R Basics Bootcamp

A Self-Guided Introduction to R and RStudio

This first set of slides is introduces the R Bootcamp

Objective of this Basics to R Bootcamp

To gain more familiarity with R

- Exposure to R if no prior experience
- A refresher and update of R version if its been awhile
- Learn approaches to R, example: if not used R Markdown

Prepare for material for a STAT or STAA course

- The R Bootcamp should be completed before first class day
- Statistics course material focus on statistical concepts, not R coding
- Bootcamp is not a comprehensive for coding in R, just a head start
- **Do not aim for mastery of material presented here!**
    - Primary goal is simply to get R and RStudio up and running
    - Content for general exposure to various functions, plotting, packages
    - If you keep getting errors, just skip it!

Self-guided in Canvas

This R Basics Bootcamp program is intended to be self-contained

Instructor not available for questions of Bootcamp material

- An intentional component to this Bootcamp is learning to seek answers on own
- Resources can also be found within Canvas

Technical issue with the Bootcamp (e.g. broken Canvas links, video malfunction, or missing content)

- You may email course instructor or Ramtech depending on the type of issue
- If new to Canvas please see : https://canvas.colostate.edu/student-support/

Four main components to Bootcamp

Bootcamp Material Provided in "pages" of Canvas

- Follow links to each component of each module

1. There are 8 topics of pdf slides
    - Might print 3 or 4 per page to take notes
    - Each topic should be covered sequentially (after each prior topic is completed)

2. Short videos that accompany slides
    - Echo360 is part of CSU and Canvas which are played within your browser
    - Videos are listed according to topic

3. R Markdown files to help guide coding (more on this in next set of slides)

4. Exercises
    - **NOT Graded and not for course credit!!**
    - Represented as Quizzes in Canvas

Nine Topics of R Basics Bootcamp

1. Getting Started with R and RStudio
2. Basics of Coding in R
3. Basics of R Packages
4. Basics of Importing Data into R
5. Basics of Using Data in R
6. Basics of Plotting in R
7. Using R for Coursework
8. Various Useful Functions and Packages

Proceeding Through Bootcamp

An outline of course material is provided
Start at beginning and sequentially complete Bootcamp
For each Topic

- Click on each pdf (as you did for this one) to download and view
- View video for narrative and follow screen (may minimize instructor window)
- Download R Markdown file and save to an appropriate folder
    - Either with slides or separate R Code Example folder
    - Later topics require data files to be downloaded from Canvas
        - Data files will be imported into R
        - The location and path will be important (consider shorter path and/or folder names)
- Exercise (Quiz) also may have accompanying R Markdown and data files

Topic 1: Getting Started with R & RStudio

1. What is R?
2. What is RStudio?
3. Basics to Using RStudio
4. What is RMarkdown?
5. Getting Started with R scripts and RMarkdown

1. What is R

- R is a command-line programming language for statistical computing
- R is a new implementation of S (a similar stat programming precursor)
- R has a wide range of packages for statistical analysis and graphing
- R is increasingly popular for data management and analysis
- **Download and Install R for Free!**

Command-Line Programming

Can be used as point and click
- R Commander (Rcmdr)
- Not covered here (I don't use it)
- See Rcmdr https://www.rcommander.com
- Package https://cran.r-project.org/web/packages/Rcmdr/index.html

R needs coding instructions
- Then code needs to be ran before anything happens
- Some code defines variables etc... which provides no output
- R Syntax (coding language) requires patience

Command-line programming

Helps achieve "reproducible research"
- Saved code and data files implies analysis & results are preserved
- Any one with R (with appropriate version and packages installed) can run same analysis
- Again, R is free (and so are the packages)

Save your work
- Forethought on organizing saved scripts and data
- Organization is important!
  - Save so you can find later for coursework material and your own research
  - Consider cloud options

R Background

Created by Ross Ihaka and Robert Gentleman at University of Auckland
- Beta release in 2000
- Named after first letter of first names (play on S language)

Comprehensive R Archive Network (CRAN)
- https://cran.r-project.org
- "R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear

and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project homage for further information”

- spend time at CRAN website for more background/documentation/versions

Packages and Popularity of R

Packages are bundles of R code that perform certain tasks

- Authors contribute for free
- Over 15,000 packages available
- Packages make R awesome!
- Packages make R more challenging
- A lot more on specific packages in this R Bootcamp

R has been in and out of top 20 most popular programming languages

- https://www.tiobe.com/tiobe-index/ currently at 22nd
- SAS vs R
- R flexibility, free, AND packages

Downloading R (forever free)

https://cran.r-project.org

Linux, Mac, Windows options for downloading

Follow Instructions for downloading latest version of R

- Ideally, all students will have same version (“replace” earlier versions)
- Frequent updates (multiple per year)
- Some packages/functions may not be compatible with older/newer versions

Just Install R per instructions

No need to open this application of R...at all. Not this

We will use RStudio to open R files. USE THIS

2. What is RStudio

Integrated Development Environment (IDE) for R

What is IDE?

