

Getting Started with R and RStudio

Jessica Minnier, PhD & Meike Niederhausen, PhD
OCTRI Biostatistics, Epidemiology, Research & Design (BERD) Workshop

2019/02/26

Slides available at http://bit.ly/berd_r_intro

pdf version: http://bit.ly/berd_r_intro_pdf

Pre-course installation

Install R

- Windows : Download from <https://cran.rstudio.com/bin/windows/base/>
- Mac OS X: Download the latest .pkg file (currently R-3.5.2.pkg) from <https://cran.rstudio.com/bin/macosx/>

Install RStudio Desktop Open Source License

- Select download file corresponding to your operating system from <https://www.rstudio.com/products/rstudio/download/#download>

Questions

- Who has used R?
- What other statistical software have you used?
- Has anyone used other programming languages (C, java, python, etc)?
- Why do you want to learn R?

Learning Objectives

- Basic operations in R/RStudio
- Understand data structures
- Be able to load in data
- Basic operations on data
- Be able to make a plot
- Know how to get help

Introduction

Rrrrrr?

What is R?

- A programming language
- Focus on statistical modeling and data analysis
 - import data, manipulate data, run statistics, make plots
- Useful for "Data Science"
- Great visualizations
- Also useful for most anything else you'd want to tell a computer to do
- Interfaces with other languages i.e. python, C++, bash



For the history and details: [Wikipedia](#)

- an interpreted language (run it through a command line)
- procedural programming with functions
- Why "R"?? Scheme (?) inspired S (invented at Bell Labs in 1976) which inspired R (**free and open source!** in 1993)

What is RStudio?

- R is like a car's engine
- RStudio is like a car's dashboard

R: Engine



RStudio: Dashboard



- R is a programming language
- RStudio is an integrated development environment (IDE) = an interface to use R (with perks!)

from [Modern Dive](#); see also [DataCamp's video discussion on the difference](#)

Start RStudio

2.1.2 Using R via RStudio

Recall our car analogy from above. Much as we don't drive a car by interacting directly with the engine but rather by using elements on the car's dashboard, we won't be using R directly but rather we will use RStudio's interface. After you install R and RStudio on your computer, you'll have two new programs AKA applications you can open. We will always work in RStudio and not R. In other words:

R: Do not open this



RStudio: Open this



from [Modern Dive](#)

RStudio anatomy

BuzzR

RStudio anatomy

<https://buzzrbeeline.blog/>

Emma Rand

Script file

Write code here

To run code put your cursor on the line and click the run button

Edit to correct errors

⇒ record of commands that worked

Save scripts with the .R extension

⇒ syntax will be highlighted

⇒ good practice

<- is the assignment operator

⇒ puts what is on the right in to the object on the left

⇒ Assign results if you want to use them again

Console

When you click run, code is sent to the console and executed

> is the prompt

⇒ do not type it

⇒ appears when R is ready for next command

Command output goes here by default

⇒ output is in a different colour

⇒ [1] indicates 3.4 is the first element of the output

⇒ many commands will not have output, the prompt just reappears

Script: where you write code

Console: where output goes

Environment

Name objects by assignment to use them again

All the objects you created in your session

Saving the environment saves all the objects, but not the code with a .RData extension

History

A history of every command you sent to the console, mistakes included.

File can be saved but usually you just need the script

Packages

Many functions come with R
A huge amount of extra functionality is available in packages

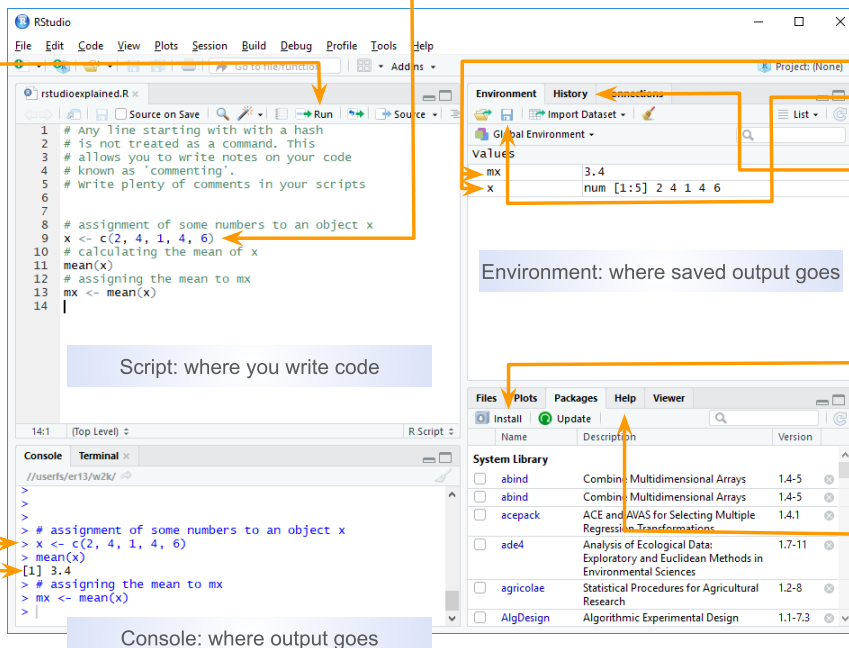
Packages can be installed by clicking the Install button

Help

Access to manual pages for all installed packages

Plots

Figure output appears here



from Emma Rand

Rstudio demo

Let's code!

