Introduction to R and RStudio

Stephen Salerno

March 4, 2022

Installing R and RStudio

Mac Users

To Install R:

1. Open an internet browser and go to www.r-project.org.
2. Click the "download R" link in the middle of the page under "Getting Started."
3. Select a CRAN location (a mirror site) and click the corresponding link.
4. Click on the "Download R for (Mac) OS X" link at the top of the page.
5. Click on the file containing the latest version of R under "Files."
6. Save the .pkg file, double-click it to open, and follow the installation instructions.
7. Now that R is installed, you need to download and install RStudio.

To Install RStudio:

1. Go to www.rstudio.com and click on the "Download RStudio" button.
2. Click on "Download RStudio Desktop."
3. Click on the version recommended for your system, or the latest Mac version, save the .dmg file on your

Windows Users

To Install R:

1. Open an internet browser and go to www.r-project.org.
2. Click the "download R" link in the middle of the page under "Getting Started."
3. Select a CRAN location (a mirror site) and click the corresponding link.
4. Click on the "Download R for Windows" link at the top of the page.
5. Click on the "install R for the first time" link at the top of the page.
6. Click "Download R for Windows", save and run the executable (.exe) file and follow the installation instructions.
7. Now that R is installed, you need to download and install RStudio.

To Install RStudio:

1. Go to www.rstudio.com and click on the "Download RStudio" button.
2. Click on "Download RStudio Desktop."
3. Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions.

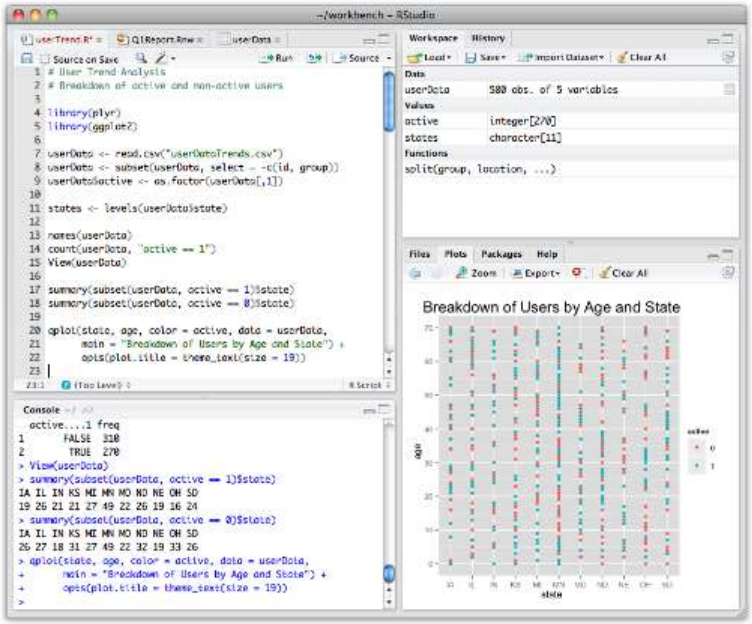
No R? No Problem!

https://rstudio.cloud/

The R Environment

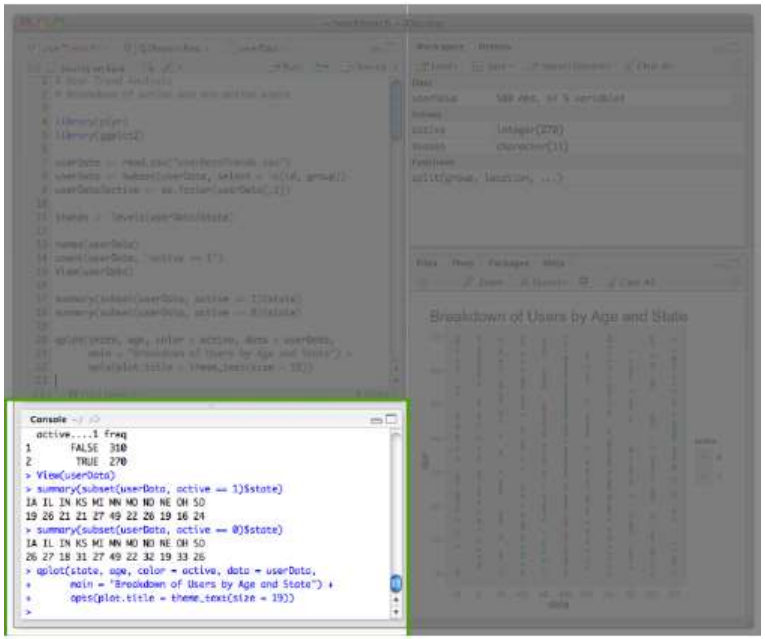
RStudio

When you first open RStudio, you will see that it is segmented into different windows.