- You have R, the program
- You interface with the program thru RStudio, i.e. open RStudio, not R

Chief developer Hadley Wickham, available in 2009

RStudio is open and Free!

Spend time on Website https://www.rstudio.com

- Explore functionality of RStudio
- Useful Resources e.g. cheat sheets: https://www.rstudio.com/resources/cheatsheets/

“R” will refer to RStudio (or R code) from now on

Downloading RStudio (also forever free)

https://www.rstudio.com/products/rstudio/download/

Choose most recent version for operating system

Again just install RStudio per Instructions

Do open this RStudio application

Make sure it is running ok

- i.e. note the version of R is shown in Console window

This will be our interface with R from now on

3. Basics to Using RStudio

The 4 Windows in RStudio (clockwise)
- Editor (will have tabs for each open application)
- Environment/History/Connections
- Plots/Help/Packages
- Console

Feel free to explore the Menu options

Commonly Used “Buttons”
- Create new application
- Open File
- Save
- Run (maybe)

Basics to Using RStudio (Applications)

Click New Application Button
- Lists many different options

R Script
- Basic R code editor
- Most commonly used to start coding in R
- May use this or R Markdown for coursework

R Markdown
- Is the application we will be using most often for this Bootcamp
- Saved as .Rmd files

Note the many other options
- Will not be using other applications in Bootcamp or initial coursework
- Consider exploring later (a reminder at end of Bootcamp)

4. What is RMarkdown

A markup language

Performs 4 main things
- Edit R code
- Run R code
- Write narrative along with R code (which can be ‘marked up’)
- Produce static (and dynamic) output formats

Output to MS Word or pdf documents covered in later topic of this Bootcamp

there is much more info at https://rmarkdown.rstudio.com

5. Basics to Using R Script and RMarkdown

Open Saved R Markdown File from Canvas (Topic1: Getting Started)

Read the narrative
- this content is similar to what is given when creating a new R Markdown file

Note the shaded area (these are called code chunks, or just chunks)
- This is where R code lives for R Markdown file
- Per instructions (which there will be throughout Bootcamp)
- In the upper right of the shaded area click play button

Create a new R Script by clicking

Note there are now two tabs
- Click on GettingStarted.Rmd tab
- Copy content of first shaded area for summary function
- Paste in new Untitled R Script and click run
- Same output, but now in Console window

Basics of Using R Markdown
Back to the R Markdown Code
Note three ticks and brace with ‘r’, then close with 3 ticks.
Run the next chunk of code for a scatter plot

- Note that there is also a Run button for R Markdown
- Explore the different run options and note the short-cut keys

The ‘cars’ in the brace is a built-in dataset

- Most often you will load data into R from a file
- There are many built-in datasets in R (more on that later)

For now, there’s text, R code, and R output all within an R Markdown application file in the editor
Will come back to description on **Knitting** later in Bootcamp

More Basics to R Scripts/R Markdown
When closing an RStudio...

- Tabbed scripts that are red indicate unsaved scripts
- **Save scripts with organization** (and your future self) in mind (more on this later)
- Quit an R Session from File Menu or by closing RStudio window (or similar)
- Note that if R Script or R Markdown is not saved, RStudio will ask about saving
- If items appear in the Global Environment window, RStudio will also ask about saving Workspace Image
    - Generally, no need to save Workspace and click this checkbox off
    - Many Objects will get stored which may cause problems for rerunning/reusing code
    - Though some have found saving everything useful, RStudio will have to deal with clutter

When opening R

- You can ‘typically’ pick up where you left off in code
- Need to reload objects/packages (more on this later)

Topic 2: Basics for Coding in R

1. Initial Guidelines for R code

2. Objects

3. Examples of Assigning Objects

4. Functions

5. Getting Help with R

1. Elements of R Coding

R is case-sensitive

Number of spaces not important
- But spacing can make code easier to read

Wrapping lines of code ok
- Limiting lines to 80 characters

`#` is Comment character for line or for after line of R code
- Get in habit of commenting code, for others and yourself
- Narrative in R Markdown
    - Before/after chunk of code
    - No need for comment character, but can still use # within code chunk

Elements of R Coding, running R Code

Can code in R script or Create RMarkdown file
- Code in script with no need for any mark-up to code
- RMarkdown, must code inside ` ```{r} ` and closing ticks ` ``` `

To Run code in R Script
- Highlight code to run, click Run button
- Or to run one line of code,
    - just put cursor ANYWHERE on line, click Run
    - Notice that this advance to next line.... Can keep clicking to run sequential lines
- Instead of clicking Run button
    - **Windows: Push Control + Return keys**
    - **Mac: Push Command + Return keys**