Coding in the console

Typing and executing code in the console

- Type code in the console
- Press **return** to execute the code

Coding in the console is not advisable for most situations!

- We only recommend this for short pieces of code that you don't need to save

```
> 7
```

```
[1] 7
```

```
> 3 + 5
```

```
[1] 8
```

```
> "hello"
```

```
[1] "hello"
```

```
> # this is a comment, nothing happens  
> # 5 - 8
```

We can do math

```
> 10^2
```

```
[1] 100
```

```
> 3 ^ 7
```

```
[1] 2187
```

```
> 6/9
```

```
[1] 0.6666667
```

```
> 9-43
```

```
[1] -34
```

R follows the rules for order of operations and ignores spaces between numbers (or objects)

```
> 4^3-2* 7+9 /2
```

```
[1] 54.5
```

The equation above is computed as

$$4^3 - (2 \cdot 7) + \frac{9}{2}$$

Logarithms and exponentials

Logarithms: `log()` is base e

```
> log(10)
```

```
[1] 2.302585
```

```
> log10(10)
```

```
[1] 1
```

Exponentials

```
> exp(1)
```

```
[1] 2.718282
```

```
> exp(0)
```

```
[1] 1
```

Check that `log()` is base e

```
> log(exp(1))
```

```
[1] 1
```

Variables

Data, information, everything is stored as a variable

- Can assign a variable using either `=` or `<-`
 - Using `<-` is preferable

Assigning just one value:

```
> x = 5  
> x
```

```
[1] 5
```

```
> x <- 5  
> x
```

```
[1] 5
```

Assigning a **vector** of values

- Consecutive integers

```
> a <- 3:10  
> a
```

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

- **Concatenate** a string of numbers

```
> b <- c(5, 12, 2, 100, 8)  
> b
```

```
[1] 5 12 2 100 8
```

We can do math with variables

Math using variables with just one value

```
> x <- 5  
> x
```

```
[1] 5
```

```
> x + 3
```

```
[1] 8
```

```
> y <- x^2  
> y
```

```
[1] 25
```

Math on vectors of values: element-wise computation

```
> a <- 3:6  
> a
```

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

```
> a+2
```

```
[1] 5 6 7 8
```

```
> a*3
```

```
[1] 9 12 15 18
```

```
> a*a
```

```
[1] 9 16 25 36
```


Variable can include text (characters)

```
> hi <- "hello"  
> hi
```

```
[1] "hello"
```

```
> greetings <- c("Guten Tag", "Hola", hi)  
> greetings
```

```
[1] "Guten Tag" "Hola"      "hello"
```






Viewing list of defined variables

- The R command to see what objects have been defined is `ls()`.
- This list includes all defined objects (including dataframes, functions, etc.)

```
> ls()
```

```
[1] "a"      "b"      "greetings" "hi"      "x"      "y"
```

- You can also look at the list in the Environment window:

Environment		History	Connections
   Import Dataset ▾ 			
 Global Environment ▾			
Values			
a	int [1:4]	3 4 5 6	
b	num [1:5]	5 12 2 100 8	
greetings	chr [1:3]	"Guten Tag" "Hola" "hello"	
hi		"hello"	
x		5	
y		25	

Removing defined variables

- The R command to delete an object is `rm()`.

```
> ls()
```

```
[1] "a"      "b"      "greetings" "hi"      "x"      "y"
```

```
> rm("greetings", hi) # Can run with or without quotes  
> ls()
```

```
[1] "a" "b" "x" "y"
```

- Remove EVERYTHING - *Be careful!!*

```
> rm(list=ls())  
> ls()
```

```
character(0)
```

- Can also remove everything using the *Clear Workspace* option in the *Session* menu.

Common console errors

Incomplete commands

- When the console is waiting for a new command, the prompt line begins with `>`
 - If the console prompt is `+`, then a previous command is incomplete
 - You can finish typing the command in the console window

Example:

```
> 3 + (2*6  
+ )
```

```
[1] 15
```

Object is not found

- This happens when text is entered for a non-existent variable (object)


Example:

```
> hello
```


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

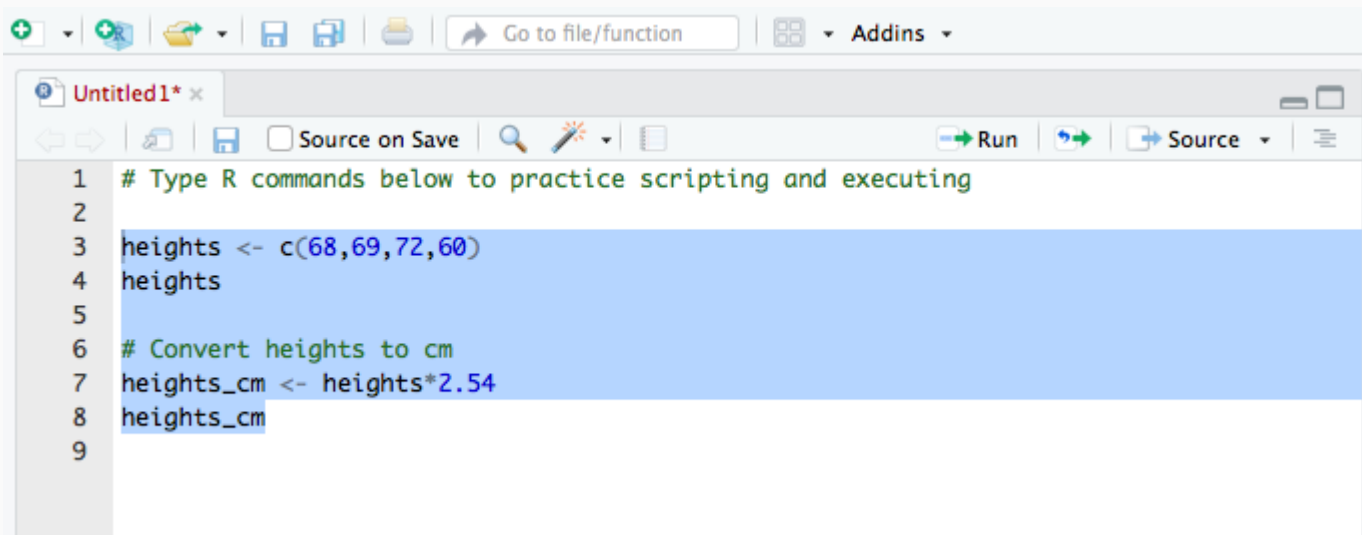
R scripts (save your work!)