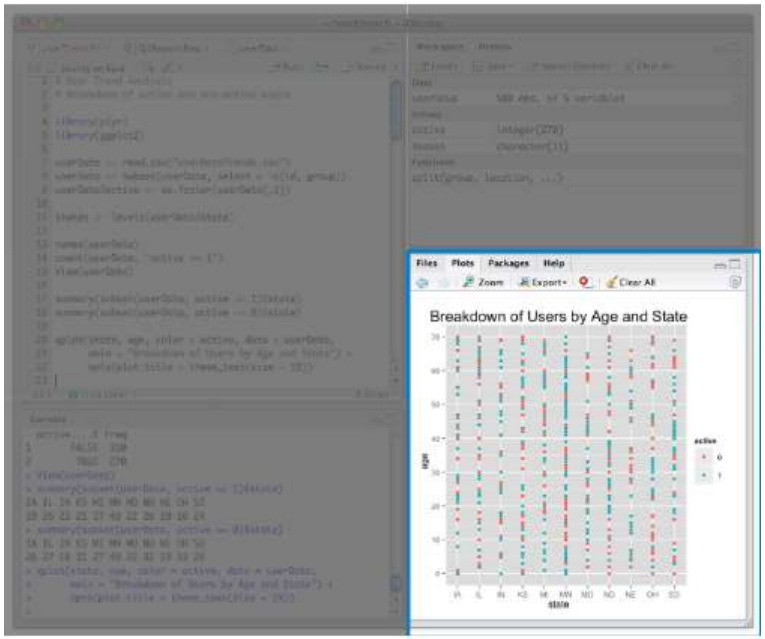
Console

- This is where Rstudio interfaces with R, and where all of the inputs and outputs happen.



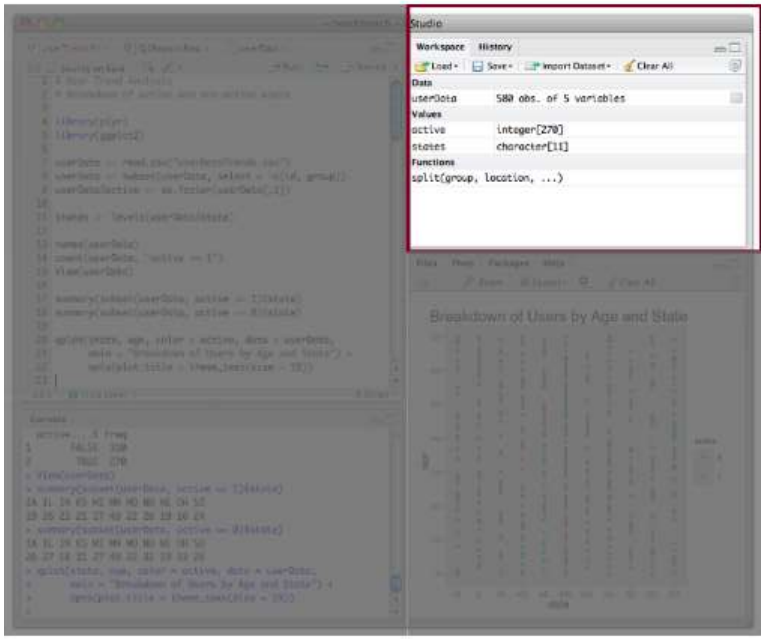
Plots and Help

- This panel displays plots as well as documentation files automatically when called.



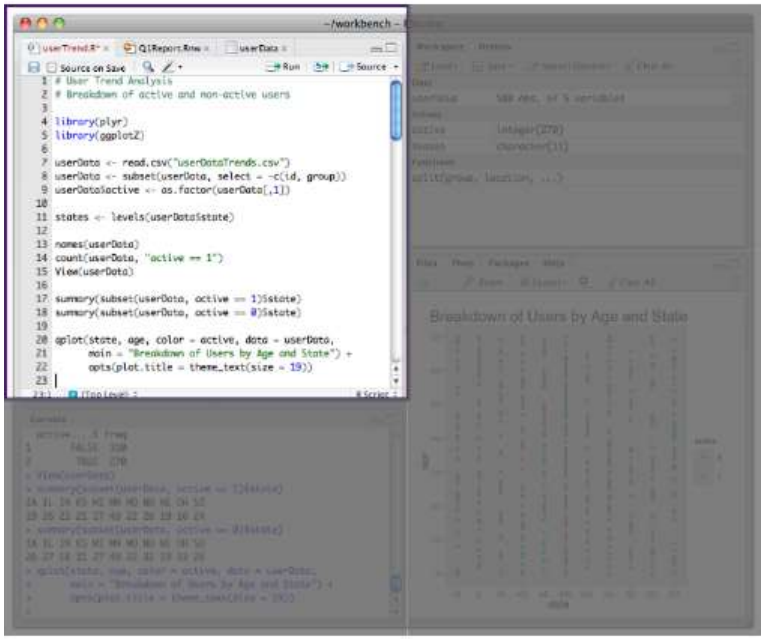
Environment

- This will show all of the data and variables you are working with throughout your current R session.



