risk
##                                  (1)                          (2)
## -------------------------------------------------------------------------------
## age                          -0.001 (0.004)              -0.0001 (0.004)
## gender                      -0.265** (0.112)            -0.259** (0.111)
## ideol                       -1.063*** (0.032)
## poly(ideol, 3)1                                        -83.163*** (2.471)
## poly(ideol, 3)2                                        -12.078*** (2.429)
## poly(ideol, 3)3                                          -3.535 (2.424)
## education                    0.052* (0.032)              0.061* (0.031)
## Constant                    10.789*** (0.348)           5.710*** (0.311)
## -------------------------------------------------------------------------------
## Observations                      1,980                       1,980
## R2                                0.380                       0.388
## Adjusted R2                       0.378                       0.386
## Residual Std. Error          2.436 (df = 1975)           2.421 (df = 1973)
## F Statistic             302.117*** (df = 4; 1975) 208.430*** (df = 6; 1973)
## ===============================================================================
## Note:                                            *p<0.1; **p<0.05; ***p<0.01
```

It actually looks like the cube term does not describe the data very well. The adjusted R squared appears to marginally increase in the cubed model, but that does not tell us much. Let's use ANOVA:

```
anova(lm1, cubed)
```

```
## Analysis of Variance Table
##
## Model 1: glbcc_risk ~ age + gender + ideol + education
## Model 2: glbcc_risk ~ age + gender + poly(ideol, 3) + education
##   Res.Df   RSS Df Sum of Sq      F     Pr(>F)
## 1   1975 11720
## 2   1973 11562  2    157.55 13.443 0.00000159 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

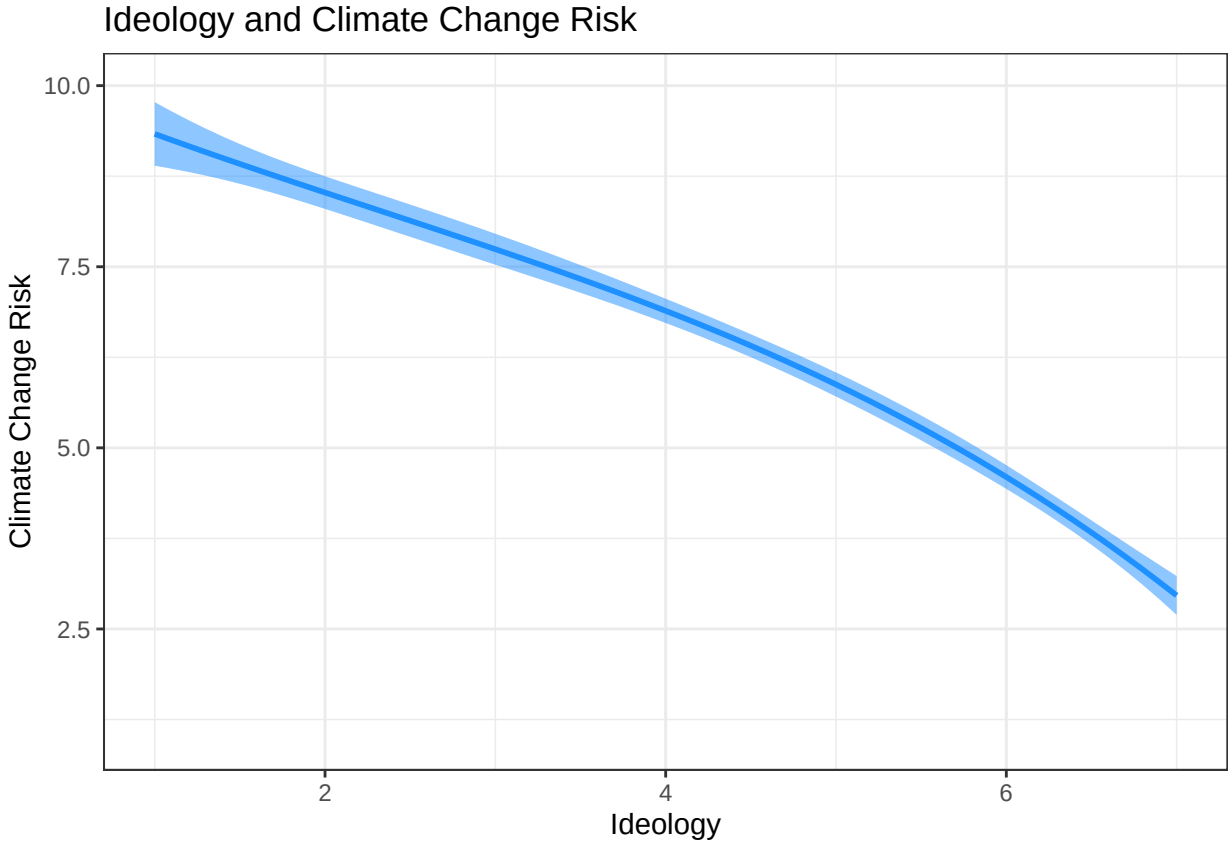
The ANOVA test tells us that our cubed model is a better fit. This is likely due to the square term, which is statistically significant. Nonetheless, let's visualize this so you have experience seeing cubed lines.

Follow the same steps as last time, predicting values, calculating the confidence interval, and plotting using `geom_line()` and `geom_ribbon()`.

```
cubed %>%
  augment(newdata = data.frame(ideol = seq(1,7,.1),
                               education = mean(sub$education),
                               age = mean(sub$age),
                               gender= mean(sub$gender))) %>%
```

```
  mutate(upper = .fitted + 1.96 * .se.fit,
         lower = .fitted - 1.96 * .se.fit) -> cube.df

ggplot(cube.df, aes(ideol, .fitted)) +
  geom_line(size = 1, color = "dodgerblue") +
  geom_ribbon(aes(ymin = lower, ymax = upper), fill = "dodgerblue", alpha = .5) +
  coord_cartesian(ylim = c(1:10), xlim = c(1:7)) +
  xlab("Ideology") +
  ylab("Climate Change Risk") +
  ggtitle("Ideology and Climate Change Risk") +
  theme_bw()
```



11.2 Non-normality

This section will go over the problem of non-normality and how to deal with it. One of the key assumptions of OLS is that the residuals of the model are normally distributed. It is also important to make sure your variables are not skewed too far either negatively or positively. Let's cover how to examine whether residuals are normally distributed, as well as how to handle greatly-skewed variables.

To begin, suppose you want to examine how different independent variables are related to a vote for Donald Trump in the 2016 presidential election. We need to clean up the class data set variable on 2016 presidential votes. Let's look at it:

```
table(ds$vote_cand)
```

```
##
##    0    1    2    3    4
```

```
## 1272  722  125    7   39
```

The codebook for the class data set tells us that a response of 0 indicates a vote for Trump, 1 is for Clinton, 2 for Gary Johnson, 3 for Jill Stein, and 4 for a different candidate. Change the variable so that it only includes Trump and Clinton. To transform this variable into a binary indicator of a vote for Trump or Clinton, we need to recode the variable so that responses of 1 equals 0, a response of 0 equals 1, with all else an NA:

```
ds$v.trump <- car::recode(ds$vote_cand, "0 = 1;1 = 0;else = NA;NA = NA")
table(ds$v.trump)
```

```
##
##    0    1
##  722 1272
```

Now pull the variables into a new data set and remove missing observations:

```
new.ds <- ds %>% dplyr::select("income", "gender", "ideol",
                                        "v.trump", "education") %>%
  na.omit()
```

Now review independent variables that are not binary:

```
psych::describe(new.ds$income)
```

```
##    vars    n     mean       sd median  trimmed     mad   min    max  range
## X1    1 1819 72889.47 58269.83  60000 64601.34 41512.8 10000 900000 890000
##    skew kurtosis      se
## X1 4.13    37.96 1366.24
```

```
psych::describe(new.ds$education)
```

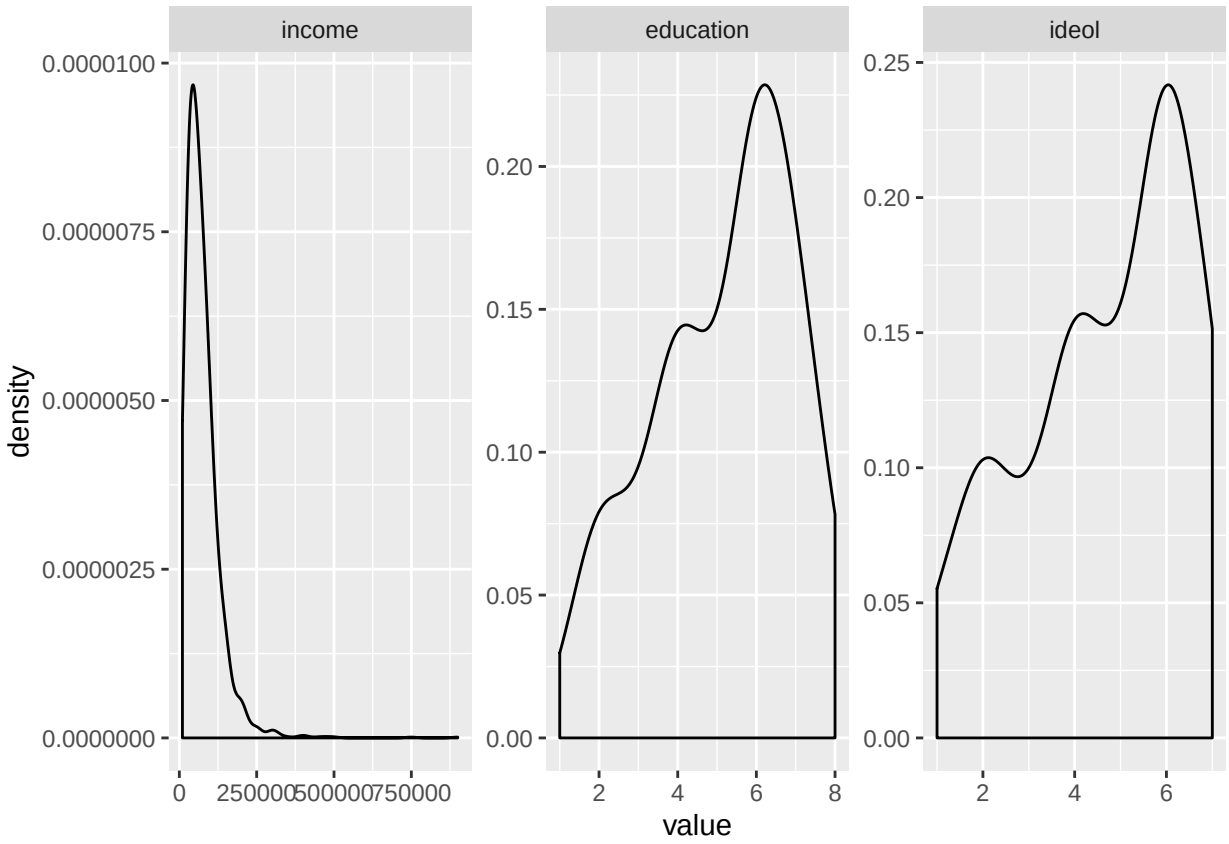
```
##    vars    n mean   sd median trimmed  mad min max range  skew kurtosis
## X1    1 1819 5.18 1.78      6    5.28 1.48   1   8     7 -0.45    -0.79
##      se
## X1 0.04
```

```
psych::describe(new.ds$ideol)
```

```
##    vars    n mean  sd median trimmed  mad min max range skew kurtosis   se
## X1    1 1819 4.69 1.8      5    4.81 1.48   1   7     6 -0.5    -0.89 0.04
```

Look at the density distributions:

```
new.ds %>%
  melt(measure.vars = c("income", "education", "ideol")) %>%
  ggplot(., aes(value)) +
  geom_density(adjust = 2) +
  facet_wrap(~ variable, scales = "free")
```



It is clear that the income variable has a large positive skew. Education and ideology are slightly skewed. One way to fix a skewed variable is to transform it, often by a log. Let's try that:

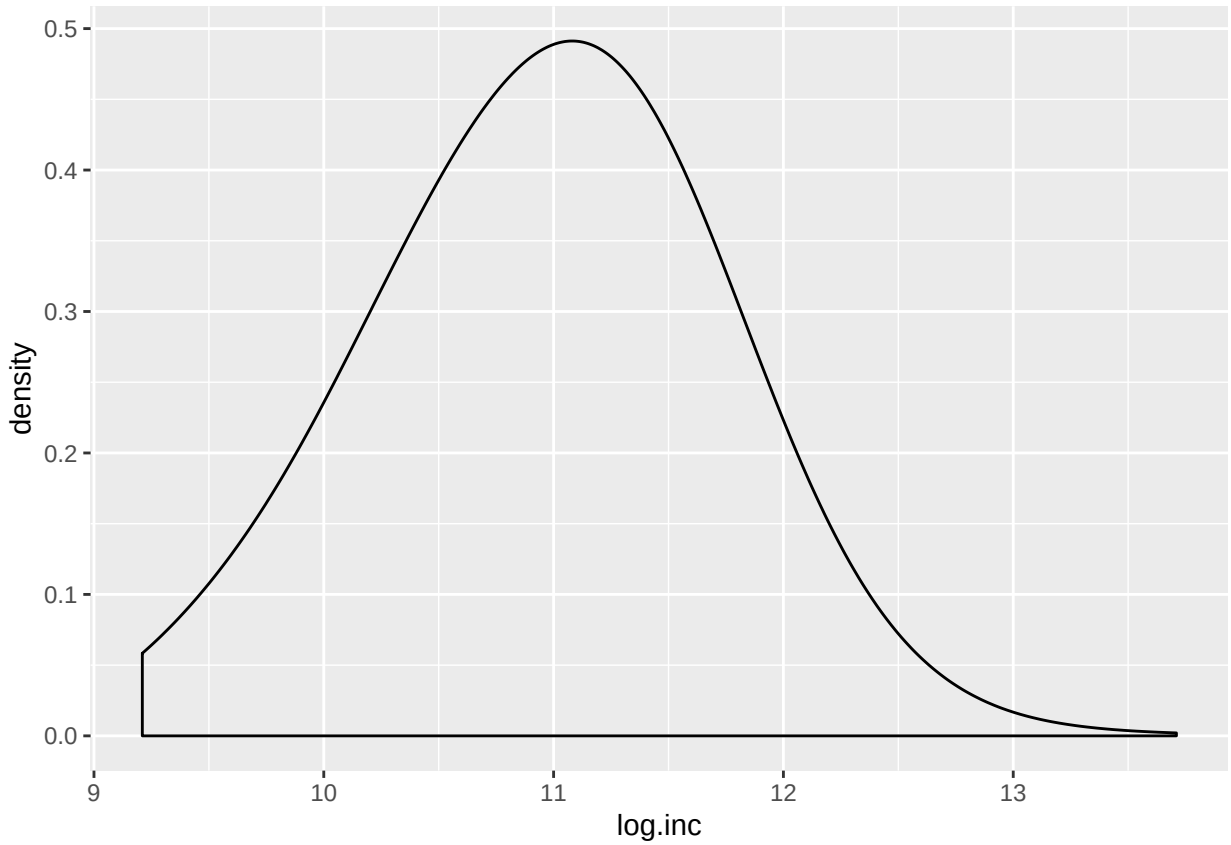
```
new.ds$log.inc <- log(new.ds$income)
```

Now review it:

```
psych::describe(new.ds$log.inc)
```

```
##    vars    n  mean  sd median trimmed  mad  min   max range  skew kurtosis
## X1    1 1819 10.96 0.7     11   10.97 0.74 9.21 13.71   4.5 -0.12    -0.03
##      se
## X1 0.02
```

```
ggplot(new.ds, aes(log.inc))+
  geom_density(adjust = 3)
```



The skew appears to be reduced. Transforming a variable does not really change how you use it in a model, but it does change the interpretation. Now the variable is ordered in logs. So a mean of 10.96 indicates 10.96 natural logged income. Now let's build the model, using the log of income instead of the income variable:

```
lm.trump <- lm(v.trump ~  log.inc + education + gender + ideol, data = new.ds)
summary(lm.trump)
```

```
##
## Call:
## lm(formula = v.trump ~ log.inc + education + gender + ideol,
##     data = new.ds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.11760 -0.12826  0.05173  0.16066  1.09163
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -0.273694   0.129610  -2.112               0.0349 *
## log.inc      0.015387   0.012458   1.235               0.2169
## education   -0.024763   0.004906  -5.048          0.000000492 ***
## gender       0.015522   0.016451   0.944               0.3455
## ideol        0.184062   0.004517  40.748 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3389 on 1814 degrees of freedom
## Multiple R-squared:  0.5044, Adjusted R-squared:  0.5033
```

```
## F-statistic: 461.6 on 4 and 1814 DF,  p-value: < 0.00000000000000022
```

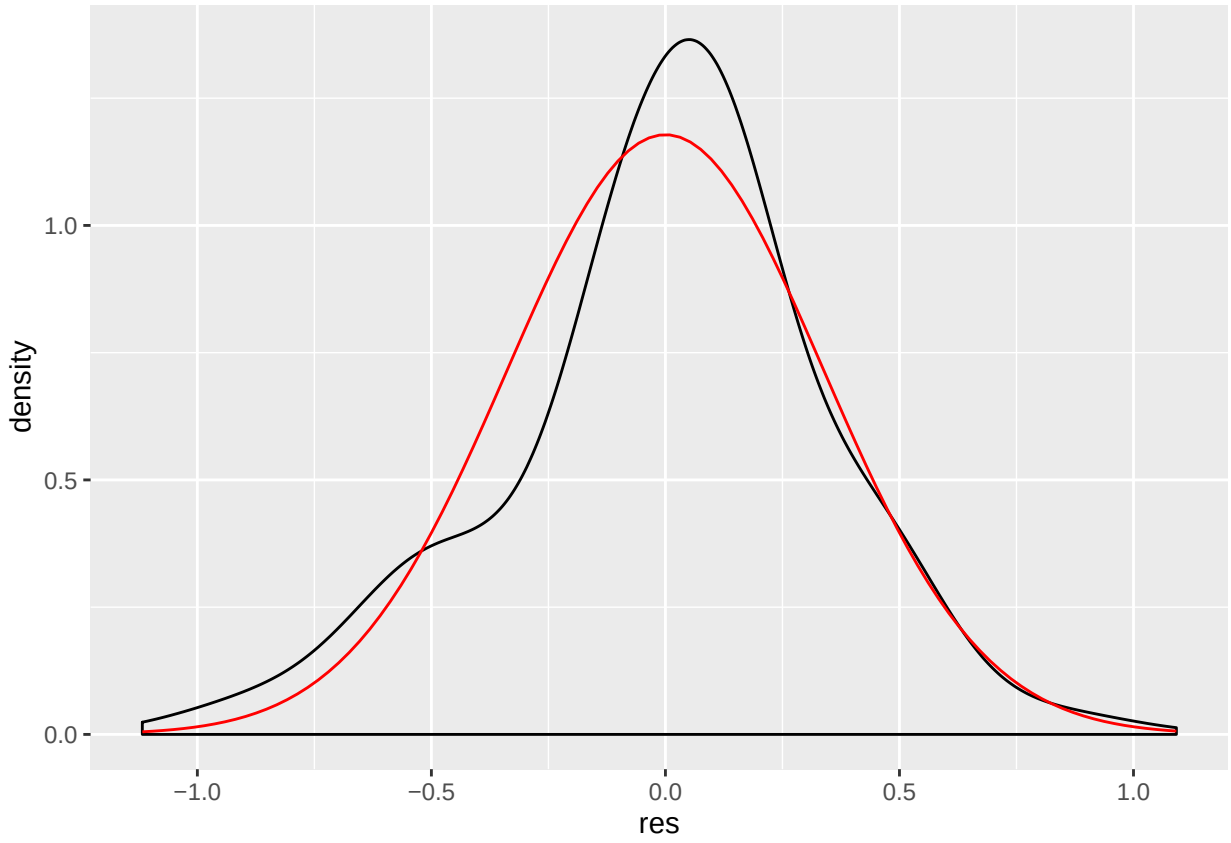
Regarding the ideology variable, there really should not be any surprises. More conservative individuals voted for Trump. Education is significant, but the coefficient is rather small. Our model suggests that education helps us explain voting, with more education tending to go with voting for Clinton. Our log.income variable does not help us to explain voting.

Let's now look at the normality of the residuals. First assign the residuals to our data set:

```
new.ds$res <- residuals(lm.trump)
```

Now make a density plot of the residuals, but also include a normal curve that has the mean and standard deviation of the residuals:

```
ggplot(new.ds, aes(res))+
  geom_density(adjust = 3) +
  stat_function(fun = dnorm, args = list(mean = mean(new.ds$res), sd = sd(new.ds$res)),
                color = "red")
```



The eye test indicates that we might have an issue with non-normality of the residuals. Let's run the Shapiro-Wilk test as well:

```
shapiro.test(lm.trump$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  lm.trump$residuals
## W = 0.97524, p-value < 0.00000000000000022
```

Recall that the Shapiro-Wilk test tests against the null hypothesis that data are normally distributed. Our

test result indicates that the residuals might not be normal, which is corroborated by the visualization. In a future lab we will go over one way to correct this, robust estimators.

11.3 Multicollinearity

Multicollinearity occurs when one independent variable of a model can be predicted with a high degree of accuracy by another independent variable. Even though perfect multicollinearity is very rare, checking for multicollinearity is an important process. The first way to explore potential multicollinearity is to check the collinearity between independent variables:

```
new.ds %>%
  dplyr::select(ideol, log.inc, education) %>%
  cor()
```

```
##                  ideol     log.inc  education
## ideol      1.000000000 0.009092405 -0.1700741
## log.inc    0.009092405 1.000000000  0.3732611
## education -0.170074114 0.373261119  1.0000000
```

There does not appear to be any extremely highly-correlated variables. We should find the variance inflation factor, which measures the increase in variance of the other coefficients due to the inclusion of a particular variable:

```
vif(lm.trump)
```

```
##   log.inc education    gender     ideol
##  1.193055  1.206849  1.039107  1.043909
```

Generally speaking, you do not want to have a value greater than 5. This model does not appear to have an issue with multicollinearity.

Now let's use an example that combines everything we've gone over so far. Let's examine the relationship between square footage of the respondent's home and income, age, and education. Start by selecting the data and removing missing observations:

```
d <- ds %>%
  dplyr::select("footage", "income", "age", "education") %>%
  na.omit()
```

Like earlier, we should do a log-transformation of income:

```
d$log.inc <- log(d$income)
```

Now build the model:

```
mod <- lm(footage ~ age + education + log.inc, data = d)
summary(mod)
```

```
##
## Call:
## lm(formula = footage ~ age + education + log.inc, data = d)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1951.5  -526.0  -176.6   296.6 18199.8
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
```

```
## (Intercept) -4598.212    403.704 -11.390 < 0.0000000000000002 ***
## age             9.633      1.695   5.683          0.000000015 ***
## education      32.968     14.126   2.334               0.0197 *
## log.inc       536.881     36.366  14.763 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1111 on 2232 degrees of freedom
## Multiple R-squared:  0.1186, Adjusted R-squared:  0.1174
## F-statistic: 100.1 on 3 and 2232 DF,  p-value: < 0.00000000000000022
```

Check for multicollinearity:

```
d %>%
  dplyr::select(age, education, log.inc) %>%
  cor()
```

```
##                   age   education    log.inc
## age        1.00000000 -0.04921609 -0.1327773
## education -0.04921609  1.00000000  0.3720851
## log.inc   -0.13277728  0.37208510  1.0000000
```

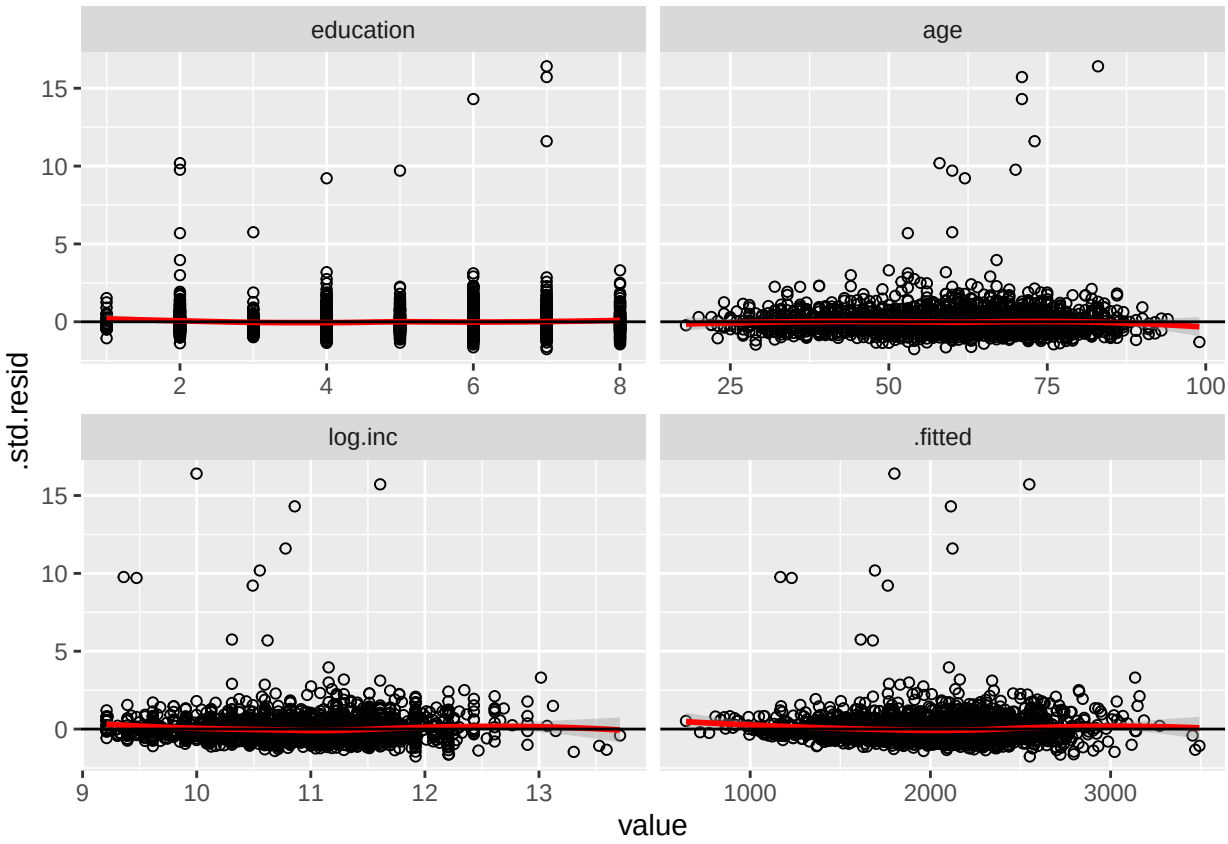
```
vif(mod)
```

```
##       age education   log.inc
##  1.017946  1.160695  1.178663
```

Taking all this into account, there does not appear to be a problem with multicollinearity.