Coding in a script (1/3)

- **Create a new script** by
 - selecting `File -> New File -> R Script`,
 - or clicking on  (the left most button at the top of the scripting window), and then selecting the first option `R Script`
- **Type code** in the script
 - Type each R command on its own line
 - Use `#` to convert text to comments so that text doesn't accidentally get executed as an R command

Coding in a script (2/3)

- **Select code** you want to execute, by
 - placing the cursor in the line of code you want to execute,
 - or highlighting the code you want to execute
- **Execute code** in the script, by
 - clicking on the  **Run** button in the top right corner of the scripting window,
 - or typing one of the following key combinations to execute the code
 - **Windows: ctrl + return**
 - **Mac: command + return**



The screenshot shows an R script editor window titled "Untitled1* x". The window has a toolbar at the top with icons for file operations (new, open, save, print) and a search bar labeled "Go to file/function". Below the toolbar is a menu bar with "Addins". The script content is as follows:

```
1 # Type R commands below to practice scripting and executing
2
3 heights <- c(68,69,72,60)
4 heights
5
6 # Convert heights to cm
7 heights_cm <- heights*2.54
8 heights_cm
9
```

Coding in a script (3/3)

- The screenshot below shows code in the scripting window (top left window)
- The executed highlighted code and its output appear in the console window (bottom left window)

The screenshot displays the RStudio interface with the following components:

- Scripting Window (Top Left):** Contains an R script titled "Untitled1*". The code is as follows:


```
1 # Type R commands below to practice scripting and executing
2
3 heights <- c(68,69,72,60)
4 heights
5
6 # Convert heights to cm
7 heights_cm <- heights*2.54
8 heights_cm
9
```

Lines 3 through 8 are highlighted in blue.
- Console Window (Bottom Left):** Shows the output of the executed code:

```
> heights <- c(68,69,72,60)
> heights
[1] 68 69 72 60
>
> # Convert heights to cm
> heights_cm <- heights*2.54
> heights_cm
[1] 172.72 175.26 182.88 152.40
>
```
- Environment Window (Top Right):** Displays the current environment with the following values:

Variable	Value
heights	num [1:4] 68 69 72 60
heights_cm	num [1:4] 173 175 183 152
- Files Window (Bottom Right):** Shows the file explorer with the following structure:
 - Home > Box Sync > R_practice
 - ..

Saving a script

- **Save a script** by
 - selecting **File -> Save**,
 - or clicking on  (towards the left above the scripting window)
- You will need to specify
 - a **filename** to save the script as
 - ALWAYS use **.R** as the filename extension for R scripts
 - the **folder** to save the script in

Practice time!

Practice questions

1. Create a vector of all integers from 4 to 10, and save it as `a1`.
2. Create a vector of *even* integers from 4 to 10, and save it as `a2`.
3. What is the sum of `a1` and `a2`?
4. What does the command `sum(a1)` do?
5. What does the command `length(a1)` do?
6. Use the commands to calculate the average of the values in `a1`.
7. The formula for the first n integers is $n(n + 1)/2$. Compute the sum of all integers from 1 to 100 to verify that this formula holds for $n = 100$.
8. Compute the sum of the squares of all integers from 1 to 100.
9. Take a break!

Object types

Data frames

Vectors vs. **data frames**: a data frame is a collection (or array or table) of vectors

```
> df <- data.frame(IDs=1:3,  
+                 gender=c("male", "female", "Male"),  
+                 age=c(28, 35.5, 31),  
+                 trt = c("control", "1", "1"),  
+                 Veteran = c(FALSE, TRUE, TRUE))  
> df
```

	IDs	gender	age	trt	Veteran
1	1	male	28.0	control	FALSE
2	2	female	35.5	1	TRUE
3	3	Male	31.0	1	TRUE

- A data frame allows different columns to be of different data types (i.e. numeric vs. text), and even allows both numeric and text within a column (stored together as text).
- Vectors and data frames are examples of *objects* in R.
 - There are other types of R objects to store data, such as matrices, lists, and tibbles.
 - These will be discussed in future R workshops.

Variable types

- integer: integer-valued numbers
- numeric: numbers that are decimals
- factor: how categorical variables are stored
- character: text
- logical (TRUE, FALSE)

Each variable (column) in a data frame can be of a different type.