File

- This shows the `.R` files where your code is stored (critical for **reproducible research**).



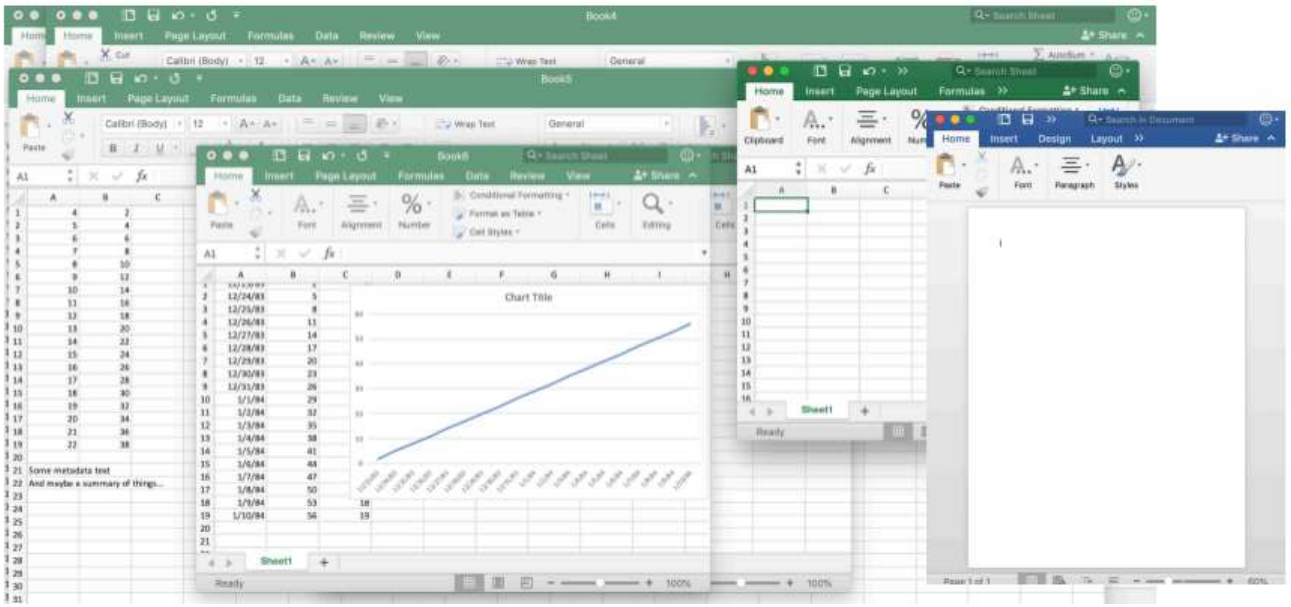
Some Notes on Reproducibility

From Victoria Stodden (as summarized in the ROpenSci Reproducibility Guide):

- **Computational Reproducibility**: "When detailed information is provided about code, software, hardware and implementation details."
- **Empirical Reproducibility**: "When detailed information is provided about non-computational empirical scientific experiments and observations. In practise this is enabled by making data freely available, as well as details of how the data was collected."
- **Statistical Reproducibility**: "When detailed information is provided about the choice of statistical tests, model parameters, threshold values, etc. This mostly relates to pre-registration of study design to prevent p-value hacking and other manipulations."

Credit: Allison Horst, 2020

What about this workflow might be hard to reproduce?



Credit: Allison Horst, 2020

Some Thoughts...

- No clear or organized history of what's been done to the data from raw data through final figures/results
- Lack of comments/annotation describing the steps
- Tedious & time consuming for a collaborator to recreate the analyses
- Backed up w/version control? Probably not...
- How do we transfer this into a final report or presentation? Is that reproducible?

Credit: Allison Horst, 2020

Do your *data sci* like it's going to need an *alibi!*

A more reproducible workflow

Credit: Allison Horst, 2020

Starting a Project

Projects in `R` create a self-contained directory to store all of your code, data, output, figures, and more! When a new project is created RStudio:

`File` → `New Project` → `New Directory` → `New Project`

- Creates a project file (`.Rproj`) in the project directory with project options and a shortcut from the filesystem
- Creates a hidden directory (`.Rproj.user`) where project-specific temporary files (e.g. auto-saved source documents, window-state, etc.) are stored
- Loads the project into RStudio and display its name in the Projects toolbar

Writing a Script

- To save your commands, we can write them into a document known as an `R` script.
- To start a new script: click `File` → `New File` → `R Script`.
- Code can be run in R by copy-pasting directly from a script to the **Console** and pressing `Enter`.
- Better yet, hitting `Ctrl+Enter` will send the current line to the **Console** directly and run it.
- You can try running some of the example commands contained in `R_Introduction.R`.

Adding Comments to Your Code

- R will ignore all code that follows a `#` character - these are referred to as **comments**.
- Comments are a crucial part of the code writing process, so others can understand the thought process behind your code.
- Also helpful when revisiting code months later to document methodology.

