Data Science for Social Scientists: An applied course using IPUMS data

Developed by:
Daniel E. Ehrlich, IPUMS, University of Minnesota
Anna Tremblay, Dept of Soc, Anth, & CJ, Clemson University
Current Version compiled: 2024-02-26

Contents

Preface

An applied methods class for social scientists that uses real-world IPUMS data. This course is:

Open-source and customizable -
All materials available on Github

Made with open-source tools -
R, RStudio, bookdown

Driven by $^{\text{(nearly)}}$ open-source data -
Harmonized across time and space: IPUMS

Why make this course

In a world where information and data are increasingly accessible, it is of utmost importance for individuals to understand data science and the interpretation of data. We believe that education should be easily accessible and teaching resources should be freely available to aid in this endeavor. While we (DEE) may be slightly biased, we think IPUMS is a fantastic resource for both **Education** and **Research**. Real-world example datasets provide the bulk of the content for this course, providing an applied context we hope students (and instructors) will find engaging. We also know many instructors may be teaching across multiple disciplines, in large departments, or be the only "data person" at their institution. We think IPUMS data is useful to virtually any social science field. We provide some example lessons, and encourage instructors to develop their own, using our `lesson_template.Rmd` to tailor this course to their subject or interest.

What is IPUMS

IPUMS started as a project to digitize the historical records of the US census. It has expanded to include 9 data collections, which are united in their methods and principles of making social science research easier. IPUMS data consists of individual-level census and survey data from more than 100 countries around the world. Notably:

- IPUMS **harmonizes** these data - ensuring consistently coded values across time and space.
- IPUMS provides harmonized **GIS Shapefiles** for most census and survey data.
- IPUMS provides extensive **metadata**, including:
  - Original questionnaire text.
  - Universe definition and comparability statements.
  - Alerts about notable changes in variable definition, universe, or coding schema

IPUMS data is free to use for education and research purposes. Researchers need only to register with an email address and brief project description. Nothing too formal - we're just trying to understand what kinds of questions researchers are interested in. For educators, we have additional resources to facilitate set up of classroom accounts - making it easy to get your students registered and share IPUMS data with them.

What is R

R is a programming language. Learning to use R is learning how to code, which teaches logic, and programmatic thinking. Since R is a *statistical* programming language, it has many built-in features to facilitate a range of mathematical calculations. Since R is open-source, it is customizable and expandable! Base R refers to the core set of functions needed to run R. It's the bare-minimum to use R, available from CRAN. Base R can be expanded by downloading add-on R packages, either from CRAN, from a github.page, or by making your own!

While you *can* interact with R by itself, pretty much everyone agrees the experience can be better. We reccomend using Rstudio, which provides a GUI and many additional handy features that make coding in R fun!

In addition to a GUI, Rstudio is an Integrated Development Environment (IDE), which allowes a user to both write and run code, but also develop R packages (or textbooks).

NOTE: * Rstudio as an organization is now known/rebranding as posit * Rstudio as a program is now known/will be produced as quarto * This is because the quarto IDE supports python, javascript, etc in addition to R

This book, and all analyses, are done using R. **ADD IN CONTENT FROM CURRENT 1.2 - WHAT IS R**

Getting Started

In order to use this textbook, you will need to:

- download and install RStudio
    - This link also contains instructions and links to download R from CRAN
    - Be sure you download the appropriate file for your Mac or PC
- Register for an with IPUMS account. We provide **limited example data**, but in order take full advantage of these exercises:
    - IPUMS registration for individuals
    - IPUMS registration for instructors

Course Description

This course is broken down into 3, 5-week units. Unit 1 focuses on familiarizing yourself with R and the IPUMS dataset. In Unit 2, each week will showcase a method/analysis using preselected variables. In class, students will walk through a given problem set and produce a lab report by the end of class. In Unit 3, students will work towards answering a research question that they pose, creating a research paper with literature review, data analysis, conclusion, and data outputs.

Course Aims

Provide students with relevant, hands on, methodological training in data literacy and visualization.

Learning Outcomes

After this course, students will be able to:

- Understand the depth of the IPUMS database and the variables it has to offer
- Compose R code to analyze the IPUMS data
- Produce visually pleasing data outputs in R
- Synthesize the information in a written report
- Present the analysis in a poster format for other students

Guiding Principles

- Phenomenon-based learning
  - try to start the class with a **question** or **problem**
  - *why* does the data look the way it does
  - structure class so students work towards solving the problem
- Relevant examples
  - Drawn from multiple disciplines (eg, economics, demography)
  - Can be added as modular examples/exercises

Syllabus - General

This syllabus is initially envisioned as 3 5-week sections. However, compilation and content are intended to be modular with templates for instructors to include their own specialties.

The basic structure of this course is:

Unit 1 (Weeks 1-5): Understanding and Testing Data

- Students use simple datasets bundled with the course or provided by the instructor.
- Simplified data to illustrate trends.
  - EG: plotting continuous variable (AGE); Table of categorical variable (SEX); Crosstabs

Unit 2 (Weeks 6-10): Finding Data and Asking Questions

- Students begin to analyze real world, IPUMS, datasets, provided by course/instructor.
- Students begin to model real world phenomena
  - EG: SEX ~ EDUATTAIN ; SEX ~ EDATTAIN + EMPSTAT
- Students learn to perform exploratory analysis, hypothesis testing, and statistical inference.
- Students learn to navigate IPUMS website,and find relevant data to thier research interest.

Unit 3 (Weeks 11-15): Discussing Data and Student Research

- Students develop a research question to be answered with IPUMS data.

  - Students are encouraged to fit it to their interests/major/discipline.
- Course time should be devoted to individual/small-group research.
- Instructor/class present on recent research.
  - Instructor models constructive / scholarly criticism.
  - Encourage students to critique published work - responsibly.

Syllabus - Detailed

Unit 1 Understanding and Testing Data

Students become gain familiarity and comfortability navigating RStudio, coding in R and performing simple data manipulation and visualization exercises. Datasets in this section consist of real-world (or synthetic) data, but the focus is on understanding data types (EG: using Age as a continuous variable; sex, education, employment as categorical; etc). Instructors should acknowledge these as **educational** datasets and make explicit trends found within these data are devoid of context, and must be taken with a (rather large) grain of salt, if at all.

By the end of Unit 1, students will be able to:

- Download R and RStudio
- Read data into R and
- Write (save) data out of R
- Summarize data visually
  - Using `base R`
  - Using `ggplot` (tidyverse)
- Summarize data in tables
  - Using base R
  - Using `gttable` / `tidyverse`
- Formally state and test assumptions of data
  - *EG:* t-test, anova, correlations, regression

By the end of Unit 1, students will understand

- Main types of data
  - *EG:* logical, numeric, character, etc
  - R specic vs general terms
- How to create and describe various data distributions
  - *EG:* normal, poisson, normal-skewed, etc
- Know which types statistical tests are appropriate for a given set of data.

Week 1: Intro to R, data types, data structures

Week 2: Plotting Data, Distributions

Week 3: Statisitcal testing of simple data sets

Week 4: Correlation and Relationships of simple data sets

Week 5: (TBD)

Unit 2 Finding Data and Asking Questions (Using IPUMS Data)

Here we demonstrate two **different** approaches to conducting research. Students become familiar writing up short lab reports detailing their findings. For Section regarding exploratory analysis, we/instructor provides students with simple datasets from IPUMS (or other real-world data). Students will learn exploratory data analysis techniques and how to create lab reports to summarize key findings.

For unit dealing with hypothesis testing, students will learn to develop their own simple research questions or social-science hypotheses. They will seek out data to answer these questions, learning to navigate ipums.org, and create **data extracts**, as well as hypothesis-testing statistical methods. Again, lab reports to summarize findings.

0.0.0.1 Week 6: Intro to IPUMS

Week 7: Exploratory analysis

If you've just collected a survey, or other raw data, you may not know what you're looking for. This is perfectly ok but goes against *the scientific method* most people learned in grade school.

This unit begins by presenting data/distributions and asking students to begin interpreting the data . visual exploration is encouraged and basic of data manipulation are taught * *EG:* how to subset data, how to reshape data, how to re-code data, how to convert from one `data type` to another.

Example lab exercise:

Students given a data set (xls, csv, etc) * load data, perform manipulations, basic summaries + cross tabs + group means by a covariate * inspect data visually + *DESCRIBE* the distribution - is it normal? significant? * *FIND* aquestion in the spread of the data + how can you test this (maybe small group work) * write up/ present results + think on confounding factors / biases

Week 8: Hypothesis Testing

If, on the other hand you have an a pre-existing idea you want to test. We can follow the traditional *scientific method*. With a question in mind, the first question is: where to look. What better place than IPUMS!

Begin introducing navigation of web resources - mainly IPUMS international

Students should become comfortable working through lab exercises: * Define a question (or be presented with one) * Download variables from IPUMS (course downloads possible) * Perform a basic analysis (discussed in Unit 1) * Generate a **visual argument** for your analysis + Include explanation/interpretation/reflection on the question at hand, and the data used + Any obvious biases + Any obvious confounding factors

Week 9: Statistical Inference

Week 10: (TBD)

Unit 3 Discussing Data and Student Research

Students will select their own research question that can be answered with the IPUMS data set and will spend five weeks conducting a research project complete with data analysis, visualization, and interpretation.

In this section we encourage the instructor to provide ample time for independent student/small-group research. Some class time should be devoted to modeling healthy discussion and critique of methods. Students should learn to discuss not just *how* to answer a research question but *why* they are asking/answering it. What impact does the question/answers have. Is the question releveant/meaningful, and importantly, Is this research question perpetuating racist ideas.

We provide some examples here but encourage instructors (or students) to bring in recent journal/popular articles that do (or do not) apply data science methods well.

Week 11: Students develop research Question

Week 12: Students find relevant variables from IPUMS

Week 13: Students test and evaluate results

Week 14: Students prepare presentations of results

Week 15: Students present work (slides, poster, podium, etc)

DEV NOTES

TO DO

- **UPDATE TODO LIST**
- Make chapter 1 chapter 2
- Anna Adds chapter con data science intro exclusive of R/IPUMS
- discuss style
  - key terms section for each chapter?
  - key terms in **bold**
  - italics for *emphasis*
  - are we pro-hyphens, or are they pedantic?

MISC IDEAS

- Application forward
- Present research/ analysis/results FIRST, then explain the mathematical principals behind it
- daily/weekly "i'm stuck on…"
  - Students send in questions (night before class) and instructor spends 10-15 mins talking through (or collaboratively working through with class) solutions
  - Alternatively, once a month maybe a longer class covering "common problems asked this month" daily/weekly "recent research"
- pick out a recent article with good visualization (or bad) and spend 5-10 mins discussing what makes it good (or bad)
  - Encourage students to find articles for extra credit

Documentation

This function grabs any packages in your project and adds them to a local list that can be referenced using `R-pacakgename` * **NOTE** in practice, that needs to be wrapped in markdown syntax, eg: `[@R-bookdown]` * See help files for more info - might be able to create/add a `citation` file

Testing title fix

Unit 1: The Basics

Summary

0.0.1 Lesson 0:

Lesson 0 files should contain a brief summary of the topics within each unit

Lesson 0 can also be used for a brainstorming space to sketch out ideas before creating `Unit#_Lesson#` files.

Lesson 1: What IS Data / Collecting Data @ref(unit1__ch1)

Lesson 2: Visualizing/Describing Data

Lesson 3: Hypothesis Testing: Comparisons and Correlations

Lesson 4: Hypothesis Testing: ANOVA and LM

Lesson 5: Drawing Conclusions

Unit-wide Glossary

Chapter 1

WHAT IS DATA

1.1 Engage

Brainstorm/word cloud on “what is data”

1.2 Explore

Brainstorm “what do we DO with data” Brainstorm/word cloud “where do we get data” We can collect it! * quick poll: how many people have ***(participated in a survey? analyzed data?)*** ** XX% of this class has done _____*

Pose questions to explore

1.3 Explain

1.3.1 WHAT ARE DATA

Data are defined as “facts and statistics collected together for reference or analysis.”1 As seen in Figure 1.1, there are two types of data: quantitative and qualitative. Quantitative data are able to be expressed in numerical format and are countable. These data are either discrete or continuous where discrete data uses numeric bins. For example, we use our age as discrete quantitative data, we round our age to the previous year (eg., 20, 21, 22). Continuous data does not use bins, but rather includes all of the fractions between two whole numbers. An example could be most physical measures like height, weight, the speed at which an individual runs.

Qualitative data describe characteristics or categories and can be broken down into two categories, nominal or ordinal. Nominal data have no inherent ordering but it can be categorized. Examples include country or origin, gender, hair color, race, etc. Ordinal data can both be categorized and ordered (e.g., first, second, and third place is a race).

Going back to our hypothesis of male height on campus, heights are continuous, qualitative data. It is difficult for people to report their specific height and you assume that most individuals will report it rounded to the closest inch. This makes the data you will actually use, discrete quantitative data.

> *Fun fact: A single data point is called a datum which is Latin for "something given". The word data can be either singular or plural depending on how you use it. It can be used as a mass noun the same way we discuss sand on a beach or hair on our head. However, in science we are usually referring to multiple datums within a data set making it a plural noun. Therefore, data is cool and data are cool!*
>
> — Sources

1.3.2 COLLECTING DATA

The first step to answering a research question is to collect your data. Broadly, data comes in two forms, primary and secondary. (Fig 1.2) Primary data are data that is collected directly by the researcher. Surveys, observations, experimentation, questionnaires, and interviews are all examples of primary data. Secondary data are collected from published or unpublished literature. It is collected by different researchers and compiled for use by a second scientist. These types of data include data found in published articles, books, journals, biographies, and government records like the US Census.

Once compiled, you now have a data set which is composed of observations and variables. An observation is all of the measures taken for one person or item. A variable is what is being measured.

The US CDC data is secondary, but you are collecting height data yourself in class as a comparison. The survey or questionnaire you use on your classmates is primary data. Each individual is an observation and the variable of interest is height.

1.3.2.1 "HOW do we get data??"