- View the **structure** of our data frame to see what the variable types are:

```
> str(df)
```

```
'data.frame':    3 obs. of  5 variables:
 $ IDs      : int  1 2 3
 $ gender   : Factor w/ 3 levels "female","male",...: 2 1 3
 $ age      : num  28 35.5 31
 $ trt      : Factor w/ 2 levels "1","control": 2 1 1
 $ Veteran: logi  FALSE TRUE TRUE
```

Data frame cells, rows, or columns

Show whole data frame

```
> df
```

	IDs	gender	age	trt	Veteran
1	1	male	28.0	control	FALSE
2	2	female	35.5	1	TRUE
3	3	Male	31.0	1	TRUE

Specific cell value:

`DatSetName[row#, column#]`

```
> # Second row, Third column  
> df[2, 3]
```

```
[1] 35.5
```

Entire column: `DatSetName[, column#]`

```
> # Third column  
> df[, 3]
```

```
[1] 28.0 35.5 31.0
```

Entire row: `DatSetName[row#,]`

```
> # Second row  
> df[2,]
```

	IDs	gender	age	trt	Veteran
2	2	female	35.5	1	TRUE

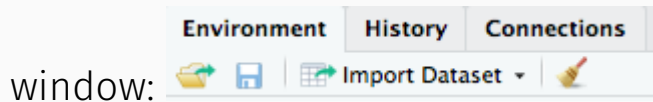
Getting the data into Rstudio

Load a data set

- Open csv file directly from the internet:

```
> mydata <- read.csv(url("http://bit.ly/berd_data_csv"))
```

- Or, download file and open saved file using Import Dataset button in Environment



- If you use this option, then copy and paste the code from the console importing the data to your script so that you have a record of from where and how you loaded the data set.

```
> View(mydata)  
> # Can also view the data by clicking on its name in the Environment tab
```

About the data

Data from the CDC's [Youth Risk Behavior Surveillance System \(YRBSS\)](#)

- complex survey data
- national school-based survey conducted by CDC and state, territorial, tribal, and local surveys conducted by state, territorial, and local education and health agencies and tribal governments
- monitors six categories of health-related behaviors that contribute to the leading causes of death and disability among youth and adults (including alcohol & drug use, unhealthy & dangerous behaviors, sexuality, physical activity); see [Questionnaires](#)
- this data is a small subset (20 rows) of data in the R package `yrbss` which includes YRBSS from 1991-2013
- we will use the full R data set in a future workshop teaching data cleaning

Data set summary

```
> summary(mydata)
```

```
      id                age      sex      grade
Min.   : 335340    14 years old      :1  Female:12    10th:8
1st Qu.: 925193    15 years old      :4  Male   : 8    11th:4
Median :1207132    16 years old      :7                      12th:4
Mean    :1093150    17 years old      :7                      9th :4
3rd Qu.:1313188    18 years old or older:1
Max.     :1316123

      race4      bmi      weight_kg
All other races      :5  Min.      :17.48  Min.      :43.09
Black or African American:3  1st Qu.:20.36  1st Qu.:57.27
Hispanic/Latino      :6  Median   :22.23  Median   :64.86
White                 :4  Mean     :23.01  Mean     :64.09
NA's                  :2  3rd Qu.:26.58  3rd Qu.:70.31
                        Max.     :29.35  Max.     :84.82

      text_while_driving_30d  smoked_ever  bullied_past_12mo
0 days                      : 5          No   :10          Mode :logical
1 or 2 days                  : 2          Yes   : 6          FALSE:11
3 to 5 days                  : 1          NA's: 4          TRUE  :7
All 30 days                  : 1                      NA's :2
I did not drive the past 30 days: 1
NA's                        :10
```

Data set info

```
> dim(mydata)
```

```
[1] 20 10
```

```
> nrow(mydata)
```

```
[1] 20
```

```
> ncol(mydata)
```

```
[1] 10
```

```
> names(mydata)
```

```
[1] "id"           "age"  
[3] "sex"          "grade"  
[5] "race4"        "bmi "  
[7] "weight_kg"    "text_while_driving_30d"  
[9] "smoked_ever"  "bullied_past_12mo"
```

Data structure

- What are the different **variable types** in this data set?

```
> str(mydata)  # structure of data
```

```
'data.frame':    20 obs. of  10 variables:
 $ id           : int  335340 638618 922382 923122 923963 925603 933724 93543
 $ age          : Factor w/ 5 levels "14 years old",...: 4 3 1 2 2 3 3 4 2 4 .
 $ sex          : Factor w/ 2 levels "Female","Male": 1 1 2 2 2 2 1 1 2 1 ...
 $ grade        : Factor w/ 4 levels "10th","11th",...: 1 4 4 4 1 1 1 3 1 4 .
 $ race4        : Factor w/ 4 levels "All other races",...: 4 NA 4 4 2 1 1 1 1 1
 $ bmi          : num  27.6 29.3 18.2 21.4 19.6 ...
 $ weight_kg    : num  66.2 84.8 57.6 60.3 63.5 ...
 $ text_while_driving_30d: Factor w/ 5 levels "0 days","1 or 2 days",...: NA NA NA NA NA
 $ smoked_ever  : Factor w/ 2 levels "No","Yes": NA 2 2 2 1 1 2 1 NA 1 ...
 $ bullied_past_12mo  : logi  NA NA FALSE FALSE TRUE TRUE ...
```