Example:

```
print("This will print!") # This comment will not
```

```
[1] "This will print!"
```

Working in R

R as a Language

- Unlike SPSS or other *point-and-click* style statistical software, R is a *scripting language*.
- This means that tasks are performed by typing basic code to instruct R what to do.
- As with any language, there is very specific syntax and grammar that must be followed for your instructions to be understood.

The R Command Line

- Code is typed line-by-line, like sentences, as a series of commands.
- The command line prompt, i.e. the `>` in your `console` window, is where you type your commands.
- At the most basic level, `R` can perform operations similar to that of your calculator using arithmetic operators: `+`, `-`, `*`, `/`, `^` (for powers), `log()`, `sqrt()`, or combinations of these.
- For example, type `2 + 2` and press the `Enter` key. Here is what appears on the screen:

```
2 + 2  # Example Arithmetic Calculation
```

```
[1] 4
```

- `R` interprets everything in terms of dimensions. It recognizes input and output as single numbers (scalars), columns of numbers (vectors), or matrices. The `[1]` before the `4` says that `2 + 2` has one element (i.e., `4`).

Accessing Help

- Most functions and concepts in `R`, including those coming from third-party packages, have a single interface for accessing help.
- While Google will usually to lead you to an acceptable answer, learning to use the documentation built-in to R can lead you to the best answer faster.
- To learn about any function, type `?<name_of_object>`, replacing `<name_of_object>` with the name of what you want to learn about.

```
?sd  # Example: Getting Help with sd() function
```

- In this example, one can see that R's `sd` function calculates sample standard deviation (dividing by `n-1`) rather than population standard deviation (dividing by `n`).

Analyzing Data

Introduction to Variables

- R can save information into a *variable* name
- For example, assume we measured the following test scores for 10 students:

Student:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Score:	50	20	90	40	60	70	70	0	10	NA

- We can save these variables into R as follows:

```
score = c(50, 20, 90, 40, 60, 70, 70, 0, 10, NA)   # Save the scores into a variable
score <- c(50, 20, 90, 40, 60, 70, 70, 0, 10, NA)  # "<-" does the exact same thing
```

Introduction to Variables

- The scores are now saved for future access in the `score` variable:

```
print(score)  # View our variable `score`
```

```
[1] 50 20 90 40 60 70 70  0 10 NA
```

- We created a `vector` of numbers in `R` using the `c()` (short for "combine") function.
- Here, you can also see that `NA` is used to represent a missing value
- **Recall:** You can learn more about `c()` and `NA` by typing `?c` or `?NA` into the console.

Exploring One Variable : More Help Tricks

- To find a specific score, we use the name `score` and the `extract` (`[]`) operator.
- This can be thought of as **slicing** a vector at a particular index.
- To find multiple values, we can use the `sequence` (`:`) operator as well. For example:

```
score[3]          # Find the score for the third student
```

```
[1] 90
```

```
1:5               # Make a sequence of numbers from 1 to 5
```

```
[1] 1 2 3 4 5
```

```
score[1:5]        # Find the scores for the first five students
```

```
[1] 50 20 90 40 60
```

Exploring One Variable: More Helpful Tricks

- Back-quotes are necessary to tell R we want to know about operators, like +, −, [, or :

```
?`[`  # To learn about slicing/indexing
?`:`  # To learn about what the colon really does
```

- When in doubt, adding the back-quotes is always okay when using the help documentation.

```
?sd     # Versus
?`sd`
```

Exploring One Variable: Functions

- A **function** is a set of statements organized together to perform a specific task.
- Functions, like variables, are objects that can be called, along with specific arguments.

```
function_name <- function(arg_1, arg_2, ...) {

  # Function Body
  # Return Value
}
```

The different parts of a function are:

- **Function Name**: This is the actual name of the function. It is stored in R as an object with this name.
- **Arguments**: An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments or default values.
- **Function Body**: The function body contains a collection of statements that defines what the function does.
- **Return Value**: The return value of a function is the last expression in the function body to be evaluated.

For Example:

- We can find out how many observations we have using the `length()` **function**:

```
length(score)   # How many observations are there?
```

```
[1] 10
```

- See whether each student has a missing value for their test score using the `is.na()` **function**:

```
is.na(score)    # See which observations are missing (TRUE/FALSE)
```

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

- Or see which students have missing values using combinations of **functions**:

```
which(is.na(score))    # Function calls read innermost -> outermost
```

```
[1] 10
```

Exploring One Variable: Logical Comparisons

Many operations in R revolve around making **logical** comparisons between our observations and certain values, or between observations. Logical comparisons each come with their own specific operators:

Symbol	Meaning
`a == b`	Equal to
`a < b`	Less than
`a > b`	Greater than
`a <= b`	Less than or equal to
`a >= b`	Greater than or equal to
`a != b`	Not equal to
`a | b`	Or (inclusive: one or the other or both)
`a & b`	And