Elements of R Coding, running code continued

Running R Markdown Code
- Text outside the 3 ‘open and closing ticks’ will not run
- Only run code in grey area: a chunk
- As before, to run an entire chunk, use play button
- Otherwise, can run code inside chunk as with regular R script
    - See previous slide
    - The Run button provides more options

Revisit methods for running R code (will remind during Bootcamp)
- Save time by running certain parts of code
- Learn efficient key strokes
- Use **copying and pasting effectively** to rerun certain parts of altered code
- Comment out parts that you don’t want to run using #
- Elements of R Coding continued

Errors

- Will learn to interpret some types of errors
- Good red and bad red (warnings are good and often allow code to run anyway)

Keep track of open and closed parentheses, brackets, braces

Keep track of commas

- Errors caused by parens and commas can be hard to interpret

The two basic elements of R code: objects & functions

- Objects store information
- Functions do something

2. Objects

Types of objects

- Vectors (R doesn't not rally have scalars, vectors length = 1)
- Matrices (not really covered here)
- Data.frames (tables of data, we will refer to this object a lot)
- Define function results as object (more on this later)

Stored Objects appear in Global Environment window

$ operator

- References element of an object
- Syntax: `object_name$element_name`

3. Assigning Objects

`<-` vs =

- The same assignment operator (almost)
- '<−' might be thought of as defining an object to a name
- '=' might be thought of as the mathematical equality

Assign a vector

- `Y <- c(2,3,4)`
- `y = c(5, 6, 7)`

Assigning data.frame

- `Data <- cars`
- Will load in R data in Topic 4.

4. Functions

Obvious functions

- `mean()`
- `round()`
- `median()`
- `anova()`

Not Obvious

- `sd()` standard deviation
- `lm()` linear model (later)

Function can depend on type of object

- Same function name can perform different tasks depending on object More o
- e.g. `plot()`, example later

Function with same name in different packages (more on this later)

Functions... continued

Functions require argument(s)

Often functions need an object in parentheses

- `mean(data$y)`

Can specifying detail of a function (often multiple specifications)

- Separated arguments by commas
- `round(data$y, 2)`

Default arguments if not specified

- `round(data$y,2)`

Functions within functions

- Parenthetical caution
- `round(mean(data$y),2)`

5. Getting Help with R

Functions

- Can hit tab key to see arguments
- Note suggested arguments when typing

Help in Rstudio

- Help in Menu
- Help Tab in lower right window R documentation (when package loaded)
- Includes examples
- Examples often use built-in R datasets (copy, paste, and run)

`help()` or `?function_name`

Google

- This is a skill
- You will become increasingly better at finding answers/examples online

Ask colleague

Topic 3: Basics of R Packages

1. What are R packages

2. Installing Packages

3. Loading Packages

4. General notes on R Packages

1. What are packages

The best thing AND the worst thing about R (2 sides of same coin)

Best:

- Packages are Free
- Packages that provide functions for most any type of analytical need
- Packages may include useful datasets
- Functions from packages are ‘often’ easy to implement in R code
- Packages contribute to more universal use of R
- Packages compartmentalize functions
- Anyone can create R packages

Worst

- The are many, many R packages ~ 15,000
- **Anyone can create R packages**
- Must to install and load packages

2. Installing R Packages

Beyond ‘base-R’ functions (stat package)

Installing = Downloading library of R code (that contain functions)

- ‘Adds blocks’ of R code to version of R that was installed
- Some packages come with R
- **You only need to install a package once!**

View names of packages

- RStudio in lower right window under Packages tab
- Names appearing are already **Installed**
- Names themselves are not always informative for what they contain

How do you know what packages does what?

- Colleagues, Coursework, **Google**
- https://cran.r-project.org/web/packages/

Easiest way to Install R Packages

Use RStudio lower-right Window with Packages tab

Click the Install   button

You’ll see the following window

Start to type the name of the package you wish to install

- RStudio will start to fill in package names
- Hit return or click Install button

Note the Install dependencies is checked... This is what you want!