The first step to answering a research question is to collect your data. Broadly, data comes in two forms, primary and secondary. (Fig 1.2) Primary data are data that is collected directly by the researcher. Surveys, observations, experimentation, questionnaires, and interviews are all examples of primary data. Secondary data are collected from published or unpublished literature. It is collected by different researchers and compiled for use by a second scientist. These type of data include data found in published articles, books, journals, biographies, and government records like the US Census. Once compiled, you now have a data set which is comprised of observations and variables. An observation is all of the measures taken for one person or item. A variable is what is being measured. The US CDC data is secondary, but you are collecting height data yourself in class as a comparison. The survey or questionnaire you use on your classmates is primary data. Each individual is an observation and the variable of interest is height.

1.3.2.2 TYPES OF DATA

There are a lot of different ways to record observations. Its important to choose an appropriate format to record your data. Some pretty broad categories are:

- Yes/No
- Tallies
- Categorization
- Measurments
- Open-ended text

If you were to ask your friend: "What is your favorite food?" You would not expect them to give an answer of "yes" or "no". Asking an open-ended question often/always prompts an open-ended response. This can be informative and inclusive, allowing respondents to answer with exactly the answer they feel best answers the question. However, it can make analyzing and drawing interpretations from the data dificult to impossible!

If instead, we ask "Is your favorite food pizza?" We expect a yes/no answer. Asking this question of 100 people, we can easily answer the question "What percentage of people's favorite food is pizza?"

1.3.2.2.1 Continuous vs categorical One of the biggest differences in classifying data is based on the uniuque values we expect from the data. Continuous variables are…

Categorical variables can be ordered (EG Factors, ordinations) or they can be undoredered (EG, categories)

1.3.3 POPULATIONS AND SAMPLING

Random Sampling: It is a sampling method in which all the items have an equal chance of being selected and the individuals who are selected are just like the ones who are not selected

Stratified Random Sampling: It is a process to gather data by separating the actual population into the distinct subset or strata, and then choosing simple random samples from each stratum Your research question is about the height of all males at your college but recording height data for each individual would be very difficult and time consuming. You instead decide to use a sample of males in your data science class. This is a random sample as each male individual has an equally likely chance of being samples (that is, unless a prerequisite exists).

Sampling strategy can lead to bias. **Statistical bias** is a systematic tendency which causes differences between results and facts. If instead of your classmates, you had chosen a different sample, like the men's basketball team, your results would have been biased as basketball players are taller on average.

1.3.3.1 Sample vs Population

Is the study sample a representative sample of the population?

1.3.3.2 How to draw samples

Random Sampling: It is a sampling method in which all the items have an equal chance of being selected and the individuals who are selected are just like the ones who are not selected Stratified Random Sampling: It is a process to gather data by separating the actual population into the distinct subset or strata, and then choosing simple random samples from each stratum Your research question is about the height of all males at your college but recording height data for each individual would be very difficult and time consuming. You instead decide to use a sample of males in your data science class. This is a random sample as each male individual has an equally likely chance of being samples (that is, unless a prerequisite exists). Sampling strategy can lead to bias. Statistical bias is a systematic tendency which causes differences between results and facts. If instead of your classmates, you had chosen a different sample, like the men's basketball team, your results would have been biased as basketball players are taller on average.

1.3.4 Study Design Considerantions - Bias

is there bias in the sampling? is there bias in the data types collected?

If so, Be explicit Our classroom represents a subset of individuals in this country: college aged attending college specific geography specific time period

we group categories a,b,c to make new groups for analysis.

In doing so, we limit our interpreations to ________

1.3.5 EXPLORATORY DATA ANALYSIS

The first step in understanding and interpreting our data is called an exploratory data analysis. We will use a few measurements to quickly look at the data and then we can use some simple graphing techniques to turn our data into visualizations. The first three M's, you are likely familiar with and are often referred to as measures of central tendency: Mean, Median, and Mode. These go along with range, outliers and sample size.

- Mean
- Median
- Mode
- Outlier
- Range

  What is a statistic?

1.3.6 Exploring height

If you do not have class data on height, we will be using the following simple dataset of 5 individuals:

- Can you describe what is happening in the following `codechunk`??
- What do `person, height` represent in relationship to `ex_height`??

Click to show answer

> We **create** the **R object** `ex_height` using the **assignment operator** `<-` `ex_height` is a **data.frame**, a table, with two columns: `person` and `height` `person, height` are the two columns, or variables, of `ex_height`

```
ex_height <- data.frame(
  "person" = paste("Ind",
                   c("a", "b","c","d","e"),
                   sep = "_"),
  "height" = c(5.5, 5, 6, 5.25, 5)
)
```

```r
knitr::kable(ex_height)
```

person	height
Ind_a	5.50
Ind_b	5.00
Ind_c	6.00
Ind_d	5.25
Ind_e	5.00

1.3.7 Mean height

You probably already know this one. The (arithmatic) mean is calculated by adding all **values** together, and dividing by the **number of observations**. For our dataset, we add all 5 heights together and divide by the number of individuals (5):

$\frac{(5.5+5+6+5.25+5)}{5}$

$\frac{26.75}{5}$

5.35

In `R`, we can write this out "by hand." Since `R` is a **statistical programming language**, r recognizes basic mathematical expressions. We can `code` the following to calcualte the mean:

```r
(5.5 + 5 + 6 + 5.25 + 5)/5
```

```
## [1] 5.35
```

Since `R` is an **object-oriented programming language**, we don't need to write out individual numbers for each calculation. Instead, we can refer to the `ex_height` **object**. Since `ex_height` is a **data.frame**, we can refer to it variables by name with the `$` operator. We also take advantage of some of the built-in mathematical functions of `R`: `sum(), length()`

```r
sum(ex_height$height)/length(ex_height$person)
```

```
## [1] 5.35
```

1.3.8 Median height

The **median** is calculated by ordering all values from small to large. If there are an **odd** numberof values, there will be a single value at the middle.

Our original data:

5.5, 5, 6, 5.25, 5

Our data re-ordered from small to large, then we simply cross off values from either end to find the middle value:

5, 5, 5.25, 5.5, 6

~~5~~, 5, 5.25, 5.5, ~~6~~

~~5~~, ~~5~~, 5.25, ~~5.5~~, ~~6~~

If we had an even number of values, we would wind up with two "middle values", in which case we take the mean of these two.

If we had one more value, let's say `5.75`, we wind up with both `5.25` and `5.5` as middle values.

5, 5, 5.25, 5.5, 5.75, 6

~~5~~, 5, 5.25, 5.5, 5.75, ~~6~~

~~5~~, ~~5~~, 5.25, 5.5, ~~5.75~~, ~~6~~

The median is the **mean** of these two values:

$\frac{5.25+5.5}{2}$

5.375

In a small dataset, it's easy to pick out the middle value. Fortunately, there's an `R` function for this as well:

```
## With 5 Individuals
median(ex_height$height)
```

```
## [1] 5.25
```

```
## With 6 individuals
median(c(ex_height$height, 5.75))
```

```
## [1] 5.375
```

> Did you notice the `c()` function above? What does it do?

1.3.9 Modal height

The **mode** is the most common value in the dataset. Here again, it's easy to pick out there are 2 people with a `height` of `5`, and all other values are represented by just one person. There's actually no build-in `R` function to calculate the **mode**...

But don't worry, there are functions that let you decide the **mode** and more!! A very useful function is `table()`, used to make **counts** of values. In a small dataset, we see that two individuals have a height of 5. Our **mode** is 5!

```
table(ex_height$height)
```

```
##
##    5 5.25  5.5    6
##    2    1    1    1
```

1.3.10 Range

The **range** of the data is two numbers, the lowest, and highest values within the data:

```
range(ex_height$height)
```

```
## [1] 5 6
```

Though, if you want just the **minimum** or **maximum** value, you can use `min()` or `max()`:

```
min(ex_height$height)
```

```
## [1] 5
```

```
max(ex_height$height)
```

```
## [1] 6
```

1.3.11 Outliers

So far, some of these examples may have so obvious you may be thinking: *what's the point??*

For a slightly more practical demonstration, try out the above functions on your `class_data`. You can also access our sample dataset, curtousey of, IPUMS-Health Surveys. The heights and weights dataset is included in the accompanying `ipumsED` R package.

There are a few ways to actually get your data into `R`, depending on the file type your data is saved as. If it is a `.csv`, you can use the base-R function `read.csv()`. If you have data in an Excel file (.xls, .xlsx) you will need install the `readxl` R package in order to use the add-on function of `read.xlsx()`.

> Notice the simliarity between `read.csv()` and `read.xlsx()`?? While packages are created by the R community, they often try to maintain syntanx/coding conventions from base R.

In order to keep things organized, we will first create an object called `my_data_path` that contains the **file path** to the data I want to read. You can type this out as a single character string, or use the `file.path()` function, which adds in `/`s for you. Next we **pass** our new object, `my_data_path`, as the **argument** to either `read.csv()` or `read.xlsx`

```
## for now working copy for dan
my_data_path <- file.path("..", "ipumsED", "data", "nhis_sample.csv")



my_data <- read.csv(my_data_path)
```

If you have the `ipumsED` package installed, you can load the dataset into your local environment simply by using the `data()` function.

```
data(nhis_data)
```

We can inspect our data object with some basic functions. Notice how the output differs when the function is called on a `data.frame` versus when it is called on a `vector`.

```
class(my_data)
```

```
## [1] "data.frame"
```

```
summary(my_data)
```

```
##      SERIAL            AGE              SEX             HEIGHT
##  Min.   :   1.0   Min.   : 0.00   Min.   :1.000   Min.   : 0.00
##  1st Qu.: 247.0   1st Qu.: 29.00   1st Qu.:1.000   1st Qu.:63.00
##  Median : 495.0   Median : 49.00   Median :2.000   Median :66.00
##  Mean   : 498.3   Mean   : 47.56   Mean   :1.517   Mean   :62.84
##  3rd Qu.: 748.0   3rd Qu.: 66.00   3rd Qu.:2.000   3rd Qu.:70.00
##  Max.   :1000.0   Max.   :997.00   Max.   :2.000   Max.   :99.00
##      WEIGHT
##  Min.   :  0.0
##  1st Qu.:136.0
##  Median :170.0
##  Mean   :216.4
##  3rd Qu.:205.0
##  Max.   :999.0
```

```
class(my_data$HEIGHT)
```

```
## [1] "integer"
```

```
summary(my_data$HEIGHT)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   63.00   66.00   62.84   70.00   99.00
```

Calling the R object itself usually prints the contents. For something this big, that's probably inadvisable, so we print just the first 10 records using the `head()` function.

There's also a `tail()` function for printing the last 10 recods

```
my_data
```

```
head(my_data)
```

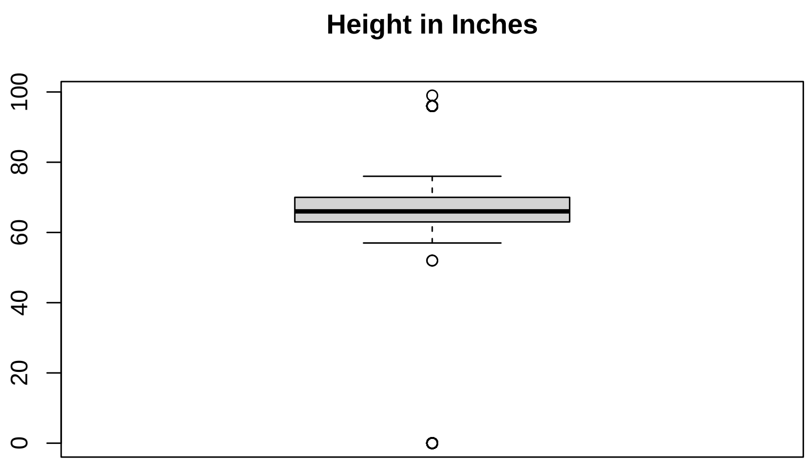
```
##   SERIAL AGE SEX HEIGHT WEIGHT
## 1    698  18   1     68    130
## 2    651  28   1     72    275
## 3    651   3   2      0      0
## 4    542  32   2     68    174
## 5    870  44   1     71    180
## 6    870   5   2      0      0
```

1.3.12 Find Outliers

1000 records is probably too much to review manually (and who would want to!?). Instead, we can **visualize** our data as one method to detect outliers. a **box-and-whisker plot** or **boxplot** is a common graph to plot continuous data like height.

Box-and-whisker plots, by default, show you which values are outliers! The box represents the middle 75% of the dataset. The whiskers account for another 11% on either end of the box. The **median** is represented by a thick black line. Any observations outside the whiskers are considered outliers

```
boxplot(my_data$HEIGHT, main = "Height in Inches")
```



In this case, the values of 0, 95,96,97,98,99 all represent special coded values. `0` Indicates someone Not In Universe - someone for whom this question wasn't asked. This may be an infant too young to meaningfully be measured. In this case higher values represent a value of `unknown` for various reasons. Both of these values represent missing data, but they mean different things.

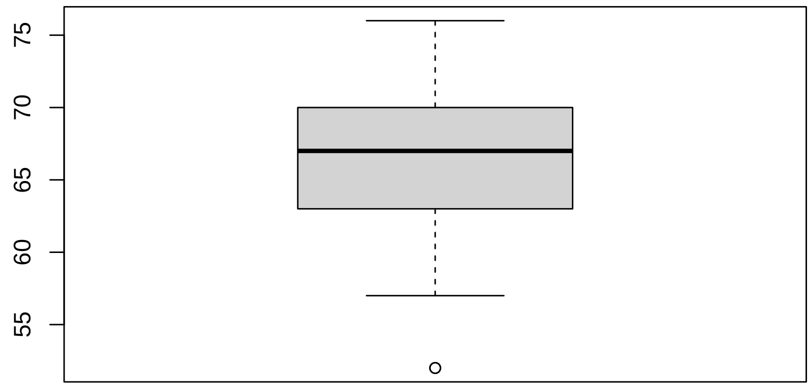
> Why do you think we have more than one value to represent missing data? How does a value of 0 compare to a value of 99??

We may want to exclude these special cases from our analysis. We can easily perform a data modification by filtering out values that equal any of our special cases. You can see the **range** of the graph is smaller, though visually, the box and whiskers are larger. Do we still see outliers in height?