View the beginning of a data set

```
> head(mydata)
```

	id	age	sex	grade	race4	bmi
1	335340	17	years old	Female	10th	White 27.5671
2	638618	16	years old	Female	9th	<NA> 29.3495
3	922382	14	years old	Male	9th	White 18.1827
4	923122	15	years old	Male	9th	White 21.3754
5	923963	15	years old	Male	10th	Black or African American 19.5988
6	925603	16	years old	Male	10th	All other races 22.1910

	weight_kg	text_while_driving_30d	smoked_ever	bullied_past_12mo
1	66.23		<NA>	NA
2	84.82		<NA>	Yes
3	57.61		<NA>	Yes
4	60.33		<NA>	Yes
5	63.50		<NA>	No
6	70.31		<NA>	No

```
> head(mydata, 2)
```

	id	age	sex	grade	race4	bmi	weight_kg
1	335340	17	years old	Female	10th	White 27.5671	66.23
2	638618	16	years old	Female	9th	<NA> 29.3495	84.82

View the end of a data set

```
> tail(mydata)
```

	id	age	sex	grade	race4
15	1313153	16 years old	Female	11th	Hispanic/Latino
16	1313291	16 years old	Female	11th	White
17	1313477	16 years old	Female	10th	All other races
18	1315121	17 years old	Female	11th	<NA>
19	1315850	17 years old	Female	12th	Hispanic/Latino
20	1316123	18 years old or older	Female	12th	Black or African American

	bmi	weight_kg	text_while_driving_30d	smoked_ever
15	26.5781	68.04	0 days	No
16	24.8047	63.50	3 to 5 days	No
17	25.0318	76.66	0 days	No
18	22.2687	54.89	I did not drive the past 30 days	Yes
19	19.4922	49.90	0 days	<NA>
20	27.4894	74.84	All 30 days	Yes

	bullied_past_12mo
15	TRUE
16	FALSE
17	TRUE
18	FALSE
19	FALSE
20	FALSE

Working with the data

The \$

Suppose we want to single out the column of BMI values.

- How did we previously learn to do this?

```
> mydata[, 6]
```

```
[1] 27.5671 29.3495 18.1827 21.3754 19.5988 22.1910 20.9913 17.4814  
[9] 22.4593 26.5781 21.1874 19.4637 20.6121 27.4648 26.5781 24.8047  
[17] 25.0318 22.2687 19.4922 27.4894
```

The problem with this method, is that we need to know the column number which can change as we make changes to the data set.

- Use the `$` instead: `DatSetName$VariableName`

```
> mydata$bmi
```

```
[1] 27.5671 29.3495 18.1827 21.3754 19.5988 22.1910 20.9913 17.4814  
[9] 22.4593 26.5781 21.1874 19.4637 20.6121 27.4648 26.5781 24.8047  
[17] 25.0318 22.2687 19.4922 27.4894
```

Basic plots of numeric data (1/3)

Histogram

```
> hist(mydata$bmi)
```



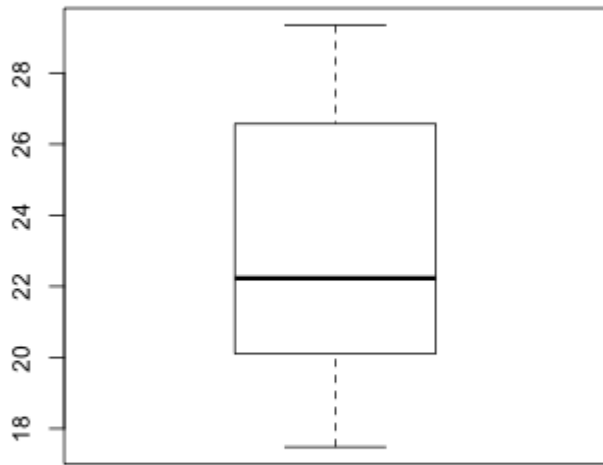
With extra features:

```
> hist(mydata$bmi, xlab = "BMI", main="BMI's of students")
```

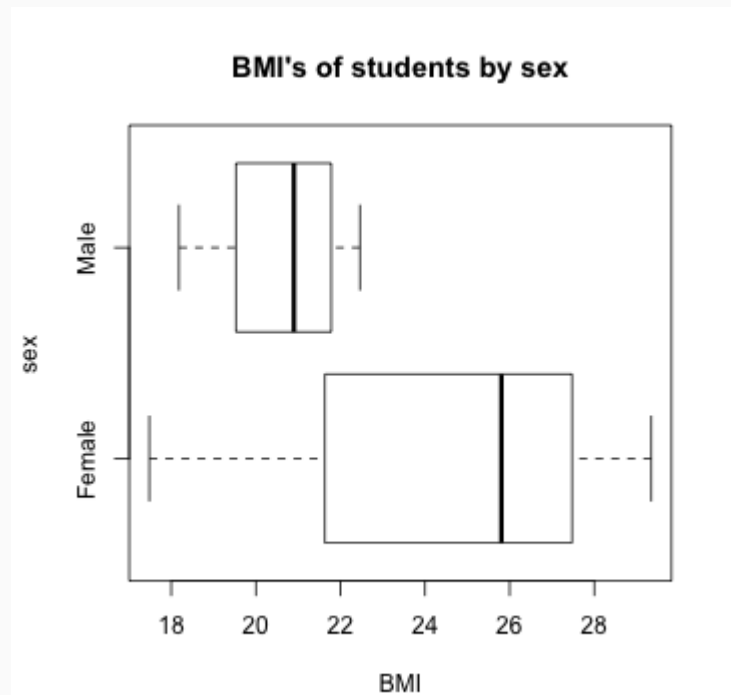
Basic plots of numeric data (2/3)