Code runs in console as dependent packages take time to install

`install.packages()` function

Not used as part of coursework

Allows code to be shared more easily

Can use if not using RStudio to run R code

Can quickly list multiple packages to install

Will need to specify or be sure to include dependent packages

Will still need to access network to download packages

Note: other than viewing in RStudio packages window, can view installed packages using `installed.packages()` or `library()`

3. Loading Packages

An installed package must be LOADED in to an R session to use functions

Loading packages:

- **`require`**`(package_name)` or **`library`**`(package_name)`
- Can click on check box in Packages window (not recommended)

In order to access functions in a package

- The package must be installed (keep this in mind when using different computers)
- The package must be loaded in to the current active R session (typically)
- If not installed AND loaded
    - An error like: could not find function "leveneTest"
    - You will see this error... It's easy to forget that for example `leveneTest()` is in the 'car' package

4. General Notes on R packages

Install to current version of R library, updating R means re-installing any packages previously installed

Some packages may not work under previous/old versions of R

Referencing a package (remember this for your future work with R)

- Authors create packages basically for free, we should reference their work
- Use this code: `citation("car")` note that the citation function requires quotes

More Notes on R packages

Have function, need package

- If many packages loaded (assuming multiple packages are loaded)
    - type the function name and run (no parens)
    - for Base-R function, also get source code (nice to know for future R coding)
    - note the environment at end of output, includes package for function

Have package, need function syntax

- Install/Load package first
- Then use help as you would with any function

More Notes on coding with R packages

Typically, required packages loaded at beginning of R code

Possible to have same function names in different loaded packages

Use double colons to call function from specific package

- Example, `car::leveneTest`
- Can use double colons to demonstrate which package function belongs

Which packages are **loaded** in current R session?

- `search()`

Topic 4: Basics of Importing Data into R

1. Data files for use in R

2. Ways to Import Data

3. Basic Functions to Check Data

1. Data files in R

Built-in data files
- Base R (and R packages) include example data frames
- Use data() to view built-in datasets
  - Use '`data(package = .packages(all.available = TRUE))`' to list the data sets in all *available* packages
- Easiest way to load data (use with examples in R help)

Importing datasets
- Comma separated values (.csv files) are common
- Can import from MS Excel (as .xlsx, but more easily as .csv)
- Multiple functions and methods to load data
- Important to examine data once imported!

2. Ways of Importing Data

Point and click via Global Environment window
- Can click Import Dataset button
- Not recommended
  - Some datatype errors may occur
  - Must reload and browse for data each time running new session
  - Can have difficulty maintaining data frame naming throughout code

Running code to load data
- `read.table()`
- `read.csv()` this is the one we will use

Other methods (largely beyond scope of coursework)
- API's -- Application Programming Interface
- Read in from webpage – (some instructors may use this to provide datasets)
- etc...

Importing Data via read.csv()

`read.table()` more versatile, but works similarly as `read.csv()`
Can supply pathname in R Code (much preferred)
Or can browse for Dataset using `file.choose()`
Pathname of dataset
- Can type path name into code (preferred)
- Must not move file
- May need different paths if using different computers to analyze dataset

Browsing for dataset
- Must search for data file each time wanting to load
- Not ideal for using with R Markdown
-

Importing Data via `read.csv()`

Pathname syntax: `dataframe_name <- read.csv("pathname")`
Getting the pathname with Mac!

- Use finder to find dataset
- bottom of finder window shows path
- Right click on the datafile, then push and HOLD option key
- 2/3 down is: Copy “datafilename.csv” as Pathname
- Then paste pathname into R code inside quotes of `read.csv("")`
- May also use ‘~’ to shorten path name
    - Change: `"/Users/bensharp/Dropbox/STAT511/Assignments/Assign6/RatLiver.csv"`
    - to:

`"~Dropbox/STAT511/Assignments/Assign6/RatLiver.csv"`

Importing Data via `read.csv()`

Getting the pathname with Windows

- Use Windows Explorer
- path name can be viewed and copied
- Paste pathname into R code inside of quotes of `read.csv("")`
- BUT need forward slashes in R !!!
- Change all backslashes to forward slashes
- Example code:

```
RatLiver<-read.csv("C:/Users/sharp/Dropbox/STAT511/Assignments/Assign6/RatLiver.csv")
```

- OR can right click on datafilename.csv
    - copy/pasting into Rstudio editor already converts forward slashes
    - Must delete file:/// that appears at the beginning of this pathname
- May be a way to shorten pathname as with Mac?

Importing Data via read.csv()

`file.choose()`

- Syntax: `dataframename <- read.csv(file.choose())`
- Run code, RStudio gives window to browse and select datafilename.csv
- Run `file.choose()` without `read.csv`, click data file to see file pathname in console

Other considerations for long pathnames

- Can create a folder for datafiles near root directory (of cloud storage)
- No path necessary if working directory is in same folder of datafiles
    - Possible to open R script files from Windows Explorer for default working directory to be set in same folder (this may not work for everyone)
    - Try: `getwd()` to see pathname of R session working directory
    - Try: `setwd()` to establish pathname to folder where working with datafiles
    - Syntax without pathname just becomes: `read.csv("datafilename.csv")`

3. Checking Data Loaded into R

“running” data frame is simplest way to see your data

- Create data frame object using `read.csv()`
- “Send” data frame object to Console, i.e. just run the data frame
- Will show limited rows of what was loaded as data frame

