BIOSTATS 640 – Introduction to R
Fall 2024
https://people.umass.edu/biep640w/webpages/demonstrations.html

Source: https://www.memedroid.com/memes/tag/coding?page=2

02
R Essentials
September 13, 2024

Right click to download R dataset
framingham.Rdata

Welcome to Lesson 02!
Dear BIOSTATS 640 Fall 2024, This is a new lesson for 2024, that was not included in my 2023 lessons. I thought that between the first lesson ("Up and Running with R") and the next lesson ("Numerical Summarization and 1 & 2 Sample Inference"), it would be convenient to have a resource of some R essentials. Fall 2024 Lesson 02, "R Essentials" is just that. You will see many of the tools introduced here in subsequent R lessons. So, no worries if this resource feels like a lot.

1. Highlights of Lesson 01 - Up and Running with R Studio

We will be doing all our work in R Studio	R Studio is an application that sits "on top" of R which is then "under the hood". The R Studio utility provides a very friendly environment for doing lots of things: writing and executing code, managing files and directories, and working with packages.
`>` is the command prompt, located in the console pane **`#` denotes a comment; R ignores the rest of the line**	
R is case sensitive	And is unforgiving!
`c( )` to create vectors of data and separate arguments by commas	**For example:** `v1 <- c(14,35,81,99)`

`dataframe`
A "dataset" (analogous to excel spreadsheet or SAS dataset or Stata dataset) is called a `dataframe` in R
	Examples:
	`df1<-data.frame(A=c(1,2,3),B=c(2,3,4))`
	
	`df2<-as.data.frame(matrix(c(1,2,3,4),`
	`nrow = 2))`

R can work with more than one dataset at a time.	
To identify a variable in a dataframe, R utilizes a two-part naming convention: **dataframename$variablename**	**For example:** `arthritis$Age`

Statistical functions run on COMPLETE data only.
Tip: use option na.rm=T to remove missings (NA)
	For example:
	
	`mean(v1, na.rm=T)`
	`mean(v1, na.rm=TRUE)` `# This also works.`

2. How R Works

2.1. What is R

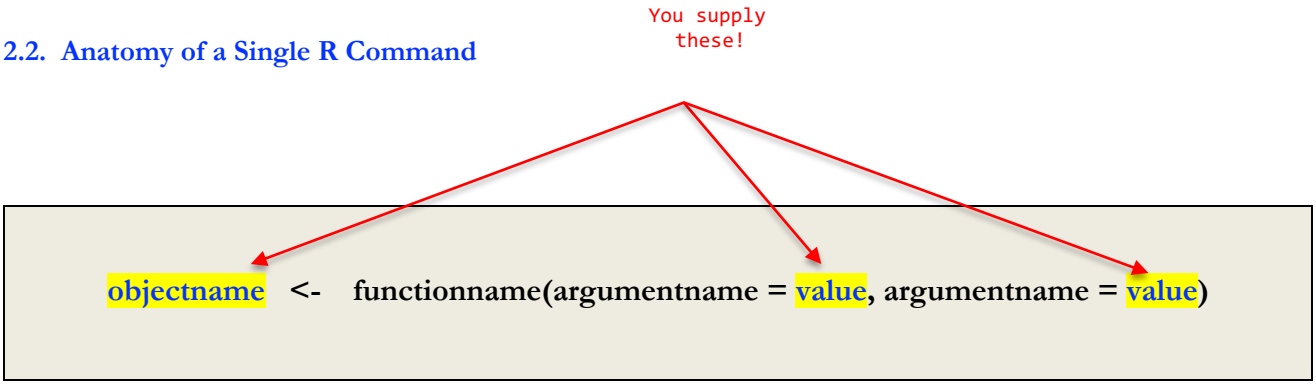
R is an object oriented program language that is ideally suited to data management, analysis, and visualization.

(Source: Zuur AF, Ieno EN, and Meesters EHWG. A Beginner's Guide to R. Springer. 2009)
"R is computer language that allows the user to program algorithms and use tools that have been programmed by others ... With R you can write functions, do calculations, apply most available statistical techniques, create simple or complicated graphs and even write your own library functions". A large user group supports it. Many rsearch institutes, companies and universities have migrated to R. .... Nearly everything you may need in terms of statsitics has already been programmed and made available in R (either as pat of the main package or as user-cintrobuted package)".