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
new_data <- my_data %>% filter(HEIGHT > 0 & HEIGHT < 95)
boxplot(new_data$HEIGHT)
```



1.4 Elaborate

1.4.1 Your Turn!

For this example, you can use data collected in class, or the example dataset available in `ipumsED` R package: `class_ipums_data`

For in-class *survey* data, you are looking at the age of each individual in class. For the IPUMS data, you are looking at age as recorded in the American Community Survey (ACS). This survey is fielded annually. It contains more detailed questions that supplement the decennial census. **Raw data** are available directly from the US Census Bureau. However, we will be working with IPUMS data, **harmonzied.**. This process standardizes variable coding schema across all samples within the IPUMS dataset. IPUMS makes it easy for researchers to conduct analysis across time and space.

> IPUMS began as a PhD project digitizing historic US census data and reconciling it with the coding schema of the current US Census (1990s). IPUMS-International grew out of this by employing our harmonization practices on Census samples from other countries. IPUMS has partnerships with over 100 countries and is working on building partnerships with the remainng **82** countries!
>
> Partnerships with IPUMS may look slightly different for each country, but they always involve IPUMS commitment to being responsible stewards of the partner-country's data. Our goal is to make data as open as possible, while maintaining responsible disclosure risk controls.

Example data used throughout this book come from IPUMS-International so that values may be contextualized for a readers interested in one of the 103 countries that have data sharing agreements with IPUMS.

Write and `Rscript` that reads in your chosen dataset, and calculates the following statistics:

- `mean()`
- `meadian()`
- `mode`
- `range()`

In addition, include at least one plot or table and use it to discuss if your dataset contains any outliers. What criteria are you using to define an outlier?

> Where's the `mode()`?? As a statistical programing language, R was developed with built-in functions for `mean()`, `median()`, and `range()`. But `mode` wasn't important enough to warrant its own function. In part, because it's so easy to code, even you can do it!
>
> Yep, we're serious!

1.5 Evaluate/Exercises

1.5.1 Context:

Using the data set from Section 1.4 and the R statistics you produced, answer the following questions about our data set.

1.5.2 Questions:

Are the data quantitative or nominal?

Click to show answer

> Age data is quantitative.

Are the data discrete or continuous?

Click to show answer

> Age data is could be continuous depending on how you measure and record it, but the way we generally talk about 'Years of Age' is actually discrete.

Is the dataset primary or secondary data?

Click to show answer

> That depends on which data you are using. If it was collected in your class, it is primary data. If you are using the IPUMS data, it is secondary data as the information was collected by IPUMS from the US Census.

What sampling method was used?

Click to show answer

> Age data is quantitative.

Does any statistical bias exist?

Click to show answer

> Age data is quantitative.

Which measure of central tendency is best to describe this data? Mean, Median or Mode.

Click to show answer

> Age data is quantitative.

1.6 GLOSSARY

Quantitative Data Discrete Data Continuous data: Nominal Data: Ordinal Data: Primary Data: Secondary Data: Data Set: Observation: Variable: Random Sampling: Stratified Random Sampling: Statistical Bias: Mean: Median: Mode: Range: Outlier: Statistic: Sample Size:

1.7 Additional Content

1.7.0.1 examples from recent academic articles

article graphs, citations, instructors research, etc

1.7.0.2 examples from non-academic

customer satisfaction; workplace/department culture; r&D;

Chapter 2

Visualizing Data

CURRENTLY JUST A COPY OF THE OLD 1.2, what is R; needs to be updated

2.1 Engage

Which visualization best represents the data?

Are there any problems with the visualizations?

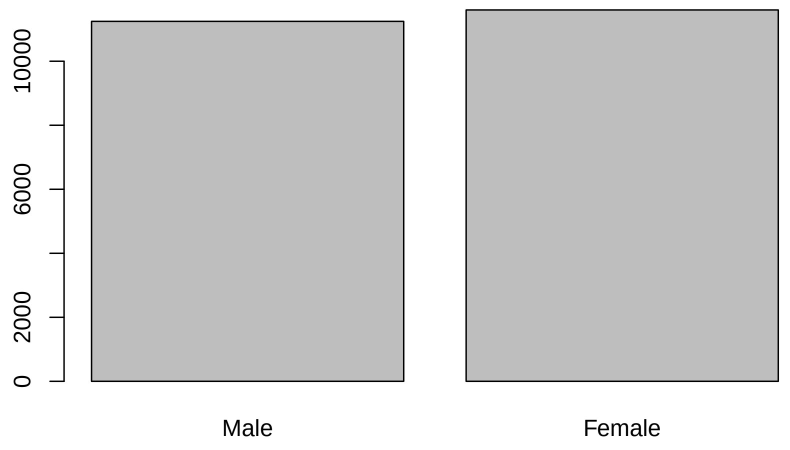
2.1.1 Vis 1

Click to show answer

> Barplots tend to be good choices to represent categorical data. They categories can be visualized side-by-side or stacked.

.pull-left[

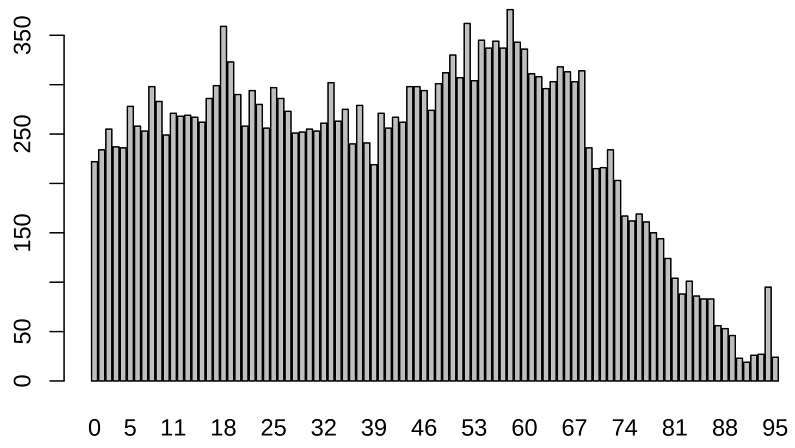
```
barplot(table(ff_sex))
```



]

.pull-right[

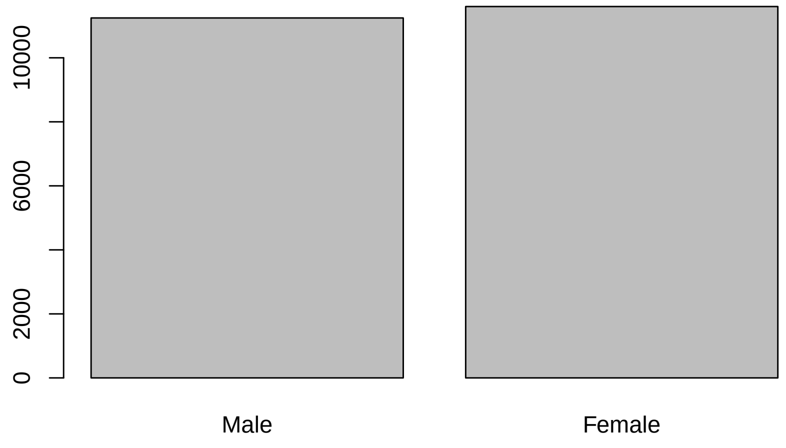
```r
barplot(table(ipums_data$AGE))
```



]

2.1.2 Vis 2

```r
plot(ff_sex, beside = FALSE)
```



2.1.3 Vis 3

```r
table(ff_marst)
```

```
## ff_marst
##    Single    Married Separated    Widowed
##      9452       9688      2394       1313
```

```r
round(prop.table(table(ff_marst)),2)
```

```
## ff_marst
##    Single    Married Separated    Widowed
##      0.41       0.42      0.10       0.06
```

2.2 Explore

What do you want it to show vs what it shows. Maybe find a real data image from news that is clearly not good visualization

2.3 Explain

this section should mostly describe, examples will come in elaborate

Visualizing data is the process of crafting a visual argument. All data visualizations require some level of subjective decsion making. Therefore any graph or table has some level of inherent bias. Effective data visualization involves crafting a visual argument, often one that conveys a direct story to the viewer.

As responsible researchers, educators, and data analysts we have a responsiblility to carefully consider our visual arguments.

An effective **and responsible** data visualization will lay out all relevant information (within reason), providing all the information for the reader to understand and build the visual argument for themselves.

2.3.1 Single Variable - Continuous

In general, your choice of visualization will depend on the **data type** that you are interested in. Continuous variables often get visually summarized using histograms, or box and whisker plots. These plots all serve to visually summarize the distrubtion and skew of a continuous variable.

- Continuous
  - Histogram
  - Box Plot
  - Summary Tables
  - plot by index - *example of not so helpful*

2.3.2 Single Variable - Categorical

Categorical variables, are often better visualized using bar graphs, pie charts, or other **quantity/count plots**.

- Categorical
  - bar graphs
  - pie charts
  - freq/prop tables

2.3.3 Multi variable - continuous

- scatter plots
- line graphs
- Cross Tabs

2.3.4 Mutli variable - mixed

- continuous ~ categorical
    - series of boxplots
- categorical ~ categorical
    - *does this exist??*
- categorical ~ continuous
    - logistic regression - kind of

2.3.5 Data Tables

This could go into the R section

2.4 Elaborate

In the next sections we will demonstrate some common visualzation using R. We will highlight common misteps and tips and tricks along the way. For each function we encourage you to try to come up with the syntax on your own, but you can click the `code` button to see how to generate the plots.

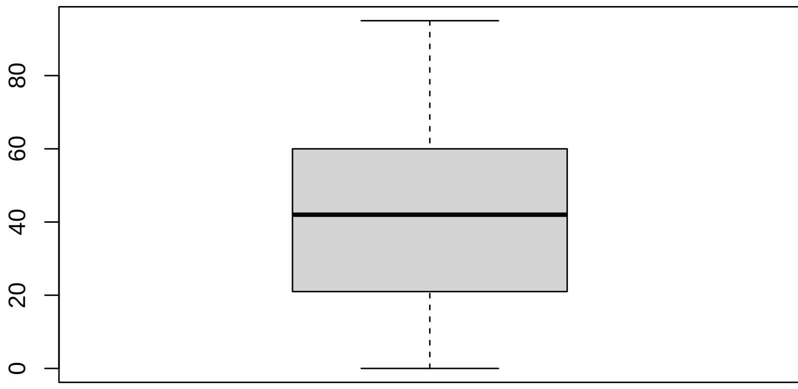
2.4.1 Making visualizations in R

2.4.1.1 Boxplot

Boxplots provide a great way to understand the distribution of the data at a glance. Three-quarters of the data are represented within the box portion of the plot. The full range of the data represented by the fences at the end of the vertical lines.

If there is an extreme range in the data, individual values will be plotted for the most extreme (3% of)

```r
boxplot(ipums_data$AGE)
```



2.4.1.1.1 Visualizations

2.4.1.1.2 Summary Tables The `summary()` function can be applied to many data types. When applying to a continuous variable, it will returns a **5-number summary** along with the **mean**.

Which numbers are included in the 5-number summary? How do these values relate to the box plot?

Click to show answer

> The 5-number summary describes a dataset by providing the minimum, maximum, and median values. In a addition, the lower and upper quartiles are returned.
>
> These 5 numbers represent key points on the boxplot. The extent of whiskers (minimum and maximum), the extent of the box (lower quartile and upper quartile), and the median.

```r
summary(ipums_data$AGE)
```

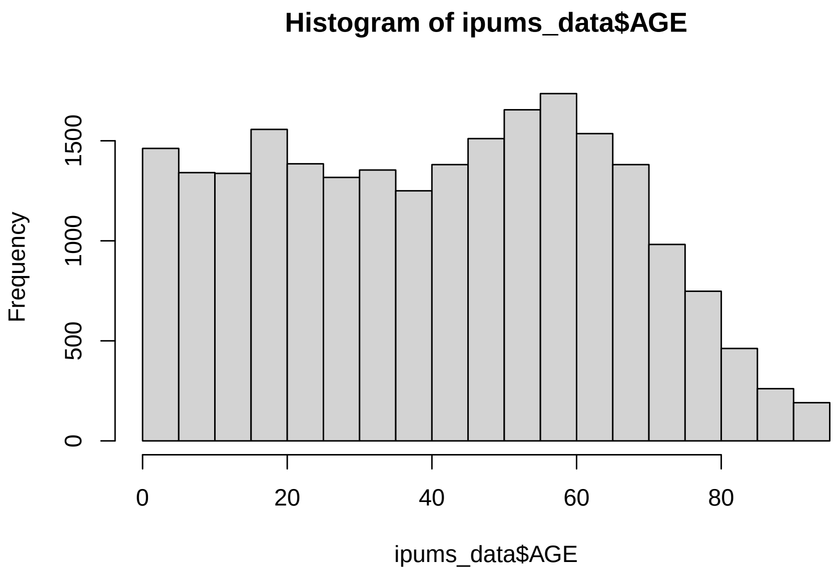
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   21.00   42.00   41.17   60.00   95.00
```

2.4.1.2 Histograms

Histograms divide a continuous variable into several `bins`.

Try out the `hist()` function on a continuous varible like `AGE`.

```
hist(ipums_data$AGE)
```



2.4.1.2.1 Visualizations

2.4.1.2.2 Summary Tables

The same summary information applies to histograms.

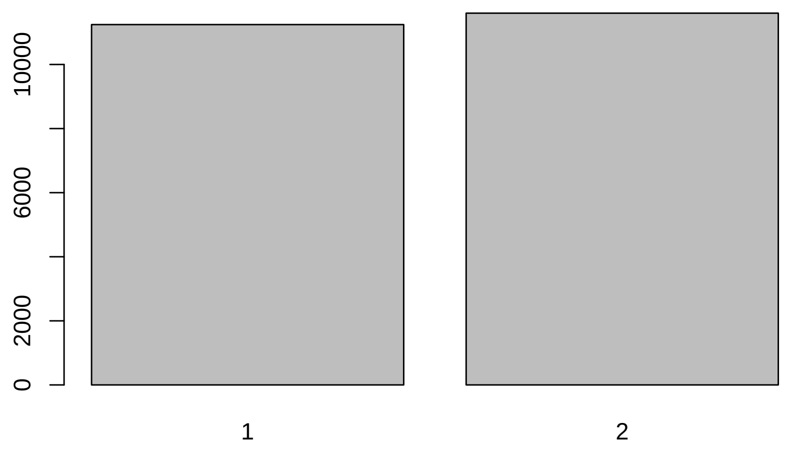
2.4.1.3 Bar Chart

```
class(ipums_data$SEX)
```

2.4.1.3.1 Visualizations

```
## [1] "integer"
```

```
barplot(table(ipums_data$SEX))
```



2.4.1.3.2 Summary Tables

2.4.1.4 Pie Charts

2.4.1.4.1 Visualizations

2.4.1.4.2 Summary Tables

2.4.1.5 Scatter Plots

2.4.1.5.1 Visualizations

2.4.1.5.2 Summary Tables

2.4.1.6 Line Graphs

2.4.1.6.1 Visualizations

2.4.1.6.2 Summary Tables

2.4.1.7 Cross Tabs

can these be worked into the above sections?