Boxplot

```
> boxplot(mydata$bmi)
```



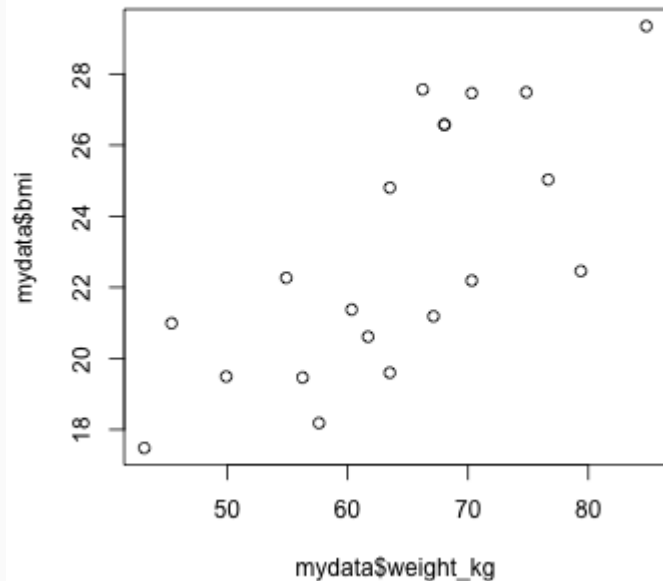
```
> boxplot(mydata$bmi ~ mydata$sex,  
+         horizontal = TRUE,  
+         xlab = "BMI", ylab = "sex",  
+         main = "BMI's of students by sex")
```



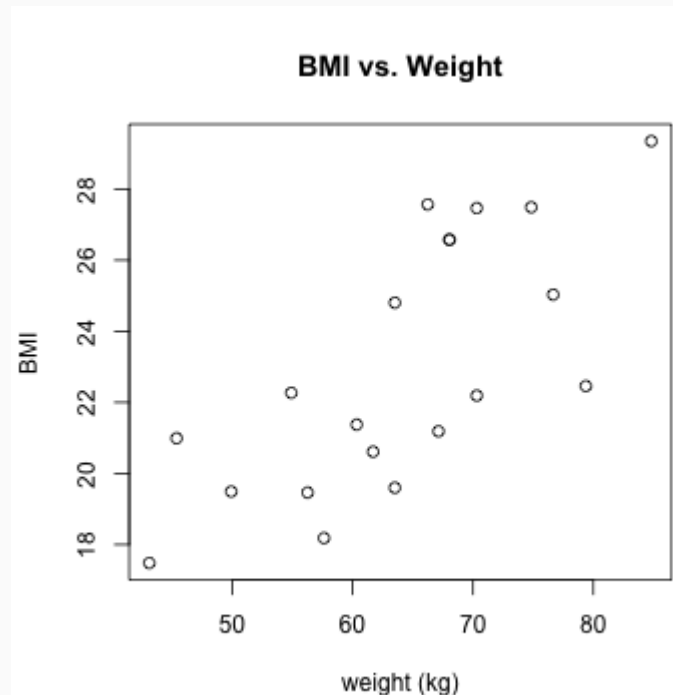
Basic plots of numeric data (3/3)

Scatterplot

```
> plot(mydata$weight_kg, mydata$bmi)
```



```
> plot(mydata$weight_kg, mydata$bmi,  
+       xlab = "weight (kg)", ylab = "BMI",  
+       main = "BMI vs. Weight")
```



Summary stats of numeric data (1/2)

Standard R `summary` command

```
> summary(mydata$bmi)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
17.48	20.36	22.23	23.01	26.58	29.35

Mean and standard deviation

```
> mean(mydata$bmi)
```

```
[1] 23.00838
```

```
> sd(mydata$bmi)
```

```
[1] 3.56471
```

Summary stats of numeric data (2/2)

Min, max, & median

```
> min(mydata$bmi)
```

```
[1] 17.4814
```

```
> max(mydata$bmi)
```

```
[1] 29.3495
```

```
> median(mydata$bmi)
```

```
[1] 22.22985
```

Quantiles

```
> quantile(mydata$bmi, prob=c(0, .25, .5, .75, 1))
```

0%	25%	50%	75%	100%
17.48140	20.35878	22.22985	26.57810	29.34950

Add height column to data frame

Since $\text{BMI} = \frac{\text{kg}}{\text{m}^2}$, we have $\text{height}(m) = \sqrt{\frac{\text{weight}(kg)}{\text{BMI}}}$

```
> mydata$height_m <- sqrt( mydata$weight_kg / mydata$bmi)
> mydata$height_m
```

```
[1] 1.550000 1.699999 1.779999 1.680001 1.799998 1.780000 1.469998
[8] 1.570002 1.879998 1.600001 1.779998 1.699999 1.730001 1.600001
[15] 1.600001 1.600000 1.750001 1.569998 1.599999 1.650001
```

```
> dim(mydata); names(mydata)
```

```
[1] 20 11
```

```
[1] "id"           "age"
[3] "sex"          "grade"
[5] "race4"        "bmi "
[7] "weight_kg"    "text_while_driving_30d"
[9] "smoked_ever"  "bullied_past_12mo"
[11] "height_m"
```

Access specific columns in data set

Previously we used `DatSetName[, column#]`

```
> mydata[, c(2, 6)] # 2nd & 6th columns
```

		age	bmi
1	17 years old	27.5671	
2	16 years old	29.3495	
3	14 years old	18.1827	
4	15 years old	21.3754	
5	15 years old	19.5988	
6	16 years old	22.1910	
7	16 years old	20.9913	
8	17 years old	17.4814	
9	15 years old	22.4593	
10	17 years old	26.5781	
11	16 years old	21.1874	
12	17 years old	19.4637	
13	17 years old	20.6121	
14	15 years old	27.4648	
15	16 years old	26.5781	
16	16 years old	24.8047	
17	16 years old	25.0318	

The code below uses *column names* instead of numbers.