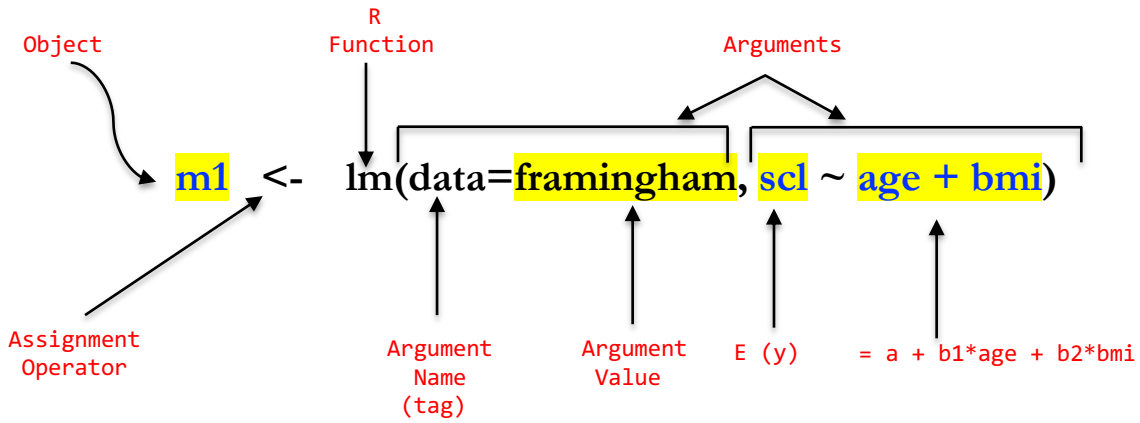
(Source: Carroll J. Beyond Spreadsheets with R. Manning. 2019)
"At its most basic level, R is a useful tool for interacting with data. It stores *values* (data) and *functions* (code that interacts with data) as *variables* (names for things) and complex *objects* (structures). In technical terms, R is an open source, intepreted, general purpose functional language:

- *source* – The underlying source code can be freely obtained and (if desired) modified.
- *Interpreted* – R doesn't require compiling your code into a standalone program. Some languages require the code to be built into an executable in order to run it.
- *General purpose* – It isn't restricted to doing just one thing in a particular domain.
- *Functional* – It uses functions on unchanging data, rather than depending on the current state of the system and modifying data in place."

2.2. Anatomy of a Single R Command



Example:
The Framingham data set (`framingham.Rdata`) has measurements of several things (data) on a large cohort of consenting individuals who have been followed for several years. The goal of the Framingham study was to identify the risk factors for cardiovascular disease. Here, a simple linear model of serum cholesterol (scl) is fit to age at baseline (age) and body mass index (bmi).



Key:

`Object` – After execution of the function (right hand side), the result is assigned to an object (left hand side)

`Assignment operator` – Wow! The assignments operator is itself a function! It stores the result in an object

`R Function` – lines of code that, when executed/called, interacts with the data according to arguments

`Arguments` – Options passed to the R function, separated by commas. Each option specifies an input to be used.

`Argument name` – R name of the input used during execution of function

`Argument value` – User supplied value of the argument to be used as the input during execution of function

2.3. Good to Know. Parentheses ( ), Brackets [ ], and Braces { }

Parentheses ( )	Brackets [ ]	Braces { }
Used in **commands** and **functions**	Used in **indexing** of items in an R object	Used in **functions** and **loops**, to hold a set of commands to be executed in the function or loop

Parentheses ( ): Used in commands and functions

- Parentheses' content tells R what to work on
- Within a parentheses, commas options
- Parentheses are also used in order of operations
- **Examples**:
  - `3 * (x + y)` is not the same as `3 * x + y`
  - ```
    nondiabetes <- subset(hersdata,diabetes=="no",
                          select=c(glucose,exercise,age,drinkany,BMI,physact))
    ```

Brackets [ ]: Used in indexing

- Structure is: [rowfirst:rowlast, columnfirst:columnlast]
- **Examples**:
  Select rows 1through 10, EVERY column: `new < - old[1:10, ]`
  Select EVERY row, but only columns 1 through 10: `new < - old[, 1:10]`
  Select EVERY row, but only some columns: `new < - old[, c("id", "age", "wt")]`

Braces { }: Used in functions and loops

- **Example**:

```
If (state="MA") {
    timezone="EDT"
   numseasons=4
}
```
```

3. Code and Execute in R Script

So far, we have done all our work in the console pane
Think of the console as the engine.
From here, commands are submitted and R executes them.
Commands in the console are NOT saved.

In practice, we do our coding in the source/editor pane and, in particular, in a R Script
An R Script file is a **plain text file** that contains: 1) R commands; and 2) comments.
When you execute commands from an R Script, they are sent to the console to be executed.

Advantages of Working with R Scripts
Saved R Scripts are a permanent records of your analysis code;
They can be recycled, edited and used again; and
They can be shared because they are text files.

Tips. Console versus R Script:

- Do not regularly work from the console pane
- Use the console pane for just TWO THINGS ONLY: (1) as a calculator and/or (2) when requesting help
- Make it a habit to do all your programming work in a saved R Script file (unless you are creating an R Markdown file)