`View(dataframename)`

- Yes, its capital ‘V’
- Creates a spreadsheet view of data, but in new tab among R Code tabs
- May copy/paste more easily

- Requires toggling between R code and the View of data
- Not ideal for R Markdown

Checking Data Loaded into R continued

`head(dataframename)`

- Shows the first few rows of the data frame
- Can specify number of rows to be shown

`Tail(dataframename)`

- Shows the LAST few rows of the data frame
- Can specify number of rows to be shown
- Depending on sorting, can show if last few rows were properly loaded

Checking Data Loaded into R continued

`str(dataframename)`

- Super helpful to see information about the data
- Shows list of columns of data
- Data types are shown -- important to know what R believes the data types are!!
- First few elements of each column shown

Other self-explanatory functions for data frames

- `length()`
- `nrow(), ncol()`
- `dimensions()`

Topic 5: Basics of Dealing with Data in R

1. Data Types
2. Factors
3. Basic Use of Data Frames
4. Basic Functions to Manipulate Data
5. Other data structures

1. Data Types

Two basic types of variable (columns)

- Numeric – quantitative measures (discrete or continuous)
- Factors – qualitative or categorical information

Numeric

- If appropriate, can force R to make factor to number
  - `as.numeric()`
- Different classes of numbers
  - `class()` shows wither integer or double etc... (not important in most cases)

- Factor
  - Often appropriate to change a number to a factor!
  - Datasets often have numbers to label different levels of a factor
  - Change number to factor: `as.factor()`

2. Factors

First determine if R is considering variable as factor with `str()`
Important because functions will run different if factor vs. numeric

- Plotting
- Modeling
- Errors for some functions: e.g. R will not do `mean(factor_variable)`

Use `as.factor()` to change numeric to factor & check again with str()
See factor levels with `levels()`
R often defaults to alpha-ordering of levels

- Change order: (move 6th level to the first)

```
clover$Strain <-
factor(clover$Strain,levels(clover$Strain)[c(6,1:5)])
```

3. Using Data Frame

Referencing data frame

- Consider name of data frame that is short and informative
- Need to refer to data frame any time referencing variables (columns) of data
  - Can attach data frame (not recommended)
  - **Use $ operator (as in prior examples, dataframename$variablename)**
  - To specify particular variable that may share same name among multiple datasets

Attaching data frame

- Do not need $ operator to refer to variables (keeps references to variables shorter)
- Should detach data frame

- May create potential issues if working with multiple data frames

Using Data Frame continued

Functions requiring data

- May not need to specify variable with $
- Often use variable names then specify data set
  - `functionname(variablename1, variablename2, data = dataframename)`
  - Appropriately referencing data and variables in functions takes practice
  -

4. Basics to Managing Data

To create a new column in data frame

- `dataframename$newcolumn <- somevalue`
- New column will be added to last of columns of data frame
- Use `str()` to verify new column was added as intended

Example1: Double a particular numeric variable

- `dataframename$newcolumn <- 2*dataframename$originalcolumn`

Example2: Add natural log transformation of a numeric variable

- `dataframename$logcolumn <- log(dataframename$originalcolumn)`

Example3: Difference between two columns

- `dataframename$diffcolumn = dataframename$columnA- dataframename$columnB`
-

Basics to Managing Data continued

Sort using order()

```
data(mtcars)
mtcars
mtcars[order(mtcar$cyl),]
```

Sort from plyr package

```
library(plyr)
arrange(mtcars, cyl)
# to include column names
myCars <- cbind(vehicle=row.names(mtcars), mtcars)
arrange(myCars, cyl, mpg)  #sorts first by cyl then by mpg
# sort with displacement in descending order
arrange(myCars, cyl, desc(disp))
```

5. Other data structures

Vectors & Matrices are common objects to store data (but others exist)

Vectors are a sequence of values

- R actually has no scalar, just vectors of length one
- Functions may treat vectors differently than a column in a data frame

```
Y <- c(5,4,3,2,1)  #Y is a vector
sort(Y) #sort() applies to vectors unlike order() or arrange()
```

Matrices store information by both a column and row

- Can reference elements in two-dimensional indexing

```
B <- matrix(c(2, 4, 3, 1, 5, 7), nrow=3, ncol=2)
B
```

Topic 6: Basics of Plotting in R

1. Managing Plots in RStudio window
2. Functions for plotting

- `plot(x,y)`
- `hist()`
- `boxplot()`
- `More with plot(x,y)`

3. Plotting a Linear Model Object

1. Managing Plots with RStudio

RStudio plot Window vs. R Markdown plots

- R scripts will send plots to the Plots window
- R Markdown will plot in editor Window
- R Markdown can print plot in document with code when Knitting to file (more later)