Now let's examine the linearity of the variables. Recall the plot we made earlier that plots the independent variables by the residuals. Let's do that again:

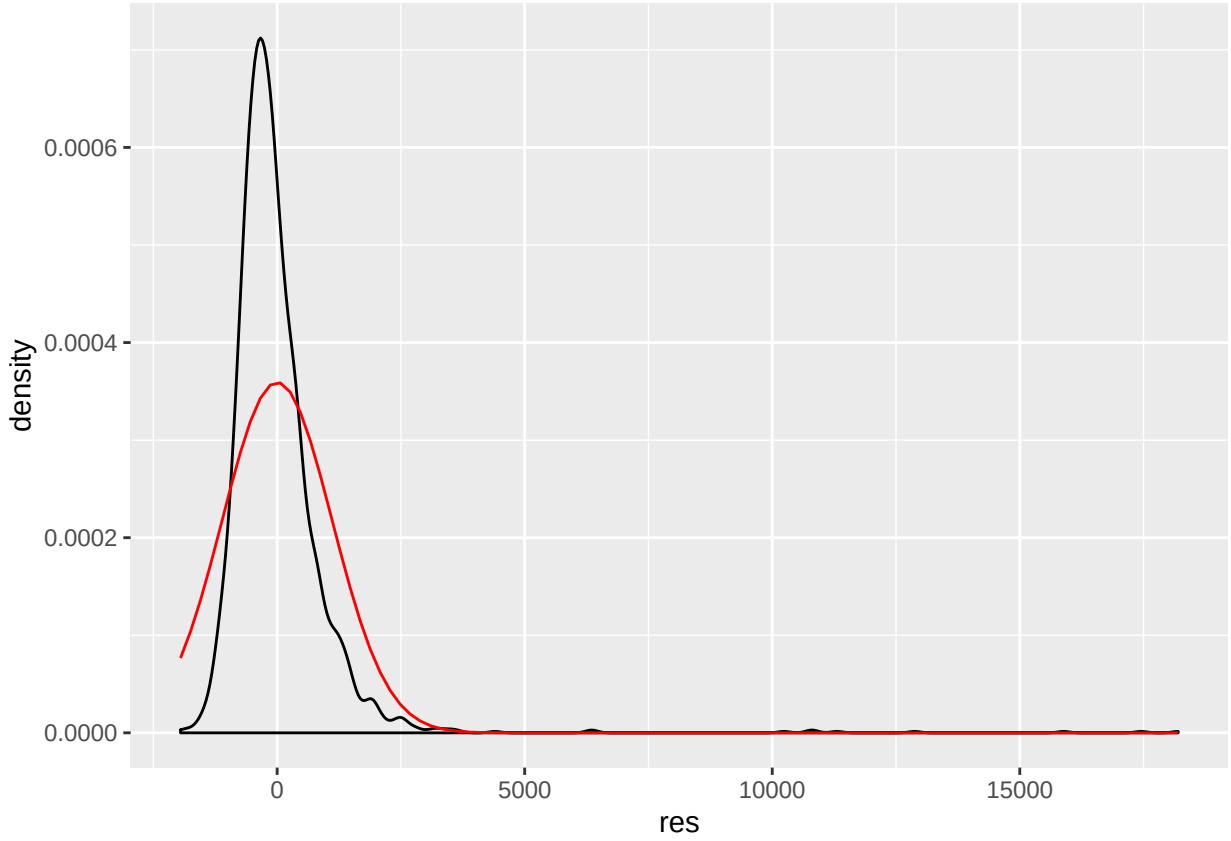
```
mod %>%
  augment() %>%
  melt(measure.vars = c("education", "age", "log.inc", ".fitted")) %>%
  ggplot(., aes(value, .std.resid)) +
  geom_point(shape = 1) +
  geom_smooth(method = loess, color = "red") +
  geom_hline(yintercept = 0) +
  facet_wrap(~variable, scales = "free_x")
```



There does not appear to be an issue with non-linearity either, so we have no reason to include exponents in the model.

The next step is to check for non-normality of the residuals:

```
d$res <- residuals(mod)
ggplot(d, aes(res))+
  geom_density() +
  stat_function(fun = dnorm, args = list(mean = mean(d$res), sd = sd(d$res)),
                color = "red")
```




```
shapiro.test(mod$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  mod$residuals
## W = 0.56711, p-value < 0.00000000000000022
```

Our results and visualization indicate that there could be a problem with non-normality.

Let's take a look at the results of the model again:

```
summary(mod)
```

```
##
## Call:
## lm(formula = footage ~ age + education + log.inc, data = d)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1951.5  -526.0  -176.6   296.6 18199.8
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -4598.212    403.704 -11.390 < 0.0000000000000002 ***
## age             9.633      1.695   5.683          0.000000015 ***
## education      32.968     14.126   2.334               0.0197 *
## log.inc       536.881     36.366  14.763 < 0.0000000000000002 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1111 on 2232 degrees of freedom
## Multiple R-squared:  0.1186, Adjusted R-squared:  0.1174
## F-statistic: 100.1 on 3 and 2232 DF,  p-value: < 0.00000000000000022
```

To interpret this model, we would say that a one unit increase in age corresponds with a 9.633 unit increase in square footage of home. Looking at education, we would say that a one unit increase in `log income` corresponds with a 536.9 unit increase in home square footage. Practically speaking though, how large is the difference between the age increase and the log income increase? We can see it is almost a 530 square foot difference, but when thinking about the overall distribution of the square footage variable, is that a lot?

11.4 Standardizing Coefficients

If you want to compare coefficients of different scales, you need to standardize them. There are three options when standardizing:

1. Standardize the dependent variable.
2. Standardize the independent variables.
3. Standardize all the variables.

Standardizing a variable refers to scaling it in standard deviations. This allows us to compare variables that were originally measured in different units. Let's use our previously developed model, but this time we will standardize the dependent variable only. Use the `scale()` function on the footage variable to standardize it:

```
d$z.footage <- scale(d$footage)
```

Now build the model and look at the results:

```
z.mod1 <- lm(z.footage ~ age + education + log.inc, data = d)
summary(z.mod1)
```

```
##
## Call:
## lm(formula = z.footage ~ age + education + log.inc, data = d)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6500 -0.4447 -0.1493  0.2508 15.3881
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -5.595500   0.341336 -16.393 < 0.0000000000000002 ***
## age          0.008145   0.001433   5.683          0.000000015 ***
## education    0.027875   0.011944   2.334               0.0197 *
## log.inc      0.453940   0.030748  14.763 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9395 on 2232 degrees of freedom
## Multiple R-squared:  0.1186, Adjusted R-squared:  0.1174
## F-statistic: 100.1 on 3 and 2232 DF,  p-value: < 0.00000000000000022
```

Since we only standardized the dependent variable, we would interpret this as saying that a one unit increase in age corresponds with a .008 standard deviation increase in square footage. For log income, we would say that a one unit increase in log income corresponds with a .45 standard deviation increase in square footage.

Now let's standardize the independent variables only:

```
d$z.age <- scale(d$age)
d$z.log.income <- scale(d$log.inc)
d$z.education <- scale(d$education)
```

Next build the model:

```
z.mod2 <- lm(footage ~ z.age + z.log.income + z.education, data=d)
summary(z.mod2)
```

```
##
## Call:
## lm(formula = footage ~ z.age + z.log.income + z.education, data = d)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1951.5  -526.0  -176.6   296.6 18199.8
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept)   2019.67      23.50  85.951 < 0.0000000000000002 ***
## z.age          134.76      23.71   5.683          0.000000015 ***
## z.log.income   376.71      25.52  14.763 < 0.0000000000000002 ***
## z.education     59.10      25.32   2.334               0.0197 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1111 on 2232 degrees of freedom
## Multiple R-squared:  0.1186, Adjusted R-squared:  0.1174
## F-statistic: 100.1 on 3 and 2232 DF,  p-value: < 0.00000000000000022
```

Now we would say that a one standard deviation increase in age corresponds with a 134.8 unit increase in square footage, and a one standard deviation increase in log income corresponds with a 376.71 unit increase in square footage. Comparing the coefficients here is rather simple and intuitive. Of course, we next need to standardize all the variables and interpret those.

```
z.mod3 <- lm(z.footage ~ z.log.income + z.education + z.age, data = d)
summary(z.mod3)
```

```
##
## Call:
## lm(formula = z.footage ~ z.log.income + z.education + z.age,
##     data = d)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6500 -0.4447 -0.1493  0.2508 15.3881
##
## Coefficients:
##                            Estimate            Std. Error t value
## (Intercept)  0.0000000000000004024 0.0198677514270765854   0.000
## z.log.income 0.3185111526573831675 0.0215744999251085111  14.763
## z.education  0.0499665777182138754 0.0214094220526226224   2.334
## z.age        0.1139389459055853149 0.0200497182548632288   5.683
##                           Pr(>|t|)
## (Intercept)                 1.0000
```

```
## z.log.income < 0.0000000000000002 ***
## z.education                0.0197 *
## z.age                 0.000000015 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9395 on 2232 degrees of freedom
## Multiple R-squared:  0.1186, Adjusted R-squared:  0.1174
## F-statistic: 100.1 on 3 and 2232 DF,  p-value: < 0.00000000000000022
```

Being careful to interpret this correctly, we would say that a one standard deviation change in log income corresponds with a .32 standard deviation increase in square footage.

12 Diagnosing and Addressing Problems in Linear Regression

This lab helps us understand how to diagnose and address potential problems in OLS regression. In the last lab, we addressed the OLS assumptions of linearity, multicollinearity, and normality of residuals. This lab addresses outliers and heteroscedasticity while also providing a refresher on exploring and visualizing your data. The following packages are required for this lab:

1. tidyverse
2. psych
3. car
4. stargazer
5. reshape2
6. stargazer
7. MASS
8. plotly
9. sandwich
10. broom

12.1 Introduction to the Data

For this lab we will use a data set that contains information on movies from the IMDB website. The initial data set contains almost 60,000 observations, so we will filter the data to make it a little smaller:

```
ds <- filter(ds, year>=1995 & votes>1000 & Short!=1) %>% dplyr::select(rating, length, budget,
                                                                       votes, title, year) %>%
  na.omit()
```

Now explore the data. Start with examining the structure of the data set:

```
str(ds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1359 obs. of  6 variables:
##  $ rating: num  6.7 4.7 6.4 6.1 6.1 5.1 5.4 2.5 7.6 8 ...
##  $ length: int  97 100 98 102 120 107 101 99 129 124 ...
##  $ budget: int  16000000 85000000 37000000 85000000 42000000 76000000 6000000 26000000 12000000 2000
##  $ votes : int  19095 1987 7859 14344 10866 9556 4514 2023 2663 21857 ...
##  $ title : chr  "10 Things I Hate About You" "102 Dalmatians" "13 Going On 30" "13th Warrior, The" .
##  $ year  : int  1999 2000 2004 1999 2001 2003 1999 2000 2004 2003 ...
##  - attr(*, "spec")=List of 2
##   ..$ cols   :List of 14
##   .. ..$ title      : list()
```

```
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ year       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ length     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ budget     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ rating     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ votes      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ mpaa       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ Action     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Animation  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Comedy     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Drama      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Documentary: list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Romance    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Short      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   ..$ default: list()
##   .. ..- attr(*, "class")= chr  "collector_guess" "collector"
##   ..- attr(*, "class")= chr "col_spec"
##  - attr(*, "na.action")= 'omit' Named int  1 3 4 6 10 16 17 23 28 30 ...
##   ..- attr(*, "names")= chr  "1" "3" "4" "6" ...
```

Look at the first five observations in the data set:

```
head(ds)
```

```
## # A tibble: 6 x 6
##   rating length   budget votes title                       year
##    <dbl>  <int>    <int> <int> <chr>                      <int>
## 1    6.7     97 16000000 19095 10 Things I Hate About You  1999
## 2    4.7    100 85000000  1987 102 Dalmatians              2000
## 3    6.4     98 37000000  7859 13 Going On 30              2004
## 4    6.1    102 85000000 14344 13th Warrior, The          1999
## 5    6.1    120 42000000 10866 15 Minutes                 2001
## 6    5.1    107 76000000  9556 2 Fast 2 Furious           2003
```

To make analysis easier, we can name each row by the title of the movie:

```
row.names(ds) <- ds$title
```

```
## Warning: Setting row names on a tibble is deprecated.
```

Now look at the first observations:

```
head(ds)
```

```
## # A tibble: 6 x 6
```

```
##   rating length   budget votes title                       year
##    <dbl>  <int>    <int> <int> <chr>                       <int>
## 1    6.7     97 16000000 19095 10 Things I Hate About You  1999
## 2    4.7    100 85000000  1987 102 Dalmatians              2000
## 3    6.4     98 37000000  7859 13 Going On 30              2004
## 4    6.1    102 85000000 14344 13th Warrior, The           1999
## 5    6.1    120 42000000 10866 15 Minutes                  2001
## 6    5.1    107 76000000  9556 2 Fast 2 Furious           2003
```

Explore some descriptive statistics:

```
describe(ds)
```

```
## Warning in describe(ds): NAs introduced by coercion
```

```
##        vars    n        mean          sd     median     trimmed
## rating    1 1359        6.17        1.16        6.3        6.22
## length    2 1359      109.26       20.54      105.0      107.12
## budget    3 1359 37038211.77 32450599.29 28000000.0 32614123.05
## votes     4 1359    11552.81    15216.95     6728.0     8413.21
## title*    5 1359     1050.00     1408.56     1050.0     1050.00
## year      6 1359     1999.99        2.76     2000.0     2000.04
##                mad    min       max       range  skew kurtosis        se
## rating        1.19    1.7         9         7.3 -0.55     0.53      0.03
## length       17.79   69.0       275       206.0  1.89     7.96      0.56
## budget 26686800.00 6000.0 200000000 199994000.0  1.36     2.15 880264.11
## votes      6594.60 1007.0    157608    156601.0  3.77    20.75    412.78
## title*     1476.67   54.0      2046      1992.0  0.00    -2.75     38.21
## year          2.97 1995.0      2005        10.0 -0.16    -1.02      0.07
```

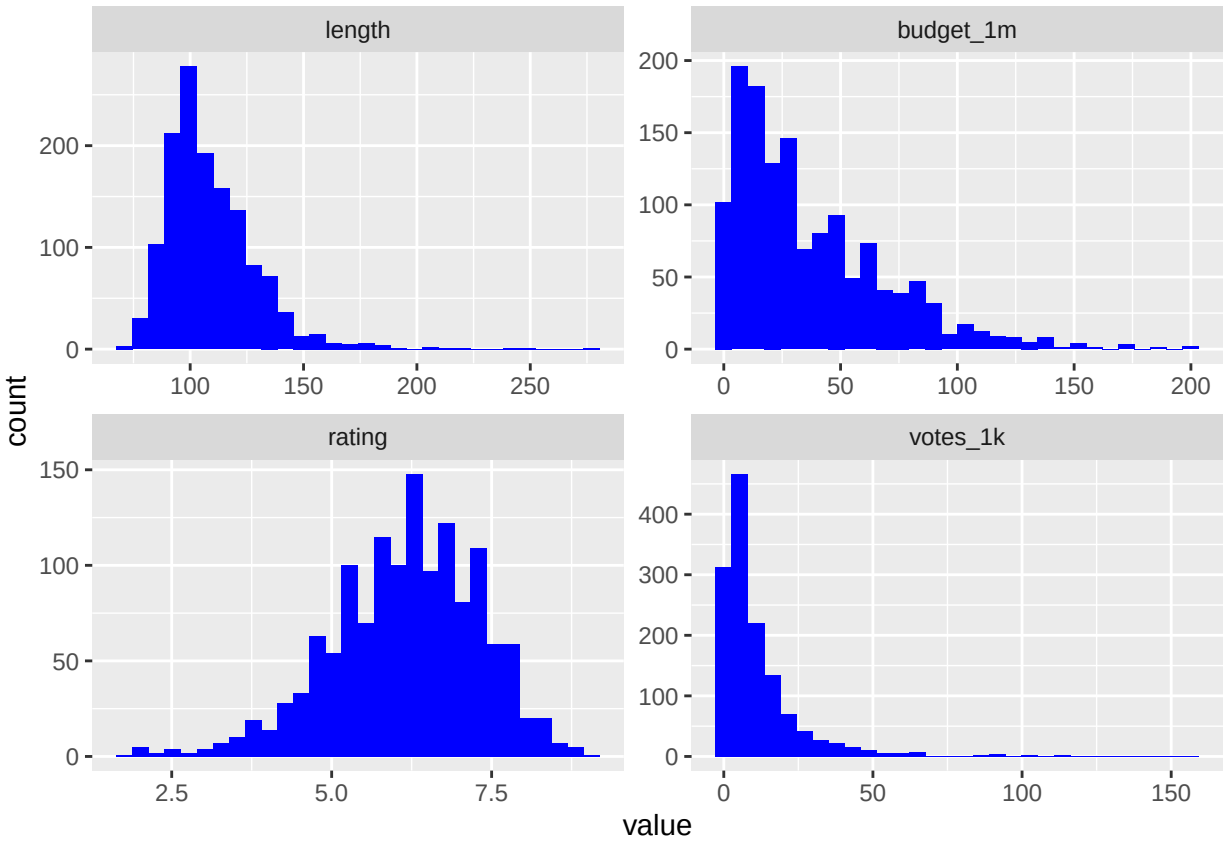
We will want to start by cleaning up a couple variables. First, we can scale the budget variable by millions of dollars, scale votes by thousands, and factor the year variable:

```
ds %>%
  mutate(budget_1m = budget/1000000,
         votes_1k = votes/1000,
         f.year = factor(year)) -> ds
```

The next step should be to look at the univariate distributions. Create histograms for the length, budget, user ratings, and votes variables. First you'll need to melt the data:

```
melt.ds <- melt(ds, measure.vars = c("length", "budget_1m", "rating", "votes_1k"))
ggplot(melt.ds, aes(value)) +
  geom_histogram(fill="#0000FF") +
  facet_wrap(~variable, scale="free")
```

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



Now let's look at the bivariate plots for the relationship between length, budget, votes, and rating. This is where we might find potential outliers. Build each visualization and then use `ggplotly()` to create an interactive interface that will allow you to identify individual observations.

```
vote <- ggplot(ds, aes(votes_1k, rating, label = title)) +
  geom_point(color = "#0000FF50") +
  geom_smooth(method = "loess", se = FALSE, color = "green") +
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  ggtitle("# of Votes and Rating")
ggplotly(vote)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```

```
length <- ggplot(ds, aes(length, rating, label = title)) +
  geom_point(color = "#0000FF50") +
  geom_smooth(method = "loess", se = FALSE, color = "green") +
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  ggtitle("Length and Rating")
ggplotly(length)
```

```
budget <- ggplot(ds, aes(budget_1m, rating, label = title)) +
  geom_point(color = "#0000FF50") +
  geom_smooth(method = "loess", se = FALSE, color = "green") +
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  ggtitle("Budget and Rating")
ggplotly(budget)
```

12.2 Outliers

The next step is to construct a model:

```
fit1 <- lm(rating ~ length + budget_1m + votes_1k, data = ds)
stargazer(fit1, type = "text", single.row = TRUE)
```

```
##
## ================================================
##                         Dependent variable:
##                     ---------------------------
##                                rating
## ------------------------------------------------
## length                    0.016*** (0.001)
## budget_1m                 -0.010*** (0.001)
## votes_1k                  0.033*** (0.002)
## Constant                  4.407*** (0.147)
## ------------------------------------------------
## Observations                   1,359
## R2                             0.312
## Adjusted R2                    0.311
## Residual Std. Error       0.962 (df = 1355)
## F Statistic           205.216*** (df = 3; 1355)
## ================================================
## Note:                *p<0.1; **p<0.05; ***p<0.01
```

Normally we are primarily interested in how this model explains our data. For this section, we are more interested in the observations that are not explained by this model. Let's identify some outliers. First create predicted ratings based on our model:

```
ds$predicted_rating <- predict(fit1)
```

One simple way to find some possible outliers is to use the `outlierTest()` function:

```
outlierTest(fit1)
```

```
##       rstudent unadjusted p-value Bonferroni p
## 476  -4.444751       0.0000095178     0.012935
## 1353 -4.343521       0.0000150670     0.020476
## 1313 -4.212031       0.0000269810     0.036667
```

We see four potential outliers... let's compare their predicted rating based on their budget, length, and number of votes, to their actual rating:

```
ds["From Justin to Kelly",c("rating", "predicted_rating")]
```

```
## # A tibble: 1 x 2
##   rating predicted_rating
##    <dbl>            <dbl>
## 1     NA               NA
```

```
ds["You Got Served",c("rating", "predicted_rating")]
```

```
## # A tibble: 1 x 2
##   rating predicted_rating
##    <dbl>            <dbl>
## 1     NA               NA
```

```
ds["Werewolf",c("rating", "predicted_rating")]
```

```
## # A tibble: 1 x 2
##   rating predicted_rating
##    <dbl>            <dbl>
## 1     NA               NA
```

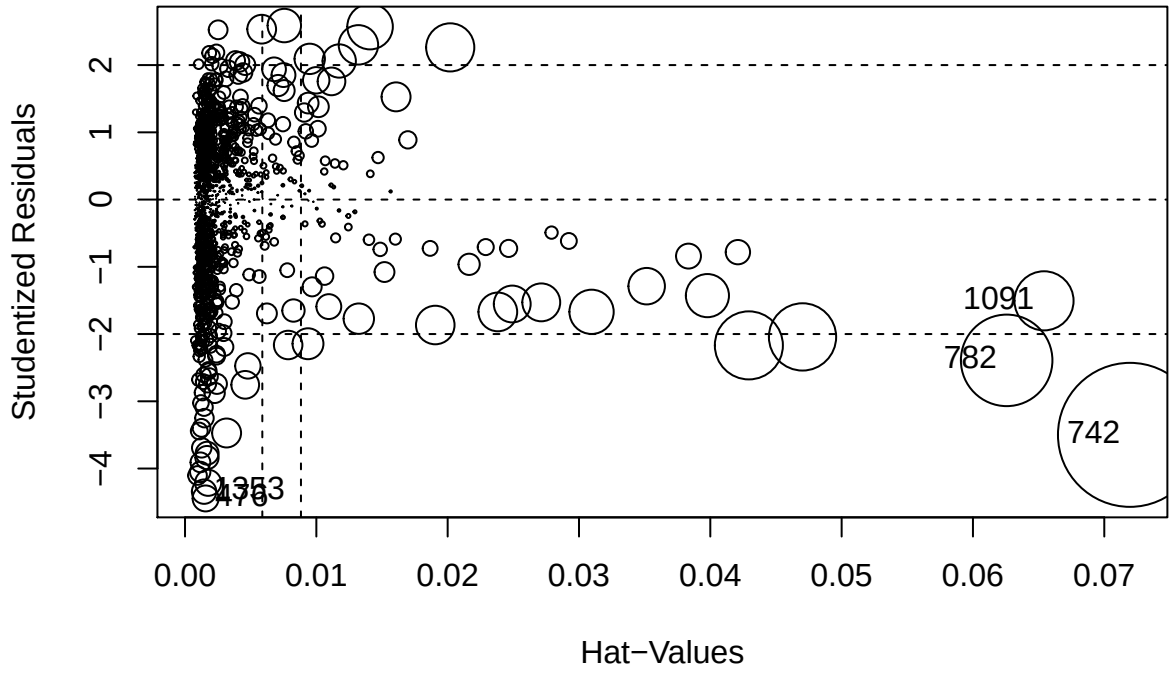
```
ds["Glitter",c("rating", "predicted_rating")]
```

```
## # A tibble: 1 x 2
##   rating predicted_rating
##    <dbl>            <dbl>
## 1     NA               NA
```

We can see that there is a large discrepancy between these movies' ratings and their predicted ratings.

There are a variety of ways to visually inspect outliers. Let's start with an influence plot:

```
influencePlot(fit1)
```




```
##         StudRes         Hat       CookD
## 476  -4.444751 0.001567730 0.007649222
## 742  -3.499031 0.071956182 0.235367503
## 782  -2.391247 0.062564978 0.095075700
## 1091 -1.506006 0.065411451 0.039647906
## 1353 -4.343521 0.001451519 0.006766880
```

An influence plot shows the residuals by their hat-values. This identifies the Matrix and the first Lord of the Rings as potential outliers, so let's compare their ratings to predicted ratings:

```
ds["Lord of the Rings: The Fellowship of the Ring, The",c("rating", "predicted_rating")]
```

```
## # A tibble: 1 x 2
##   rating predicted_rating
##    <dbl>            <dbl>
## 1     NA               NA
```

```
ds["Matrix, The",c("rating", "predicted_rating")]
```

```
## # A tibble: 1 x 2
##   rating predicted_rating
##    <dbl>            <dbl>
## 1     NA               NA
```

These movies are flagged outliers for a different reason than the movies we found earlier. These two movies are rated high, but their predicted ratings are even higher (above the maximum possible value of 10).

Another way to examine outliers is to look at the DFBetas. DFBetas measure the influence of case `i` on the `j` estimated coefficients. Put another way, measuring DFBetas asks how many standard errors a particular beta changes when case `i` is removed. The rule of thumb is if the absolute value of a DFBETA is greater than 2 divided by the square root of n, there could be cause for concern. Let's calculate that value:

```
df <- 2/sqrt(1261)
df
```

```
## [1] 0.05632127
```

We can find the DFBetas easily:

```
dfbs.fit1 <- dfbetas(fit1)
head(dfbs.fit1)
```

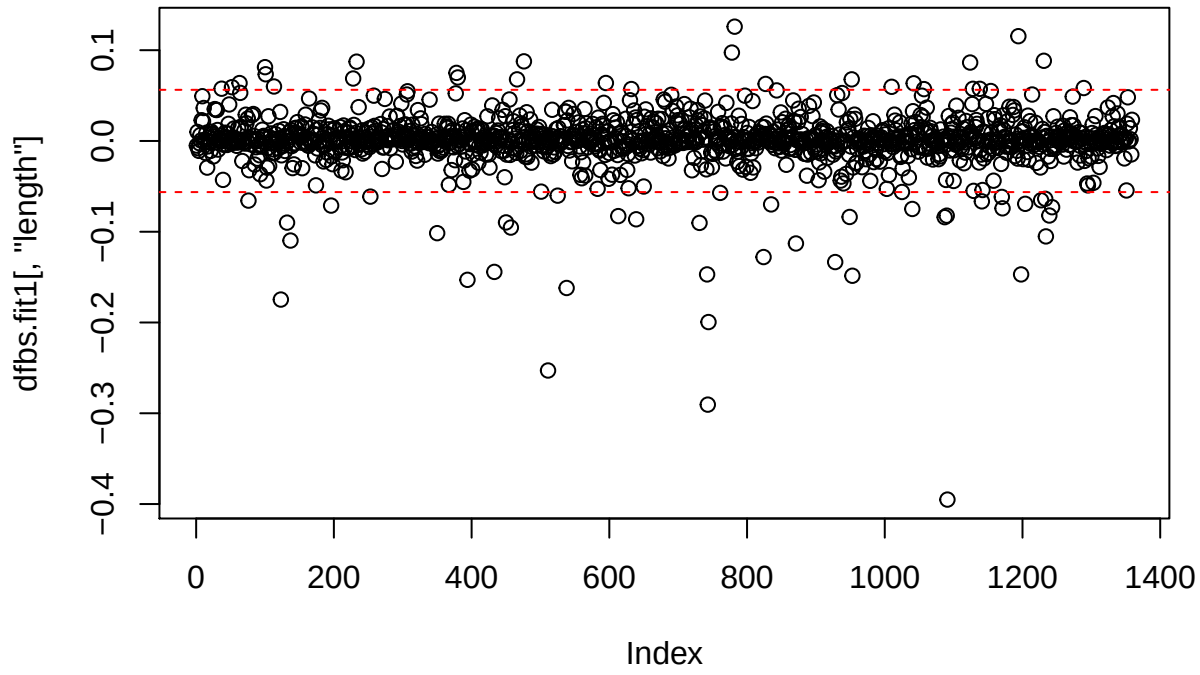
```
##    (Intercept)       length     budget_1m      votes_1k
## 1  0.006806296 -0.005084189 -0.0050276250  0.0070121773
## 2 -0.008979320  0.009910553 -0.0275525338  0.0130756825
## 3  0.010690935 -0.008032946  0.0032307871 -0.0011083722
## 4  0.009230482 -0.010909459  0.0210387730 -0.0003584795
## 5  0.001794457 -0.002610743 -0.0001487667  0.0012840947
## 6 -0.005795187  0.006463715 -0.0220719600  0.0064232528
```

With a large data set, listing every DFBeta value is not efficient. Instead, we should plot the DFBetas with lines at the calculated value. We will use the `identify()` function to mark specific observations. Each of the coefficients will have its own plot. **Note:** The `windows()` (or `quartz()`) function does not knit with RMarkdown; however, the function is used to build the plot on the screen as you follow along within the RMD document.