Others...

2.4.2 Which visualization is best for my data? Things to consider

List things to consider for a visualization

2.5 Evaluate/Exercises

Give them a data set and have them think through why they choose a specific visualization. Have the create and answer questions about it. We will go through the list of possible visualizations for the same data and explain which work, which dont, and why

Chapter 3

Describing Data

see old_ for some examples, not very exstenive

3.1 Engage

post-it note histogram of height: * one with MANY bins * one with FEW bins * *do these graphs tell the same story??*

3.2 Explore

How do we describe this meaningfully?? can we determine the "average"

have class guess at average (likely the heighest point of histo) * Interrogating a commonly held assumption + the average height of humans is 5'5"

Pose questions to explore

what does "average" mean??

3.3 Explain - lecture/read

3.3.1 How do we describe data - summary statistics

3.3.2 How do we describe the distribution of data

3.3.3 How do we manipulate data

3.3.3.1 coding/categorizing free response/ "binning" a continuous variable

Height, while we treat as continuous, is actually "binned" into ~60 unique values.

3.4 Normality

What does it mean to be normally distributed. (how measures of central tendency compare) discuss skew / kurtosis specialized vocabulary to describe data Why people care about normality (what tests can/t you use and why)

3.5 Elaborate

Read/write data Visualize data - describe the distribution; compare to board results Tabulate Data - describe with summary statistics Test of normality: one-way t-test ; QQ plot ; shapiro-wilks (note high rate of rejection) Demo binnig of ages

3.6 Evaluate/Exercises

Write an R script that: 1. reads in data 1. tests normality across variables 1. print results as table 1. prints results as graph.

Reflect: Are any variables non-normal? Do you have any interpretations as to why? Do you have any questions to test

Chapter 4

Hypothesis Testing - Comparison and Correlations

4.1 Engage

Post-it histogram - height separated by birth month (may require a lot of board space)

Are the distributions similar? different? **too many small samples to tell**?

4.2 Explore

How do we **group** data in order to compare it.

The most simple question in statisitics is: Is x bigger than y

How can we do that for our groups here?

Summer/Winter?

conduct demo/small-scale experiment/analysis Ask leading questions, form hypotheses; * can the example be grouped / broken into subgroups + does the same pattern/phenomenon apply? * Is there a corollary / inverse phenomenon?

Pose questions to explore

Discuss/interrogate the pattern of the data * does the shape imply anything * try to have it student led / guided * with time, students guess at what the data shows/doesn't show without labels

4.3 Explain - lecture/read

demo recode: birth month -> Summer/Winter (daylight-savings or not); recode favorite food to sweet/savory

4.3.1 Why 2 groups?

Historical problem of statistics. We've developed very good methods for very un-reasonable situations. In order to make these tests work, we often have to **manipulate** our data in order to meet the **assumptions** of the specific test or analysis that we'd like to perform.

4.3.2 Common tests to compare 2 groups

t-test and chisq | box-whisker, histogram, density distribution

4.3.3 Correlation(s)

correlation (pearson, spearman) | scatter plot height ~ hrs_slept ; height ~ fav_food_category

4.4 Elaborate

demo recode: birth month -> Summer/Winter (daylight-savings or not); recode favorite food to sweet/savory t-test and chisq | box-whisker, histogram, density distribution height ~ birth_SW correlation (pearson, spearman) | scatter plot height ~ hrs_slept ; height ~ fav_food_category What is an hypothesis??

4.5 Evaluate/Exercises

Formulate a hypothesis and test it using class survey data. Recode birth month to birth_quarter; recode favorite food to 3 or more categories

Chapter 5

Hypothesis Testing - ANOVA and LM

5.1 Engage

Compare MORE than 2 groups

Post-it histogram - height separated by Birth Mo (reclassified to birth quarter)

5.2 Explore

Pose questions to explore

Are the four distributions similar? different?

5.3 Explain - lecture/read

What is ANOVA, how to interpret tabularly, graphically (box-whisker, histo, density); non-parametric alternative (KS) What is a linear model? LM vs Correlation vs ANOVA What does a linear relationship imply? LM = continuous ~ continuous ANOVA = continuous ~ categorical

5.4 Elaborate

Conduct recoding for birth quarter conduct ANOVA Conduct Linear Model

Work through the same process/analysis on new data. Either a different variable(s) in the same sample, or applying/demonstrating the phenomenon using real-world data/examples.

- Reiterate/reinforc the process/task of the lesson
- Does this fit in to other steps/analysis
- does this fit in to other thematic topics?

(Possibly) Build on the first one to show variation. * Other ways to represent the phenomenon/data + tabular + visually

5.5 Evaluate/Exercises

Formulate a hypothesis and test it using class survey data.

Unit 2: IPUMS

Lesson 6 Introduction to IPUMS

Some text to break up the sub-section headers

Intro to IPUMS website

background on ipums

navigating website

Find certain (very common) variables to answer (common) social science questions.

Lesson 7 Exploratory analysis

If you've just collected a survey, or other raw data, you may not know what you're looking for. This is perfectly ok but goes against *the scientific method* most people learned in grade school (More on that to follow(***include_link***)).

This unit begins by presenting data/distributions and asking students to begin interpreting the data . visual exploration is encouraged and basic of data manipulation are taught * *EG:* how to subset data, how to reshape data, how to recode data, how to convert from one `data type` to another.

Example lab exercise:

Students given a data set (xls, csv, etc) * load data, perform manipulations, basic summaries + cross tabs + group means by a covariate * inspect data visually + *DESCRIBE* the distribution - is it normal? significant? * *FIND* aquestion in the spread of the data + how can you test this (maybe small group work) * write up/ present results + think on confounding factors / biases

Advanced Exploration - Change Over Time

Here we demonstrate an approach to looking at how Family Structure (inferred from household relationships) has changed over time.

Setup / Load Data

Install/update R packages

```
install.packages("ipumsr")
install.packages("tidyverse")
```

Data extract created online using the datacart system.

```
library(ipumsr)
library(dplyr)


ddi <- read_ipums_ddi("Data/ipumsi_00005.xml")
data <- read_ipums_micro(ddi)
```

Inspect the Data Using `haven` labeled values.

```
data$RELATE[1:100]
class(data$RELATE)

data %>% count(RELATE)
data %>% count(SEX)
```

What were those codes ??

```
## need to convert this to an image or something similar; kable table?
ipums_view(ddi)
```

Visualize A simple plot

```
plot(AGE ~ YEAR, data = data)
```

A fancier plot

```
plot(AGE~YEAR, data = data, type = "n", main = "Age by Sex, over Time, CO")
points(data$YEAR[data$SEX==1]-1, data$AGE[data$SEX==1], pch = 16, col = hsv(.6,.6,.8,.2))

points(data$YEAR[data$SEX==2]+1, data$AGE[data$SEX==2], pch = 16, col = hsv(1,.6,.8,.2))

abline(lm(AGE~YEAR, data = data), col = "green")
```

Asking (logical) questions Here we demonstrate how setting up logical questions can be used to easily filter/subset data.

```
age_test <- data$AGE > 18

class(age_test)

age_test
```

Logical vectors are stored as `TRUE` or `FALSE`, but can also be evaluated numerically as `1` or `0` respectively. We can therefore `sum()` the number of `TRUE` values and divide by total rows for a proportion.

```
sum(age_test)/nrow(data)
```

HH vs persons A unique characteristic of census and some survey data is the nested-structure with individuals being grouped into households. Often times it is necessary to choose to work at the hh or person level, and data must be appropriately manipulated to fit that case.

```
hh_total <- length(unique(data$SERIAL))
hh_total
ipums_view(ddi)
```

Nuclear Family

First we look at a nuclear family, comprising only parents and their immediate children.

```
library(ipumsr)
library(dplyr)

ddi <- read_ipums_ddi("/pkg/ipums/personal/ehrli097/AABA_2022/Data/ipumsi_00005.xml")
all_data <- read_ipums_micro(ddi)
```

```
census_years <- c(1860, 1870, 1880, 1900, 1910, 1960, 1970, 1980, 1990, 2000, 2010)

## subset census only
d2 <- all_data %>% filter(YEAR %in% census_years)

## make a household dataframe
hhs <- d2 %>% distinct(YEAR, SERIAL, .keep_all = TRUE) %>% select(YEAR,SERIAL,GEO1_US)

hhs %>% View()
```

```
hhs <- d2 %>% filter(RELATE ==4) %>%


  distinct(YEAR, SERIAL) %>% mutate(extended_test=TRUE) %>% right_join(hhs, by = c("YEA










hhs <- d2 %>% filter(!RELATE %in% c(1, 2, 3) |
                  (RELATE == 3 &
                     MARST %in% c(2, 3, 4))
                ) %>%




  distinct(YEAR, SERIAL) %>% mutate(nuclear_test = FALSE) %>% right_join(hhs, by = c("



table(hhs$extended_test,hhs$nuclear_test)
```

```


  hhs <- d2 %>% filter(RELATED %in% c(4200, 4210, 4211, 4220, 4500, 4510, 4600)) %>% di


  hhs <- d2 %>% filter(RELATED %in% c(4100, 4110, 4120, 4130, 4300, 4301, 4302)) %>% di
  res_tabs <- list(
    "nuclear_test" = hhs %>% group_by(YEAR, nuclear_test,GEO1_US) %>% summarize(.groups="drop",n
  "extended_test" = hhs %>% group_by(YEAR, extended_test, GEO1_US) %>% summarize(.groups="drop",n
  "parent_test" = hhs %>% group_by(YEAR, parent_test, GEO1_US) %>% summarize(.groups="drop",n = n
  "children_test" = hhs %>% group_by(YEAR, children_test, GEO1_US) %>% summarize(.groups="drop",n
  )
```

```
collapsed_results <- res_tabs %>% purrr::map(function(x){
  x <- x %>% group_by(across(names(x)[1:3])) %>% summarize(.groups="drop",n = sum(n))

})


collapsed_results <- lapply(collapsed_results, function(x){
  colnames(x)[2] <- "test"
  colnames(x)[3] <- "state"
  return(x)
})

combined <- collapsed_results %>% purrr::reduce(full_join, by = c("YEAR", "test", "state"))



colnames(combined) <- c("YEAR","test", "state", "n_nuclear", "n_extended", "n_parent", "n_childre

combined[is.na(combined)] <- 0


to_plot <- combined %>% group_by(YEAR, state) %>% mutate(n_tot = sum(n_nuclear)) %>% ungroup() %>
```

Tabulate results

```
to_plot <- to_plot %>% filter(test==TRUE)



plot(to_plot$YEAR, to_plot$pct$n_nuclear, col = hsv(.4, .6,.8), pch = 16, ylim =c(0,1), xlab = "
```

Visualize Nuclear Families

Extended Family

Next we look at hhs with extended families present. IE, any that contain more relationships than just Parent/Child/Sibling (between children only)

```
to_plot <- to_plot %>% filter(test==TRUE)


glm_hist <- glm(pct$n_extended ~ YEAR, data = to_plot[to_plot$YEAR < 1950,], family = 



glm_hist_x <- seq(from=1860, to = 1910, length.out = 100)
glm_hist_y <- predict(glm_hist, list(YEAR = glm_hist_x), type = "response")

glm_mod <- glm(pct$n_extended ~ YEAR, data = to_plot[to_plot$YEAR> 1950,], family = qua

glm_mod_x <- seq(from = 1960, to = 2010, length.out = 100)
glm_mod_y <- predict(glm_mod, list(YEAR = glm_mod_x), type = "response")

mods <- list("hist"=list(),
             "mod" = list()
             )
mods_plots <- list("hist"=list(),
                   "mod" =list()
                   )

for(i in names(to_plot$pct)){

  hist_x <- to_plot$YEAR[to_plot$YEAR < 1950]
mod_x <- to_plot$YEAR[to_plot$YEAR > 1950]

  mods$hist[[i]] <- lm(pct[[i]] ~ YEAR, data = to_plot[to_plot$YEAR < 1950,])

  mods_plots$hist[[i]] <-
    data.frame("x" = hist_x,
               "y" = predict(mods$hist[[i]],
                             list(YEAR =hist_x),
                             type = "response")
               )
  mods$mod[[i]] <- lm(pct[[i]] ~ YEAR, data = to_plot[to_plot$YEAR > 1950,])


  mods_plots$mod[[i]] <-
    data.frame("x" = mod_x,
               "y" = predict(mods$mod[[i]],
                             list(YEAR =mod_x),
                             type = "response")
               )
}
```

Gernerate models

```
plot(to_plot$YEAR, to_plot$pct$n_extended, col = hsv(.95, .6,.8), pch = 16, ylim =c(0,.25), bg =


lines(glm_hist_x,glm_hist_y, col = hsv(.95, .3, 1), lwd = 2)
lines(glm_mod_x, glm_mod_y, col = hsv(.95, .3, 1), lwd = 2, lty = 2)




points(to_plot$YEAR,
       to_plot$pct$n_extended,
       pch = 23,
       bg = hsv(.95,.6,.8))
```

Visualize

Even more DETAIL - maybe remove

```
ipums_view(ddi)
```

```
  hhs <- d2 %>% filter(RELATED %in% c(4200, 4210, 4211, 4220, 4500, 4510, 4600)) %>%

  distinct(YEAR, SERIAL) %>% mutate(parent_test=TRUE) %>% right_join(hhs, by = c("YEAR", "SERIAL'

  hhs <- d2 %>% filter(RELATED %in% c(4100, 4110, 4120, 4130, 4300, 4301, 4302)) %>% distinct(YEA
plot(to_plot$YEAR, to_plot$pct$n_extended, col = hsv(.95, .6,.8), pch = 16, ylim =c(0,


lines(glm_hist_x,glm_hist_y, col = hsv(.95, .3, 1), lwd = 2)
lines(glm_mod_x, glm_mod_y, col = hsv(.95, .3, 1), lwd = 2, lty = 2)


lines(mods_plots$hist$n_parent,col = hsv(.8, .3,1), lwd = 2)

lines(mods_plots$mod$n_parent, col = hsv(.8, .3,1), lwd = 2, lty = 2)

points(to_plot$YEAR,
       to_plot$pct$n_parent,
       pch = 23,
       bg = hsv(.8, .6,.8))


points(to_plot$YEAR,
       to_plot$pct$n_extended,
       pch = 23,
       bg = hsv(.95,.6,.8))
```

Parents Supporting Parents

```
plot(to_plot$YEAR, to_plot$pct$n_extended, col = hsv(.95, .6,.8), pch = 16, ylim =c(0,


lines(glm_hist_x,glm_hist_y, col = hsv(.95, .3, 1), lwd = 2)
lines(glm_mod_x, glm_mod_y, col = hsv(.95, .3, 1), lwd = 2, lty = 2)



lines(mods_plots$hist$n_children, col = hsv(.55,.3,1), lwd = 2)


lines(mods_plots$mod$n_children, col = hsv(.55,.3,1), lwd = 2, lty = 2)


points(to_plot$YEAR,
       to_plot$pct$n_children,
       pch = 23,
       bg = hsv(.55,.6,.8))

points(to_plot$YEAR,
       to_plot$pct$n_extended,
       pch = 23,
       bg = hsv(.95,.6,.8))
```

Parents Supporting (extended) children

Lesosn 8: Hypothesis Testing

If, on the other hand you have an a pre-existing idea you want to test. We can follow the traditional *scientific method*. With a question in mind, the first question is: where to look. What better place than IPUMS!