Exploring One Variable: Logical Comparisons

- Logical operators return `TRUE` or `FALSE` depending on truth of condition

```
score[1]                          # Print Student 1's score
```

```
[1] 50
```

```
score[1] == 10                    # Is Student 1's score equal to 10?
```

```
[1] FALSE
```

```
score[1] < 60 & score[1] > 40     # Is Student 1's score less than 60 and greater than 40?
```

```
[1] TRUE
```

```
score[1] == score[2]              # Is Student 1's score equal to Student 2's score?
```

```
[1] FALSE
```

Exploring One Variable: which() function

- If instead, we wish to know which students match certain comparisons, we can use the `which()` function in combination with our previous commands as follows:

```
score                   # Print all scores
```

```
 [1] 50 20 90 40 60 70 70  0 10 NA
```

```
which(is.na(score))     # See which student is missing a test score
```

```
[1] 10
```

```
which(score == 10)      # See which student has a score of 10
```

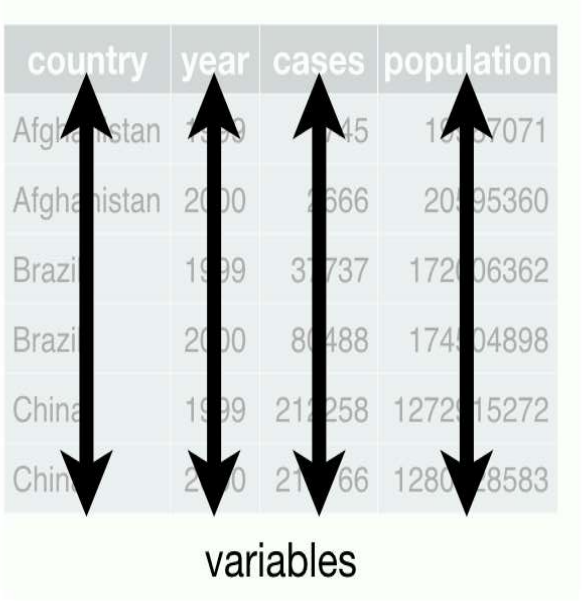
```
[1] 9
```

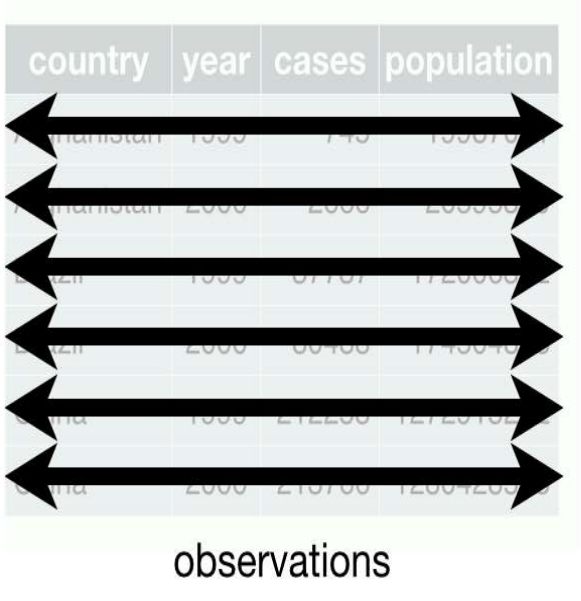
Exploring Data

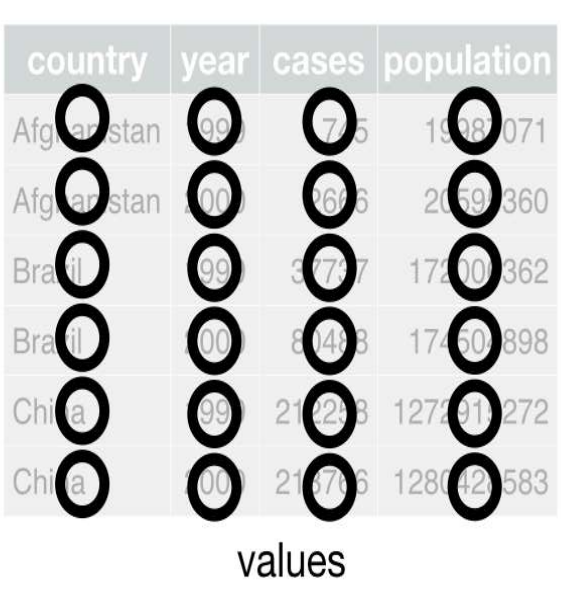
Working with Tidy Data

- In most cases, we will be interested in working with more than one variable at a time.
- To organize multiple variables into a dataset, `R` uses a specific type of object such as a **data frame** or **tibble**.
- Reproducible research works best with **tidy data**:
    1. Each variable in the data set has its own column
    2. Each observation has its own row
    3. Each value has its own cell
- It is (usually) best practice to have a single data source (see: cfss.uchicago.edu/notes/tidy-data/)

Tidy Data


country
year
cases
population
variables


country
year
cases
population
observations


country
year
cases
population
values

Importing Data

- There are many different functions in `R` for reading in different data sources found in the wild, such as `read.csv` (comma-seperated-value) and `read.table` (text files).
- Because of `R`'s great package ecosystem, we can read in just about any data with the right package.
- For example, if we want to import an Excel file, we can use the `readxl` package.
- Also check out the `haven` package for reading in Stata, SAS, and SPSS files.
- **Caveat**: Usually easiest to use original program to export to CSV or text format.