```
windows()
```

```
## Error in windows(): could not find function "windows"
```

```
plot(dfbs.fit1[,"length"])
abline(h = c(2/sqrt(1261), -2/sqrt(1261)), lty = 2, col = "red")
identify(dfbs.fit1[,"length"],labels = ds$title)
```

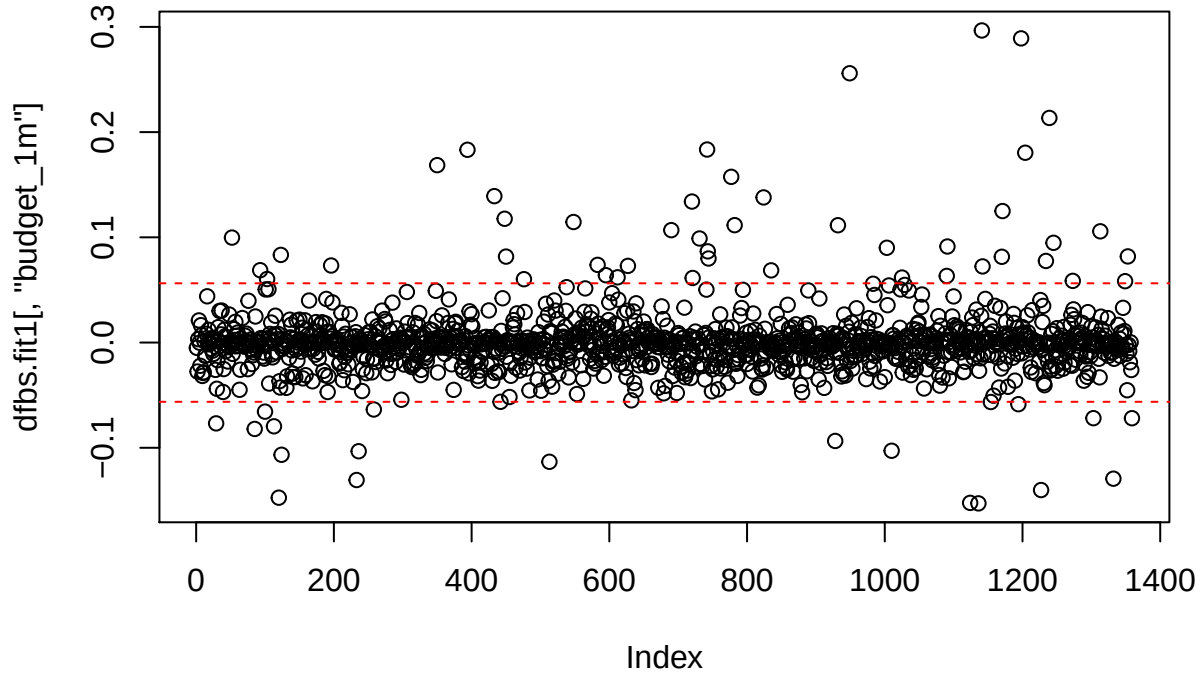



```
## integer(0)
```

```
windows()
```

```
## Error in windows(): could not find function "windows"
```

```
plot(dfbs.fit1[,"budget_1m"])
abline(h = c(2/sqrt(1261), -2/sqrt(1261)), lty = 2, col = "red")
identify(dfbs.fit1[,"budget_1m"],labels = ds$title)
```

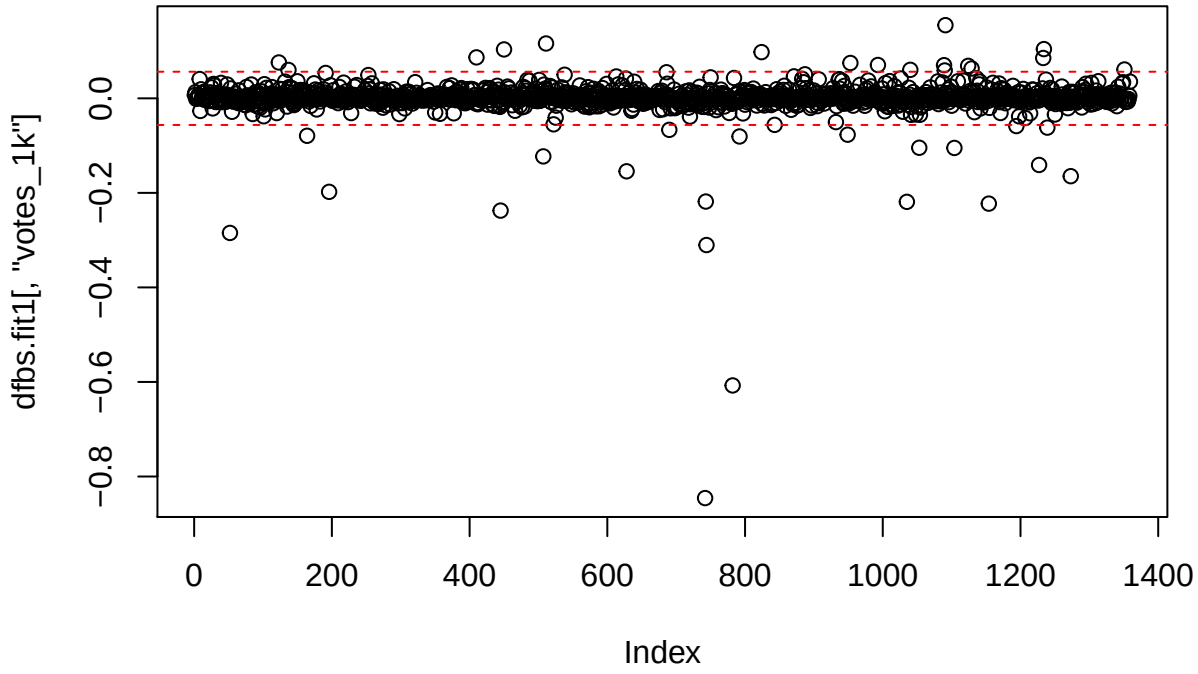



```
## integer(0)
```

```
windows()
```

```
## Error in windows(): could not find function "windows"
```

```
plot(dfbs.fit1[,"votes_1k"])
abline(h = c(2/sqrt(1261), -2/sqrt(1261)), lty = 2, col = "red")
identify(dfbs.fit1[,"votes_1k"],labels = ds$title)
```




```
## integer(0)
```

All of the diagnostics so far indicate that there are outliers to address. There are a few ways to deal with this: First, you can keep them in the model. This is a perfectly viable method, especially if you don't have a technical or theoretical reason to remove them. Another method of dealing with outliers is to omit them and re-run the model. Let's look at the outliers identified by the outlier test again:

```
outlierTest(fit1)
```

```
##       rstudent unadjusted p-value Bonferroni p
## 476  -4.444751       0.0000095178     0.012935
## 1353 -4.343521       0.0000150670     0.020476
## 1313 -4.212031       0.0000269810     0.036667
```

Omit these using the following operator. Essentially this tells R not to include the rows with the titles of the outlier movies.

```
ds.omit <- ds[ !(ds$title %in% c("From Justin to Kelly", "You Got Served", "Werewolf", "Glitter")),]
```

Next make a new model with the `ds.omit` data:

```
fit.omit <- lm(rating ~ length + budget_1m + votes_1k, data = ds.omit)
```

Compare the two models side by side:

```
stargazer(fit1, fit.omit, type = "text", single.row = TRUE)
```

```
##
## ===============================================================
```

```
##                                 Dependent variable:
##                     ---------------------------------------------
##                                         rating
##                               (1)                     (2)
## ------------------------------------------------------------------
## length                   0.016*** (0.001)        0.016*** (0.001)
## budget_1m                -0.010*** (0.001)       -0.010*** (0.001)
## votes_1k                 0.033*** (0.002)        0.033*** (0.002)
## Constant                 4.407*** (0.147)        4.450*** (0.143)
## ------------------------------------------------------------------
## Observations                  1,359                   1,355
## R2                            0.312                   0.322
## Adjusted R2                   0.311                   0.321
## Residual Std. Error     0.962 (df = 1355)       0.937 (df = 1351)
## F Statistic         205.216*** (df = 3; 1355) 214.265*** (df = 3; 1351)
## ==================================================================
## Note:                                   *p<0.1; **p<0.05; ***p<0.01
```

Notice the minimal changes. The omitted observations changed 3 of the four coefficients, and increased the adjusted R squared value.

Another option when dealing with outliers is to use robust regression, which weights the observations based on influence. Make a new model using robust regression using the `rlm()` function. There are two methods, "M" and "MM", and both should be evaluated to determine which model best represents your needs.

```
fit.m <- rlm(rating ~ length + budget_1m + votes_1k, data = ds, method = "M")
fit.mm <- rlm(rating ~ length + budget_1m + votes_1k, data = ds, method = "MM")
```

Compare the four models:

```
stargazer(fit1, fit.omit, fit.m, fit.mm, type = "text",single.row = TRUE)
```

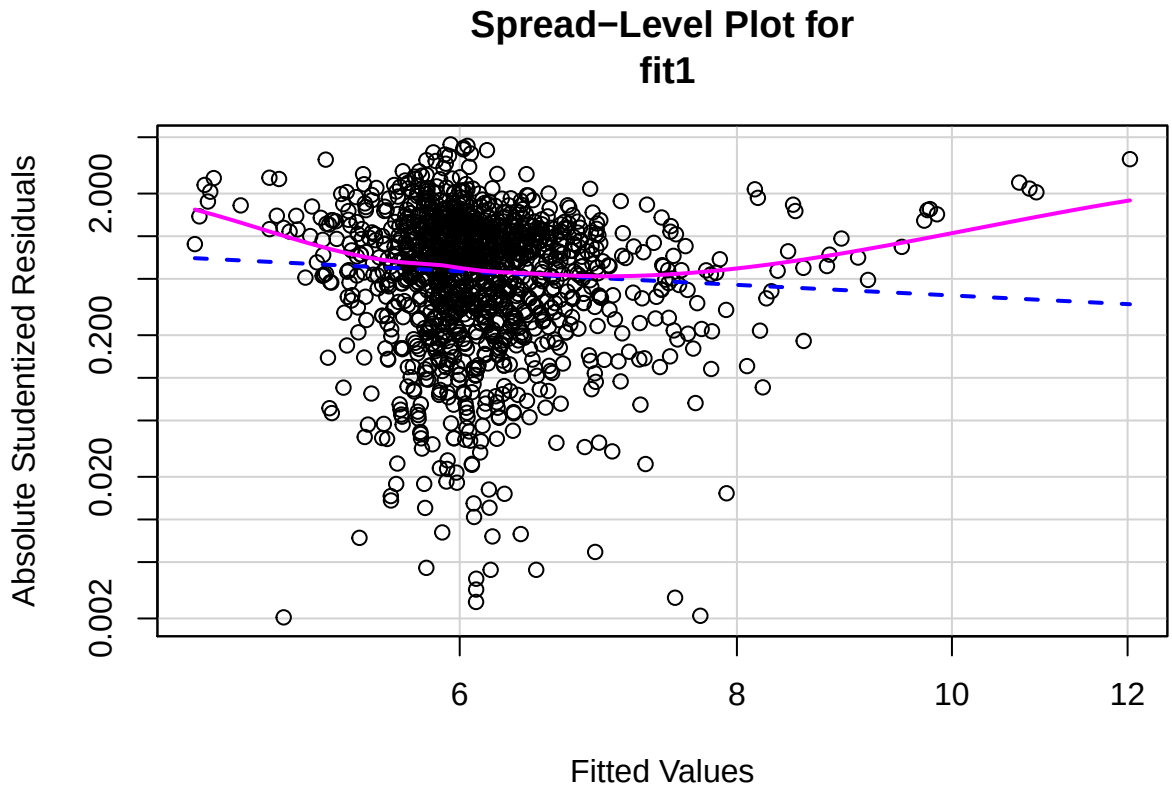
```
##
## ==========================================================================================
##                                          Dependent variable:
##                     ---------------------------------------------------------------------
##                                                rating
##                                    OLS                                  robust
##                                                                         linear
##                               (1)                     (2)                 (3)            (4)
## ------------------------------------------------------------------------------------------
## length                   0.016*** (0.001)        0.016*** (0.001)    0.017*** (0.001)  0.016*** (0
## budget_1m                -0.010*** (0.001)       -0.010*** (0.001)   -0.011*** (0.001) -0.012***
## votes_1k                 0.033*** (0.002)        0.033*** (0.002)    0.035*** (0.002)  0.036*** (0
## Constant                 4.407*** (0.147)        4.450*** (0.143)    4.432*** (0.138)  4.472*** (0
## ------------------------------------------------------------------------------------------
## Observations                  1,359                   1,355               1,359            1,35
## R2                            0.312                   0.322
## Adjusted R2                   0.311                   0.321
## Residual Std. Error     0.962 (df = 1355)       0.937 (df = 1351)   0.864 (df = 1355) 0.836 (df =
## F Statistic         205.216*** (df = 3; 1355) 214.265*** (df = 3; 1351)
## ==========================================================================================
## Note:                                                          *p<0.1; **p<0.05; ***
```

The biggest difference here is the residual standard error for the robust models is quite a bit lower. There are also differences in the coefficients. With outliers, there is not a one-size-fits-all solution. Let your theory contribute to what solution you use.

12.3 Heteroscedasticity

One of the key assumptions of OLS is homoscedasticity (constant error variance). One way to check for this is by making a spread level plot, which allows us to see the spread of the residuals:

```
spreadLevelPlot(fit1)
```




```
##
## Suggested power transformation:  1.771347
```

There does not appear to be constant spread of residuals, which could indicate a problem with heteroscedasticity. We can further investigate this by doing a Non-constant Variance Test. This tests the null hypothesis that error variance changes (heteroscedasticity). That is, if you fail to reject the null there exists heteroscedasticity:

```
ncvTest(fit1)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 3.205979, Df = 1, p = 0.07337
```

Based on the test and visualization, it is clear there is an issue with heteroscedasticity. There are a couple ways to deal with heteroscedasticity. One method is robust standard errors. Robust standard errors don't change the beta estimates, but rather affect the value of the standard errors, which improve the p-values accuracy. To use robust standard errors for the model:

```
se.fit1 <- fit1 %>% vcov() %>% diag() %>% sqrt()
vcov.fit1 <- vcovHC(fit1, method = "white1",type = "HC1")
rse.fit1 <- vcov.fit1 %>% diag() %>% sqrt()
```

Now compare the original model to the model using robust standard errors. Use `se=list(se.fit1,rse.fit1)` for R to use the original standard errors for the first model and robust for the second.

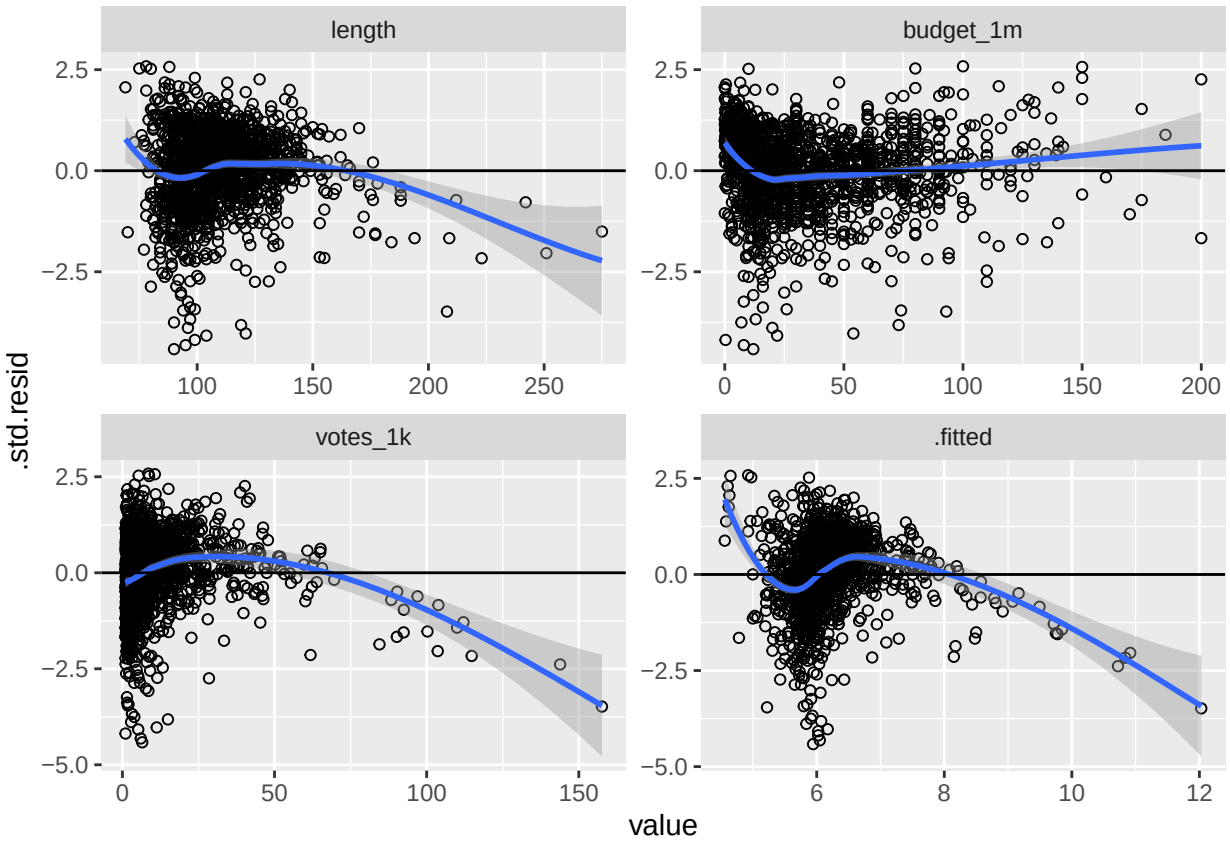
```
stargazer(fit1, fit1, type = "text", single.row = TRUE, se = list(se.fit1, rse.fit1))
```

```
##
## ===============================================================
##                                        Dependent variable:
##                                    -----------------------------
##                                                rating
##                                          (1)            (2)
## ---------------------------------------------------------------
## length                                0.016*** (0.001)  0.016*** (0.002)
## budget_1m                             -0.010*** (0.001) -0.010*** (0.001)
## votes_1k                              0.033*** (0.002)  0.033*** (0.003)
## Constant                              4.407*** (0.147)  4.407*** (0.168)
## ---------------------------------------------------------------
## Observations                               1,359             1,359
## R2                                         0.312             0.312
## Adjusted R2                                0.311             0.311
## Residual Std. Error (df = 1355)            0.962             0.962
## F Statistic (df = 3; 1355)              205.216***        205.216***
## ===============================================================
## Note:                                        *p<0.1; **p<0.05; ***p<0.01
```

12.4 Revisiting Linearity

Let's revisit addressing the assumption of linearity by constructing the residual plots that we made in the last lab. Recall that these are made by using `augment()` to predict values and calculate residuals, melt the data into long form and identify the independent variables and fitted values are your measure variables, then pipe it all into `ggplot2` and use `facet_wrap()` to create a visualization for each variable.

```
fit1 %>%
  augment() %>%
  melt(measure.vars = c("length", "budget_1m", "votes_1k", ".fitted")) %>%
  ggplot(., aes(value, .std.resid)) +
  geom_point(shape=1) +
  geom_smooth(method = loess) +
  geom_hline(yintercept = 0) +
  facet_wrap(~variable, scales = "free")
```



There appears to be a linearity problem. The budget graphics appears to be the most linear, and the others suggest non-linear relationships. Let's examine some more information about the variables:

```
describe(ds$length)
```

```
##    vars    n   mean    sd median trimmed   mad min max range skew kurtosis
## X1    1 1359 109.26 20.54    105  107.12 17.79  69 275   206 1.89     7.96
##      se
## X1 0.56
```

```
describe(ds$budget_1m)
```

```
##    vars    n  mean    sd median trimmed   mad  min max  range skew
## X1    1 1359 37.04 32.45     28   32.61 26.69 0.01 200 199.99 1.36
##    kurtosis   se
## X1     2.15 0.88
```

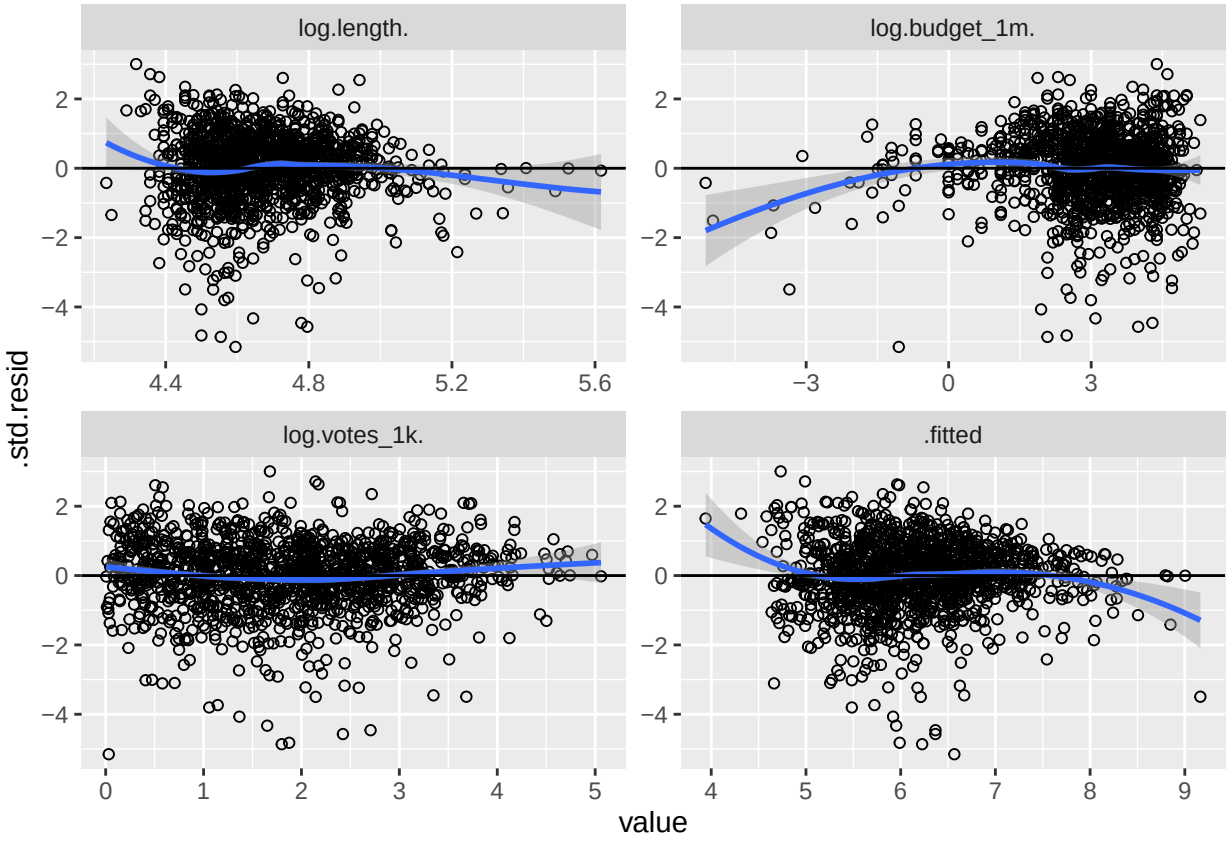
```
describe(ds$votes_1k)
```

```
##    vars    n  mean    sd median trimmed  mad  min    max range skew
## X1    1 1359 11.55 15.22   6.73    8.41 6.59 1.01 157.61 156.6 3.77
##    kurtosis   se
## X1    20.75 0.41
```

There is skew for all three variables, so let's respecify the model by using the log of each variable, then create the same visualization as before:

```
fit.log <- lm(rating ~ log(length) + log(budget_1m) + log(votes_1k), data = ds)
fit.log %>%
  augment() %>%
```