Begin introducing navigation of web resources - mainly IPUMS international

Students should become comfortable working through lab exercises: * Define a question (or be presented with one) * Download variables from IPUMS (course downloads possible) * Perform a basic analysis (discussed in Unit 1) * Generate a **visual argument** for your analysis + Include explanation/interpretation/reflection on the question at hand, and the data used + Any obvious biases + Any obvious confounding factors

Lesson 9: Statistical Inference

Lesson 10: (TBD)

We describe our methods in this chapter.

Math can be added in body using usual syntax as follows. This may be useful, particularly for explaining the math side of things.

Unit 3: Independent Research

Students will select their own research question that can be answered with the IPUMS data set and will spend five weeks conducting a research project complete with data analysis, visualization, and interpretation.

In this section we encourage the instructor to provide ample time for independent student/small-group research. Some class time should be devoted to modeling healthy discussion and critique of methods. Students should learn to discuss not just *how* to answer a research question but *why* they are asking/answering it. What impact does the question/answers have. Is the question releveant/meaningful, and importantly, Is this research question perpetuating racist ideas.

We provide some examples here but encourage instructors (or students) to bring in recent journal/popular articles that do (or do not) apply data science methods well.

Lesson 11: Students develop research Question

Lesson 12: Students find relevant variables from IPUMS

Lesson 13: Students test and evaluate results

Lesson 14: Students prepare presentations of results

Lesson 15: Students present work (slides, poster, podium, etc)

By this point, students should be familiar with basic concepts from Chapter Unit 1. These include:

- Basic Coding
    - read/write data in/out of R
    - basic manipulations
- Theoretical Basis
    - looking at data distributions
    - formal assessment of distributions

Students will also be familiar with how these concepts are applied from Chapter Unit 2. Hopefully students will be able to:

- Come up with a social science question they are interested in
    - Critically think about target variable(s) of interest. Any *a priori* covariates? confounders?
    - Acquire relevant data from IPUMS
    - Analyze, Summarize, Visualize Data
        - scope and complexity at student/teach discretion
    - Present research to class
        - **potentially** critically discuss/evaluate each others work.
        - **science is collaborative** everyone should be out to do their best work and represent the data as best we can. We all have conscious and unconscious biases, and the best way to confront them is share and receive (respectful) feedback.

During this Unit, we suggest giving ample class time for independent student research, peer-to-peer collaboration, and basic R/stats troubleshooting. This would also be a great time to model how to give respectful criticism by discussing recent research papers. * We could maybe come up with 1-2 seed examples, with a few talking points

5.5.1 Example one

5.5.2 Example two

Appendix 1: Intro to R/Rstudio

A crash course to teach you everything you need to know…

To teach yourself more with the rest of this book!

Basics

R vs RStudio

Let's start with some analogies:

- If `R` is the engine, then `RStudio` is the car.
- If `R` is the text, `RStudio` is the text-editor.

What do you think some differences are between R and RStudio?

Click to show answer

> `R` is the **programming language**, `RStudio` is the **Integrated Development Environment (IDE)**

`RStudio` is a program/app just like Google Chrome or Microsoft Word. Each of these programs provide a **Graphical User Interface (GUI)**, a pretty way for a user to use a mouse and keyboard to do *something.*

An **IDE** is a special kind of program/app that provides MANY tools for writing and running code.

Throughout this text, we will use **RStudio**, to write (aka, **code**) **R-scripts** that will transform, analyze, and/or visualize data.

R is a language that can DO a lot. **RStudio** is a program that lets **YOU** do a lot **USING** the **R language** (and other languages).

RStudio vs POSIT

One last bit of definitions: **POSIT** is a new name/re brand of the company that produces and maintains the RStudio software. In the past, the RStudio (company) produced the RStudio (software).

Now, POSIT (company) produces RStudio (software), along with Quarto (software), and other resource for data scientists.

How to install

5.5.3 R, RStudio

R is available directly from the Comprehensive R Archive Network, CRAN, located at https://cran.r-project.org/. You can find the latest official and development versions of **base-R** and all contributed packages hosted by CRAN.

You can also go straight to the POSIT website, https://posit.co/download/rstudio-desktop/, where you can find download links for both R and Rstudio.

Beyond Base R

Base R Generally refers to the core set of **R packages** that come bundled together when you download R from CRAN or POSIT. **Packages** are collections of R code that can expand the functionality that is not included with the base R packages. For example, this book was made using the `bookdown` Rpackage which itself extends the `Rmarkdown` package. Together, these packages make it easy to produce high quality textbook style materials using only R!.

This section will focus on using base R commands to orient users, however this book does draw on several additional packages, namely the IPUMS packages:

- ipumsr
- ipumsED
- ipumsEDbook (that's the text you're reading!)

R packages need to be **installed** one time which downloads a copy to your local R package library. This is most often done using the `install.packages()` command, which whill download R packages straight from CRAN.

For example, you could run the following to download the packages mentioned above. Note how the `c()` function is needed in order to provide a list of R packages the user wishes to download.

```r
## One package
install.packages("ipumsr")


## Several packages

install.packages(c("ipumsr", "ipumsED", "ipumsEDbook"))
```

Then, any time you want to use functionality from that package, you can use the `library()` function at the start of your **R session**. In general, your **R session** lasts for the entire time you are actively working. You can **restart** your session in order to clear your workspace, memory, etc. Or, if your computer goes to sleep, your session will also (most likely) automatically restart.

Note that `library()` does not expect a characer string (in quotes `""`), and that it only takes a single package name.

```r
library(ipumsr)
library(ipumsED)
```

5.5.4 ipumsEDbook

If you want to edit the book text directly, you'll need a few additional tools.

First, you'll need the R `devtools` package. This can be downloaded with `install.packages(devtools)`. You will also need `Rtools`, which is not an R package and must be downloaded manually from CRAN.

OS-specfic versions can be found here: Windows Mac Linux

In addition to devtools and Rtools, you will need **GIT**. GIT is a version-control system that is commonly used to track and maintain software. GIT provides MANY tools that most researchers won't really need to interact with. The main reason we use GIT is to host a **repository** on a website, like **github.com**.

By hosting our project on a remote site like github, we ensure that a version is always available. Other researchers can create a copy of their own, and download whatever the latest version.

5.5.4.1 Setting up GIT

Download instructions for GIT can be found here.

5.5.4.2 Setting up GITHUB

5.5.4.3 Downloading from GITHUB

Note also that while downloading `ipumsEDbook` provides all the source code for this document, it is unlikely that you would ever need to load the package with `library()`. Instead, you can use the following notation to call a specific function without loading the entire package. In this case, we call a function which will regenerate the html version of this document.

```
## need to add a function to make it easy to recompile
ipumsEDbook::regenerate_book()
```

Navigating RStudio

Lots of Panes, not a pain!

> You get used to it, promise.

5.5.5 Console

The console is where you interact with `R` directly.

The console is technically where you input the code to be run in R.

5.5.5.1 Simple Code

At its simplest, R is a calculator. You can enter basic math equations into the console like. R will read the input, interpret it, and print some kine of output.

The exact output you receive depends on the input!

```
2+2
```

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

As a statistical programming language, R is capable of basic arithmetic and following order of operations.

```
(2+2)^2/2
```

```
## [1] 8
```

5.5.5.2 R Objects

R is also an object oriented programming language. This means you can create R-objects. These objects can store complicated input or output. R objects can really be anything, and R objects can be used to create new types, or **classes** of other R objects.

Let's start simple. Try to describe, in plain language, what's going on in the code below. In other words, how do you as a human, interpret the code.

What do you think will happen when we **run the code**

```
my_answer <- 1+2+3+4+5+6+7+8+9
```

Click to show answer

> In the code chunk above, we create a new R object called my_answer. This object is the result or sum of the numbers 1 through 9.

So what *is* `my_answer`???

In the above code chunk, we add up the numbers 1 through 9 and save the result to a new R object, which we call `my_answer`. R will print results by default, as we saw in the first math expressions. But in the above codechunk, we "catch" the result, so to speak, in out R object. We can access the result by calling the R object by name.

```
my_answer
```

```
## [1] 45
```

R functions

In addition to objects, a second important concept to programming in R is **functions**. Functions are actually just a special **class** of R object. What makes functions special is that they **do** something. In general functions take one or more **arguments** as **input**, they do some kind of action like calculations, visualizations, or data transformations, and the produce some kind of **output**

R functions are easily recognizable because they end in parentheses `()`. Arguments to the function go within the parentheses.

Instead of typing the `+` sign multiple times, as we did above, we could use the `sum()` function. In the code on the right, we pass the numbers 1 though 9 as arguments to the `sum()` function. The arguments go within the `()` separated by commas.

.pull-left[

```
1+2+3+4+5+6+7+8+9
```

```
## [1] 45
```

]

.pull-right[

```
sum(1,2,3,4,5,6,7,8,9)
```

```
## [1] 45
```

]

When we create objects in R, we use `<-`, which is known as the **assignment operator**. The assignment operator is a special function. instead of using `()`, Everything to the right of the assignment operator is taken as the input. Whatever is to the left of the assignment operator is also input! This becomes the name of the R object.

The output of the assignment operator is the R object itself.

```
my_sum <- sum(1,2,3,4,5,6,7,8,9)

my_sum
```

```
## [1] 45
```

> Notice how we create the object, `my_sum` and then call the object, by name, on the next line, in order to print the results.

Be careful! The assignment operator **does** have some limitations on what it can/can't take as input. Try to run the following code in your own R session:

```
my_sum <- 1,2,3,4,5
```

If you can't run the code, click to show result

```
    Error: unexpected ',' in "my_sum <- 1,"
```

So what went wrong??

Technically, the assignment operator is expecting just one thing to the right of it. In the working code `my_sum <- sum(1,2,3,4,5,6,7,8,9)` , R interprets, or evaluates, the `sum()` function, calculating a result that is a single value, 45, which we give the name `my_sum`.

If you want to store a list of values, you need to **concatenate** them first. This is probably a common concept to anyone with some kind of programing experience before. To concatenate something is simply to put it together. In `R`, you will get **very** used to using the concatenate function - in fact, it's so common the creators of R wisely gave it a very simple, easy to remember name: `c()`.

Using `c()` is easy. It takes any number of inputs, and joins them together into a single **vector**. A **vector** is an R object that lists several values. Notice the difference?

.pull-left[

```
my_value <- sum(1,2,3,4,5,6,7,8,9)

my_value
```

```
## [1] 45
```

]

.pull-right[

```
my_vector <- c(1,2,3,4,5,6,7,8,9)

my_vector
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

]

5.5.5.3 Calling functions on objects

We can use some basic R functions to interact with our R objects. The first function tells us the `class` of our R object. While `length()` tells us how long the object is. Notice how `my_value` and `my_vector` have the same class, but different lengths.

.pull-left[

```
my_value <- sum(1,2,3,4,5,6,7,8,9)

class(my_value)
```

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

```
length(my_value)
```

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

]

.pull-right[

```
my_vector <- c(1,2,3,4,5,6,7,8,9)

class(my_vector)
```

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

```
length(my_vector)
```

```
## [1] 9
```

]

There is technically no end to the number of R object classes that can exist in R. That's because classes can be built upon and modified to suit specialized purposes. Creating your own object classes is an advanced topic, for now, you will be interacting with pretty basic classes of objects.

Below are several different vectors that represent some of the most basic object classes. Notice how these classes align closely with data classes.

```
my_logical <- c(TRUE, TRUE, FALSE, TRUE, FALSE,TRUE,FALSE,FALSE, TRUE)

my_numeric <- c(1,2,3,4,5,6,7,8,9)

my_character <- c("a", "b", "c", "d", "e", "f", "g","h", "i")
```

A vector stores one or more values of the same type of data.

If we want to combine multiple vectors together, we can do that with the `data.frame()` function. This function takes multiple vectors as inputs and combines them together into a single object, as long as they're the same length!

.pull-left[

Why does this data.frame fail??

```r
my_df <- data.frame(
  "logical" = my_logical,
  "numeric" = my_numeric,
  "character" = my_character,
  "numbers" = c(2,4,6)
)
```

]

.pull-right[

```r
my_df <- data.frame(
  "logical" = my_logical,
  "numeric" = my_numeric,
  "character" = my_character
)
```

]

Notice the arguments we pass to `data.frame()`. Do you see how each piece fits together?

Can describe how each argument goes from input to output?

Click to show answer

> In this example, we pass `data.frame()` three arguments, each separated by a comma `,`. In each argument, we specify an R object to the right of an equal sign `=` and a character string within quotes `""` to the left of the equal sign.
>
> The characters to the left of the equal sign become the **names** of each column. The R objects, specified verbatim, become the contents of each column.