Importing Data: readxl

- We install a package using the `install.packages` function
- Once installed, we load the package for use with the `library()` function.

```
# install.packages("readxl") - Use quotation marks around name

library(readxl) # Don't use quotation marks around name
```

- We can see the help pages available for an entire package using double question marks `??readxl`.
- Often, package authors write long-form documents called **vignettes**. These are nicely formatted HTML pages designed to orient you to the package.
- Use `vignette(package = "readxl")` to see the available vignettes for the package `readxl`.
- Try `vignette("sheet-geometry")` for a detailed explanation on how to navigate and import more complicated Excel files.

60+ Survey of Washtenaw County

- Our example dataset comes from a Statistics in the Commumnity (STATCOM) project
- STATCOM is a community outreach program that provides the expertise of graduate students, free of charge, to non-profit and community organizations in the areas of data organization, analysis, and interpretation.
- STATCOM partnered with the Ann Arbor Area Community Foundation (AAACF), a philanthropic organization which aims to improve the quality of life of citizens within Washtenaw County, Michigan.
- AAACF received over 18 million dollars to help improve the quality of life of older adults who are aging in place within the county, especially those with lower life expectancy and lower socioeconomic status.
- In order to both identify these at-risk populations and understand their needs, AAACF partnered with a team of four STATCOM students to develop, administer, and analyze a survey assessing the quality of life of older adults (60 years of age and older) residing in Washtenaw County.

T. NeCamp, E. Morris, E. Reynolds, S. Salerno. (2018) Data for Good In Your Neighborhood: A case study on how data can benefit your local community. *2018 Bloomberg Data For Good Exchange (D4GX)*.

Importing Data:

Once the data are read in, you can double click on the name of the data frame in the `Environment` panel to view it, use the `View()` function, or obtain summaries of the data using the `summary()` function:

```
dat <- read_excel("data/60plus_v4.xlsx")      # Read-in 60+ survey data

# View(dat)

# str(dat)

names(dat)[1:10]    # Provides names of the first five variables in the dataset
```

```
 [1] "SurveyNum"                          "NameofLocation"
 [3] "SiteLocationNumber"                 "DateofCollection"
 [5] "Envelope Number"                    "D1: How do you describe yourself"
 [7] "D2: What is your age"               "D3a: Asian/Pacific Islander"
 [9] "D3b: American Indian/Alaskan Native" "D3c: Hispanic/Latino(a)"
```

summary()

- `dat` is the variable name that stores our full dataset.
- To access a specific variable in our data, we use the `$` operator.
- For a list of summary statistics, we can use the `summary()` function:

```
summary(dat$`D2: What is your age`) # Summary statistics for participant age
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  35.00   69.00   77.00   92.36   85.00  999.00 
```

- The `summary()` function is context-specific. Its output should change depending on the type of the input!
- When in doubt, `?summary`.

Importing Data: Cleaning

We can see from the previous slide that some values are coded as `999` for missing values. To change these for the entire data frame at once, we can use the indexing and subsetting from earlier to assign a new value:

```
dat[dat == 999] = NA  # Assigns NA to any values in the data frame equal to 999
```

Now that the data are corrected, we can be more confident in using our previous functions such as `summary()`:

```
summary(dat$"D2: What is your age")   # Summary statistics for the age variable
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  35.00   69.00   76.00   76.37   84.00  102.00      13
```

Descriptive Statistics, Continued

- R has built-in functions for most descriptive statistics

```
mean(dat$`D2: What is your age`)   # Find the average age
```

```
[1] NA
```

- Because missing data shouldn't be ignored, `R` functions by default make functions with `NA` inputs `NA` as well
- This is a **feature**, not a **bug**!
- If we know that the `NA` values can be removed without harm, then we can specify `na.rm = TRUE` to calculate the statistic over only non-missing values.

```
mean(dat$`D2: What is your age`, na.rm = TRUE)   # Mean age for available respondents
```

```
[1] 76.36771
```

table()

- To tabulate the number of individuals at each age, we can use the `table()` function.

```
table(dat$`D5: What is your zip code?`, useNA="always") # Frequency of each zip-code
```