```
  melt(measure.vars = c("log.length.", "log.budget_1m.", "log.votes_1k.", ".fitted")) %>%
ggplot(., aes(value, .std.resid)) +
  geom_point(shape = 1) +
  geom_smooth(aes(value, .std.resid), method = "loess") +
  geom_hline(yintercept = 0) +
  facet_wrap(~variable, scales = "free")
```



This method fixed some problems and created new ones. The votes graphic suggests a more linear relationship, but problems persist. Perhaps a polynomial model is more appropriate. Let's square every IV in the next model:

```
fit.poly <- lm(rating ~ poly(length, 2) + poly(budget_1m, 2) + poly(votes_1k, 2), data = ds)
```

Compare the last three models:

```
stargazer(fit1, fit.log, fit.poly, single.row = TRUE, type = "text")
```

```
##
## ===============================================================================================
##                                                Dependent variable:
##                                    ---------------------------------------------------------
##                                                        rating
##                                       (1)                 (2)                    (3)
## -----------------------------------------------------------------------------------------------
## length                          0.016*** (0.001)
## budget_1m                       -0.010*** (0.001)
## votes_1k                        0.033*** (0.002)
## log(length)                                          2.235*** (0.155)
```

```
## log(budget_1m)                                -0.390*** (0.020)
## log(votes_1k)                                 0.551*** (0.026)
## poly(length, 2)1                                                                12.399*** (1.028)
## poly(length, 2)2                                                                -3.986*** (0.945)
## poly(budget_1m, 2)1                                                            -14.445*** (1.021)
## poly(budget_1m, 2)2                                                             5.735*** (0.917)
## poly(votes_1k, 2)1                                                              19.283*** (1.025)
## poly(votes_1k, 2)2                                                              -8.690*** (0.962)
## Constant                 4.407*** (0.147)          -4.132*** (0.703)          6.165*** (0.025)
## ------------------------------------------------------------------------------------------------
## Observations                    1,359                     1,359                     1,359
## R2                              0.312                     0.411                     0.383
## Adjusted R2                     0.311                     0.410                     0.381
## Residual Std. Error      0.962 (df = 1355)         0.890 (df = 1355)         0.912 (df = 1352)
## F Statistic         205.216*** (df = 3; 1355) 315.495*** (df = 3; 1355) 140.073*** (df = 6; 1352)
## ================================================================================================
## Note:                                                              *p<0.1; **p<0.05; ***p<0.01
```

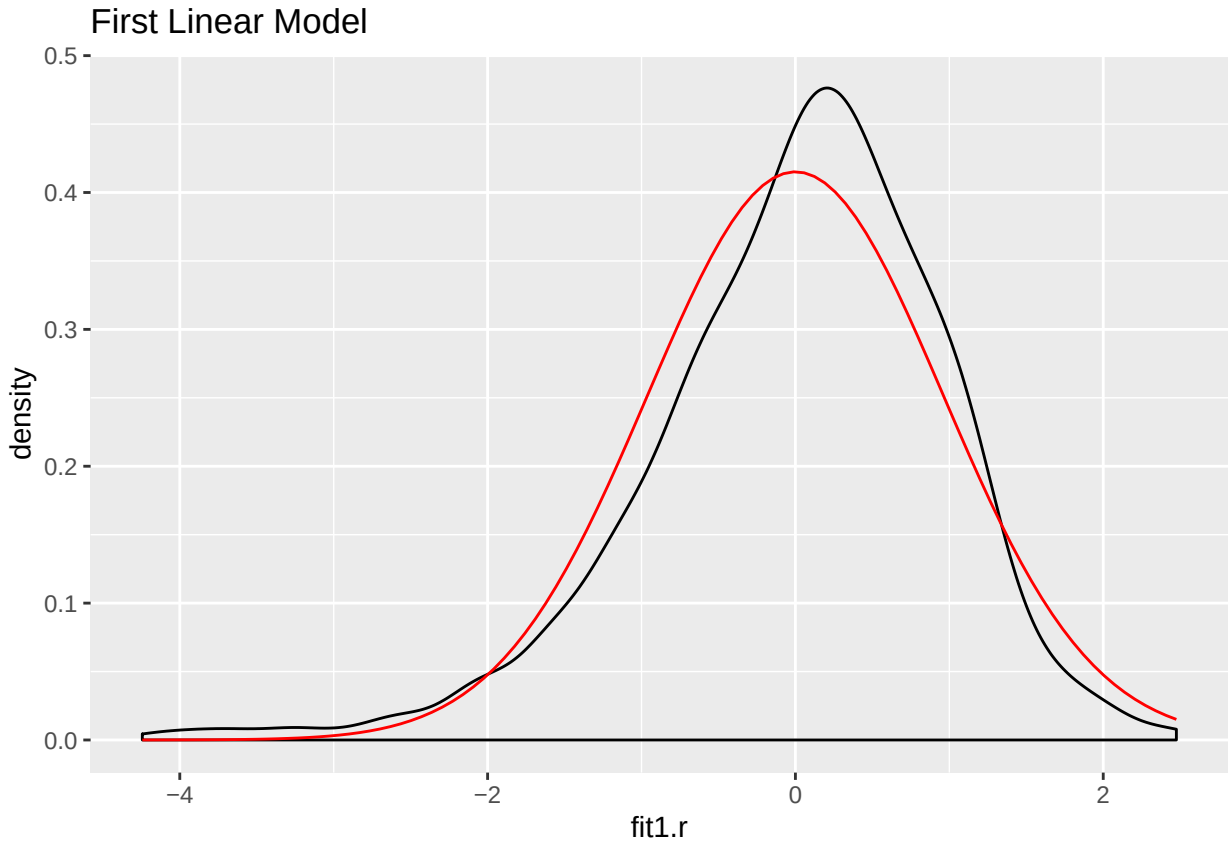
The log model has the highest adjusted R squared and lowest residual standard error.

12.4.1 Normality

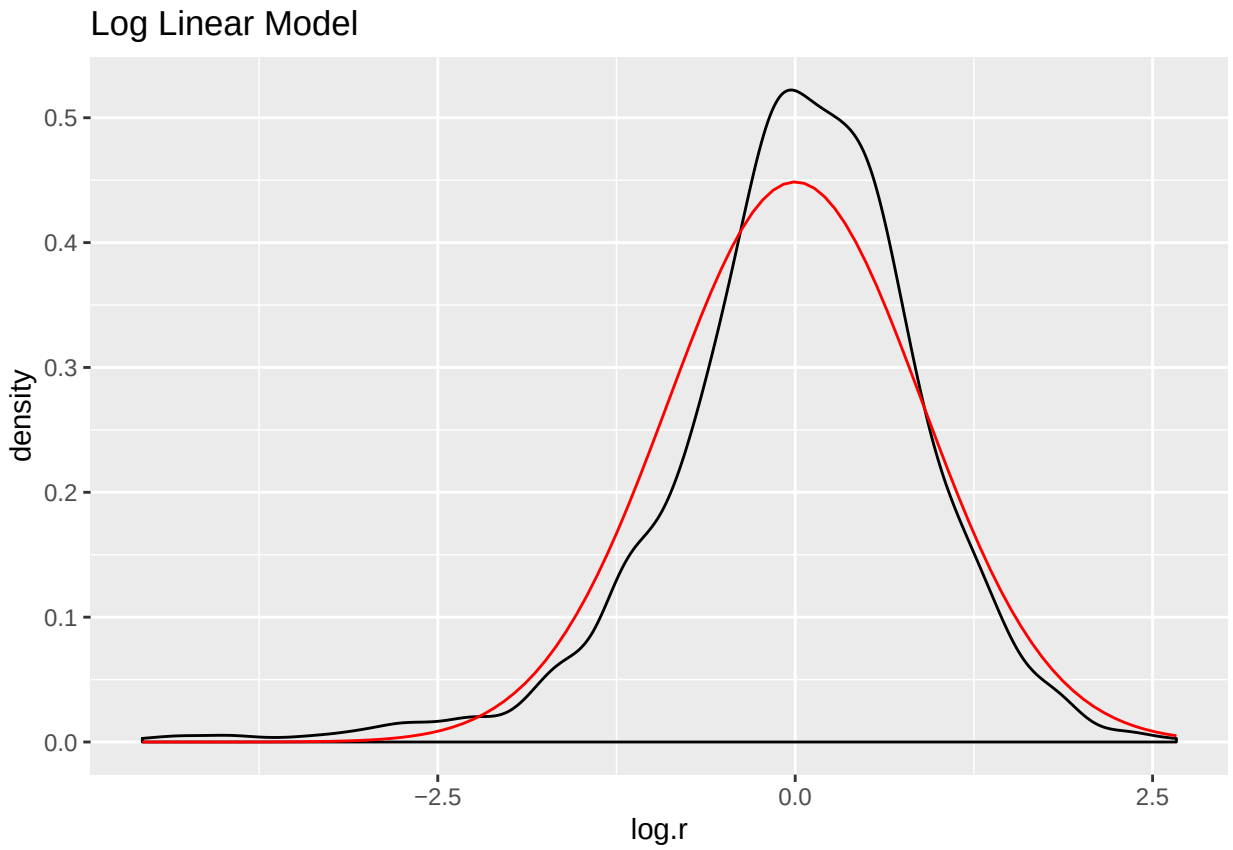
Let's look at the normality of the residuals for the models:

```
ds %>%
  mutate(fit1.r = residuals(fit1)) ->ds

ggplot(ds,aes(fit1.r)) +
  geom_density() +
  stat_function(fun = dnorm, args = list(mean = mean(ds$fit1.r),
                                         sd = sd(ds$fit1.r)), color = "red") +
  ggtitle("First Linear Model")
```

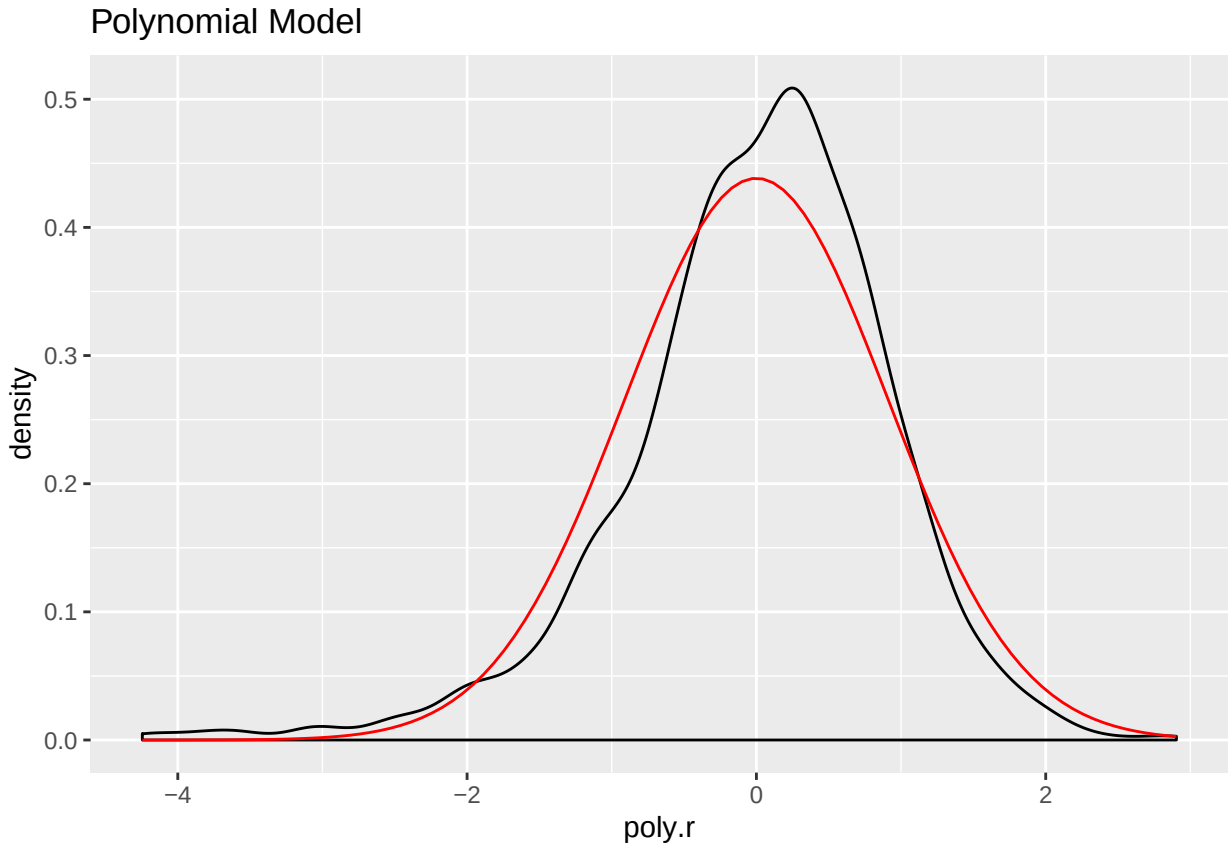



```
ds %>%
  mutate(log.r = residuals(fit.log)) -> ds
ggplot(ds, aes(log.r)) +
  geom_density() +
  stat_function(fun = dnorm, args = list(mean = mean(ds$log.r),
                                         sd = sd(ds$log.r)), color = "red") +
  ggtitle("Log Linear Model")
```




```
ds %>%
  mutate(poly.r = residuals(fit.poly)) -> ds

ggplot(ds, aes(poly.r)) +
  geom_density() +
  stat_function(fun = dnorm, args = list(mean = mean(ds$poly.r),
                                         sd = sd(ds$poly.r)), color="red") +
  ggtitle("Polynomial Model")
```



The log model has the highest adjusted R squared value, the lowest residual standard error, and its residuals appear to approximate the normal distribution better than the other two models. Let's use it to make predictions and create visualizations.

First create predicted values for movie ratings by holding all IVs constant at their means except one at at time, using the `augment()` function. Then use `mutate()` to calculate the upper and lower bounds of the confidence interval. Create separate data frames for length, budget, and votes.

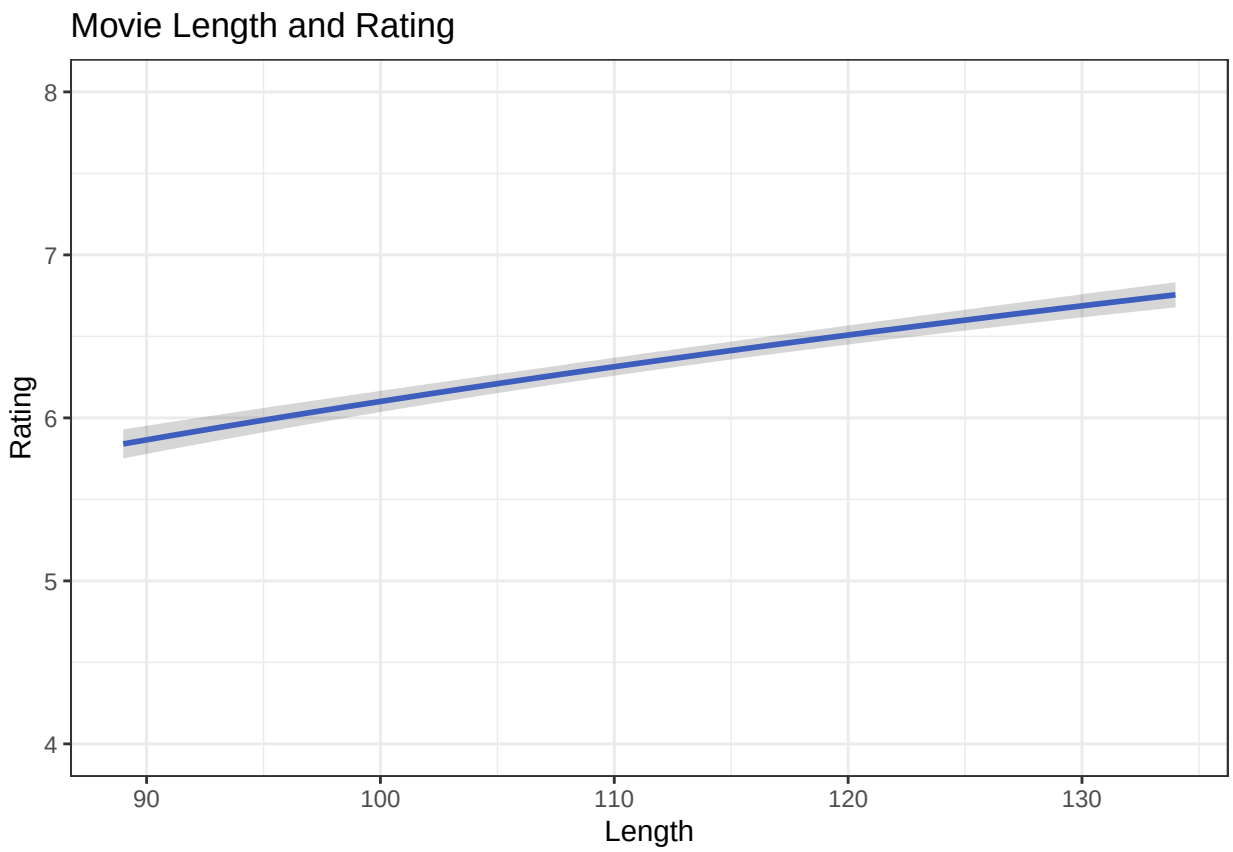
```
df.length <- fit.log %>%
  augment(newdata = data.frame(length = 89:134,
                               budget_1m = mean(ds$budget_1m),
                               votes_1k = mean(ds$votes_1k))) %>%
  mutate(upper = .fitted + 1.96 * .se.fit,
         lower = .fitted - 1.96 * .se.fit)

df.budget <- fit.log %>%
  augment(newdata = data.frame(length = mean(ds$length),
                               budget_1m = 5:80,
                               votes_1k = mean(ds$votes_1k))) %>%
  mutate(upper = .fitted + 1.96 * .se.fit,
         lower = .fitted - 1.96 * .se.fit)

df.votes <- fit.log %>%
  augment(newdata = data.frame(length = mean(ds$length),
                                       budget_1m = mean(ds$budget_1m),
                                       votes_1k = 1.645:25.964)) %>%
  mutate(upper = .fitted + 1.96 * .se.fit,
         lower = .fitted - 1.96 * .se.fit)
```

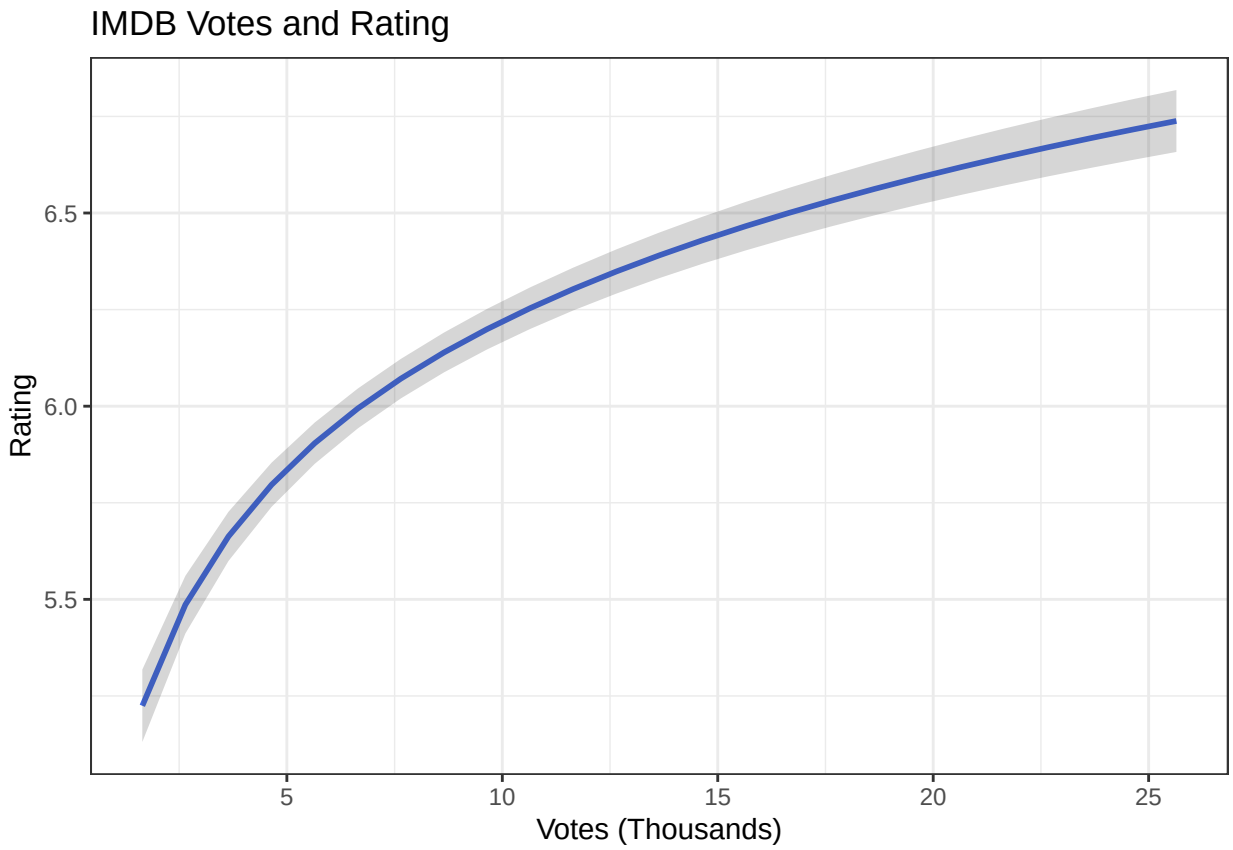
Now make the visualization for each data frame~

```
ggplot(df.length, aes(length, .fitted)) +
  geom_line(size = 1, color = "royalblue") +
  geom_ribbon(aes(ymin = lower, ymax = upper), alpha = .2) +
  coord_cartesian(ylim = c(4:8), xlim = c(89:134)) +
  ggtitle("Movie Length and Rating") +
  xlab("Length") +
  ylab("Rating") +
  theme_bw()
```

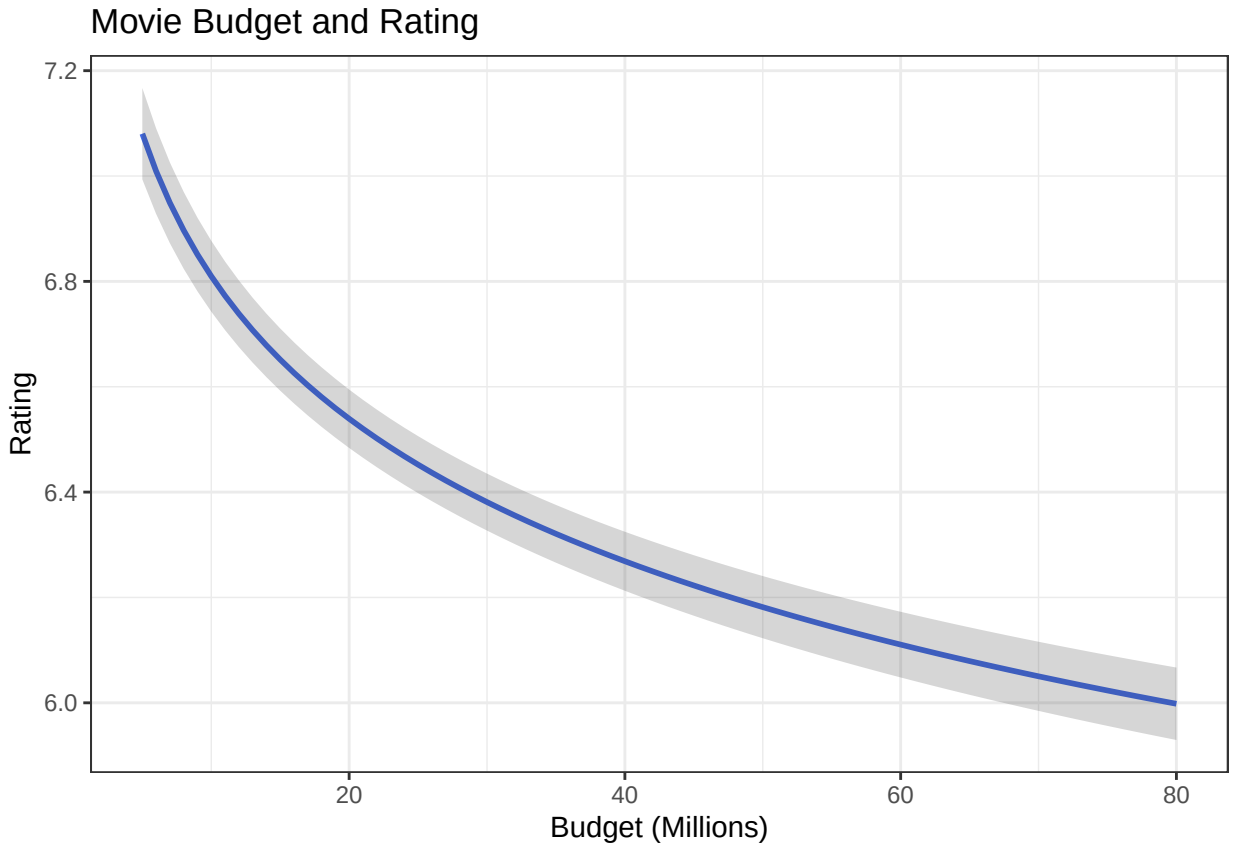


Now do the same for thse next two IVs:

```
ggplot(df.votes, aes(votes_1k, .fitted)) +
  geom_line(size = 1, color = "royalblue") +
  geom_ribbon(aes(ymin = lower, ymax = upper), alpha = .2) +
  ggtitle("IMDB Votes and Rating") +
  xlab("Votes (Thousands)") +
  ylab("Rating") +
  theme_bw()
```




```
ggplot(df.budget, aes(budget_1m, .fitted)) +
  geom_line(size = 1, color = "royalblue") +
  geom_ribbon(aes(ymin = lower, ymax = upper), alpha = .2) +
  ggtitle("Movie Budget and Rating") +
  xlab("Budget (Millions)") +
  ylab("Rating") +
  theme_bw()
```



13 Logistic Regression

This lab covers the basics of logistic regression, part of the broader generalized linear model family. GLMs relate a linear model to a response variable that does not have a normal distribution. Often you will use "logit" regression when working with a dependent variable that has limited responses, like a binary DV or an ordered DV. Logit regression uses Maximum Likelihood Estimation, which aims to identify the probability of obtaining the observed data as a function of the model parameters. The following packages are required for this lab:

1. tidyverse
2. psych
3. car
4. stargazer
5. reshape2
6. MASS
7. pscl
8. broom
9. DAMisc

13.1 Logistic Regression with a Binary DV

Recall from lab ten that we attempted to use OLS regression to explore the relationship between a number of independent variables and a vote for Trump. While using OLS provided useful information, some would consider logistic regression more appropriate in that instance. This is because of the binary DV (voted for

Trump or did not) that does not follow the normal distribution. Let's construct a logit regression model that explores how certain IVs predict a vote for Trump. First we need to recode and factor the candidate variable to make it binary and exclude candidates other than Trump and Clinton, where a vote for Trump is 1 and Clinton is 0. We start by factoring the variable:

```
ds$trump <- car::recode(ds$vote_cand, "0 = 1;1 = 0;else = NA;NA = NA")
ds$f.trump <- factor(ds$trump, levels = c(0, 1), labels = c("Clinton", "Trump"))
table(ds$f.trump)
```

```
##
## Clinton   Trump
##     722    1272
```

```
ds$f.party.2 <- factor(ds$f.party.2)
```

Next, select a subset of the data and remove missing observations:

```
ds.sub <- ds %>%
  dplyr::select("f.trump", "gender", "ideol", "income", "education", "race") %>%
  na.omit()
```

Build the generalized linear model:

```
logit1 <- glm(f.trump ~ ideol + gender + education + income, data = ds.sub,
              family = binomial(link = logit), x = TRUE)
summary(logit1)
```

```
##
## Call:
## glm(formula = f.trump ~ ideol + gender + education + income,
##     family = binomial(link = logit), data = ds.sub, x = TRUE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9402  -0.3615   0.2564   0.4587   2.8576
##
## Coefficients:
##                 Estimate   Std. Error z value             Pr(>|z|)
## (Intercept) -4.029590067  0.326208525 -12.353 < 0.0000000000000002 ***
## ideol        1.246695960  0.056725872  21.978 < 0.0000000000000002 ***
## gender       0.130433860  0.147900989   0.882                0.378
## education   -0.206200640  0.043473125  -4.743            0.0000021 ***
## income       0.000001201  0.000001315   0.913                0.361
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2381.2  on 1816  degrees of freedom
## Residual deviance: 1288.6  on 1812  degrees of freedom
## AIC: 1298.6
##
## Number of Fisher Scoring iterations: 5
```

The coefficients returned are logged odds, so there really is not much we can get from looking at them alone; however, from looking at the coefficients alone, we can tell that ideology and education both affect the probability of voting for Trump. In order to understand the sense of magnitude, we need to convert these

from logged odds to odds, and then to percentages. To convert logged odds to odds, take the exponent of the coefficients using the *exp()* function:

```
logit1 %>%
  coef() %>%
  exp()
```

```
## (Intercept)       ideol      gender   education      income
##  0.01778162  3.47882976  1.13932258  0.81366981  1.00000120
```

Odds are difficult to interpret intuitively, but to get a sense of what they're telling us, remember that odds greater than 1 indicate increased probability, and odds less than one indicate a decrease in probability. The statistically significant coefficients from the model are ideology and education. Based on the odds of each IV, we can tell that an increase in ideology improves the probability of a Trump vote, and an increase in education reduces the probability of a Trump vote. To get a more intuitive understanding, we can convert these to percentages. To do this you subtract the odds from 1. Do this for the ideology and education variables only, because those are the only siginificant ones:

Ideology:

```
1 - exp(logit1$coef[2])
```

```
##    ideol
## -2.47883
```

Education:

```
1- exp(logit1$coef[4])
```

```
## education
## 0.1863302
```

This may seem counter-intuitive, but since we subtracted 1 from the odds, a negative percentage is actually an increase in probability. The -2.48 for ideology can be interpreted as a 248% increase in the odds of voting for Trump. The point is that an increasing ideology score (liberal to conservative) drastically increased the probability of a vote for Trump. The .19 for education indicates that an increase in education decreases the odds of voting for Trump by about 19%.

Notice that even at this point, we are still dealing with some level of abstraction (a 248% increase in odds is hard to understand). Perhaps the best reason to use a logit model is that it allows us to generate predicted probabilities of some outcome. Similar to how we used OLS and the *predict()* function to describe and predict a relationship, logit regression allows us to obtain a predicted probability that a particular outcome occurs, given a certain set of parameters. In our case, we can generate predicted probabilities of voting for Trump. Let's find the predicted probabilities of voting for Trump as ideology increases and all other IVs are held constant at their means. We first need to generate some simulated data that sequences ideology from 1 to 7 and holds all other values at their means:

```
ideol.data <- with(ds, data.frame(education = mean(education, na.rm = T),
                                  gender = mean(gender, na.rm = T),
                                  income = mean(income, na.rm = T),
                                  ideol = 1:7))
ideol.data
```

```
##   education    gender   income ideol
## 1  5.023987 0.4029851 70641.77     1
## 2  5.023987 0.4029851 70641.77     2
## 3  5.023987 0.4029851 70641.77     3
## 4  5.023987 0.4029851 70641.77     4
## 5  5.023987 0.4029851 70641.77     5
## 6  5.023987 0.4029851 70641.77     6
```

```
## 7  5.023987 0.4029851 70641.77     7
```

Now use the `augment()` function to calculate predicted probabilities of voting for Trump at the various ideology levels. To do so, include `type.predict = "response"`. This tells `augment()` to generate predicted probabilities:

```
logit1 %>%
  augment(newdata = ideol.data, predict = "response")
```

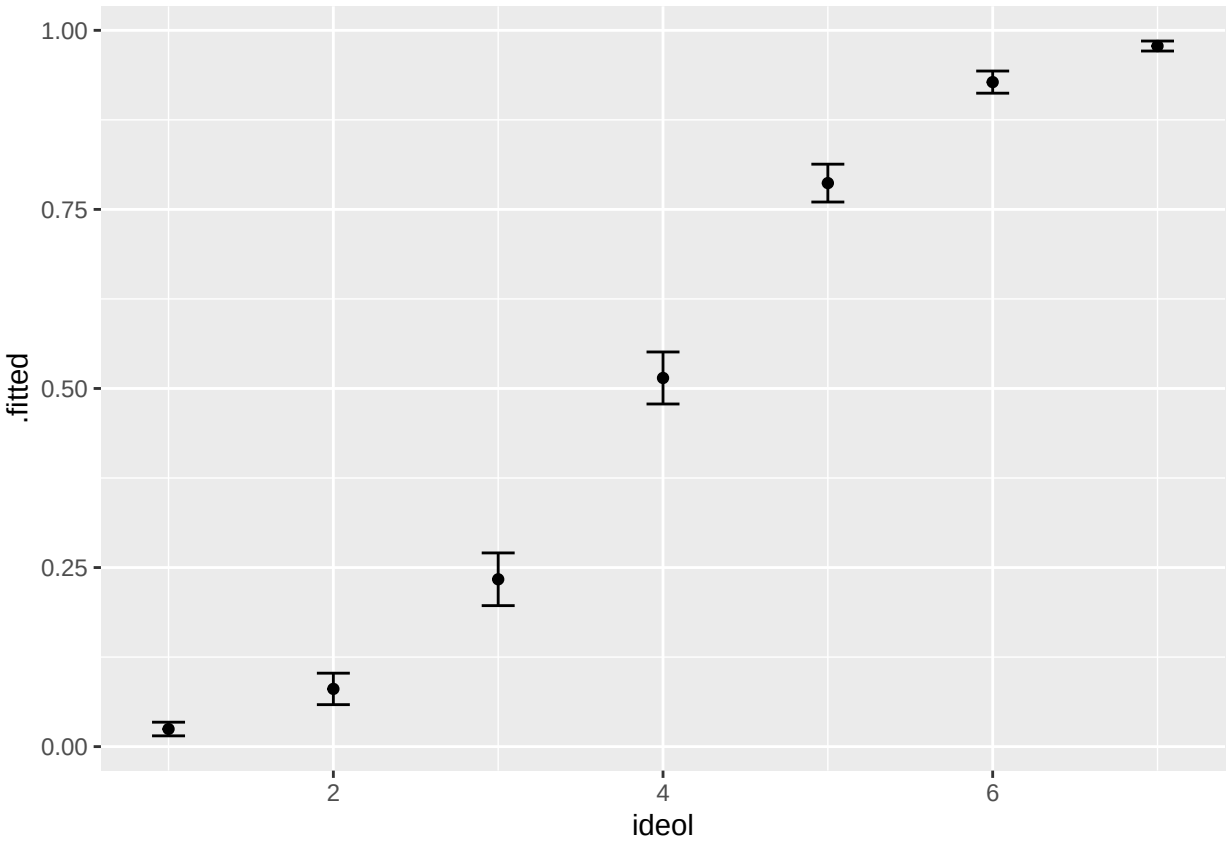
```
## # A tibble: 7 x 6
##   education gender income ideol .fitted .se.fit
##       <dbl>  <dbl>  <dbl> <int>   <dbl>   <dbl>
## 1      5.02  0.403 70642.     1 -3.68    0.203
## 2      5.02  0.403 70642.     2 -2.43    0.151
## 3      5.02  0.403 70642.     3 -1.19    0.105
## 4      5.02  0.403 70642.     4  0.0586  0.0742
## 5      5.02  0.403 70642.     5  1.31    0.0803
## 6      5.02  0.403 70642.     6  2.55    0.118
## 7      5.02  0.403 70642.     7  3.80    0.166
```

As we would likely expect, increasing ideology increases the probability of voting for Trump. At an ideology level of 7, there is almost a guarantee of voting for Trump. To get a sense of what this would look like, we can visualize these predicted probabilities rather easily. We need to calculate lower and upper bounds of the confidence interval first, which is done just like with other models. Assign the data frame to an object.

```
logit1 %>%
  augment(newdata = ideol.data, type.predict = "response") %>%
  mutate(upper = .fitted + 1.96 * .se.fit,
         lower = .fitted - 1.96 * .se.fit) -> log.data
```

Visualizing the predicted probabilities is similar to how we have visualized in the past. Use `geom_point()` and `geom_errorbar()`:

```
ggplot(log.data, aes(ideol, .fitted)) +
  geom_point(size = 1.5) +
  geom_errorbar(aes(ymin = lower, ymax = upper), width = .2)
```



13.1.1 Goodness of Fit, Logit Regression

Determining model fit when performing logit regression is different than when doing OLS. There are three main methods of exploring model fit, Pseudo-R squared, Log-likelihood, and AIC. The best way to understand logit model fit is by comparison, so let's create a null model that tries to predict a Trump vote only by the intercept term:

```
logit.null <- glm(f.trump ~ 1, data = ds.sub, family = binomial(link = logit))
summary(logit.null)
```

```
##
## Call:
## glm(formula = f.trump ~ 1, family = binomial(link = logit), data = ds.sub)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4232  -1.4232   0.9501   0.9501   0.9501
##
## Coefficients:
##             Estimate Std. Error z value            Pr(>|z|)
## (Intercept)  0.56135    0.04878   11.51 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2381.2  on 1816  degrees of freedom
```

```
## Residual deviance: 2381.2  on 1816  degrees of freedom
## AIC: 2383.2
##
## Number of Fisher Scoring iterations: 4
```

To test model fit via log-likelihood, we can calculate what is called the deviance statistic, or G squared. G squared tests whether the difference in the log-likelihoods of the null model and the demographic model (our initial model) are statistically distinguishable from zero. If so, our demographic model is a better fit. First let's the log-likelihoods for each model:

```r
loglikli.null <- logLik(logit.null)
loglikli.demo <- logLik(logit1)
```

To fing G squared, you subtract the null log-likelihood from the demographic log-likelihood:

```r
G <- loglikli.demo - loglikli.null
```

To test if the G statistic is significant, you use a Chi-Square test with q degrees of freedom, where q is the difference in the number of IVs in the model. Our demographic model has 4 IVs (ideology, age, education, income) and the null model has 1, so q is 3:

```r
pchisq(G, df = 3, lower.tail = FALSE)
```

```
## 'log Lik.' 0.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

We can conclude with confidence that the demographic model better explains a vote for Trump than the null model.

A similar approach can be made to compare nested models, similar to a nested F-test. Using the `anova()` function and specifying chi-squared, we can test if adding or subtracting a particular variable improves model fit. Let's include race in a new model and compare it to our first model:

```r
logit2 <- glm(f.trump ~ ideol + gender + education + income + race, data = ds.sub,
              family = binomial(link = logit), x = TRUE)
summary(logit2)
```

```
##
## Call:
## glm(formula = f.trump ~ ideol + gender + education + income +
##     race, family = binomial(link = logit), data = ds.sub, x = TRUE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8492  -0.3708   0.2520   0.4528   2.8359
##
## Coefficients:
##                 Estimate   Std. Error z value             Pr(>|z|)
## (Intercept) -3.737995285  0.331991444 -11.259 < 0.0000000000000002 ***
## ideol        1.259534233  0.057318244  21.974 < 0.0000000000000002 ***
## gender       0.121195088  0.149323750   0.812                0.417
## education   -0.200958175  0.043940078  -4.573            0.0000048 ***
## income       0.000001120  0.000001336   0.838                0.402
## race        -0.267085133  0.062456084  -4.276            0.0000190 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 2381.2  on 1816  degrees of freedom
## Residual deviance: 1270.4  on 1811  degrees of freedom
## AIC: 1282.4
##
## Number of Fisher Scoring iterations: 5
```

Now compare models:

```
anova(logit1, logit2, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: f.trump ~ ideol + gender + education + income
## Model 2: f.trump ~ ideol + gender + education + income + race
##   Resid. Df Resid. Dev Df Deviance   Pr(>Chi)
## 1      1812     1288.7
## 2      1811     1270.3  1   18.296 0.00001891 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The test indicates that including race improves the model.

Another way to examine model fit is pseudo-R squared. This is not completely analogous to R squared, because we're not trying to simply explain the variance in Y. However, pseudo-R squared compares the residual deviance of the null model to that of the actual model and ranges of 0 to 1, with higher values indicating better model fit. Deviance in a logit model is similar to the residual sum of squares in an OLS model. To find pseudo-R squared you take 1 minus the deviance of the actual model divided by the deviance of the null model. Let's use the new model that includes race:

```
psuedoR2 <- 1 - (logit2$deviance/logit2$null.deviance)
psuedoR2
```

```
## [1] 0.4665082
```

The final method we go over is AIC, or Akaine Information Criteria. AIC is only useful in comparing two models, and like adjusted R squared it penalizes for increased model parameters. Fortunately, AIC is calculated for us when we look at the summary of the model. A smaller AIC value indicates better model fit. Let's again compare the two actual logit models (not the null model)

```
stargazer(logit1, logit2, type="text", single.row = TRUE)
```

```
##
## ===================================================
##                          Dependent variable:
##                     ------------------------------
##                                 f.trump
##                            (1)               (2)
## ---------------------------------------------------
## ideol                1.247*** (0.057)  1.260*** (0.057)
## gender                 0.130 (0.148)     0.121 (0.149)
## education           -0.206*** (0.043) -0.201*** (0.044)
## income              0.00000 (0.00000) 0.00000 (0.00000)
## race                                  -0.267*** (0.062)
## Constant            -4.030*** (0.326) -3.738*** (0.332)
## ---------------------------------------------------
## Observations             1,817             1,817
## Log Likelihood         -644.325          -635.177
## Akaike Inf. Crit.      1,298.650         1,282.354
```

```
## ================================================
## Note:                    *p<0.1; **p<0.05; ***p<0.01
```

The AIC values indicate the the model including race is a better fit, which our log-likelihood test also indicated.

13.1.2 Percent Correctly Predicted

Another way to assess how effective our model is at describing and predicting our data is by looking at the percent correctly predicted. Using the `hitmiss()` function found in the `pscl` package, we can look at how well the model predicts the outcomes for when y and 0 and when y equals 1, and we can immediately compare it to how well a null model predicts outcomes. Let's do this for both the logit model that does not include race and the one that does:

```
hitmiss(logit1)
```

```
## Classification Threshold = 0.5
##        y=0  y=1
## yhat=0 498  119
## yhat=1 162 1038
## Percent Correctly Predicted = 84.53%
## Percent Correctly Predicted = 75.45%, for y = 0
## Percent Correctly Predicted = 89.71%  for y = 1
## Null Model Correctly Predicts 63.68%
```

```
## [1] 84.53495 75.45455 89.71478
```

```
hitmiss(logit2)
```

```
## Classification Threshold = 0.5
##        y=0  y=1
## yhat=0 497   99
## yhat=1 163 1058
## Percent Correctly Predicted = 85.58%
## Percent Correctly Predicted = 75.3%, for y = 0
## Percent Correctly Predicted = 91.44%  for y = 1
## Null Model Correctly Predicts 63.68%
```

```
## [1] 85.58063 75.30303 91.44339
```

It appears the model with race better predicts outcomes, which our other diagnostics so far have also suggested. One other method is to examine proportional reduction in error, which looks at how a model reduces error in predictions versus a null model. Let's look at the PRE for the logit model that includes race. To do so, use the `pre()` function from the `DAMisc` package:

```
pre(logit2)
```

```
## mod1:  f.trump ~ ideol + gender + education + income + race
## mod2:  f.trump ~ 1
##
## Analytical Results
##  PMC =  0.637
##  PCP =  0.856
##  PRE =  0.603
## ePMC =  0.537
## ePCP =  0.785
## ePRE =  0.535
```

The ? function can help you remember what all of the acronyms mean, but for now, know that PMC is the percent correctly predicted by the null model, PCP is percent correct predicted by the actual model, and PRE is proportional reduction in error. As all of our diagnostics have indicated, the actual model is better at predicting a vote for Trump than the null model.

13.1.3 Logit Regression with Groups

Now let's go over logit regression with groups. Let's continue looking into the probability of a vote for Trump, but let's include political party into the mix. We can use logit regression to find the probability of voting for Trump as ideology varies across political parties. Let's pull a new subset of the data that removes missing observations and includes the factored party variable:

```
ds.sub2 <- ds %>% dplyr::select("f.trump", "gender", "ideol", "income",
                                "education", "race", "f.party.2") %>%
  drop_na() #%>%
  #mutate(f.part = factor(f.party.2))
```

Notice that we used the factored party variable that only includes Democrats, Independents, and Republicans. Let's build the model:

```
logit3 <- glm(f.trump ~ ideol + gender + education + income + race + f.party.2,
              family = binomial(link = logit), data = ds.sub2, x = TRUE)
summary(logit3)
```

```
##
## Call:
## glm(formula = f.trump ~ ideol + gender + education + income +
##     race + f.party.2, family = binomial(link = logit), data = ds.sub2,
##     x = TRUE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.0531  -0.2845   0.1689   0.3204   2.9330
##
## Coefficients:
##                  Estimate   Std. Error z value             Pr(>|z|)
## (Intercept)  -3.118886176  0.370541003  -8.417 < 0.0000000000000002 ***
## ideol         0.901913313  0.063659790  14.168 < 0.0000000000000002 ***
## gender        0.094554608  0.173273089   0.546             0.585274
## education    -0.272118671  0.050596914  -5.378     0.00000007524784 ***
## income        0.000000066  0.000001596   0.041             0.967020
## race         -0.261920110  0.077118302  -3.396             0.000683 ***
## f.party.2Ind  1.465607702  0.214207780   6.842     0.00000000000781 ***
## f.party.2Rep  2.982488437  0.202712936  14.713 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2325.2  on 1769  degrees of freedom
## Residual deviance:  977.9  on 1762  degrees of freedom
## AIC: 993.9
##
## Number of Fisher Scoring iterations: 6
```

With Democrats as the reference group, Independents and Republicans have an increased probability of voting for Trump, which makes sense for Oklahoma voters. Next we generate predicted probabilities. First create data frames for each party:

```
rep.data <- with(ds.sub2, data.frame(gender = mean(gender),
                                     education = mean(education), race = mean(race),
                                     income = mean(income), ideol = (1:7),
                                     f.party.2 = c("Rep")))

dem.data <- with(ds.sub2, data.frame(gender = mean(gender),
                                         education = mean(education), race = mean(race),
                                         income = mean(income), ideol = (1:7),
                                         f.party.2 = c("Dem")))

ind.data <- with(ds.sub2, data.frame(gender = mean(gender),
                                         education = mean(education), race = mean(race),
                                         income = mean(income), ideol = (1:7),
                                         f.party.2 = c("Ind")))
```

Now we can calculate predicted probabilities of voting for Trump for each party and ideology score, holding all other IVs constant, as well as upper and lower bounds of the confidence intervals:

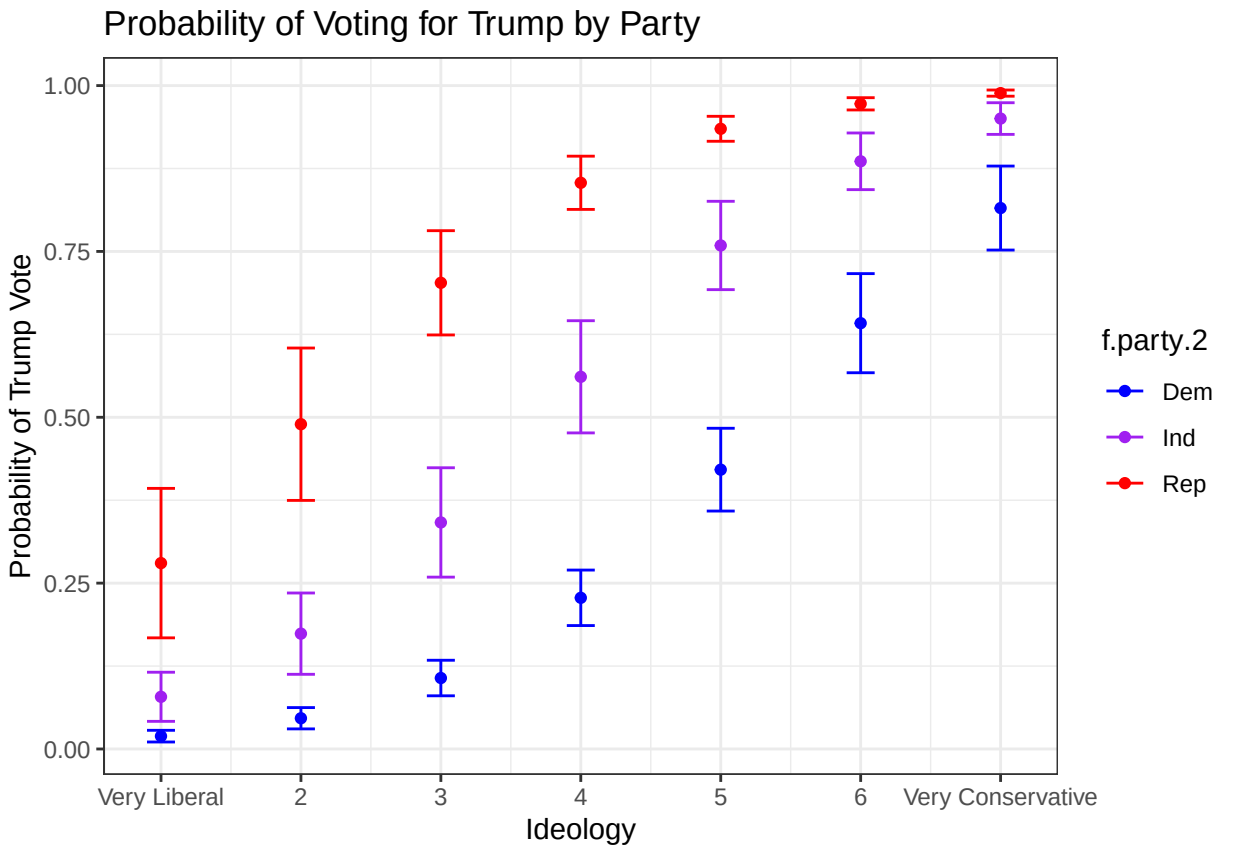
```
rep.prob <- augment(logit3, newdata = rep.data, type.predict = "response") %>%
                    mutate(upper = .fitted + 1.96 * .se.fit,
                           lower = .fitted - 1.96 * .se.fit)
dem.prob <-  augment(logit3, newdata = dem.data, type.predict = "response") %>%
                     mutate(upper = .fitted + 1.96 * .se.fit,
                           lower = .fitted - 1.96 * .se.fit)
ind.prob <-  augment(logit3, newdata = ind.data, type.predict = "response") %>%
                     mutate(upper = .fitted + 1.96 * .se.fit,
                           lower = .fitted - 1.96 * .se.fit)
```

Now we combine everything into one data frame using `rbind()`.