```r
knitr::kable(my_df)
```

logical	numeric	character
TRUE	1	a
TRUE	2	b
FALSE	3	c
TRUE	4	d
FALSE	5	e
TRUE	6	f
FALSE	7	g
FALSE	8	h
TRUE	9	i

You're bound to create **MANY** R objects throughout your work, and that can be a lot to keep track of. Fortunately, you don't need to track this all yourself - that's what the environment tab is for!

5.5.6 Environments (The 2nd Pane)

Finally, we come to the second tab of Rstudio. In addition to Environment, you may see tabs for `History, Connections`. You may also see tabs for `Build` or `Git` depending on if you have these tools installed.

While you *may* interact with these other tabs depending on your project needs, you will **always** want to reference R objects stored in your **local environment**. In the environment panel, you will find each R object listed by name and organized broadly into categories of `Values` (vectors), `Data` (data frames, matrices, and other more complex structures), User-created `Functions` will also receive their own section.

Your local environent is a temporary space. When you close your R session, you will lose any work you do not save or **write out** to a separate file.

Of course, that doesn't mean you can't re-run old code, and it doesn't mean that you can't save your progress in between sessions. In fact, there are **Several** options for how RStudio handles your environment. By default, R will attempt to preserve a copy of all data in your local environement. It will also try to restore whatever you were working on most recently when you open a new session.

These may seem like handy convenience features, but in practice we find them prone to cause issues (outlined below).

5.5.6.1 Our Reccomendations

Here we outline what we consider to be best practice for maintaining an R environment/ R project.

5.5.6.1.1 Global Options For the most part, we strive to use the default installations/parameters when using R/Rstudio. This facilitates reproducibility for a researcher working across different computers, and reproducibility if a colleague wishes to replicate the code themselves.

However, one setting we **DO** recommend changing the following under `Tools/Global Options/General` Uncheck all boxes that begin `Restore`..

- Restore last R project
- Restore previous source files
- Restore .RData

Also, we recommend changing the option for `Save a copy of .RData` to `NEVER`.

The reason for this is that if an R session crashes, trying to load the latest autogenerated backup is likely to cause the error and make the session fail again. It can also be timely if you last worked on a really large project that takes a while to open and restore all files.

In practice, it's better to intentionally save and close your project. Then open up whichever project you want to work on each time you start a new R session.

5.5.7 Files, Plots, Packages, oh my! (The 3rd Pane)

The bottom-right pane (by default) contains a lot of tabs, many of which you will likely use!

They functions of these tabs are largely self explanatory. `Files` provides a file browser, which defaults to your **working directory**. Usually, your working directory is the same as your R Project Folder.

`Plots` is a graphical device, this is where (most) data visualizations will be rendered by default. Past plots can be viewed with the arrow keys, and new ones will be placed last in the sequence.

NOTE: these plots are temporary renders made for your current R session. Be sure to export any visualizations if you will need to access them later.

`Packages` lists all the R packages you currently have installed on your machine. The check box *can* be used to load packages as desired. In fact, clicking the check box is the same as the `library()` function, which in practice, is usually the preffered way to load packages. Remember, you only need to `install.packages()` once, but you need to load R packages with `library()` each session.

The `Help` tab provides direct access to the help files housed within R. Each function *should* have a helpfile which you can search for by name. Alternatively, you can press `F1` while your cursor is located at the end of a function name, but before the opening parentheses.

The `Viewer` and `Presentation` panes will be used if you are using certain R packages that produce special outputs. See various package-specific help fules for those.

5.5.8 Writing R in Scripts (Breaking the 4th Pane)

Save your work, find typos, re-run easy!

Up until now, you've likely have been passing code directly to Console. While it makes it easy to hit the ground running, this aproach is not very efficient for actually conducting a project. Entering code line-by-line is error prone and does not provide an easy reference of the work you have done.

Instead of entering code directly to console, we usually want to save our work in the form of an R script.

An R script is simply a text document that contains R code. Importantly, it is **assumed** that everything within the Rscript is R code **to be run**.

Reveiw of R-specific Quirks

Here is a summary of some of the information outlined previously.

5.5.9 Global options

refer to the global options outlined above

R objects, Functions

R is an object-oriented programming language. Everything in R is either an object or a function. R objects generally store or define some kind of data. R functions, generally, take R objects as input, do some kind of operation, and produce some kind of output.

In the context of **language**, objects are the nouns and functions are the verbs.

When you write R code, you usually create R objects that represent data of interest, then apply various R functions to transform, analyze, or visualize the data. In the below code, we are creating an **object** called `my_data`, which represents the integers one through 10. We are using the `c()` **function** which takes any number of input **objects** as **input** and **concatenates** them together into a character string.

The **input(s)** to a function go within the parentheses separated by commas. You can usually tell the difference between R objects and R functions because R functions always have `()` written after their name.

In most cases, the **output** of an R function will be printed to the console, unless it is saved into a new R object. We do this using the **assignment opperator**, `<-`. This is a special function R function that takes whatever is on the right-hand side of the `<-` and creates or overwrites an R object with the name specified to the left-hand side of the `<-`. In the example below, we use the **assignment opperator** to "catch" the result of the `c()`, which is the character string of integers from 1 to 10.

Calling the name of an R object will *usually* print its contents in the console.

```
my_data <- c(1,2,3,4,5,6,7,8,9,10)



my_data
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

We say usually because objects have different properties depending on their **class**.

> NOTE: We sometimes use the term **class** to refer to the class of data of something. Is a given number sequence comprised of integers, factors, or real numbers? In R, and in this text, when we use **class** we are explicitly referring to the R-object-class of an object. Because R is a statistical programming language, the data-class of an object and the object-class of an object are sometimes synonymous (EG factors, logical, character), other times, the names may be slightly different (EG the R-class numeric comprises both integer numbers and complex numbers).

5.5.10 R classes

R classes determine the characteristics of the R object. They may set parameters about what kind of data can/can't be included in the R object. They may also define or modify preset ways these R objects interact with other R objects. You can use the `class()` function on any R object to find out it's class(es).

That's right, an R object can have more than one class.

Some basic R classes are **vectors**, **data,frames**, and **functions**. That's right! a function is actually just a certain **class** of R object. You can see below that in our example, `my_data` hass the class **vector**.

```r
class(my_data)
```

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

5.5.11 R Projects, R packages

R projects are an essential way to organize your data analysis, research, or even scratch work in R. Projects let you create self-contained environments that can feature their own R libraries, containg packages that are different from your day-to-day operations.

R packages are collections of R functions and R objects (datasets) that extend the functionality of R. In general, an R package is also an R project. However, an R project, doesn't have to be an R package.

5.5.12 R code vs R scripts vs R markdown

R **code** can go anywhere. It gets input into the console to actually run the commands. But if you want to keep a record of your code and document it, you probably want to write an **R script**. A script is simply a text file, that contains code. Presumably, the code takes some inputs, does some operations, and produces helpful output.

Sometimes we use scripts as a start-to-finish record of what we did in our work. This is often the case for research projects. A script might read data from a source, analyze it, visualize it, and save out results.

If we ever need to re-do this work, it's as simple as "re-runing" the script (literally a click of a button).

Scripts can also be helpful to automate workflows. If the same data transofrmations need to occur across many samples, a script can be written to **iterate** across a set of inputs, perform the same calculations/transformations, and output results in some way.

R markdown is just another script, but has a few more bells and whistles that make writing and documenting code a much more enjoyable experience. In fact, this whole book was written using Rmarkdown!!

How to find help

5.5.12.1 In R

All R functions *should* come with their hown help file. There are a few different ways to access this page.

In RStudio, you can click on the Help tab of the File/Plots/Packages...(**HELP**, Viewer, Presentation) panel - the Lower-Right panel by default.

Here the serach box will search the help files for any R package you have installed.

You could also access the help files by pressing the `F1` key while your cursor is at the end of an r function.

todo: Screenshot of cursor placement after function name, before open parentheses

5.5.12.2 the web!

Google (or duck-duck-go or, i guess, bing) it!

"How to calculate area under the curve in R"

You may be directed directly to an R package tailor made for that purpose. In this case, the AUC package was the first result from the official CRAN page.

You can search CRAN if you know you want a package to do a particular thing.

Alternatively, doing an internet search for a specific error message will often yield results on stackexchange or stackoverflow. These are two public sites where users submit responses and upvote helpful responses while downvoting unhelpful ones.

NOTE: posting directly to these sites CAN be helpful, but oftentimes someone has already asked the question and doing an internet search will get you the answer more quickly.

In general, people don't put out bad or malicious information about R, so you can trust any reasonable responses from the above websites. In addition, there are a few notable people or teams who consistently put out good (read amazing) R-related tips/help.

RStudio (Posit) Yihui

Tidyverse Hadley Wickam Jenny Bryan

Example Workflow

5.5.13 Create a new R project

5.5.14 Load a dataset

5.5.15 Visualize/Analyze data

5.5.16 Save out results

Excercises

5.5.17 Interpreting R

Describe what the following R code does, in plain words.

```
my_data <- read_excel("//filepath/directory/filename.xlsx")
```

Hint: There are 4 elements in the above code: `my_data`, `<-`, `read_excel()`, `//filepath/directory/filename.xlsx`

5.5.18 Try setting up a project

- Read data in from outside the R project
- Save data within your R project
- Save data external to your R project
- Visualize a dataset or trend.

Appendix 3a: Additional lessons / content to transfer over: What *is* data?

POV:

> In a social science class, your teacher tells you that the CDC reports average male height in the United States to be 69 inches or 5ft 9in. While browsing dating apps, you notice that nearly all the men report that they are 6ft or over. You wonder if this is a bias in reporting, or if the area where you live and attend college has significantly taller men. To test your theory, you want to collect data on height from individuals in your data science class to test if males are truly taller on campus than the country average.
>
> **Source:** https://www.cdc.gov/nchs/fastats/body-measurements.htm**

5.5.19 ACTIVITY - collect data on height

If in-person * Create a histogram (x axis) on a blackboard/wall, have students place their heights with a post-it note + Start with one distribution for whole class + Repeat with separate distributions for M/F + *DEE: I want to find a way to acknowledge this dichotomy ignores non-binary individuals, and include some suggestions on how to discuss it.*

5.5.20 ACTIVITY - collect data on birth month

- Create a histogram (x axis) on a blackboard/wall, have students place their heights with a post-it note
  - Students place post-it notes on month of birth

IDEALLY: Have both a histogram of height AND histogram of birth month visible at the same time. Compare/contrast distributions of the two data sets.

Explore

Questions to consider: * *Why do we plot data* * *What does the* **distribution** *of post-it notes look like?* * *What can you infer from the distribution(s)* * *How does the distribution of height differ from birth month?*

5.5.21 Example Datasets

If you're working through this course on your own, or are unable to facilitate a classroom activity, see the companion R package, `ipumsED`, which includes example data sets for each lesson, as well as custom code and functions to facilitate learning data science with IPUMS data.

DEE: This could probably be stated at the begining of unit 1

5.5.22 ACTIVITY - Calculate by hand

In our hypothetical setup, we are interested in the **average** height. * *What does it mean to be* ***average*** * *Which height would you say is average? (eyeballing)* * *Which Birth Month would you say is average? - is there one?*

Write out steps for calculating **mean** - trivial as it may seem.

Explain

In the context of this example, we **collected** data on height and birth month for individuals in our class. Plotting our data allows us to **visualize** the data, making it easy to interpret.

Ellaborate

5.5.23 So what is data?

Data is defined as "facts and statistics collected together for reference or analysis."[1] As seen in Figure 1.1, there are two types of data: quantitative and qualitative. **Quantitative data** are able to be expressed in numerical format

[1]This is from the internet and needs to be our words

and are countable. These data are either discrete or continuous where **discrete data** uses numeric bins. For example, we use our age as discrete quantitative data, we round our age to the previous year (eg., 20, 21, 22). **Continuous data** does not use bins, but rather includes all of the fractions between two whole numbers. An example could be most physical measures like height, weight, the speed at which an individual runs.

Qualitative data describes characteristics or categories and can be broken down into two categories, nominal or ordinal. **Nominal data** has no inherent ordering but it can be categorized. Examples include country or origin, gender, hair color, race, etc. **Ordinal data** can both be categorized and ordered (e.g., first, second, and third place is a race).

> Going back to our hypothesis of male height on campus, heights are continuous, qualitative data. It is difficult for people to report their specific height and you assume that most individuals will report it rounded to the closest inch. This makes the data you will actually use, discrete quantitative data.

5.5.24 Collecting Data

The first step to answering a research question is to collect your data. Broadly, data comes in two forms, primary and secondary. (Fig 1.2) **Primary data** is data that is collected directly by the researcher. Surveys, observations, experimentation, questionnaires, and interviews are all examples of primary data. **Secondary data** is collected from published or unpublished literature. It is collected by different researchers and compiled for use by a second scientist. This type of data includes data found in published articles, books, journals, biographies, and government records like the US Census.

Once compiled, you now have a data set which is comprised of observations and variables. An **observation** is all of the measures taken for one person or item. A **variable** is what is being measured.

> The US CDC data is secondary, but you are collecting height data yourself in class as a comparison. The survey or questionnaire you use on your classmates is primary data. Each individual is an observation and the variable of interest is height.

5.5.25 POPULATIONS AND SAMPLING

Random Sampling: It is a sampling method in which all the items have an equal chance of being selected and the individuals who are selected are just like the ones who are not selected

Stratified Random Sampling: It is a process to gather data by separating the actual population into the distinct subset or strata, and then choosing simple random samples from each stratum Your research question is about the height of all males at your college, but recording height data for each individual would be very difficult and time consuming. You instead decide to use a sample of males in your data science class. This is a random sample as each male individual has an equally likely chance of being samples (that is, unless a prerequisite exists).

Sampling strategy can lead to **bias**

> If you had chosen a different sample, like the men's basketball team, your results would have been biased.

Evaluation

5.5.26 Review Questions

- What is one example of collecting/visualizing data from your own life
- Is height a **continuous** or **discrete** variable and Why?

5.5.27 Exercises

Brainstorm 3 topics/questions that are of interest to you personally, or academically. * What **variable(s)** will you need to collect to study this phenomenon? * Describe these variable(s), are they qualitative or quantitative? Continuous or ordinal?