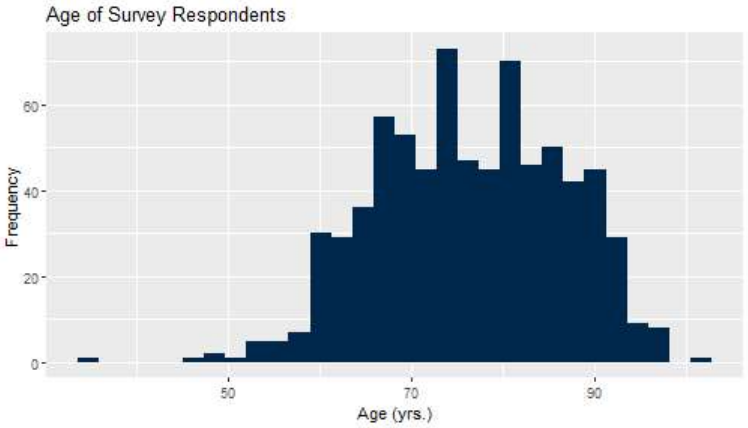
```
42153 48001 48043 48102 48103 48104 48105 48106 48108 48111 48118 48129 48130
    1     1     1     1    77    35    54     2    22     3   153     1    13
48132 48137 48139 48158 48160 48164 48169 48170 48176 48178 48179 48180 48182
    1     8     1    60    18     2     3     4    62     2     1     1     1
48188 48189 48191 48197 48198 48501 49083 49230 49240 49285 49913  <NA>
    1    11     6    83    86     1     1     2     6     1     1    23
```

- The `useNA = "always"` option tabulates missing values as well

Visualizing Data

- R has a wide range of visualization functions (this is a whole separate workshop!)
- Perhaps the most popular is the ggplot2 package, which is based off of the *G*rammar of *G*raphics

```
ggplot(dat, aes(x = `D2: What is your age`)) +
  geom_histogram(fill = "#00274C") +
  labs(x = "Age (yrs.)", y = "Frequency", title = "Age of Survey Respondents")
```



Some Other Data Processing Considerations

- For this work, we collected a convenience sample of 750 surveys.
- However, we only want to keep individuals who are:
    1. Over 60 years of age
    2. Live in Washtenaw County
- Using the `which()` function and the `[` operator, we can subset only those respondents that fit this criteria:

```
dat2 <- dat[which(dat$over.60 == TRUE & dat$washtenaw.ind == TRUE),] # Subset Data

dim(dat2) # New Dimensions
```

```
[1] 629  51
```

Creating New Variables

- We further wanted to encode three metrics for vulnerability:

  1. **Geographic** (living in zip codes 48197 or 48198)
  2. **Financial** (Receiving rent assistance, Medicaid, or answered no to having enough money)
  3. **Living Alone**

- We can use logical comparisons to construct new variables

- `as.numeric` will encode `TRUE` as `1` and `FALSE` as `0`.

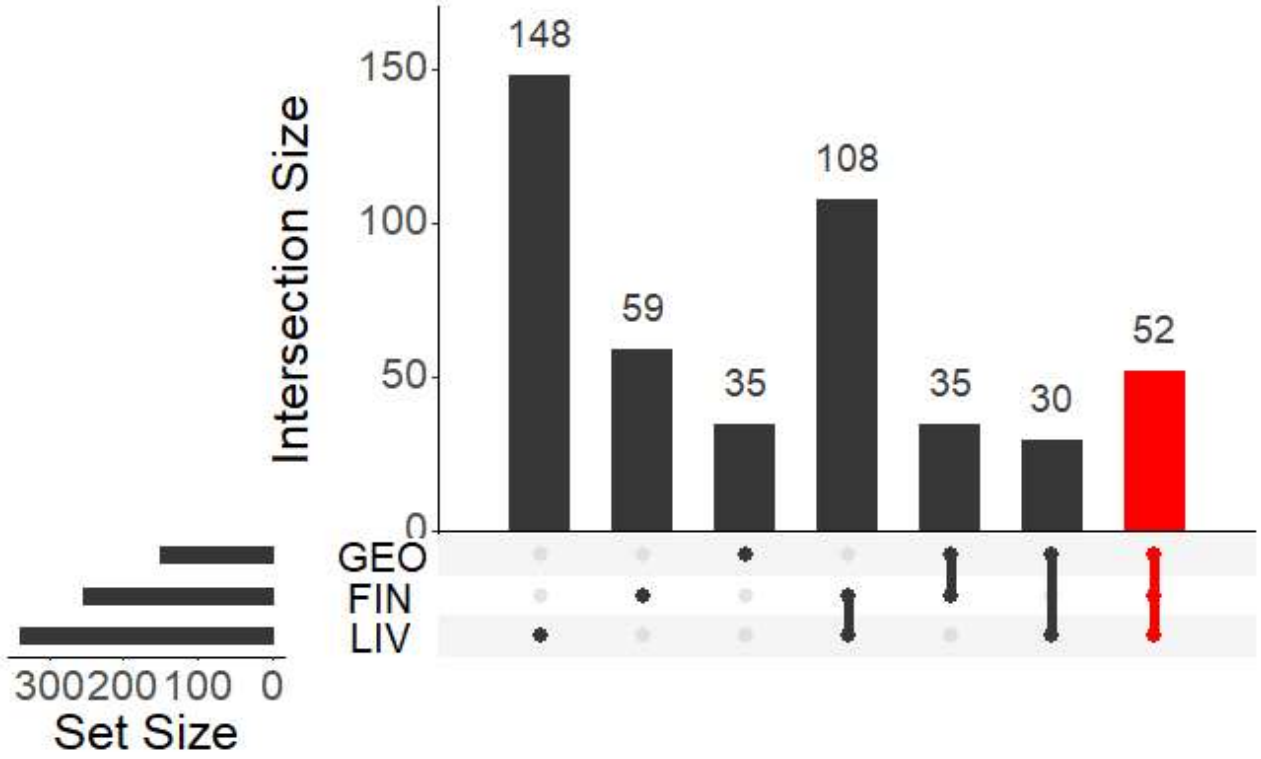
```
# Assign a value of 1 to a variable, GEO, for geographic vulnerability if the survey
# participant lives in either of the indicated zip codes.

dat2$GEO <- as.numeric(dat2$"D5: What is your zip code?" == 48197 |
                       dat2$"D5: What is your zip code?" == 48198)

# Can do this similarly for financial (FIN) and living alone (LIV) vulnerabilities
```