RStudio Plots window for convenient quick viewing of plots

- R scripts in Plots window allow scrolling through previous plots
- Will need to Export plots and Copy and paste to include plots in documents
  - Can export as file
  - Easiest to Copy to clipboard and then paste in document
- Can change size of Plots window to better fit plots
- Can get error if Plots window is too small for plot

2. Basic Plots

plot(x,y) cartesian plot (scatter plot)

```
#Create sequence of values for x  (aka vector)
x<- seq(-3,3, length = 20)
#Create values of y that are normally distributed function of x
y<- dnorm(x)
plot(x,y)
```

Look up help for plot()

- Either help(plot) or use Help tab in Rstudio
- Note the different types
- R defaults to points when type not specified, try type as line

```
plot(x,y, type = "l")
```

Now try changing length to something larger than 20, say 50...

Note: you'll need to rerun code for redefining y as well

Basic Plots continued with labeling plots

If sharing plots with others, important to have at least minimum of labels
Some plot functions will default to labeling with column names
Need to use quotes when using text labels
Set of common label arguments for base R plotting functions
xlab = "Text label for x-axis"
ylab = "Text label for y-axis"
main = "text label for the overall plot

Basic Plots continued with other options

Once a plot is created...

- Other features can be added in separate lines and functions
- Keep in mind scale and coordinate location
- Lines as example
  - abline() draws a straight line
  - lines() adds a curve (or straight line) according to specified coordinates

Color options can be applied to most any feature of plot

- col = "colorname" (example of misleading code, mistake for "column")

Syntax different for labeling with other plotting packages

- ggplot2 package has similarities to some base R syntax on plots
- more on ggplot2 (it's a popular and versatile plotting package)
- Basic Plots continued with `hist()`

Base-R histogram function to visualize distribution of quantitative values

Change number of bins (number of groups for frequency numeric)

```
data(mtcars)
hist(mtcars$mpg, breaks=20, main="Breaks=20")
hist(mtcars$mpg, breaks=5, main="Breaks=5")
```

Include normal curve with histogram

```
x <- mtcars$mpg
h <- hist(x, breaks=10, col="red", xlab="Miles Per Gallon",
     main = "Histogram with Normal Curve")
xfit <- seq(min(x), max(x), length = 40)
yfit <- dnorm(xfit, mean = mean(x), sd = sd(x))
yfit <- yfit*diff(h$mids[1:2])*length(x)
lines(xfit, yfit, col="blue", lwd=2)
```

Basic Plots continued with `boxplot()`

Base R function to visualize quantitative value for multiple categories

Order of variables in functions matters

'~' is a versatile and useful character in R

R expects quantitative variable 'by' a factor

cyl is numeric, but R will still plot

Order of cyl is intuitive here, but may re-order levels of factor (see managing data)

```
boxplot(mpg ~ cyl, mtcars, main="mpg by cylinder")
#the following throws an error
boxplot(cyl ~ mpg, mtcars)
#the following plots cyl as one box and mpg as another on same y-axis
boxplot(mtcars$cyl, mtcars$mpg) #this plots but is not correct
```

Basic Plots continued with `plot()`

Base R function to visualize quantitative values by another set of quantitative values, i.e. scatter plot

Order matters, y-axis variable is first

Typically, y-axis is dependent variable (the response variable)

Typically, x-axis is independent variable (the predictor variable)

```
  plot(mpg~wt, data= mtcars)
```

`lm()` function fits a linear least-squares model

Similarly, can add line fitted line to scatter to plot above

```
abline(lm(mpg~wt, data= mtcars))
plot(mtcars$mpg, mtcars$wt) #also creates scatter plot, but note axes
```

3. Plotting a linear model object

Same function , different argument

- `plot(linearmodel)`

simple and excellent tool for validating linear models

But can be cumbersome manage

Note instructions in console... Must click in console windowand hit return to see plots

If unfamiliar, will learn more about plots as part of coursework

```
mpgbywt <- lm(mpg~wt, data = mtcars)
plot(mpgbywt)
```

Plotting a linear model object continued

Will need to hit return 4 times to see all plots

Can scroll through plots with arrows

Or, can see all 4 plots in same Plots window

Create 4 panels to put in plots, but may need to make window larger!

```
par(mfrow = c(2,2))

plot(mpgbywt)
```

Plot window remains divided until reset session, or hit or use code like:

```
par(mfrow = c(1,1))
```