Glossary

Appendix 3b: Additional lessons / content to transfer over:

Intro to R, data types, data structures

In the previous lesson, we began thinking about **data**, how to talk about it, and how to **visualize** it. We also talked about one type of average, the **mean.**

Engage

What do you think the following code does?

```
my_data <- read_excel("//filepath/directory/filename.xlsx")
```

Hint: There are 4 "things" in the above code: `my_data`, `<-`, `read_excel()`, `//filepath/directory/filename.xlsx`

5.5.28 R vs RStudio

- If `R` is the engine, then `RStudio` is the car.
- If `R` is the text, Rstudio is the text-editor.
- `R` is the **programming language**, `Rstudio` is the **Integrated Development Environment (IDE)**.
  - *What do you think some differences are between R and RStudio?*

`RStudio` is a program/app just like Google Chrome or Microsoft Word. Each of these programs provide a **Graphical User Interface (GUI)**, a pretty way for a user to use a mouse and keyboard to do *something.*

An **IDE** is a special kind of program/app that provides MANY tools for writing and running code.

5.5.29 Orientation to RStudio

SUGGESTION: Instructor live demos interacting with RStudio while students follow along on computers

When you first open RStudio, you'll see 3 **panes**, all of which will look fairly empty at the moment.

If you're using the default layout, you should see: * On the left, the `Console` **pane** * On the Top-right, the `Environment` **pane** * On the Bottom-right, the `Files` **pane**

A keen eye will also notice that each of these **panes** contains multiple **tabs**. We will go over the uses of many of these **tabs**, but for now let's start with `Console`.

The `Console` is where you input `R code`, run it, and see the results.

At it's simplest `RStudio` is a calculator. Try typing `4 + 4` into the `Console`, then press the `[Enter]` key to run the code. Immediately, `R` prints the result as we see here:

```r
4 + 4
```

```
## [1] 8
```

The `Console` serves as a running log of all your operations for your current session, but it can be helpful to temporarily save your results as `R objects`. We do this using the **assignment operator** we saw earlier: `<-`

```r
answer <- 4+4
```

By default, `Console` only prints results if they have no where else to go. If they're stored as an `Robject`, no result is printed, but you should now see `answer` listed in the `Environment` **pane**.

Explore - Interacting with R objects

5.5.30 Load the Data

```
dir_path <- file.path("inst","unit1_data")
survey_path <- file.path(dir_path, "data_template.xlsx")

data <- readxl::read_excel(survey_path)
```

What is `data`?? Below we call the `class()` function on `data` and see that it has 3 classes: `tbl_df` , `tbl` , `data.frame`

The first two classes, `tbl_df, tbl` indicate it is a special kind of table, in the `tibble` format. In general, you can interact with these like a `matrix` or `data.frame` but they have additional features.

```
class(data)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

We can call `colnames()` on data, like a regular `data.frame` or `matrix`. Or we can take advantage of the `tibble` structure and use the `glimpse()` function which provides a succinct summary of your data.

```
colnames(data)
```

```
## [1] "individual"    "Birth_Month"   "Height_inches"
```

```
tibble::glimpse(data)
```

```
## Rows: 32
## Columns: 3
## $ individual    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1~
## $ Birth_Month   <chr> "January", "September", "March", "April", "April", "Octo~
## $ Height_inches <dbl> 70, 64, 72, 61, 55, 65, 72, 75, 69, 75, 76, 70, 70, 69, ~
```

5.5.31 Inspect the Data

What is `data`?? Below we call the `class()` function on `data` and see that it has 3 classes: `tbl_df` , `tbl` , `data.frame`

The first two classes, `tbl_df, tbl` indicate it is a special kind of table, in the `tibble` format. In general, you can interact with these like a `matrix` or `data.frame` but they have additional features.

```r
class(data)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

We can call `colnames()` on data, like a regular `data.frame` or `matrix`. Or we can take advantage of the `tibble` structure and use the `glimpse()` function which provides a succinct summary of your data.

```r
colnames(data)
```

```
## [1] "individual"    "Birth_Month"   "Height_inches"
```

```r
tibble::glimpse(data)
```

```
## Rows: 32
## Columns: 3
## $ individual    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1~
## $ Birth_Month   <chr> "January", "September", "March", "April", "April", "Octo~
## $ Height_inches <dbl> 70, 64, 72, 61, 55, 65, 72, 75, 69, 75, 76, 70, 70, 69, ~
```

Summarize Data

Continuous Data

For continuous data, we often want to summarize our data by describing the **mean, median, and/or range**. **Mean** and **median** describe the *central tendency* of the data, while **range** describes the full extant of the data, as seen below.

NOTE: if `NA` are present in the data, be sure to use the `na.rm=TRUE` flag for these operations.

```r
mean(data$Height_inches, na.rm = T)
```

```
## [1] 66.5625
```

```r
median(data$Height_inches, na.rm = T)
```

```
## [1] 66.5
```

```
range(data$Height_inches, na.rm = T)
```

```
## [1] 55 76
```

All in summary()

Mean, **median**, and **range** will all be reported by calling `summary()` on a `numeric vector`, such as `Height_inches`. In addition, the lower and upper quartiles will be reported, along with the number of `NA` responses.

NOTE: `summary()` does NOT require special handling for `NA` values, in fact - it expects them!

```
summary(data$Height_inches)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   55.00   64.00   66.50   66.56   70.00   76.00
```

Mode

You're probably familiar with **mean** and **median** being talked about with a third term, **mode**. The **mode** is the most commonly occuring value in a dataset. It's often important to know the **modal response** of survey data. While a commonly reported metric(??), there is no `mode()` function included in `base r`…

so we'll just have to create our own!

5.5.32 Mode Code

One common measure of data reported is the mode, or most frequently occuring value. For whatever reason, this is not a default function in R, but we can easily write our own function like so:

```
my_mode <- function(x){
  tt <- table(x) ## find frequencies
  tt <- tt[order(tt, decreasing = TRUE)] ## resort based on freq

  ## check number of modes
  max <- max(tt)
  n_max <- sum(tt==max)
  if(n_max > 1 ){
    warning("More than one mode detected")
    return(tt[tt==max])
  } else {
    ## return only the first value
    return(tt[1]) ## return whatever the highrst frequency is
  }


}
```

5.5.33 Mode Results

Now that we've created out own `function`, it's easy to find the **mode**

```
my_mode(data$Height_inches)
```

```
## Warning in my_mode(data$Height_inches): More than one mode detected
```

```
## x
## 64 65 69 70
##  4  4  4  4
```

Visualizing Data

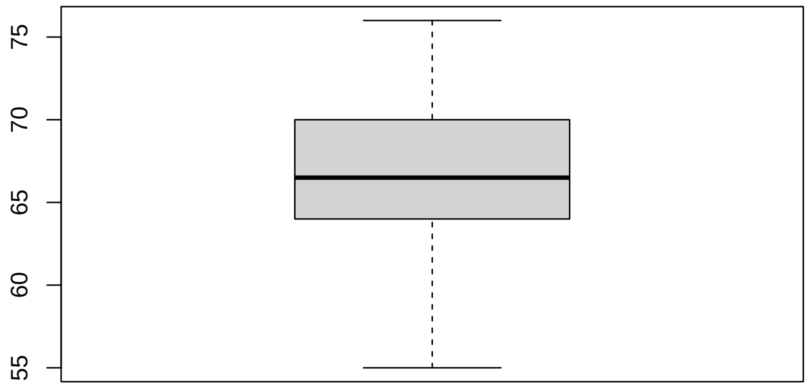
The above summaries describe data with numbers, but we can also describe data visually.

5.5.34 Continuous Data - Boxplots

Univariate continuous data, like height, can be visualized using a box and whisker plot, which shows many of the components of summary:

- the **median** is the black bar in the middle
- the **quartiles** (25th and 75th percentiles) are represented by the extents of the boxes
- The **range** is shown by the whiskers, with outliers shown indvidually, if needed.

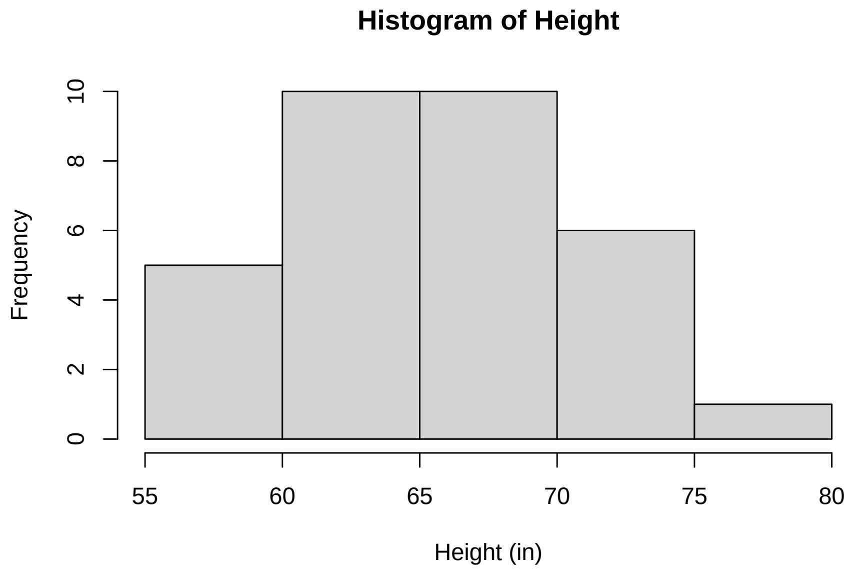
```r
boxplot(data$Height_inches)
```



5.5.35 Continuous Data - Histograms

Continuous data, can also be broken into **bins** and plotted as a **histogram**. The `hist()` function will attempt to find the optimum number of bins for you, but you can specify a different number with the `breaks` argument.

```r
hist(data$Height_inches, main = "Histogram of Height", xlab = "Height (in)")
```



5.5.36 Categorical Data

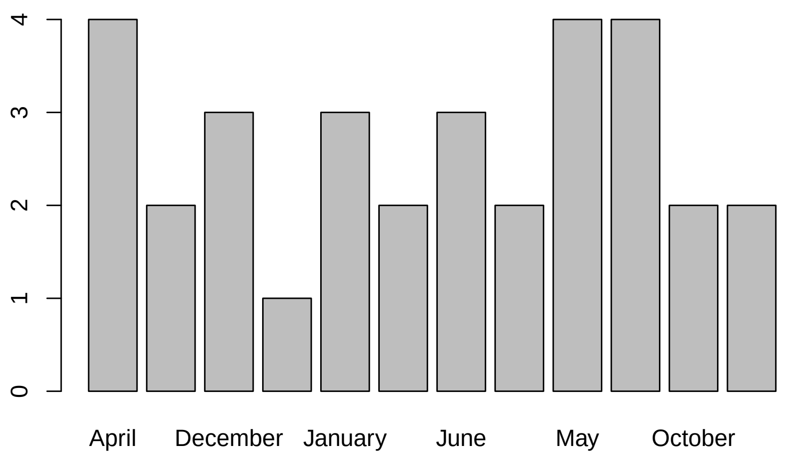
Categorical data is already in discrete units. In general with categorical data, we want to count the **frequency** of unique values. There are many ways to do this, but one of the easiest is the `table()` function. Saving the results of the table to an object, `birth_freq`, allows you to save and print the results at any time.

```
birth_freq <- table(data$Birth_Month)

birth_freq
```

```
##
##     April    August  December  February   January      July      June     March
##         4         2         3         1         3         2         3         2
##       May  November   October September
##         4         4         2         2
```

We can also visualize our tabulated results using a **barplot** as below.

```
barplot(birth_freq)
```



Glossary

Appendix 3b: Additional lessons / content to transfer over:

Comparing Data

NEEDS A LOT OF WORK

Data Distributions

5.5.37 Normal Distributions

First we'll generate a normal distribution with the `rnorm()` function. This takes 3 arguments: `n, mean, sd`, which you can see filled in below. While we could print out a list of all these values, it's not easy to *understand* a list of numbers

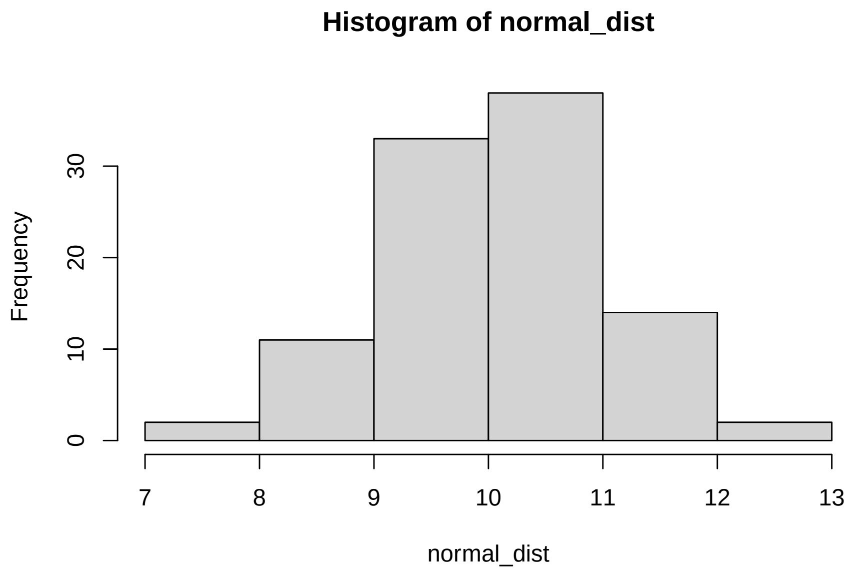
```
normal_dist <- rnorm(n = 100, ## 100 samples
                     mean = 10, ## with a mean of 10
                     sd = 1 ## and a standard deviation of 1
                     )


normal_dist
```

```
##   [1] 11.537843  9.577027 11.444075 10.518058  9.752622  9.381794  9.283329
##   [8] 11.997276 10.383081 10.716883  9.871463 10.575796  9.803151  8.695750
##  [15]  9.346073  9.349846 10.108626 11.711685 10.674153 11.827317 11.779177
##  [22] 10.896859 12.545381 11.943484 10.964451  9.924096  9.439721 10.084466
##  [29] 10.287442  9.352893 10.636215  8.703583 10.935781 10.969983  8.645146
##  [36] 11.430844 11.295499 10.003588  8.291503  9.189448 11.316452 11.729021
##  [43] 11.006216 10.525947 10.863133  9.884728  8.801202 10.653382  9.648026
##  [50]  9.906731  7.851274  9.809432  9.968136  8.717939  9.882488 11.077242
##  [57] 10.287510  9.558057  9.327232 10.197740  7.510276 10.527043  9.925029
##  [64]  9.986464 10.820894  9.069265 10.166528  8.990858 11.031247 10.943931
##  [71]  8.493954  8.986688 10.595989  8.882771 10.572699 10.811687 10.045714
##  [78]  8.082443 10.578480 10.122678 10.514887  9.696846  9.871598  9.787594
##  [85] 10.141834  9.217790 10.534329 10.177832 10.412341 10.193418 10.332295
##  [92]  9.273755  9.637714  9.729498  9.926673 10.746631 10.008932  9.251834
##  [99] 12.293185  9.264603
```

Another better way to look at data would be to **visualize** or **plot** it. One way to to that is with a **histogram**, which groups **continuous values** into **bins**, then plots the **frequency** for each bin.