UpSet Plots

We can visualize the intersection of these three vulnerabilities using an `UpSet` plot. The `R` function to produce this figure is located in the `UpSetR` package. We can import and use this package as before:

```
# install.packages("UpSetR")

library(UpSetR)

upset(dat2, sets = c("GEO", "FIN", "LIV"),

  queries = list(

    list(

      query  = intersects,
      params = list("GEO", "FIN", "LIV"),
      color  = "red",
      active = TRUE)),

  text.scale = c(2, 2, 2, 2, 2, 2))
```

UpSet Plot of Vulnerabilities



Beyond This Workshop

Concluding Remarks

- These slides provide a very basic introduction for getting started with analyzing data in `R`.
- `R` has many packages to perform more complicated statistical analysis tasks including data cleaning (`dplyr`) and data visualization (`ggplot2`)
- Because `R` is free open source, hundreds of data enthusiasts are continually developing and contributing packages for any task you can imagine!

R for Data Science

https://r4ds.had.co.nz/introduction.html

- Completely free, online textbook written in Rmarkdown (physical version available from O'Reilly)
- Assumes no prior programming knowledge
- Teaches current best-practices for tidying, analyzing, and visualizing data.

Additional References:

1. https://courses.edx.org/courses/UTAustinX/UT.7.01x/3T2014/56c5437b88fa43cf828bff5371c6a924/
2. https://cran.r-project.org/doc/contrib/usingR.pdf
3. http://imathesis.com/media/usingR.pdf
4. https://www.statmethods.net/

Using {swirl}

Swirl teaches you `R` programming and data science interactively, at your own pace, and right in the `R` console!

The first time using `{swirl}`, open RStudio and type the following into the console:

```
> install.packages("swirl")
```

Every time you want to run `{swirl}` after, load the package and call the `swirl()` function:

```
> library("swirl")
```

```
> swirl()
```

Website: swirlstats.com

Recommended `{swirl}` Classes

Module 1: R Programming

1: Basic Building Blocks
2: Workspace and Files
3: Sequences of Numbers
4: Vectors
5: Missing Values
6: Subsetting Vectors
7: Matrices and Data Frames
8: Logic
9: Functions
12: Looking at Data
14: Dates and Times
15: Base Graphics


the 5th annual
Data for Public Good
symposium
MARCH 24-25, 2022
We invite you to attend the **Data for Public Good Symposium** on **Thursday, March 24, and Friday, March 25, 2022**. The symposium will showcase many research efforts and university/community partnerships that focus on improving humanity by using data for the public good. **Registration is free and open to all!**
- **Featured Speakers:** Kat Hartman, Jeff Leek, and Sheria G. Robinson-Lane
- **Workshops From:** Equilo and the Emergence Collective
- **Alumni Trailblazers:** Lauren Beriont, Rachel Elsinga, and Tim NeCamp
- An **Interactive Poster Session**
- **Research Talks** in Public Health & Medicine, Social Justice & Equity, Education, and more
REGISTER ONLINE AT:
tinyurl.com/2022-d4pg
Interested in learning more? Schedule and updates can be found on our website:
midas.umich.edu/2022-d4pg
Or contact us at:
d4pg-admin@umich.edu
BROUGHT TO YOU BY:
MIDAS MICHIGAN INSTITUTE FOR DATA SCIENCE UNIVERSITY OF MICHIGAN
stat.com
STUDENT LIFE EDWARD GINSBERG CENTER UNIVERSITY OF MICHIGAN
ceder
CENTER FOR EDUCATION DESIGN, EVALUATION, & RESEARCH

Questions?

Feel to reach out with any questions or comments:

salernos@umich.edu

Interested in STATCOM? Visit:

https://sph.umich.edu/biostat/statcom/