```
df.party <- rbind(dem.prob, ind.prob, rep.prob)
```

Start by building the visualization. This will be similar to the last visualization, but we plot predicted probabilities for each ideology score in each party, so 21 points in all. We have everything we need to make a great visualization. We plot the points and error bars just like we did last time, but we assign colors by political party, and specify blue for Democrats, purple for Independents, and red for Republicans:

```
ggplot(df.party, aes(ideol, .fitted, color = f.party.2)) +
  geom_point(size = 1.5) +
  geom_errorbar(aes(ymin = lower, ymax = upper), width = .2) +
  scale_color_manual(values = c("blue", "purple", "red")) +
  ggtitle("Probability of Voting for Trump by Party") +
  scale_x_continuous(breaks=c(1:7),
                     labels = c("Very Liberal", "2", "3", "4", "5",
                                "6", "Very Conservative")) +
  xlab("Ideology") +
  ylab("Probability of Trump Vote") +
  theme_bw()
```



13.2 Ordered Logit and Creating an Index

Logit regression can be used in more than just situations with a binary DV. Ordered logit analysis is done in a similar way, but with an ordered DV. Instead of simply assessing the probability of one outcome, ordered logit analysis gives us the probability of moving from one level of an ordered categorical variable to the next. We're going to use ordered logit analysis to also learn how to create an index and work with one as your dependent variable.

The class data set survey includes responses that indicate whether or not the participant does a number of energy-saving activities at their home, like turning the lights off, installing insulation, unplugging appliances, etc. Perhaps you are interested in how a variety of IVs influence one's propensity to do these activities. You could use binary logit regression to find the probability of individuals doing one of these particular activities. However, you could use ordered logit regression to include all the activities and use an additive index of them as your dependent variable. Start by creating an index of the energy-saving activities:

```
energy <- with(ds, cbind(enrgy_steps_lghts, enrgy_steps_heat, enrgy_steps_ac,
                         enrgy_steps_savappl, enrgy_steps_unplug, enrgy_steps_insul,
                         enrgy_steps_savdoor, enrgy_steps_bulbs))
```

Now take a look at the index:

```
psych::describe(energy)
```

```
##                    vars    n mean   sd median trimmed mad min max range
## enrgy_steps_lghts     1 2547 0.70 0.46      1    0.75   0   0   1     1
## enrgy_steps_heat      2 2547 0.60 0.49      1    0.62   0   0   1     1
## enrgy_steps_ac        3 2547 0.60 0.49      1    0.62   0   0   1     1
## enrgy_steps_savappl   4 2547 0.12 0.33      0    0.03   0   0   1     1
```

```
## enrgy_steps_unplug     5 2547 0.23 0.42      0    0.16   0   0   1     1
## enrgy_steps_insul      6 2547 0.07 0.25      0    0.00   0   0   1     1
## enrgy_steps_savdoor    7 2547 0.07 0.26      0    0.00   0   0   1     1
## enrgy_steps_bulbs      8 2547 0.55 0.50      1    0.56   0   0   1     1
##                       skew kurtosis   se
## enrgy_steps_lghts   -0.88    -1.22 0.01
## enrgy_steps_heat    -0.40    -1.84 0.01
## enrgy_steps_ac      -0.41    -1.83 0.01
## enrgy_steps_savappl  2.32     3.38 0.01
## enrgy_steps_unplug   1.27    -0.38 0.01
## enrgy_steps_insul    3.40     9.54 0.01
## enrgy_steps_savdoor  3.29     8.84 0.01
## enrgy_steps_bulbs   -0.18    -1.97 0.01
```

Add these variables together. This will create an index that scores 1 if an individual does one of the activities, 2 if they do two, and so on and so on:

```
ds$s.energy <- with(ds, enrgy_steps_lghts + enrgy_steps_heat + enrgy_steps_ac +
                      enrgy_steps_savappl + enrgy_steps_unplug + enrgy_steps_insul +
                      enrgy_steps_savdoor + enrgy_steps_bulbs)
```

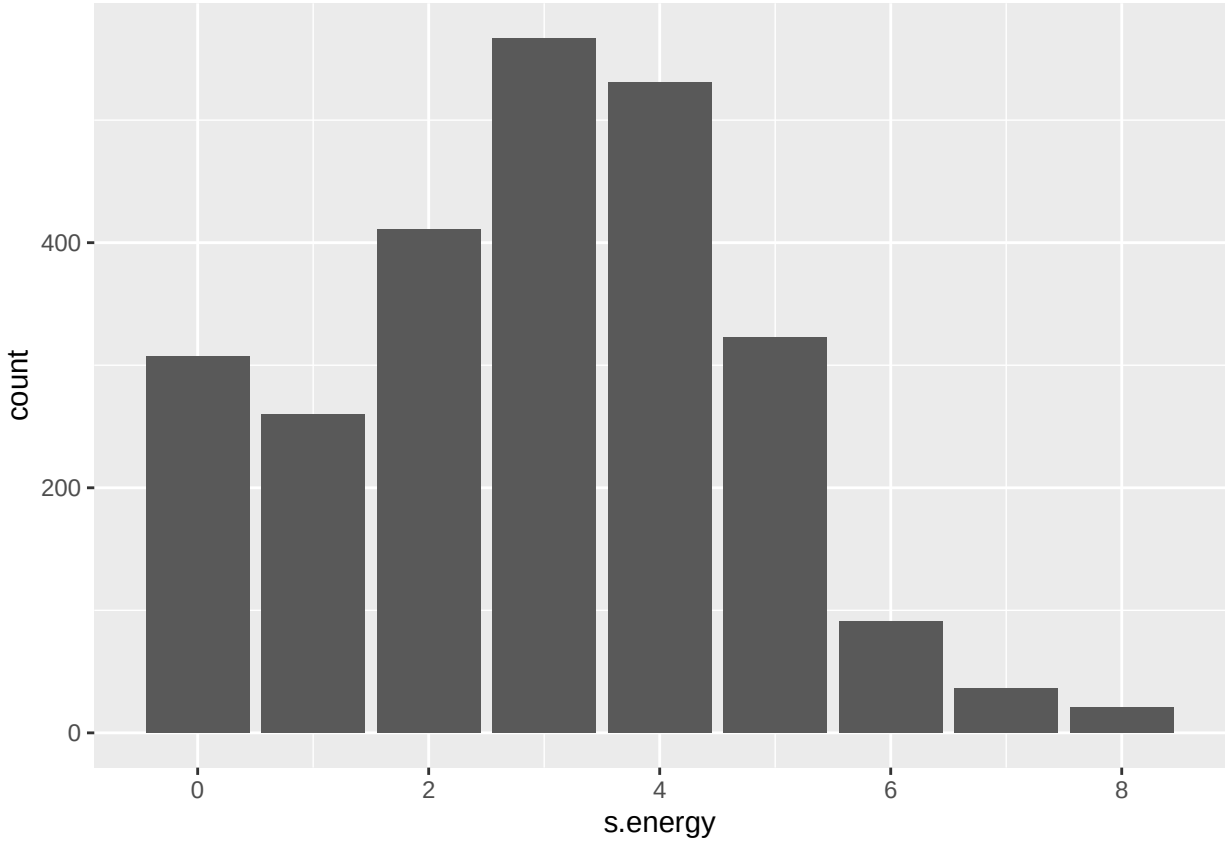
Examine the index:

```
psych::describe(ds$s.energy)
```

```
##    vars    n mean   sd median trimmed  mad min max range skew kurtosis
## X1    1 2547 2.94 1.77      3    2.94 1.48   0   8     8 0.08    -0.38
##      se
## X1 0.04
```

Make a bar plot of the index:

```
ggplot(ds, aes(s.energy)) +
  geom_bar()
```



Start building the model. First select our relevant variables and remove missing observations:

```
ds.sub3 <- ds %>% dplyr::select("s.energy", "ideol", "age", "glbcc_risk") %>%
  na.omit()
```

In order to use the energy index as a dependent variable, we treat it as a factor:

```
ds.sub3$f.energy <- as.factor(ds.sub3$s.energy)
```

There are a number of ways to do ordered logit, but for this example we use the `polr()` function found in the `MASS` package.:

```
ord1 <- polr(f.energy ~ ideol + age + glbcc_risk, data = ds.sub3, Hess = TRUE)
```

Use `stargazer()` to look at the results:

```
stargazer(ord1, type="text", style="apsr", single.row = T)
```

```
##
## ------------------------------
##                     f.energy
## ------------------------------
## ideol            0.049* (0.025)
## age              -0.002 (0.002)
## glbcc_risk     0.098*** (0.014)
## N                   2,513
## ------------------------------
## *p < .1; **p < .05; ***p < .01
```

The results indicate an increased risk associated with climate change corresponds with an an increase in the

odds of doing enery-saving techniques at home. When doing ordered logit, coefficient interpretation is even less intuitive than it is with a binary DV. This makes generating predicted probabilities even more important. We are going to generate predicted probabilities of each level of the DV (0 through 8) as perceived climate change risk increases and the other IVs are held constant at their means. It should make sense that we have to do it this way. We can't really explain the relationship in any other way with the information we have.

First we create a data frame to work with:

```
ord.df <- data.frame(ideol = mean(ds.sub3$ideol),
                     age = mean(ds.sub3$age),
                     glbcc_risk = seq(0, 10, 1))
```

Now we use the `predict()` function to generate predicted probabilities of each level of the DV as we sequence climate change risk from 0 to 10. The `augment()` function does not work with the `polr()` function, so that is why we are using `predict()`.

```
prob.df <- cbind(ord.df, predict(ord1, ord.df, type = "probs"))
prob.df
```

```
##       ideol      age glbcc_risk          0          1         2         3
## 1  4.656188 60.37724          0 0.19435827 0.14196855 0.1912928 0.2082757
## 2  4.656188 60.37724          1 0.17942987 0.13532435 0.1883238 0.2132792
## 3  4.656188 60.37724          2 0.16541268 0.12853979 0.1845691 0.2174494
## 4  4.656188 60.37724          3 0.15228730 0.12168852 0.1800926 0.2207161
## 5  4.656188 60.37724          4 0.14002866 0.11483803 0.1749682 0.2230233
## 6  4.656188 60.37724          5 0.12860710 0.10804884 0.1692771 0.2243305
## 7  4.656188 60.37724          6 0.11798933 0.10137399 0.1631054 0.2246146
## 8  4.656188 60.37724          7 0.10813937 0.09485887 0.1565416 0.2238706
## 9  4.656188 60.37724          8 0.09901939 0.08854133 0.1496742 0.2221116
## 10 4.656188 60.37724          9 0.09059043 0.08245186 0.1425895 0.2193687
## 11 4.656188 60.37724         10 0.08281302 0.07661412 0.1353698 0.2156896
##            4          5          6           7           8
## 1  0.1534703 0.07825448 0.02043592 0.007422660 0.004521296
## 2  0.1629619 0.08507611 0.02244353 0.008175297 0.004985885
## 3  0.1725244 0.09236696 0.02463709 0.009002686 0.005497949
## 4  0.1820742 0.10013563 0.02703147 0.009911920 0.006062283
## 5  0.1915187 0.10838602 0.02964226 0.010910688 0.006684154
## 6  0.2007578 0.11711639 0.03248565 0.012007320 0.007369344
## 7  0.2096849 0.12631840 0.03557834 0.013210813 0.008124197
## 8  0.2181896 0.13597605 0.03893738 0.014530868 0.008955674
## 9  0.2261594 0.14606476 0.04258003 0.015977912 0.009871402
## 10 0.2334828 0.15655041 0.04652346 0.017563123 0.010879736
## 11 0.2400522 0.16738852 0.05078447 0.019298445 0.011989821
```

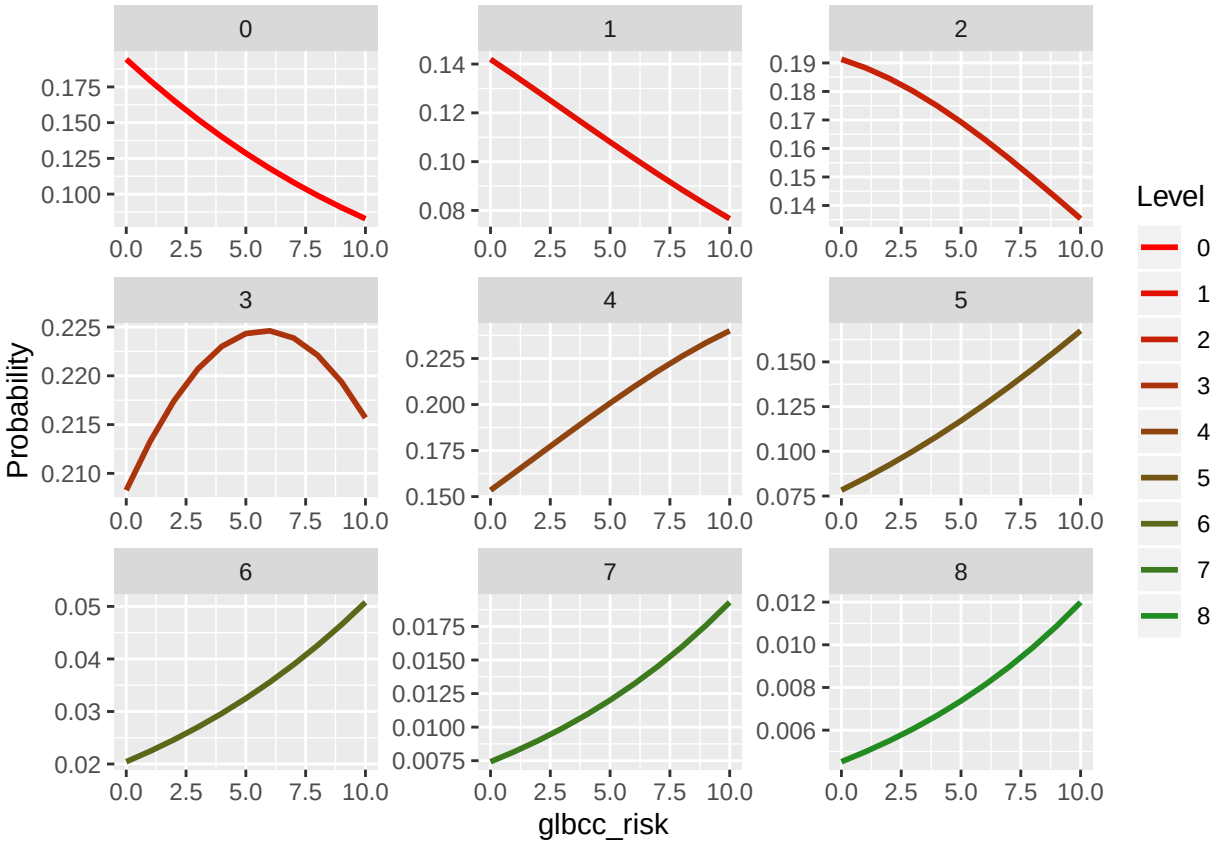
The next step is to melt the data. This will allow us to eventually generate a prediction line for each level of the DV:

```
m.df <- melt(prob.df, id.vars = c("ideol", "age", "glbcc_risk"),
             variable.name = "Level", value.name = "Probability")
```

Next we will create the visualization. With an ordered logit model, we can visualize predicted probabilities of observing each separate level of the DV (how many energy saving activities), as perceived climate change risk increases. We use `facet_wrap()` to create individaul visualizations for each level of the DV, so that the graphic does not get too hard to interpret. We also create a color scale that goes from red to green.

```
col_scale<-colorRampPalette(c("#FF0000","#228B22"))(9)
ggplot(m.df, aes(x = glbcc_risk, y = Probability, colour = Level)) +
```

```
  geom_line(size = 1) +
  scale_color_manual(values = col_scale) +
  facet_wrap( ~ Level, scales = "free")
```



Taking a quick look at these visualizations, we can see that for doing 0 to 2 energy saving activities at home, increasing climate change risk largely corresponds with decreasing probability. This makes sense. Once we reach 4 energy saving activities, increasing climate change risk largely corresponds with increased probabilities.

14 Statistical Simulations

So far we have covered many different techniques of statistical analysis. This lab will cover the basics of statistical simulations. As a foundation for this, we will be using the various functions associated with the Zelig Project. Zelig provides a method of simulating outcomes based on certain parameters, but it also provides a different way of constructing almost any model in R. The following packages are required for this lab:

1. tidyverse
2. psych
3. car
4. stargazer
5. reshape2
6. MASS
7. zeligverse

14.1 The Basics

The basics of Zelig can be broken down into four steps:

1. `zelig()` function to estimate parameters.
2. `setx()` to set values.
3. `sim()` to simulate the quantities of interest.
4. `plot()` to visualize the simulation results.

As the lab progresses, we will amend the steps, but these are almost always a good place to start.

First create a subset of data and remove missing observations.

```
ds.sub <- ds %>%
  dplyr::select("footage", "income", "education", "age") %>%
  na.omit()
```

Use OLS regression to look at the relationship between square footage of a home (DV) and the IVs income, age, and education. First we use `zelig()` to specify the model, indicate `model="ls"`. For the income variable use logged income. As we've shown many times throughout the lab, the income variable has a skew.

```
ds.sub$log.inc <- log(ds.sub$income)
ols1 <- zelig(footage ~ log.inc + age + education, data = ds.sub, model = "ls", cite = FALSE)
```

Just like any other model, we look at the results using the usual methods. Let's start with `summary()`:

```
summary(ols1)
```

```
## Model:
##
## Call:
## z5$zelig(formula = footage ~ log.inc + age + education, data = ds.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1951.5  -526.0  -176.6   296.6 18199.8
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -4598.212    403.704 -11.390 < 0.0000000000000002
## log.inc       536.881     36.366  14.763 < 0.0000000000000002
## age             9.633      1.695   5.683          0.000000015
## education      32.968     14.126   2.334               0.0197
##
## Residual standard error: 1111 on 2232 degrees of freedom
## Multiple R-squared:  0.1186, Adjusted R-squared:  0.1174
## F-statistic: 100.1 on 3 and 2232 DF,  p-value: < 0.00000000000000022
##
## Next step: Use 'setx' method
```

As we could have guessed, increased income corresponds with increased square footage, as does age and education. Let's further explore education and square footage while holding logged income and age at their means. First take a look at the education variable:

```
table(ds.sub$education)
```

```
##
##   1   2   3   4   5   6   7   8
##  24 270 110 456 180 653 406 137
```

According to the code book, a 2 indicates a High School education and 6 indicates a Bachelor's degree. We can set the x value of education to both 2 and 6, and then have Zelig run Monte Carlo simulations, creating quantities of interest that we can compare. Use `setx()` and `setx1()` to set the two x values:

```
ols1.ed <- setx(ols1, education = 2)
ols1.ed <- setx1(ols1.ed, education = 6)
ols1.ed
```

```
## setx:
##   (Intercept) log.inc  age education
## 1           1    10.9 59.9         2
## setx1:
##   (Intercept) log.inc  age education
## 1           1    10.9 59.9         6
##
## Next step: Use 'sim' method
```

The next step is to use the `Zelig::sim()` function. This will use Monte Carlo simulations to generate quantiles of interest at each of the specified levels. In the past we might have predicted a Y value based on a model, but this will allow us to see mean, standard deviation, and more, based on 1000 simulations at each level of x.

```
ols.sim <- Zelig::sim(ols1.ed)
summary(ols.sim)
```

```
##
##  sim x :
##  -----
## ev
##       mean       sd      50%     2.5%   97.5%
## 1 1915.526 49.33007 1916.169 1822.868 2011.28
## pv
##          mean      sd      50%      2.5%    97.5%
## [1,] 1947.017 1083.78 1944.384 -261.5647 4044.672
##
##  sim x1 :
##  -----
## ev
##       mean       sd      50%     2.5%    97.5%
## 1 2046.847 26.29678 2047.144 1995.798 2100.174
## pv
##          mean       sd      50%      2.5%    97.5%
## [1,] 2131.777 1127.846 2097.634 -53.42045 4340.725
## fd
##       mean       sd      50%     2.5%    97.5%
## 1 131.3212 56.90982 131.8491 26.07368 241.1586
```

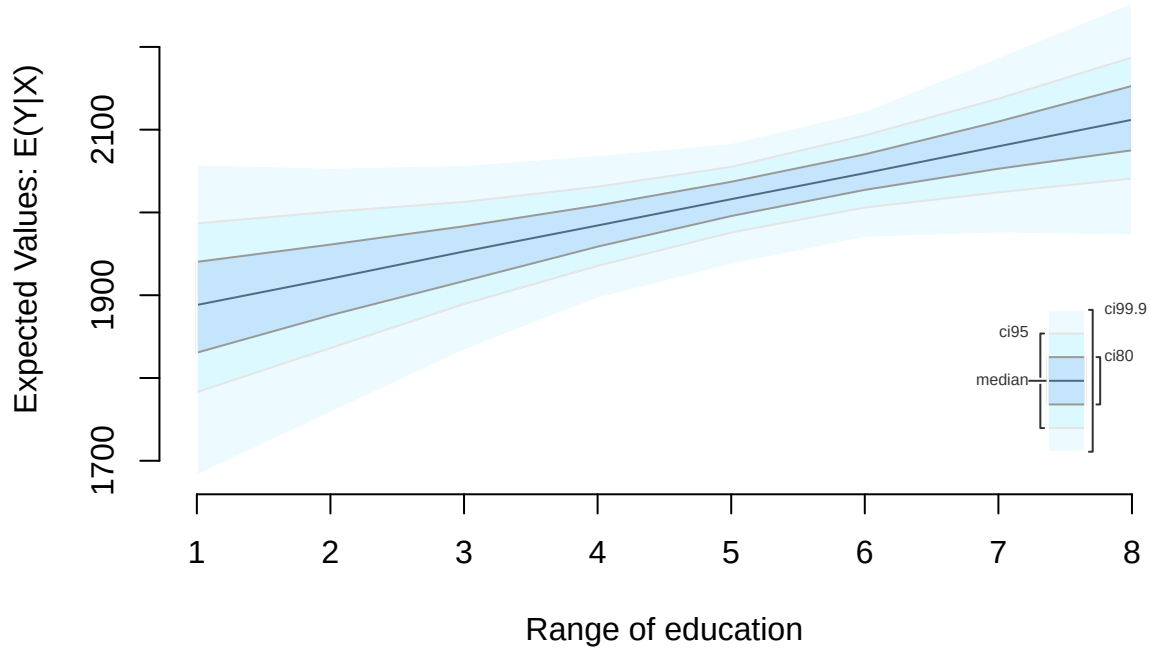
Next we use `plot()` to visualize the various QIs. I recommend clicking the "Show in New Window" button in the top right corner of the space below. **Note:** This is not visible via the knit PDF.

```
plot(ols.sim)
```

```
## Error in plot.new(): figure margins too large
```

We can use Zelig to simulate based on a range of values, similar to how we would sequence one independent variable when we would predict values in past labs. Let's simulate the data for the whole range of education values, from 1 to 8. We use the pipe operator, %>%, to simplify the syntax.

```
ols2.ed <- setx(ols1, education=1:8) %>%
  Zelig::sim()
plot(ols2.ed)
```



14.1.1 Plotting Predictions with Zelig

Zelig is useful in another way: plotting predictions. By adding only two more steps to the process, R can return a data frame of information in the tidyverse format that we plot with `ggplot2`. Let's plot predicted values of Y, square footage of home, by each education level. We've got the simulated values, so use `zelig_qi_to_df()` to transform the data into a data frame, and use `qi_slimmer()` to slim the values and generate confidence intervals for each point:

```
ols.ed.df <- zelig_qi_to_df(ols2.ed) %>%
  qi_slimmer()
```

```
## Slimming Expected Values . . .
```

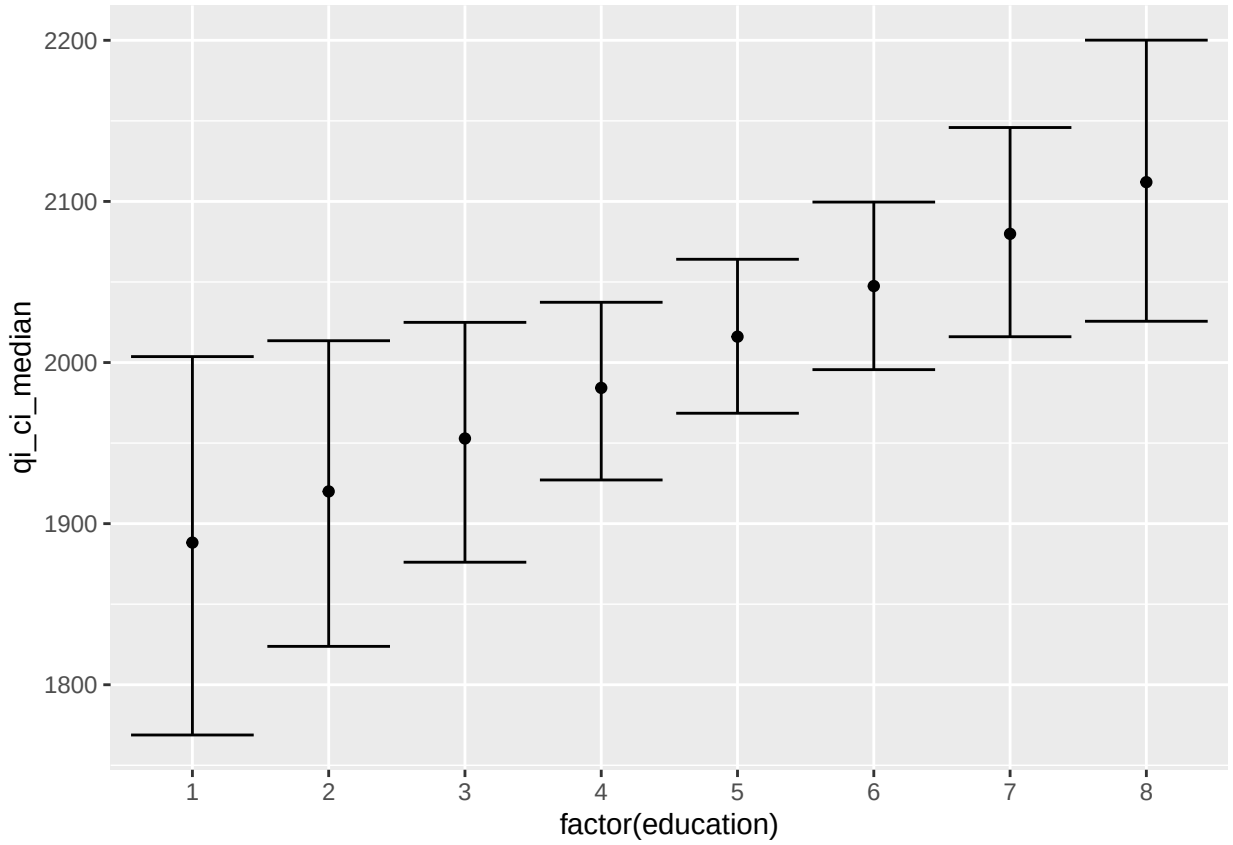
Take a look at the new data:

```
ols.ed.df
```

```
##   setx_value  log.inc      age education qi_ci_min qi_ci_median qi_ci_max
## 1          x 10.93598 59.94007         1  1768.807     1888.237  2003.685
## 2          x 10.93598 59.94007         2  1823.811     1919.995  2013.536
## 3          x 10.93598 59.94007         3  1876.057     1952.843  2024.939
## 4          x 10.93598 59.94007         4  1927.109     1984.267  2037.408
## 5          x 10.93598 59.94007         5  1968.526     2016.057  2064.091
## 6          x 10.93598 59.94007         6  1995.594     2047.471  2099.608
## 7          x 10.93598 59.94007         7  2016.014     2079.884  2145.876
## 8          x 10.93598 59.94007         8  2025.621     2111.976  2200.112
```

Logged income and age are held at their means, education is sequenced from 1 to 8, and there are three other groups of values. `Qi_ci_min` is the lower limit, `qi_ci_median` is the estimate, and `qi_ci_max` is the upper limit. Let's plot them. Make sure to use `factor(education)` to treat each level separately.

```
ggplot(ols.ed.df, aes(factor(education), qi_ci_median)) +
  geom_errorbar(aes(ymin = qi_ci_min, ymax = qi_ci_max)) +
  geom_point()
```



14.2 Other Models

Zelig can be utilized for many different types of models. Let's run through an example of logistic regression. For instance, we can run through the example we used in the last lab, predicting a vote for Trump.