```
> mydata[, c("age", "bmi")]
```

		age	bmi
1	17 years old	27.5671	
2	16 years old	29.3495	
3	14 years old	18.1827	
4	15 years old	21.3754	
5	15 years old	19.5988	
6	16 years old	22.1910	
7	16 years old	20.9913	
8	17 years old	17.4814	
9	15 years old	22.4593	
10	17 years old	26.5781	
11	16 years old	21.1874	
12	17 years old	19.4637	
13	17 years old	20.6121	
14	15 years old	27.4648	
15	16 years old	26.5781	
16	16 years old	24.8047	

Access specific rows in data set

- Rows for 14 year olds only

```
> mydata[mydata$age == "14 years old",]
```

```
      id      age  sex grade race4      bmi weight_kg
3 922382 14 years old Male   9th White 18.1827      57.61
text_while_driving_30d smoked_ever bullied_past_12mo height_m
3                <NA>          Yes                FALSE 1.779999
```

In this case the output is only one row since there is only one 14 year old.

- Rows for teens with BMI less than 19

```
> mydata[mydata$bmi < 19,]
```

```
      id      age  sex grade      race4      bmi weight_kg
3 922382 14 years old Male   9th      White 18.1827      57.61
8 935435 17 years old Female 12th All other races 17.4814      43.09
text_while_driving_30d smoked_ever bullied_past_12mo height_m
3                <NA>          Yes                FALSE 1.779999
8                <NA>          No                FALSE 1.570002
```

Access specific values in data set

- Grade and race for 15 year olds only

```
> mydata[mydata$age == "15 years old", c("age", "grade", "race4")]
```

	age	grade	race4
4	15 years old	9th	White
5	15 years old	10th	Black or African American
9	15 years old	10th	All other races
14	15 years old	10th	Hispanic/Latino

- Age, sex, and BMI for students with BMI less than 19

```
> mydata[mydata$bmi < 19, c("age", "sex", "bmi")]
```

	age	sex	bmi
3	14 years old	Male	18.1827
8	17 years old	Female	17.4814

Practice

1. Create data frames for males and females separately.
2. Do males and females have similar BMI's? Weights? Compares means, standard deviations, range, and boxplots.
3. Plot BMI vs. weight for each gender separately. Do they have similar relationships?
4. Are males or females more likely to be bullied in the past 12 months? Calculate the percentage bullied for each gender.
5. Are students that were bullied in the past year more likely to have smoked in the past? Does this vary by gender?

Save data frame

- Save **.RData** file: the standard R format, which is recommended if saving data for future use in R

```
> save(mydata, file = "mydata.RData")
```

You can load .RData files using the load() command:

```
> load("mydata.RData")
```

- Save **csv** file: comma-separated values

```
> write.csv(mydata, file = "mydata.csv", col.names = TRUE, row.names = FALSE)
```

The more you know

Installing and using packages

(Packages are to R/Rstudio like apps are to your phone/OS)

CRAN = package mothership

Comprehensive R Archive Network

Also can use the "Packages" tab in the Files/Plots/Packages/Help/Viewer window