In R, we use the `hist()` function to plot a histogram of data. We can (try to) control the number of bins with the `breaks` argument, but note that it doesn't always match up. The `hist()` function will adjust based on the distribution of the data.

```
hist(normal_dist,breaks = 5)
```



Another way to visualize this would be with a d

5.5.38 What *is* normal?

5.5.38.1 Quantitative summaries

5num summary * Min, 25th percentile, median, 75th percentile, Max

```
tab_normal_dist <- summary(normal_dist)
```

We can print the table in R by calling its name.

```
tab_normal_dist
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   7.510   9.425  10.097  10.120  10.724  12.545
```

Mean, standard deviation

5.5.38.2 Meaningful Comparisons

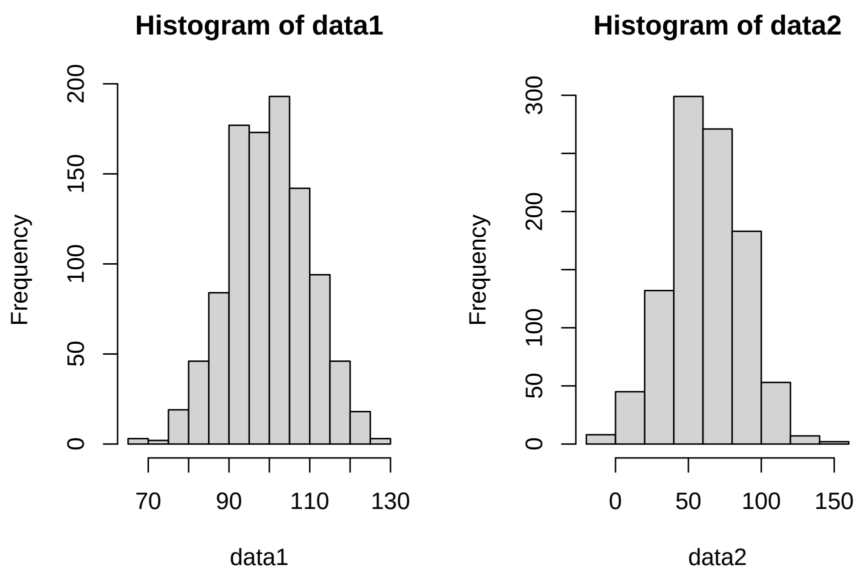
How to compare apples to oranges? Standardize the units / standardize the data

```
data1 <- rnorm(n=1000,
              mean = 100,
              sd = 10)

data2 <- rnorm(n=1000,
               mean = 60,
               sd = 25)
```

Are these the same distribution?

Any issues??

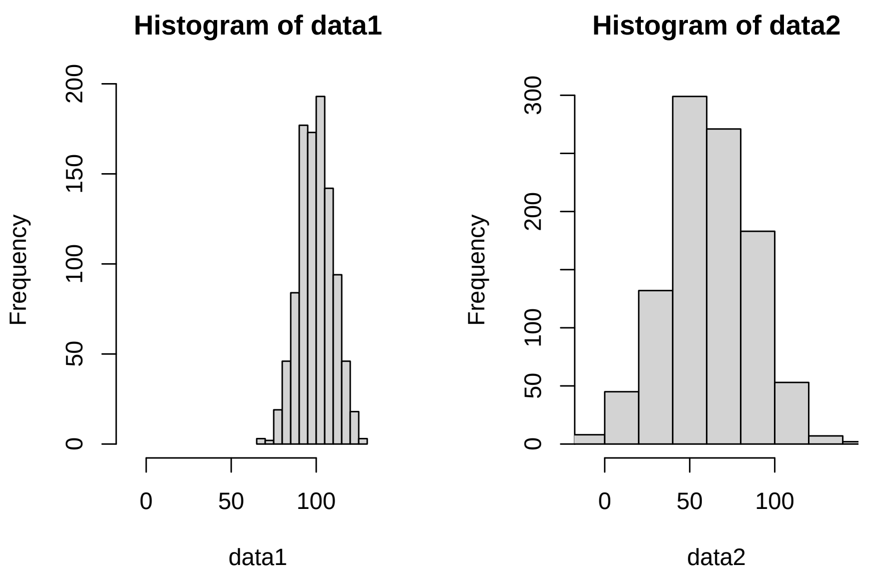
```
layout(matrix(1:2, ncol = 2))
hist(data1)
hist(data2)
```



```
total_range <- range(data1, data2)
```

Are they the same?

```
layout(matrix(1:2, ncol = 2))
hist(data1, xlim = total_range)
hist(data2, xlim = total_range)
```



Numerically / tabularly

Often times its important to tables of **summary statistics**

```
norm_comp_tab <- rbind(summary(data1),
                       summary(data2))

norm_comp_tab
```

```
##           Min.  1st Qu.   Median     Mean   3rd Qu.     Max.
## [1,]  67.50282 92.94738 99.85491 99.83723 106.34225 127.6897
## [2,] -11.47190 44.40308 60.78438 61.91661  79.79141 142.9623
```

Making the table a little nicer. Also an example of **conditional programming**.

```
rownames(norm_comp_tab) ## they're null
```

```
## NULL
```

```
if(is.null(rownames(norm_comp_tab))){
  rownames(norm_comp_tab) <- c("data1", "data2")
}
```

When working with **Rmarkdown** we can take advantage of `knitr` and `pandoc` to nice looking tables even easier.

```
knitr::kable(norm_comp_tab)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
data1	67.50282	92.94738	99.85491	99.83723	106.34225	127.6897
data2	-11.47190	44.40308	60.78438	61.91661	79.79141	142.9623

How transform the data

Simple transformation (multiply all values by 100) * to convert units * other examples?

Complex transformations * log-transformation (*DEE: not a fan*) * z-scores (*DEE: a better option*)

Why transform the data? * Real world applications? * Is it always appropriate to transform data?

5.5.39 Skews

What to do if the data are **not** normal?

Statisitcal testing of simple data sets

5.5.40 t-tests, ANOVA, chi2

Relationships between variables in simple data sets

5.5.41 Correlation, Linear Regression

5.5.41.1 Simple LM

5.5.41.2 Complex LM

5.5.42 Genearlized Linear Model

For now, I have 3 main chapters for each of the main sections: * Basics of data science / R * Applications/critiques using IPUMS data * Student-driven projects

Each of these **Chapters** contains multiple sections. We'll likely want to break these sections out into their own `.Rmd` files as they get fleshed out. For now, I'll try to keep the abundance of files limited.

NOTE: As these actually get filled out, we will probably want to insert different `parts` to the book (EG, the content of Unit 1 is covered in `Part I`). * Declare parts with `# (PART) Part I {-}` immediately before the first chapter `#` it contains.

Topics to include: * What is data? * Everything can be data * How do we interpret data * Tables * Plots * Univariate distributions * What can they tell us * Multi-modality in distributions * Categorical vs continuous data * Don't need to get ahead of this yet * Add in a grouping category - multi state/multi-national dataset * Ttest / anova

Type of Data: Age distributions Specifically generate a dataset with old/young folks over-represented to highlight a bimodal distribution

Start with single state/country Add a second state/country to demo ttest Add more to demo anova

Alternatively, income by education level - may be more interesting/relevant to college students (or depressing)

Intro to R/RStudio

Reading Data / Distributions

5.5.43 What *is* a normal distribution

5.5.43.1 How normal is it?

show increasingly unclear examples of normal vs not

introduce tests of normality

5.5.43.2 Measuring normality - single sample

reinforce [concept of statistical] **normality**

is a value from a sample? - one way ttest something about tails

5.5.43.3 comparing normality - two saples

standard / two-way t test

5.5.43.4 comparing more than two - ANOVA

Glossary

Data Quantitative Qualitative Discrete Continuous Nominal Ordinal

dev-Appendix : Styling with RMD

For now, this chapter is a bit of a placeholder. I'm not sure what/how the `references.Rmd` file actually fits in to the code/construction (it looks automatic) so I want to keep that in place and need a section to note that.

I also want a more centralized reference point to put any example code I find helpful while working in R/bookdown. This section could get really unruly really fast, but oh well.

Navigating the bookdown

All chapter files need a 2 digit prefix (except `index.Rmd`) which dictates the chapter order. Note, this is not automatic and a little tedious to re-do.

- `index.Rmd` is a required file and treated as file `00`.
- Chapters *should* be numbered for ease of sorting but custom orders are possible by specifying filenames somewhere `in a certain file...but i'm not sure where`.

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`. * Unlike usual Rmd files, chapters DO not contain a YAML header * Note that `index.Rmd` has its own YMAL for the title page and some bookwide parameters.

Include Images

5.5.44 Figures in a folder

Embed figures from a folder.

For this, it's usually best to use a code-chunk and `knitr`. There are a number of graphical paramerters you can set (or ignore) `out.width` will scale your image accordingly - irrespective of unit/display `fig.align` should be "left", "right", or "center" `fig.cap` allows you to provide "mouse over" captions for the image. `echo=FALSE` is important if you ONLY want the image (IE the result of the code). If you want the code itself to show, (IE, or echo) set `echo=TRUE`.

Figure 5.1: the ipums logo

5.5.45 Fancy renders

Embed html renders (EG, fancy tables (IPUMS_var_desc), or any shiny app) with `webshot` R package and `phantomJS`.

```
install.packages("webshot")
webshot::install_phantomjs()
```

Tips

- Each section (chapters) gets **AUTOMATIC NUMBERING**
  - Use `{-}` or `{.unnumbered}` to keep a section out of the auto-list. These will still be listed in the TOC
  - To hide the heading from the TOC you can also use {.unlisted}

Basic Fonts

In a list:

- *italics*
- **bold**
- `code`
- *equations*

In *a sentence* **with differently** `formated fonts` and maths $ a + b = c? $.

5.5.46 More on Math

Randal Pruim features an extensive list of common math expression on their github page. Here are some quick notes:

In-line equations can be written within `$` and will be displayed right there: $a^2 + b^2 = c^2$. In contrast, you can also add equation-chunks by using `$$`

This can be coded in-line,

$$\sum_{n=1}^{10} n^2$$

, but will result in a page break.

p is unknown but expected to be around 1/3. Standard error will be approximated

$$SE = \sqrt(\frac{p(1-p)}{n}) \approx \sqrt{\frac{1/3(1-1/3)}{300}} = 0.027$$

Linking and Crossrefs

There are a few different options here:

- Footnotes
- Hyperlinks
- autocite?

5.5.47 Footntes

You can also use math in footnotes like this[2]. Footnotes are helpful because they either open to a new page, where you can re-link to where you left off. Or they display as a literal footnote, depending on your file format.

We will approximate standard error to 0.027[3]

Anything that maintains the indent following the html tag (square brackets) will be included in the footnote and **excluded** from the actual rendered text. This could be helpful to easily provide more information without breaking the

[2] where we mention $p = \frac{a}{b}$

[3] p is unknown but expected to be around 1/3. Standard error will be approximated

$$SE = \sqrt(\frac{p(1-p)}{n}) \approx \sqrt{\frac{1/3(1-1/3)}{300}} = 0.027$$

narrative; however, it has the drawback that footnotes are dispersed throughout the document.

NOTE: Footnote labels can be any character string, but for our own sakes, it makes sense to follow a standard, descriptive, format.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

5.5.48 Hyperlinks

In short, if you **label** it, you can link to it. So it's good practice to **LABEL EVERYTHING DESCRIPTIVELY** in case you it later. There are a couple of differnt styles of labeling depending on what you want to link to. You already saw above how to link to a website (URL).

- code chunks that produce figures can be referenced via `@\ref(fig:[LABEL])`

5.5.48.1 Link to chapters

You can label chapter, section, or any level of header (`##`) using by including the label in curly brackets after the section. * e.g., we can reference @ref(unit1_ch1) using the following: + `\@ref(unit1_ch1)`

NOTE: If you do not manually label sections, they will get automatic labels anyway, which are hard to reference, so best to just declare them.

5.5.48.2 Link to figures, tables, or code chunks

You can link to figures or tables created by R code, or the R code itself, as long as they are labeled. **Be careful!!** if labels are not descriptive and informative, it might be hard to find them later. * We should decide on some conventions for this!*

Whatever the output (figure, table, etc), the label follows the r at the very first line of the code chunk: * `{r an_example}`` *{r an_example, fig.cap = "Example that include additional code chunk parameters"}'

To include the link in text, use the `\@ref()`.

To link to the code itself, we simply call the label (just like linking to a section header). * `\@ref(basic_code)` Link to some basic code @ref(basic_code)

If the output is a figure, you will also need to preface the label with `fig:` * `\@ref(fig:height_boxplot)`. Is the code that produces this link to a figure @ref(fig:height_boxplot)

In order to reference a nicely formatted table, made with `knitr::kable()`, use the following: * `\@ref(tab:height_table)`. In order to reference *this table* @ref(tab:height_table)

5.5.48.3 Link to websites

linking to external websites is easy, just include the text-to-be-hyperlinked

This is actuall the exact same as linking to a local .jpg or similar.

5.5.49 Citations

There do appera to be tools for citations, but I'm not super familiar with them so this section will grow as I learn more.

5.5.49.1 Citing R packages

You can easily gather a list of all R packages used in a project using `knitr::write_bib()`. By using square brackets and the `@` sign, along with the package name? or maybe the github repo? I'm not entirely sure, see below

Quick example from demo/index (may not work without write_bib() though): we are using the **bookdown** package (Xie, 2023) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

- This infor should be written to the References files, though i'm not sure if it will work

Chapter 6

Glossary

6.1 Unit 1

6.1.1 Chapter 1

Random Sampling: Stratified Random Sampling: Statistical bias

6.2 Aoendix 1

6.2.1 Intro to R

Advanced Methods

Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2023). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.35.