```
ds$trump <- car::recode(ds$vote_cand, "0 = 1;1 = 0;else = NA;NA = NA")
```

Subset the data and remove missing observations:

```
ds.sub <- ds %>% dplyr::select("footage", "trump", "gender",
                               "ideol", "income", "education", "race", "age") %>%
  na.omit() %>%
  mutate(log.inc = log(income))
```

Build a model that includes gender, ideology, logged income, education, and race. To indicate a logit model, include `model="logit"`:

```
z.log <- zelig(trump ~ gender + ideol + log.inc + education + race, data = ds.sub, model="logit", cite=
summary(z.log)
```

```
## Model:
##
## Call:
## z5$zelig(formula = trump ~ gender + ideol + log.inc + education +
```

```
##     race, data = ds.sub)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8637  -0.3625   0.2492   0.4504   2.8749
##
## Coefficients:
##             Estimate Std. Error z value             Pr(>|z|)
## (Intercept) -4.74842    1.23843  -3.834             0.000126
## gender       0.10011    0.15174   0.660             0.509394
## ideol        1.27104    0.05868  21.661 < 0.0000000000000002
## log.inc      0.10294    0.11766   0.875             0.381643
## education   -0.21835    0.04648  -4.698          0.00000263
## race        -0.25301    0.06389  -3.960          0.00007500
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2322.4  on 1772  degrees of freedom
## Residual deviance: 1225.2  on 1767  degrees of freedom
## AIC: 1237.2
##
## Number of Fisher Scoring iterations: 5
##
## Next step: Use 'setx' method
```

Since we used ideology in the last lab, let's find the predicted probabilities of voting for Trump based on education levels:

```
log.out <- setx(z.log, education = 1:8) %>%
  Zelig::sim() %>%
  zelig_qi_to_df() %>%
  qi_slimmer()
```

```
## Slimming Expected Values . . .
```

Now take a look at the data frame:

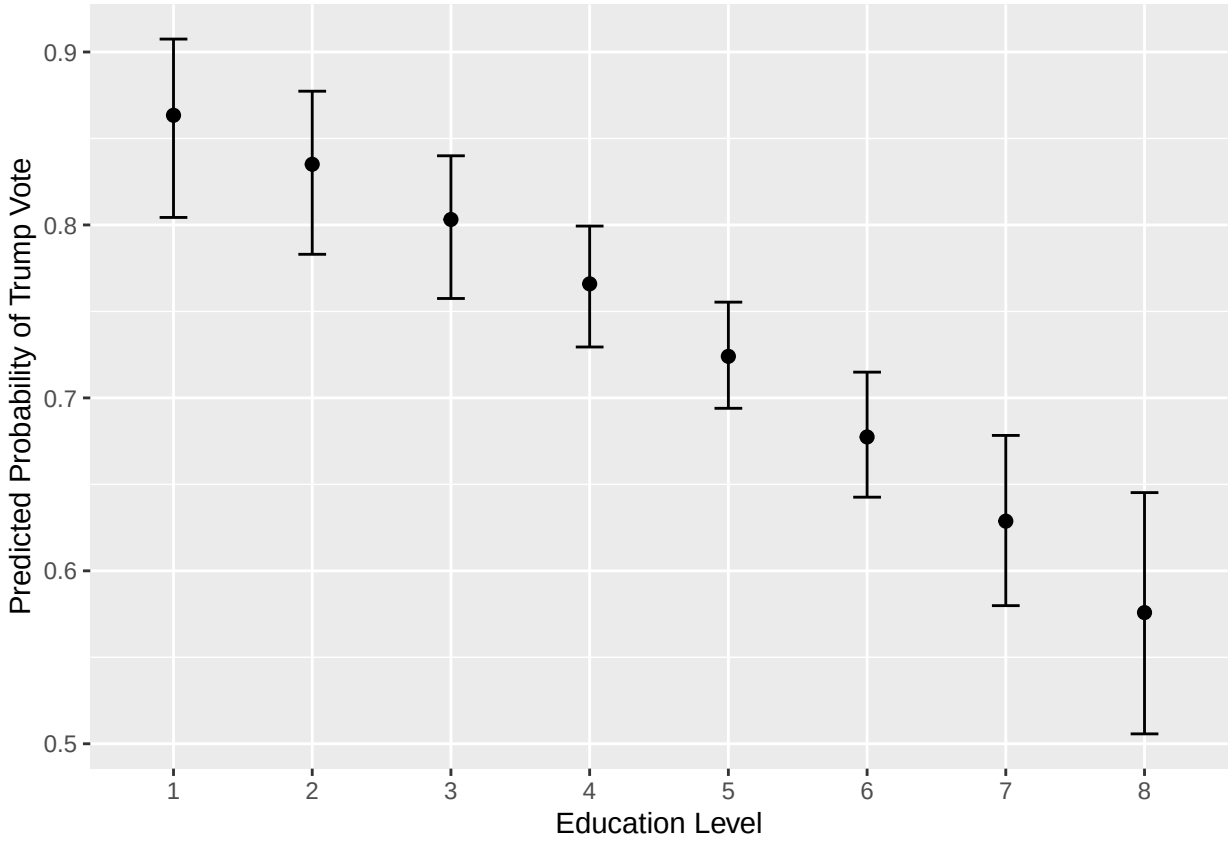
```
log.out
```

```
##   setx_value    gender    ideol  log.inc education     race qi_ci_min
## 1          x 0.4179357 4.695431 10.97955         1 1.320361 0.8043388
## 2          x 0.4179357 4.695431 10.97955         2 1.320361 0.7830302
## 3          x 0.4179357 4.695431 10.97955         3 1.320361 0.7574758
## 4          x 0.4179357 4.695431 10.97955         4 1.320361 0.7294318
## 5          x 0.4179357 4.695431 10.97955         5 1.320361 0.6939839
## 6          x 0.4179357 4.695431 10.97955         6 1.320361 0.6425855
## 7          x 0.4179357 4.695431 10.97955         7 1.320361 0.5798745
## 8          x 0.4179357 4.695431 10.97955         8 1.320361 0.5057178
##   qi_ci_median qi_ci_max
## 1    0.8634084 0.9074866
## 2    0.8350942 0.8773442
## 3    0.8031580 0.8399864
## 4    0.7659425 0.7993544
## 5    0.7240259 0.7553631
## 6    0.6774094 0.7148882
## 7    0.6287197 0.6782930
```

```
## 8    0.5758595 0.6452315
```

Next make the visualization.

```
ggplot(log.out, aes(factor(education), qi_ci_median))+
  geom_errorbar(aes(ymin = qi_ci_min, ymax = qi_ci_max), width = .2) +
  geom_point(size = 2) +
  ylab("Predicted Probability of Trump Vote") +
  xlab("Education Level")
```



14.2.1 Ordered Logit

Recall that in the last lab, we created an index of energy-saving activities and used ordered logit to assess the probability of individuals doing the activities based on their perceived climate change risk. Let's revisit that model and go one step farther than we did last time. Instead of looking at predicted probabilities, we will use Zelig to simulate predicted values, actually predicting the number of energy-saving activities individuals do based on their perceived climate change risk.

First create the index again:

```
energy <- with(ds, cbind(enrgy_steps_lghts, enrgy_steps_heat, enrgy_steps_ac,
                         enrgy_steps_savappl, enrgy_steps_unplug, enrgy_steps_insul,
                         enrgy_steps_savdoor, enrgy_steps_bulbs))

ds$s.energy <- with(ds, enrgy_steps_lghts + enrgy_steps_heat + enrgy_steps_ac +
                        enrgy_steps_savappl + enrgy_steps_unplug + enrgy_steps_insul +
                        enrgy_steps_savdoor + enrgy_steps_bulbs)
```

Subset the data and remove missing observations:

```
ds.sub3 <- ds %>%
  dplyr::select("s.energy", "ideol", "age", "glbcc_risk") %>%
  na.omit()
```

Create a factored version of the index:

```
ds.sub3$f.energy <- factor(ds.sub3$s.energy)
```

Build the model using `model="ologit"`:

```
logit1 <- zelig(f.energy ~ ideol + age + glbcc_risk, data=ds.sub3,
                model = "ologit", cite=FALSE)
```

Let's review the results:

```
summary(logit1)
```

```
## Model:
## Call:
## z5$zelig(formula = f.energy ~ ideol + age + glbcc_risk, data = ds.sub3)
##
## Coefficients:
##                Value Std. Error t value
## ideol       0.048599   0.025282  1.9223
## age        -0.002486   0.002491 -0.9978
## glbcc_risk  0.098279   0.014475  6.7895
##
## Intercepts:
##     Value   Std. Error t value
## 0|1 -1.3457  0.2420    -5.5618
## 1|2 -0.6035  0.2393    -2.5219
## 2|3  0.1868  0.2382     0.7841
## 3|4  1.1010  0.2389     4.6087
## 4|5  2.1605  0.2421     8.9254
## 5|6  3.4735  0.2524    13.7609
## 6|7  4.4917  0.2743    16.3742
## 7|8  5.4706  0.3237    16.8994
##
## Residual Deviance: 9616.121
## AIC: 9638.121
## Next step: Use 'setx' method
```

We're primarily interested in the relationship between climate change risk and energy-saving activities. Normally the next step would be to sequence climate change risk from one to ten and generate predicted probabilities, but we already did that in the last lab. This time, let's use Zelig to generate predicted values. These next steps might get a little messy, so here they are:

1. Use `setx()` and `Zelig::sim()` to simulate values for each level of climate change risk. Then use `get_qi()` to extract the predicted values. We have to do this for each level separately.
2. Put all the predicted values into a data frame:
3. Use `melt()` to melt the data frame into long form:
4. Use ggplot2 and `facet_wrap()` to create bar plots of the predicted values.

We can do steps one and two together in one line of code by piping:

```
pv.0 <- setx(logit1, glbcc_risk = 0) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")
```

```
pv.1 <- setx(logit1, glbcc_risk = 1) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")

pv.2 <- setx(logit1, glbcc_risk = 2) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")

pv.3 <- setx(logit1, glbcc_risk = 3) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")

pv.4 <- setx(logit1, glbcc_risk = 4) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")

pv.5 <- setx(logit1, glbcc_risk = 5) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")

pv.6 <- setx(logit1, glbcc_risk = 6) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")

pv.7 <- setx(logit1, glbcc_risk = 7) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")

pv.8 <- setx(logit1, glbcc_risk = 8) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")

pv.9 <- setx(logit1, glbcc_risk = 9) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")

pv.10 <- setx(logit1, glbcc_risk = 10) %>% Zelig::sim() %>% get_qi(qi = "pv", xvalue = "x")
```

Put the predicted values into a data frame:

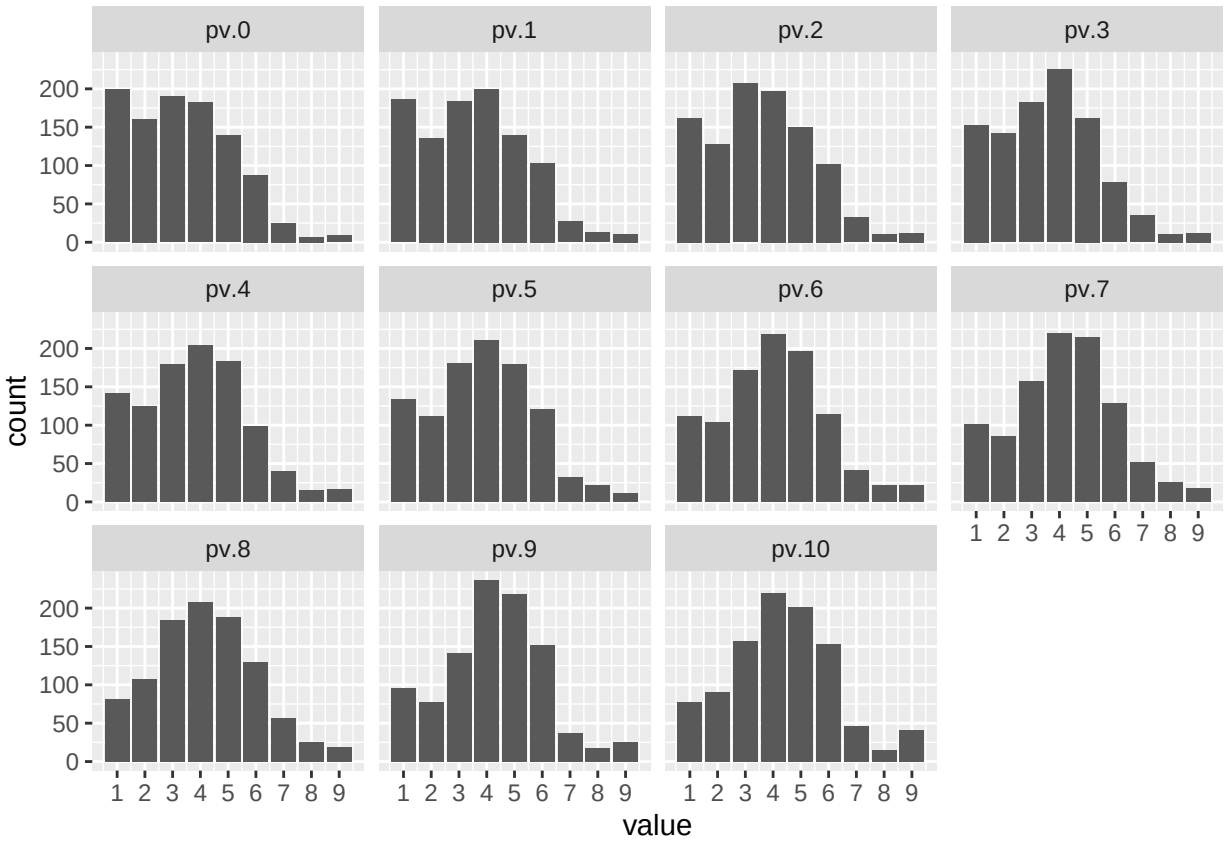
```
pv.df <- data.frame(pv.0, pv.1, pv.2, pv.3, pv.4, pv.5,
                    pv.6, pv.7, pv.8, pv.9, pv.10)
```

Melt the data:

```
pv.m <- melt(pv.df, measure.vars = c("pv.0", "pv.1", "pv.2","pv.3","pv.4","pv.5",
                                     "pv.6","pv.7","pv.8","pv.9","pv.10"))
```

Plot the predicted values. Remember, these are the predicted values that Zelig found by doing 1000 simulations at each level, not just one predicted value. Use `geom_bar()` to bar plots:

```
ggplot(pv.m, aes(value)) +
  geom_bar() +
  facet_wrap( ~ variable, scales = "fixed") +
  scale_x_continuous(breaks = c(0:10))
```



We can deduce from this visualization that the skew shifts more negative as climate change risk increases, indicating that individuals more concerned about climate change are doing more energy-saving activities.

14.2.2 Another Example

Let's go back to the example model that regressed home square footage on logged income, age, and education. Recall the model:

```
ols1 <- zelig(footage ~ log.inc + age + education, data = ds.sub,
              model = "ls", cite = FALSE)
summary(ols1)
```

```
## Model:
##
## Call:
## z5$zelig(formula = footage ~ log.inc + age + education, data = ds.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1900.9  -527.6  -178.5   312.6 17418.2
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -5121.208    456.914 -11.208 < 0.0000000000000002
## log.inc       592.457     40.386  14.670 < 0.0000000000000002
## age             9.209      1.934   4.762              0.00000207
## education      24.542     15.564   1.577                   0.115
```

```
##
## Residual standard error: 1077 on 1769 degrees of freedom
## Multiple R-squared:  0.1321, Adjusted R-squared:  0.1306
## F-statistic: 89.77 on 3 and 1769 DF,  p-value: < 0.00000000000000022
##
## Next step: Use 'setx' method
```

So far in the labs, we would often sequence one IV while holding the rest constant at their means. But that is not the only way to go about this. Perhaps you were interested in the relationship between income and square footage for people who have a Bachelor's degree, or maybe the relationship between education and square footage for individuals with a specific income. You can hold IVs constant at values other than their means. If you do so, in it important that you make note of it and are transparent about the data you are presenting. Let's use Zelig to generate simulations and predictions for respondents who went to college by their logged income. A Bachelor's degree is indicated by a 6 on the education scale. We need to know the range of logged income as well:

```
describe(ds.sub$log.inc)
```

```
##    vars    n  mean   sd median trimmed mad  min   max range  skew kurtosis
## X1    1 1773 10.98 0.69     11   10.99 0.7 9.21 13.71   4.5 -0.12     0.05
##      se
## X1 0.02
```

```
inc.out <- setx(ols1, education=6,
                log.inc = seq(min(ds.sub$log.inc), max(ds.sub$log.inc))) %>%
  Zelig::sim() %>%
  zelig_qi_to_df() %>%
  qi_slimmer()
```

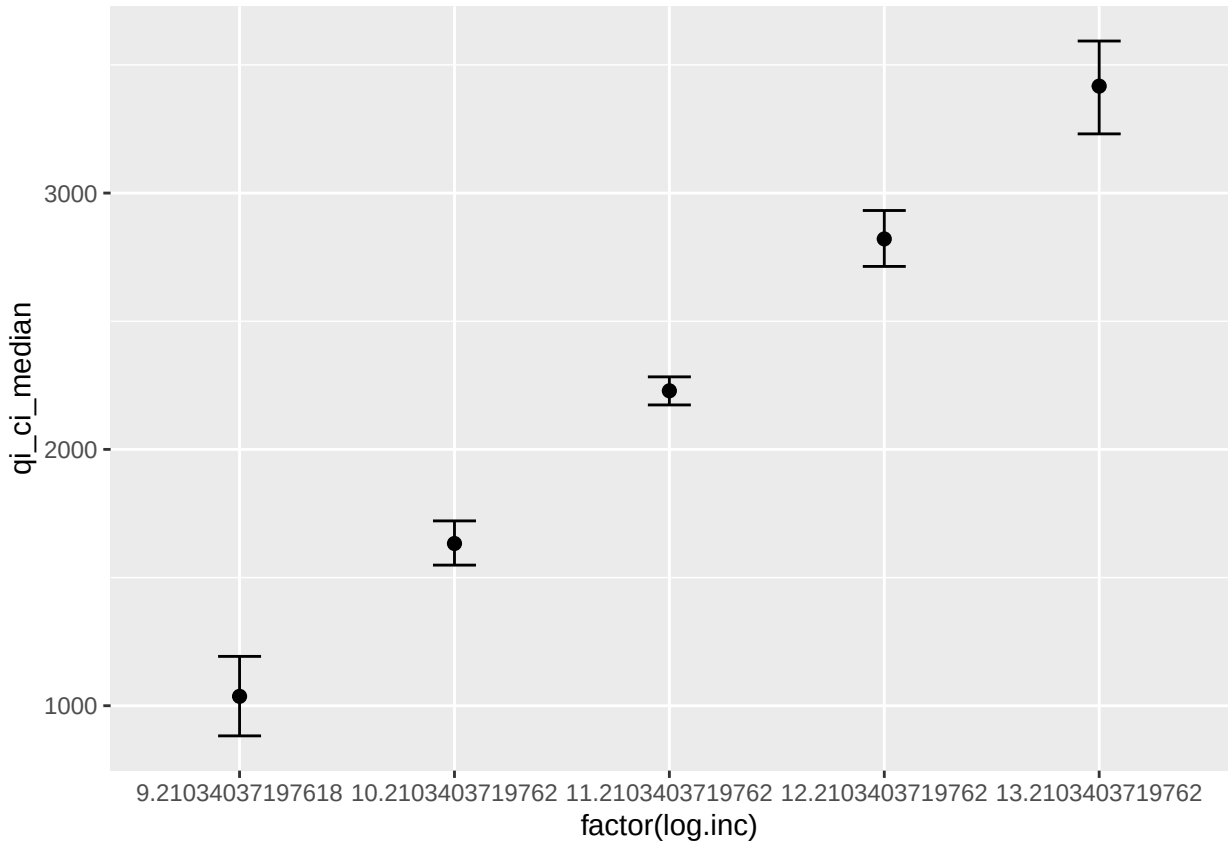
```
## Slimming Expected Values . . .
```

```
inc.out
```

```
##   setx_value  log.inc      age education qi_ci_min qi_ci_median qi_ci_max
## 1          x  9.21034 60.93739         6  882.4395     1036.623  1192.488
## 2          x 10.21034 60.93739         6 1548.5850     1632.855  1721.080
## 3          x 11.21034 60.93739         6 2173.4661     2228.582  2282.978
## 4          x 12.21034 60.93739         6 2713.8159     2821.148  2932.045
## 5          x 13.21034 60.93739         6 3230.8880     3417.201  3593.192
```

Plot the predictions:

```
ggplot(inc.out, aes(factor(log.inc), qi_ci_median)) +
  geom_errorbar(aes(ymin=qi_ci_min, ymax=qi_ci_max),width=.2) +
  geom_point(size=2)
```



14.3 Zelig with non-Zelig Models:

There are some models that can be specified outside of Zelig but that you can use the Zelig functions on. The complete list can be found on the Zelig website, but we can demonstrate with the `lm()` function. Here's a classic model from our labs:

```
ds$log.inc <- log(ds$income)
lm1 <- lm(glbcc_risk ~ glbcc_cert + log.inc + education + gender + ideol, data=ds)
summary(lm1)
```

```
##
## Call:
## lm(formula = glbcc_risk ~ glbcc_cert + log.inc + education +
##     gender + ideol, data = ds)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -7.826 -1.356  0.145  1.584  8.013
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept) 10.31811    0.78977  13.065 < 0.0000000000000002 ***
## glbcc_cert   0.27605    0.01886  14.636 < 0.0000000000000002 ***
## log.inc     -0.17187    0.07537  -2.280              0.02268 *
## education    0.03038    0.02981   1.019              0.30831
## gender      -0.34293    0.10091  -3.398              0.00069 ***
```

```
## ideol       -0.92756    0.02956 -31.378 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.325 on 2260 degrees of freedom
##   (281 observations deleted due to missingness)
## Multiple R-squared:  0.4192, Adjusted R-squared:  0.4179
## F-statistic: 326.2 on 5 and 2260 DF,  p-value: < 0.00000000000000022
```

We pass this model along to `setx()` and go from there. Let's sequence ideology from 1 to 7:

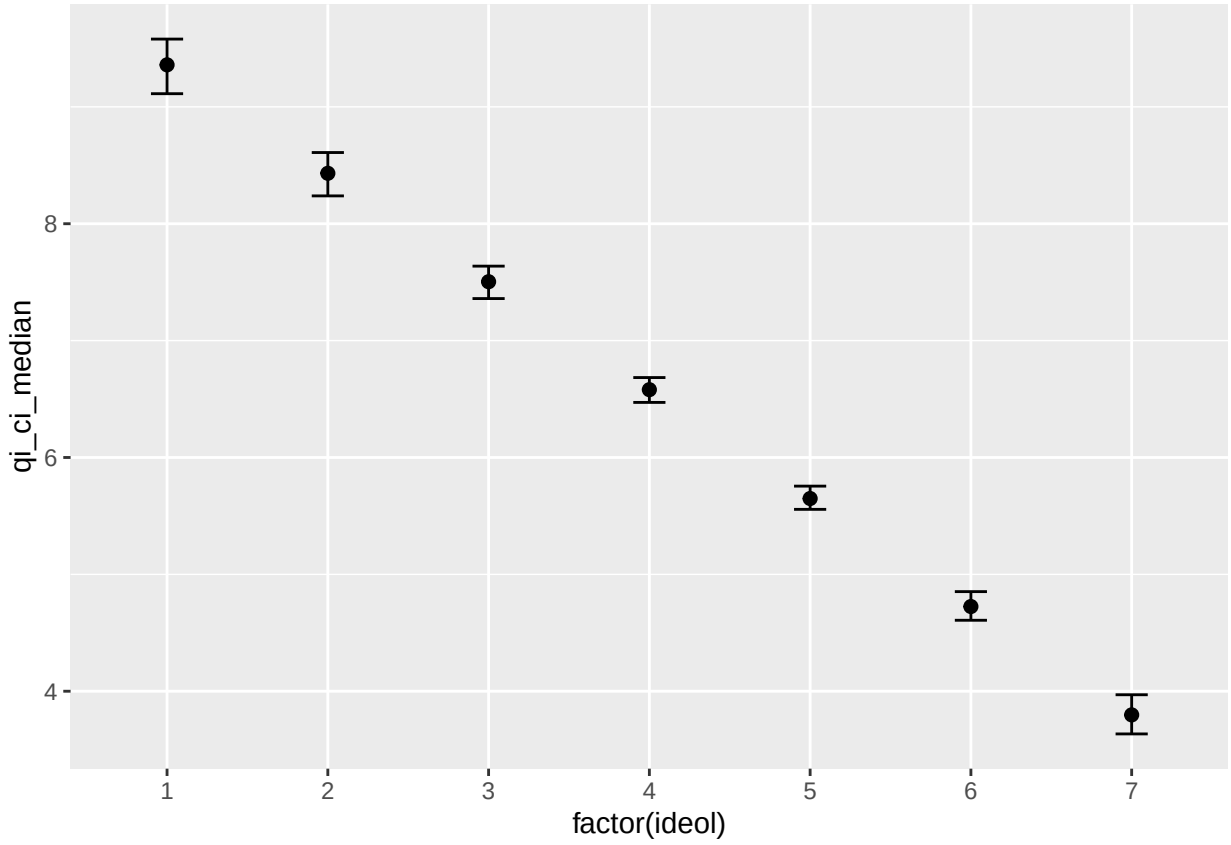
```
lm1.out <- setx(lm1, ideol=1:7) %>%
  Zelig::sim() %>%
  zelig_qi_to_df() %>%
  qi_slimmer()
```

```
## to_zelig is an experimental function.
##   Please report issues to: https://github.com/IQSS/Zelig/issues
```

```
## Assuming zls to convert to Zelig.
```

```
## Slimming Expected Values . . .
```

```
ggplot(lm1.out, aes(x=factor(ideol), y = qi_ci_median)) +
    geom_errorbar(aes(ymin = qi_ci_min, ymax = qi_ci_max), width = .2) +
    geom_point(size = 2)
```



Let's move onto a different model that might provide some interesting findings. In a previous lab we looked the relationship between ideology and support for the candidate an individual voted for. Recall that we used polynomial terms to find a better model fit and concluded that the relationship was not strictly linear.

Let's take that question one step further and break it down by political party. Perhaps there are linear relationships when we look at candidate support by ideology and party. We use Zelig to run simulations and predict values of candidate support for each of the ideology levels based on political party.

First subset the data:

```
d <- filter(ds) %>%
  dplyr::select("ideol", "education", "vote_cand_spt", "income", "age", "gender", "f.party.2") %>%
  na.omit() %>%
  mutate(log.inc = log(income),
         f.part = as.factor(f.party.2))
```

Build the model:

```
lm2 <- lm(vote_cand_spt ~ ideol + education + log.inc + age + gender + f.part, data=d)
summary(lm2)
```

```
##
## Call:
## lm(formula = vote_cand_spt ~ ideol + education + log.inc + age +
##     gender + f.part, data = d)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9741 -0.6758  0.1736  1.0439  2.2191
##
## Coefficients:
##              Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  4.073215   0.455838   8.936 < 0.0000000000000002 ***
## ideol        0.018765   0.019300   0.972             0.331009
## education   -0.040783   0.015770  -2.586             0.009776 **
## log.inc     -0.063440   0.040437  -1.569             0.116841
## age          0.007197   0.001913   3.763             0.000173 ***
## gender      -0.048230   0.052717  -0.915             0.360362
## f.partInd   -0.532600   0.085114  -6.257        0.00000000048 ***
## f.partRep   -0.076862   0.073093  -1.052             0.293129
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.123 on 1941 degrees of freedom
## Multiple R-squared:  0.0445, Adjusted R-squared:  0.04106
## F-statistic: 12.91 on 7 and 1941 DF,  p-value: 0.0000000000000002543
```

Within the `setx()` function we can sequence ideology from 1 to 7 and sequence the three party options, Democrats, Independents, and Republicans. Then use `Zelig::sim()`. Doing so will perform 1000 simulations for each level of ideology and each party, so 1000 for Democrats with an ideology score of 1, 1000 for Democrats with an ideology score of 2, and so on.