```
> # Install a package from CRAN (main package repository)
> install.packages("tidyverse") # only do this ONCE
> # Load the package
> library(tidyverse)
```

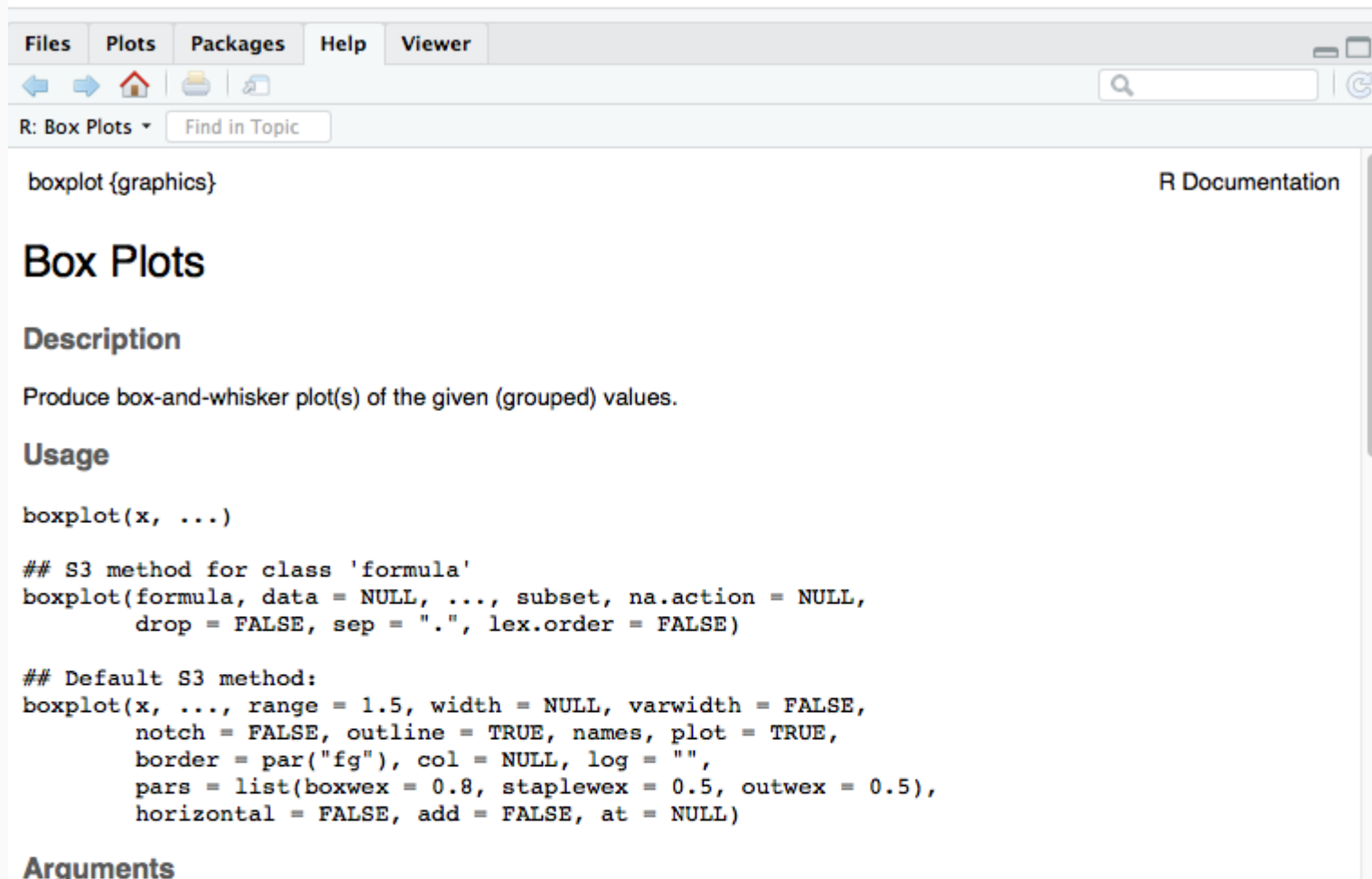
Other places (i.e. github) = wild west

```
> install.packages("devtools") # only do this ONCE
> library(devtools)
> # Install a package from github (often in development, no testing)
> # https://github.com/hadley/yrbss
> install_github("hadley/yrbss")
> library(yrbss)
```

How to get help (1/2)

Use `?` in front of function name in console. Try this:

```
> ?boxplot  
> |
```



The screenshot shows the R help window for the `boxplot` function. The window has a menu bar with 'Files', 'Plots', 'Packages', 'Help', and 'Viewer'. Below the menu bar is a toolbar with navigation icons and a search bar. The main content area displays the documentation for `boxplot`, including its description, usage, and arguments.

R: Box Plots ▾ Find in Topic

boxplot {graphics} R Documentation

Box Plots

Description

Produce box-and-whisker plot(s) of the given (grouped) values.

Usage

```
boxplot(x, ...)
```

```
## S3 method for class 'formula'  
boxplot(formula, data = NULL, ..., subset, na.action = NULL,  
        drop = FALSE, sep = ".", lex.order = FALSE)
```

```
## Default S3 method:  
boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE,  
        notch = FALSE, outline = TRUE, names, plot = TRUE,  
        border = par("fg"), col = NULL, log = "",  
        pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5),  
        horizontal = FALSE, add = FALSE, at = NULL)
```

Arguments

How to get help (2/2)

- Use `??` (i.e. `??dplyr` or `??read_csv`) for searching all documentation in installed packages (including unloaded packages)
- search [Stack Overflow #r tag](#)
- google your question + rcran or + r (i.e. "make a boxplot rcran" "make a boxplot r")
- google the error in quotes (i.e. "Evaluation error: invalid type (closure) for variable '*'")
- search [github](#) for your function name (to see examples) or error
- [Rstudio community](#)
- [twitter #rstats](#)

Resources

- [RStudio IDE Cheatsheet](#)
- [Install R/RStudio help video](#)
- [Basic Basics](#)

Interactive lessons

- [DataCamp](#)
 - [Introduction to R \(free course\)](#)
 - [Introduction to the Tidyverse](#)
 - [Intermediate R](#)

Some of this is drawn from materials in online books/lessons:

- [Intro to R/RStudio](#) by Emma Rand
- [Modern Dive](#) - An Introduction to Statistical and Data Sciences via R by Chester Ismay & Albert Kim
- [Cookbook for R](#) by Winston Chang

Local resources

- OHSU's [BioData club](#) + active slack channel
- Portland's [R user meetup group](#) + active slack channel
- [R-ladies PDX](#) meetup group
- in June in Portland, the [WNAR Annual meeting](#) (biostats conference) will have R related workshops
- in June in Redmond, the [Cascadia R conference](#) will have presentations

Possible Future Workshop Topics?

- data wrangling with the tidyverse
- reproducible reports in R
- tables
- ggplot2 visualization
- advanced tidyverse: functions, purrr
- statistical modeling in R

Contact info:

Jessica Minnier: *minnier@ohsu.edu*

Meike Niederhausen: *niederha@ohsu.edu*

This workshop info:

- Code for these slides on github: [jminnier/berd_r_courses](#)
- all the **R code in an R script**
- answers to practice problems can be found here: [html](#), [pdf](#)