Topic 7: Using R for Class/Assignments

1. R Script vs R Markdown

2. More on Using R Markdown

3. Saving & Organizing Files

4. General coding reminders

1. R Script vs R Markdown for Coursework

R Scripts is for getting code to work and quickly see results

- R Script editor in RStudio is a user friendly programming editor

R Markdown is for report with R code/ouput

- The good for R Markdown
  - Sharing code and results is best with R Markdown
  - User-friendly mark-up editor
  - Has capabilities to write equations
  - Self-contained code and output
  - Knitting to pdf, html, or document
- The not so good for R Markdown
  - Coding in R requires including braces {``` r} ``` (i.e. harder to copy and paste lots of code)
  - Outputs to editor (not always bad, but lengthy coding window)
  - Some functions do not work as effectively (more later)
  - Knew syntax to learn to use Knitting feature effectively

2. More on Using R Markdown

Knitting to file

- Need knitr package
- Generates a file of R code and output together
- Create report in html, pdf, or document
- Report saved in same directory as R Markdown script by default

Advantages to Knitting to file

- Can recreate report quickly when updating code/output
- Can create publication-ready reports
- Can choose which material to include in report
- Familiarity since course material may appear as knitted pdf files
- Easy to learn and execute

More on Using R Markdown continued

Notes on Knitting to MS Document vs knitting to pdf file

Assignment material is likely to require submission as pdf

Knitting to pdf also requires additional Latex application

- Download Latex editor (if not already)
- LaTex https://www.latex-project.org
- See Getting LaTex link, Following install instructions
- Will not need to run explicitly run LaTex

Knitted pdf files can not be edited (without additional software)

Knitted document files can be edited

Knitting to document does not require additional software

But must save document as pdf file (another step to get submission file)

More on Using R Markdown knitting to pdf

When open a new R Markdown application, can add title add specify PDF
Use provided R Markdown file
Run all chunks to make sure no errors!
Then click the Knit button
A pdf preview launches if successful (pdf file simultaneously saved)
If LaTex editor is not installed, pdf editor will compile, but error

```
No TeX installation detected (TeX is required to create PDF output). You shoul
d install a recommended TeX distribution for your platform:
    Windows: MiKTeX (Complete) - http://miktex.org/2.9/setup
    (NOTE: Be sure to download the Complete rather than Basic installation)
     Mac OS X: TexLive 2013 (Full) - http://tug.org/mactex/
    (NOTE: Download with Safari rather than Chrome _strongly_ recommended)
```

More on Using R Markdown knitting to pdf (continued)

Can add mark-up outside R code chunks

- # creates heading
- ** text ** bolds text
- Many other options (if interested in using R Markdown more broadly)

Options within braces

- ```{r, message = FALSE} or ```{r, warning = FALSE} saves space by turning off warning
- ```{r, echo = FALSE} does not include the code, just output
- ```{r, fig.width = , fig.height = } helps sizing plots to better fit pages

Other resources for additional coding options

- Check RStudio resources (R Markdown Cheat sheet included in Canvas also)
- Online searches can to help with more coding tips and resolve R Markdown errors

3. Saving and Organizing Files

May have four separate files that accompany a project or assignment

- **R Script file** to get code started and working
- Pasting R Script into **R Markdown file** to prepare report and add narrative
- R Markdown **pdf file**
- Data file as **.csv file**

Save each file with appropriate informative names

- May reuse name since extensions are different
- Code goes thru iterations, give draft names informative connotate

Organize files to find later

- Keep in mind that you will copy and paste A LOT!
- If code works recycle it and change out data and variable names
- Be able to find and reuse useful functions, plots (even years from now)
- recommend Dropbox or equivalent to access from multiple computers etc...

4. Coding Reminders

Comment your code!

- Whether R Markdown or R describe what code does
- Use the beginning of the code to explain the general purpose of the code
- Comment by line and/or by major chunks of code that you get working

Organize code in the editor

- Use some spaces(or tab) at beginning of line for lengthy functions
- Space between lines to break up code blocks

Denote/load packages at beginning of code
Denote dataset and any nuance to data
Explain warnings or errors if some code needs attention later
Note any special or useful functions at the top of code that is used

- you will likely want to copy and paste later if they work properly

Topic 8: Useful Functions (and packages)

1. Useful Base-R Functions
2. More Data Management
3. ggplot2 plotting package
4. List of useful packages
5. Some useful resources
6. Extensions to R

1. Useful Base-R Functions, calculations

`sum(), mean(), median(), sd()`
`var(x) = (sd(x))^2`
`sqrt(x) = x^.5`
`exp()` – raises e to a value, inverse: `log()` – is natural log
`summary()` – multi-use function, depends on argument

- `summary(x)` where x is a vector of numeric gives 5-number summary
- `min(x), max(x), median()`, 1st quartile, 3rd quartile... And `mean()`
- `quantile(x, .25)` = 1st quantile, `median(x) = quantile(x, .5)`

Useful Base-R Functions, distributions

Normal distribution functions

- dnorm(x) – calculates normally **distributed** value for given x
- pnorm(x) – calculates normally **probability** for all values less than a given x
- qnorm(x) – calculates the appropriate **quantile** for a given probability
  - inverse of pnorm(): qnorm(pnorm(x)) = x
- rnorm(x) – generates **random** normally distributed value(s)

Arguments vary based on type of distribution function

- Typically specify a mean and standard deviation
- For norm, default is mean = 0, sd = 1, i.e. standard normal
- Different distributions have different parameters
- For r-distribution_name, must specify number of randomly generated values

Some other distributions with similar 4 functions (with appropriate first letter)

- beta – beta distribution
- binom – binomial
- exp - exponential
- t – t distribution
- unif – uniform distribution

Other Useful Base-R Functions

`seq()`
`rep()`
`t.test()`

- Common function for simple inference about means
- Many different argument formats
- Basic syntax is similar to boxplot (but with only 2 levels to factor)
- See more about boxplot examples in this topic (i.e. using '~' or not)
- Much more about `t.test()` will be used in coursework

2. More with data management

`with()` – another way to deal with data frame and columns