```
lm2.out <- setx(lm2, ideol=1:7, f.part=c("Dem", "Ind", "Rep")) %>%
  Zelig::sim()
```

```
## to_zelig is an experimental function.
##   Please report issues to: https://github.com/IQSS/Zelig/issues
```

```
## Assuming zls to convert to Zelig.
```

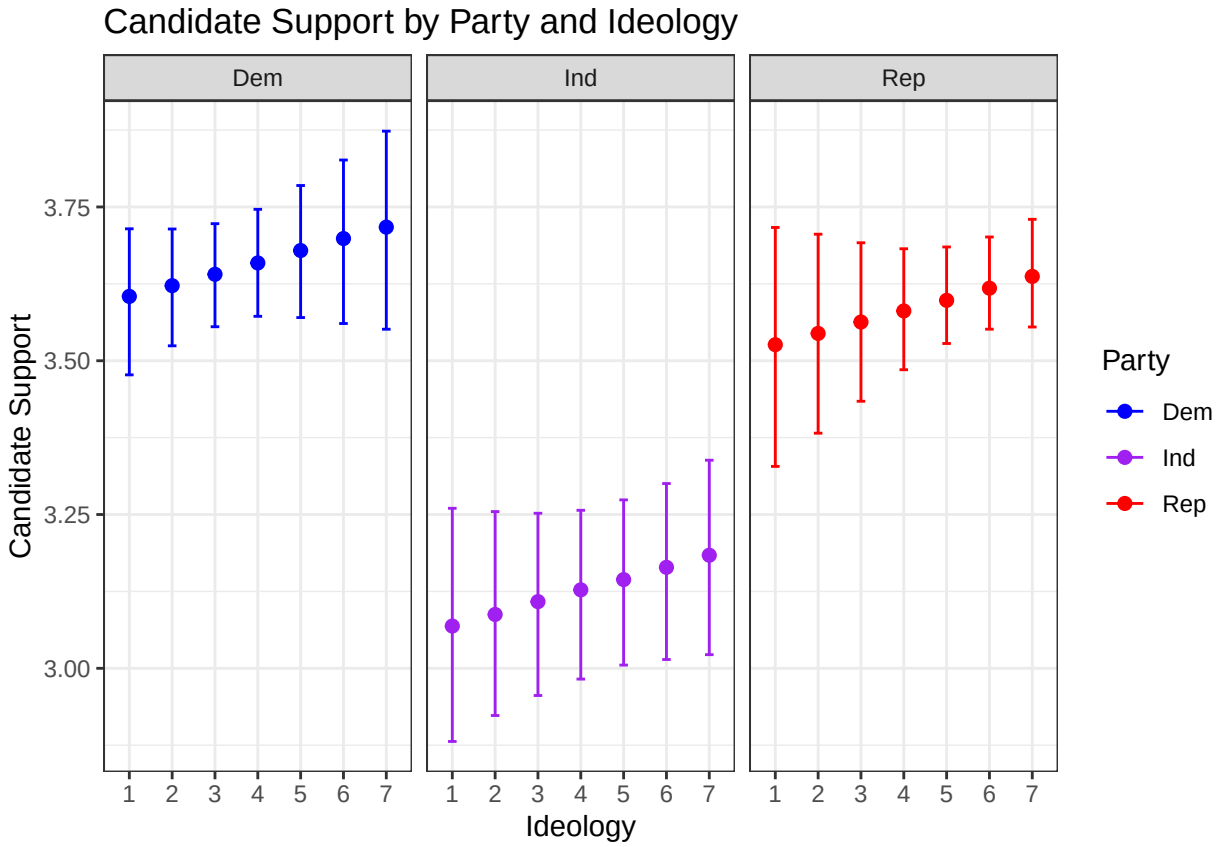
Now get the data into tidyverse data frame form, slim the QIs:

```
lm2.out <- zelig_qi_to_df(lm2.out) %>%
  qi_slimmer()
```

```
## Slimming Expected Values . . .
```

Next plot the results of the simulations:

```
ggplot(lm2.out, aes(factor(ideol), qi_ci_median, color = f.part)) +
  geom_errorbar(aes(ymin = qi_ci_min, ymax = qi_ci_max), width = .2) +
  geom_point(size = 2) +
  scale_color_manual(values = c("blue", "purple", "red"),
                     name = c("Party")) +
  facet_wrap( ~ f.part, scales = "fixed") +
  ggtitle("Candidate Support by Party and Ideology") +
  xlab("Ideology") +
  ylab("Candidate Support") +
  theme_bw()
```



This visualization tells us some interesting information: Independents do not appear to be have as much support for the candidate they voted for, regardless of their ideology. It also appears that Democrats and Republicans follow the same trend, with more conservative Democrats tending to support the candidate they voted for a little more, and the same for Republicans.

For more on Zelig, make sure to check out the website: zeligproject.org

15 Appendix: Guide to Data Visualization

Throughout these labs, we have created numerous high quality data visualizations using our class data sets. While you can always use the code we have created there - it may even help refresh your memory on the data analysis skills you've learned - this guide is designed to be a quick, additional resource for the creation and customization of data visualizations using the Tidyverse and `ggplot2`.

These examples will use the class dataset to demonstrate how to add and customize elements of your data visualizations.

Note: While this quick guide is hopefully helpful, it is by no means complete. Remember that R and the Tidyverse packages are open source; you can always find help online, such as this Data Visualization Cheat Sheet from RStudio.

15.1 Deciding Which Visualization to Use

Remember - the type of data you are using *matters* for what kind of plot you make.

For Exploring a Single Variable

When you are first exploring your data, it is often helpful to visualize a single variable.

For **continuous** data, you may be interested in producing

- density curves (geom_density), or
- histograms (geom_histogram).

For **discrete** data, you will likely be using a bar plot (geom_bar or geom_col).

For Displaying Two (or more) Variables

After you get a feel for your data, you are likely then interested in visualizing their relationships. These examples are geared towards two variables (one x and one y), but you can always add more!

With a **continuous** x and a **continuous** y, you have numerous options available. The most common include:

- a scatter plot (geom_point + geom_jitter), and
- a function curve(geom_line or geom_smooth)

With a **discrete** x variable and a **continuous** y variable, you might produce:

- a bar plot (geom_bar or geom_col),
- a box blot (geom_boxplot), or
- a violin plot (geom_violin)

Note: While these are not your only options, they are some of the most commonly used (and match what we've covered in these labs!)

Once you have the basic plot made, the customization can begin. Below, we'll cover how to add labels and titles, scale your axes, visualize error, add color, and using a theme.

15.2 Adding Labels

There are numerous ways to add titles and labels to your plot. The simplest is to use the functions:

+ xlab("X-Axis Title"), + ylab("Y-Axis Title"), and + ggtitle("Plot Title")

15.3 Scale and Limits

You may also want to change the range of values that appear on either axis, as well as the number and nature of data breaks.

To set limits on the range of data that appears in your plot, you can use the function:

+ coord_cartesian(ylim =, xlim =)

In essence, this allows you to "zoom in" on your data by specify the range of the x and y axes.

You may also want to change the number and nature of the breaks along your axes. For continuous data, you can use the functions: + scale_x_continuous(breaks =, labels = c()), and + scale_y_continuous(breaks =, labels = c())

Note: For discrete data, you will use these functions in the same way: + scale_x_discrete() + scale_y_discrete()

15.4 Visualizing Error

Often, you will need to include a representation of estimated error in your data visualizations. The primary ways to do this are through confidence intervals or error bars.

Confidence Intervals

To add confidence intervals to your function lines, you can use the geom_ribbon() function. The arguments for this function are:

+ geom_line() + geom_ribbon(aes(ymin =, ymax =), alpha =)

Note: Specifying the alpha value here changes how opaque the confidence interval is.

Error Bars

With a discrete point, you will need error bars rather than confidence intervals. To include error bars on your plot, you can use the geom_errorbar() function. The necessary arguments include:

+ geom_errorbar(aes(ymin, ymax), width =)

Note: Prior to using either of these functions, it helps to add the upper and lower limits of your confidence interval to your data frame. This can be done via the mutate function.

15.5 Adding Color

Adding color to your plots helps you distinguish between variables and categories. In the `ggplot` aesthetic, setting a variable to "color" or "fill" will automatically produce predefined color variations. However, these won't always make sense with your data and you may want to specify your own.

Note: `R` allows you to use HEX color codes for customization. Color Brewer 2 is a great tool for picking out colors.

For setting the color of an individual variable, you can add the argument within the geom you are using. Here you will specify the desired color, rather than the variable. Remember that you use "color" for points and lines, and "fill" for columns and bars. For example,

+ geom_point(color = "blue"), or + geom_col(fill = "red")

Note: You can also specify the size and alpha here, too.

You may also want to add multiple colors to a plot. These functions allow yoou to specify your own mappings from levels in the data to color values.

+ scale_colour_manual(values = c()) + scale_fill_manual(values = c())

15.6 Position Adjustments

When using multiple variables, you may often want to adjust their placement on the graph. Below are some common position adjustments you can use.

- Arrange elements side by side

+ geom_bar(position = "dodge")

- Stack elements on top of each other

+ geom_bar(position = "stack") or, + geom_bar(position = "fill")

Note: "Fill" normalizes the height of the bars, while "stack" does not

- Move labels away from data points

+ geom_label(position = "nudge")

15.7 Using Themes

Outside of the basic plot elements pertaining to your data, you can still further customize your plots by using themes.

A complete guide to the existing `ggplot2` themes is available here. These primarily control the non-data display elements, which include the background shading and internal lines of the plot. Different themes may be best suited for different kinds of projects; you can even tweak the premade themes by adjusting their arguments.

Additionally, you can create and save your own custom theme for future use! Here is an example of a custom theme:

```
theme_custom <- theme_minimal() +
 theme(plot.title = element_text(family="serif",
                                 face = "bold",
                                 hjust = .5,
                                 size = 15),
       axis.title = element_text(family = "serif",
                                 face = "bold",
                                 size=15),
       axis.text = element_text(family = "serif",
                                face = "bold",
                                color = "black"),
       panel.grid = element_line(color = "grey75"),
       legend.background = element_rect(color = "grey94",
                                        fill = "grey94"),
       legend.text = element_text(family = "serif",
                                  face = "bold",
                                  size = 9),
       legend.title = element_text(family = "serif",
                                   face = "bold",
```

```
                                  size = 9),
        legend.title.align = .5)
```

15.8 Putting it All Together

Below, we have created three plots to provide an example of how these visual elements might all interact.

First, we will establish a subset of the class dataset to use for the examples.

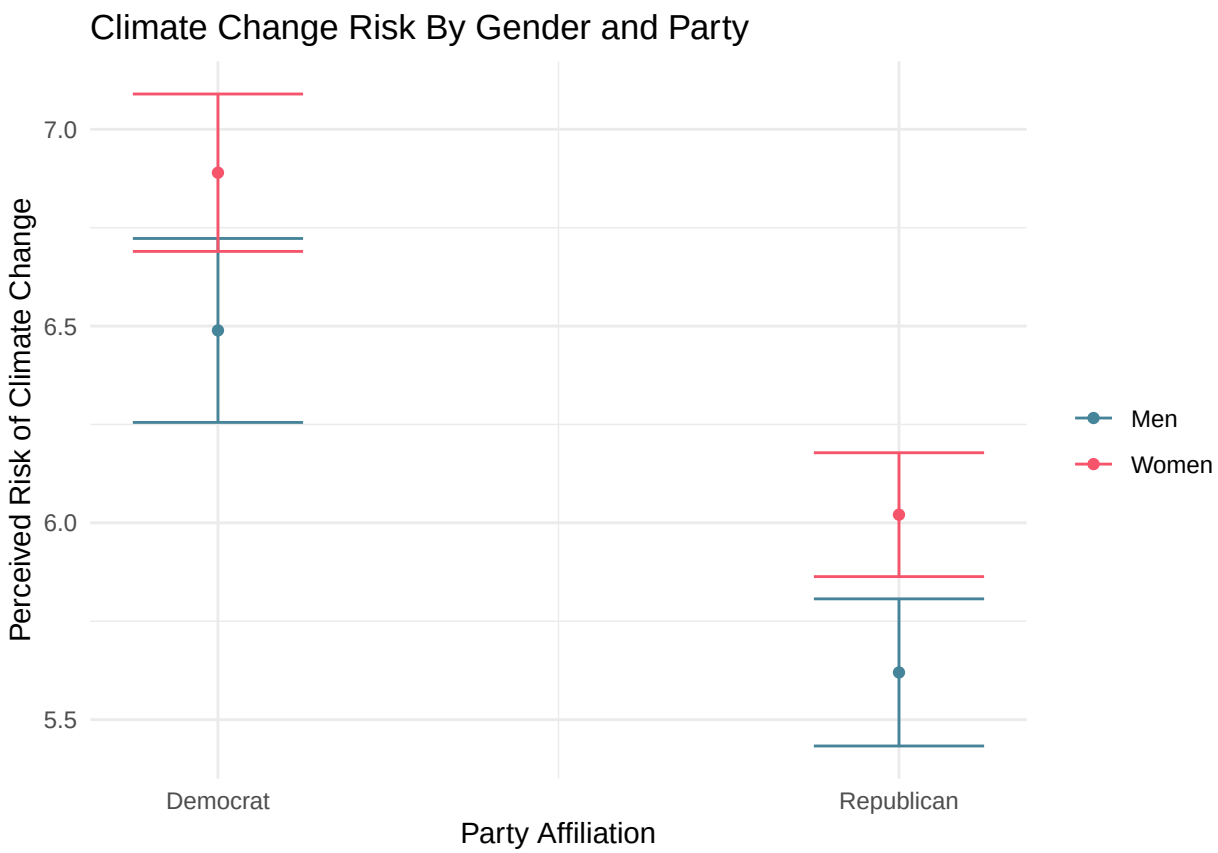
```
#create the model
lm1 <- lm(glbcc_risk ~ party + f.gender, data = sub.ds)
lm1 %>%
  augment(newdata = data.frame(party = 1:2,
                               f.gender = "Men")) -> fit1.m
lm1 %>%
  augment(newdata = data.frame(party = 1:2,
                               f.gender = "Women")) -> fit1.w

fit1.df <- full_join(fit1.m, fit1.w)
```

```
## Joining, by = c("party", "f.gender", ".fitted", ".se.fit")
```

```
## Warning: Column `f.gender` joining factors with different levels, coercing
## to character vector
```

```
#visualize the model
ggplot(fit1.df, aes(party, .fitted, color = f.gender)) +
  geom_point(stat = "identity", position = position_dodge(width = 0))  +
  scale_x_continuous(breaks = 1:2, labels = c("Democrat", "Republican")) +
  geom_errorbar(aes(ymin = .fitted - 1.96 * .se.fit,
                    ymax = .fitted + 1.96 * .se.fit), width = 0.25) +
  scale_colour_manual(name="",  values =c("#468499", "#f6546a")) +
  ylab("Perceived Risk of Climate Change") +
  xlab("Party Affiliation") +
  ggtitle("Climate Change Risk By Gender and Party") +
  labs(color="Gender") +
  scale_fill_discrete(guide=FALSE) +
  theme_minimal()
```




```
#creating the model
lm2 <- lm(glbcc_risk ~ wtr_comm + f.gender, sub.ds)
lm2 %>%
  augment(newdata = data.frame(wtr_comm = 0:10,
                               f.gender = "Women")) -> fit2.w
lm2 %>%
  augment(newdata = data.frame(wtr_comm = 0:10,
                               f.gender = "Men")) -> fit2.m
fit2.df <- full_join(fit2.w, fit2.m)
```

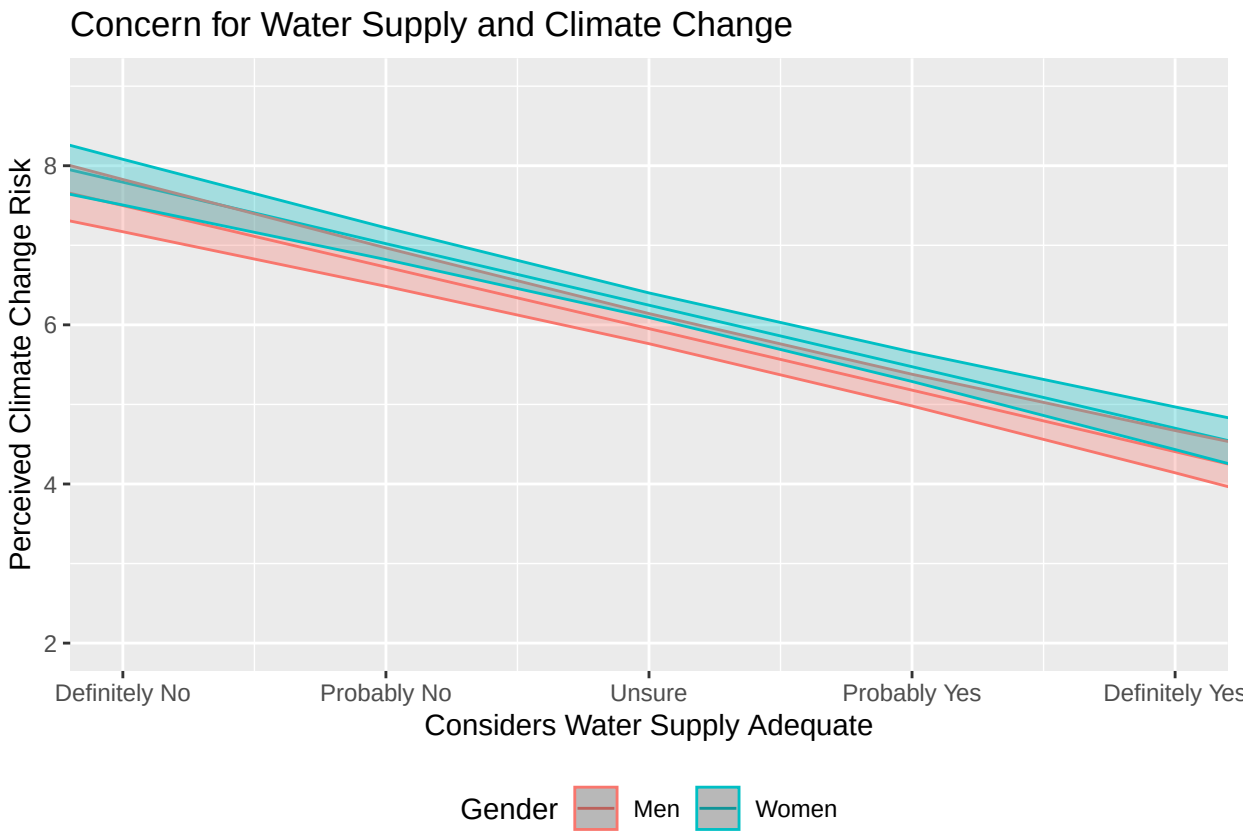
```
## Joining, by = c("wtr_comm", "f.gender", ".fitted", ".se.fit")
```

```
## Warning: Column `f.gender` joining factors with different levels, coercing
## to character vector
```

```
fit2.df <- fit2.df %>%
  mutate(up = .fitted + 1.96 * .se.fit,
         low = .fitted - 1.96 * .se.fit)

#visualizing the model
fit2.df %>%
ggplot(., aes(x = wtr_comm, y = .fitted, color = f.gender)) +
  geom_line() +
  geom_ribbon(aes(ymin = low, ymax = up, fill = f.gender), alpha = .3) +
  coord_cartesian(ylim = c(2, 9), xlim = c(1, 5)) +
  scale_x_continuous(breaks=c(1, 2 ,3 ,4 ,5), labels=c("Definitely No",
                                                       "Probably No", "Unsure",
                                                       "Probably Yes", "Definitely Yes")) +
```

```
  theme(legend.position = "bottom", legend.direction = "horizontal") +
  ggtitle("Concern for Water Supply and Climate Change") +
  xlab("Considers Water Supply Adequate") +
  ylab("Perceived Climate Change Risk") +
  labs(color="Gender") +
  scale_fill_discrete(guide=FALSE)
```



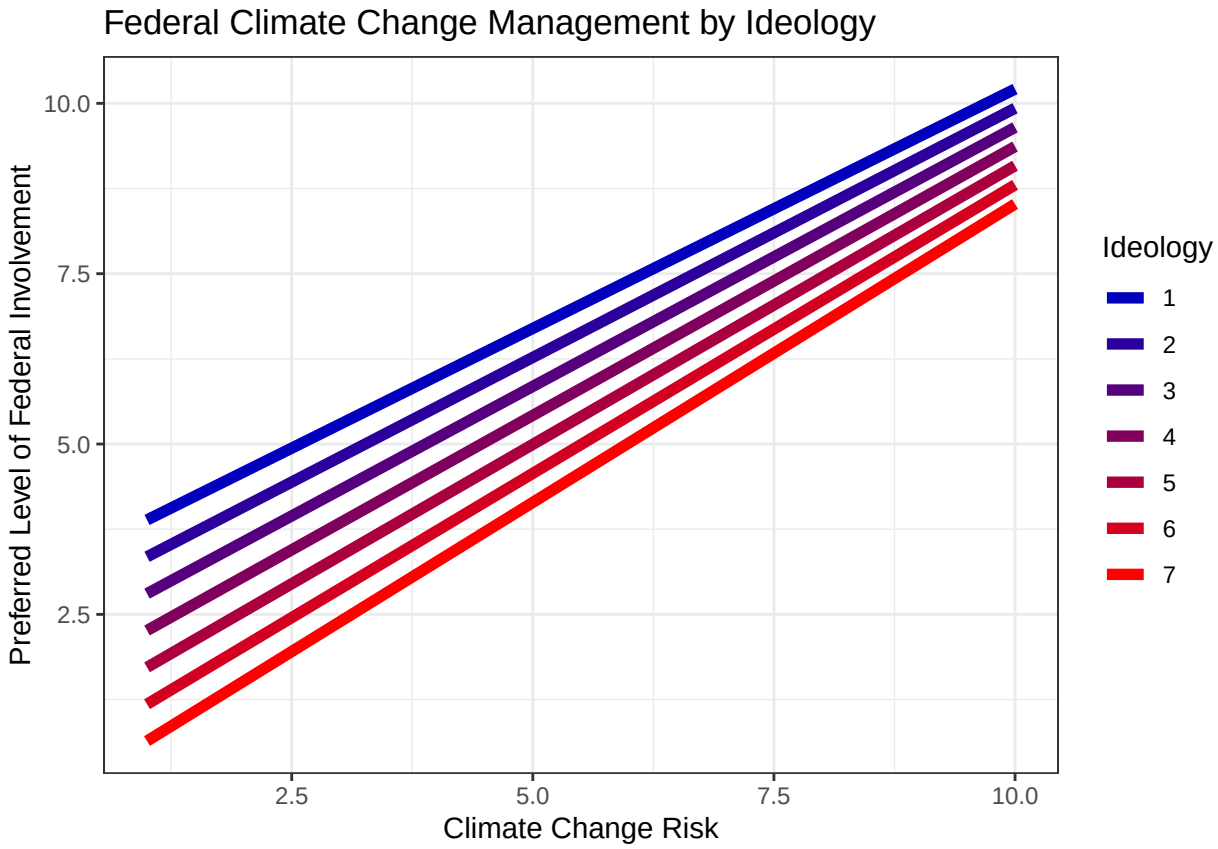

```
lm3 <- lm(glbwrm_risk_fed_mgmt ~ ideol * glbcc_risk, data = sub.ds)
lm3 %>%
  augment(newdata = data.frame(ideol = 1, glbcc_risk = seq(1, 10, 1))) -> id1
lm3 %>%
  augment(newdata = data.frame(ideol = 2, glbcc_risk = seq(1, 10, 1))) -> id2
lm3 %>%
  augment(newdata = data.frame(ideol = 3, glbcc_risk = seq(1, 10, 1))) -> id3
lm3 %>%
  augment(newdata = data.frame(ideol = 4, glbcc_risk = seq(1, 10, 1))) -> id4
lm3 %>%
  augment(newdata = data.frame(ideol = 5, glbcc_risk = seq(1, 10, 1))) -> id5
lm3 %>%
  augment(newdata = data.frame(ideol = 6, glbcc_risk = seq(1, 10, 1))) -> id6
lm3 %>%
  augment(newdata = data.frame(ideol = 7, glbcc_risk = seq(1, 10, 1))) -> id7
full_join(id1, id2) %>%
  full_join(., id3) %>%
  full_join(., id4) %>%
  full_join(., id5) %>%
  full_join(., id6) %>%
```

```
  full_join(., id7) -> fit3.df
```

```
## Joining, by = c("ideol", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "glbcc_risk", ".fitted", ".se.fit")
## Joining, by = c("ideol", "glbcc_risk", ".fitted", ".se.fit")
```

```
#create a color scale
col_scale<-colorRampPalette(c("#0200bd50","#FF000050"))(7)

#visualizing the model
ggplot(fit3.df, aes(glbcc_risk, .fitted, color = as.factor(ideol))) +
  geom_line(size = 2) +
  scale_color_manual(values = c(col_scale[1], col_scale[2], col_scale[3],
                       col_scale[4], col_scale[5], col_scale[6], col_scale[7]),
                     labels = c("1", "2", "3", "4", "5", "6", "7"),
                     name = "Ideology") +
  ggtitle("Federal Climate Change Management by Ideology") +
  xlab("Climate Change Risk") +
  ylab("Preferred Level of Federal Involvement") +
  theme_bw()
```



There are an infinite number of ways you might use these (and more) elements to customize your own data visualizations. Have fun with it!