```
with(mtcars, mpg[cyl == 8 & disp > 350])
```

`aggregate()` - summarize data frame

```
with(mtcars, aggregate(mpg, by = list(cyl), FUN = "mean"))
```

Combining data into a single data frame (no examples with these functions)

- `rbind()` – binds rows... Just stacks more rows (of same type of data and columns)
- `cbind()` – binds columns to one data frame, but will need same number of rows for each column
- `merg()` – combines columns of two data frames, but each data frame must have common identifier to link the two.
-

More with data management

`subset()`

- Selects part of a data frame

```
cyl4mpg <- subset(mtcars, cyl == "4", select = c(mpg))
```

- "stacking" two columns of a dataset
- reshape2 package, `melt()` function

- `sample()` selects a random sample from data set
- See R Example for more detail and code

More with data management, summary statistics

- create a SumStats object (name is arbitrary)
- Use `ddply()` function
  - Like `aggregate()` but more options

```
mtcars$cyl <- as.factor(mtcars$cyl)
SumStats <- ddply(mtcars, c("cyl"),summarise,
                      n = length(mpg),
                      mean = mean(mpg),
                      sd = sd(mpg),
                      SE = sd/sqrt(n))
SumStats
```

3. Plotting with ggplot2 package

common plotting tool in R

R Basics Bootcamp does not cover plotting with ggplot2

- Plotting options are diverse and unique to fields
- Coursework will provide opportunity for plotting experience

Comprehensive plotting package

- Faceting
- Multiple dimensions
- Spatial plotting

Takes some time to get used to syntax, but eventually intuitive

Please see the ggplot2 cheat sheet for examples

4.More advanced stuff (FYI)

Creating functions

- Example of mean and standard error of a vector d

```
mean.fun <- function (d)
{ m <- mean(d)
n <- length(d)
se <- sd(d)/sqrt(n)
c(m, se)
}
x <- c(2,3,5,9,2,4,6,4,9,5)
mean.fun(x)
```

More advanced stuff (FYI)

Loops/simulations are great in R

Not necessarily need ‘for loops’ or ‘while loops’

Can use ddply or random number generators (e.g. rnorm) to create many values

- Then perform operations on entire sequence of values
- Out put values to new data frame etc...
- See ddply package for potential options

Conditional statements are typically straightforward in R also

- If, ifelse or subset selections etc...
- See resources and helps for syntax
-

More advanced stuff (FYI)

Missing Values!

```
a <- c(1, 3, NA, 7, 9)
sum(a)
help(sum)
sum(a, na.rm = TRUE)
```

`x[!is.na(x)]` to remove NA’s

Saving Data frames

- `write.csv(data,”data.csv”, row.names=FALSE)`
- Row names may cause confusion when saving
- `write.table(data, “data.csv”, sep=”\t”, row.names=FALSE, col.names=FALSE)`
- “\t” separates with tabs.

5. Useful packages

- ggplot2 (of course)
- plyr – data management
- reshape2 – for melt function
- car – companion to applied regression
- emmeans – comparing means (ANOVA)
- MASS – modern applied statistics with S (various modeling/plotting)
- tidyverse – a major tool for managing data, a “game changer”
- Many others: stay tuned for suggestions from instructors, colleagues, and referenced work of others in your field

6. Useful Resources, other than google

RStudio (of course)
- Cheat sheets
- Community stuff

Data camp
- https://www.statmethods.net/index.html

Stack over flow

Github

Coding and Cookies
- CSU seminar
- https://lib.colostate.edu/services/data-management/coding-cookies/

7. Other Stuff R Can Do

- Create presentations
- Manage Data
- Embed Python Code
- Run API's
- RShiny allow for HTML interaction
- Interactive plots
- Spatial/Mapping functionality