Tips. Writing R Script Files

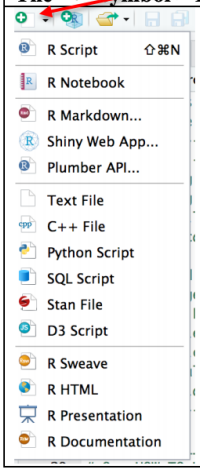
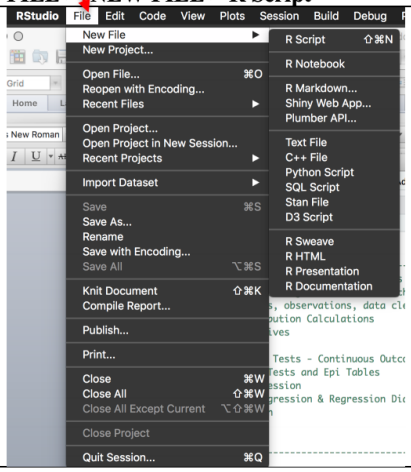
- When you create a new R script file, save it immediately to a permanent file
- Save often.
- Use lots of comments (Recall. These begin with **#**)
- Begin your R script file with a comments that describe the R Script file (title, date, name of file, etc)
- Consider using the following as your first command in your R Script file: **rm(list=ls( ))**

Tip. Clear your workspace at the start of your session using rm(list=ls( ))

- **ls( )** tells R "get every object in my workspace"
- **rm( )** tells R to remove it.
- Why this is a good idea: You don't want to mistakenly work with objects leftover from a previous session.

How to Create an R Script File

STEP 1: From the source pane, click on either (+) or do FILE > NEW FILE > R SCRIPT

The "+" symbol > R Script	FILE > NEW FILE > R Script
	

STEP 2: Immediately save it using **FILE > SAVE AS.**

How to execute commands in your R Script file

Method 1 – Execute selected commands only.
Highlight commands → At upper right, click on **RUN**

Method 2 – Execute selected commands only.
Highlight commands → Do a **<control>-enter**

Method 3 – Execute entire R Script file.
At upper right, click on **SOURCE**

All done?
Don't forget to save your R Script file before exiting R Studio

4. Set Your Working Directory

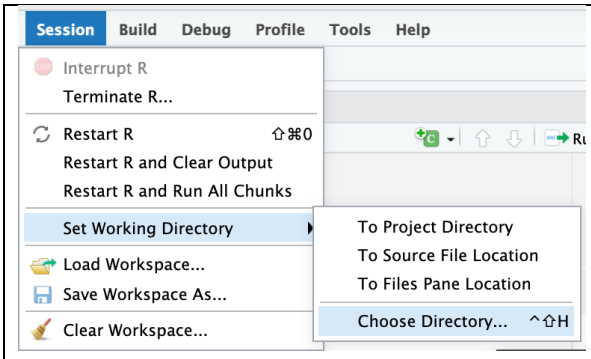
What is the working directory and why do I need to set it?

R needs to know where to find the files to **read from** and where to **write to.** This location is a directory/folder with an associated path and is known as your **working directory**.

setwd( ) - Set your working directory

getwd( ) - Show current working directory

How to Set Your Working Directory Method I - Using the R Studio/Posit Menus
From the top menu bar, click Session > Set Working Directory > Choose Directory
Browse to navigate to your desired folder. Click **CHOOSE.**



How to Set Your Working Directory Method II - Using the `setwd( )` function in the console
IMPORTANT. The path name must be enclosed in quotes.
Example (Windows): `setwd("My Documents/BIOSTATS 640/homeworks")`
Example (Mac): `setwd("~Desktop/BIOSTATS 640/homeworks")`

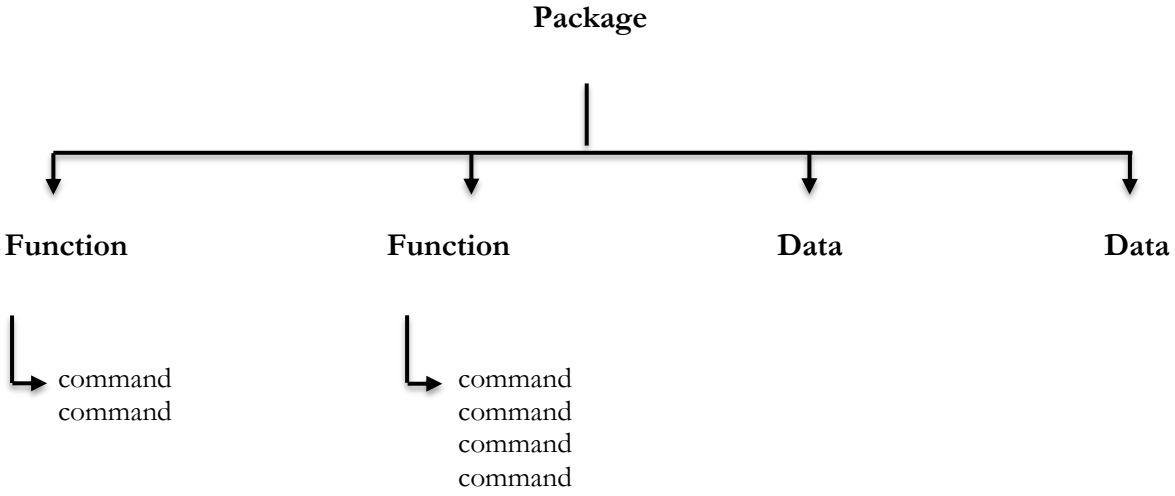
How to Show Your Current Working Directory – Use the command `getwd( )`

```
getwd( )                    # Yes. The parentheses are left empty
```

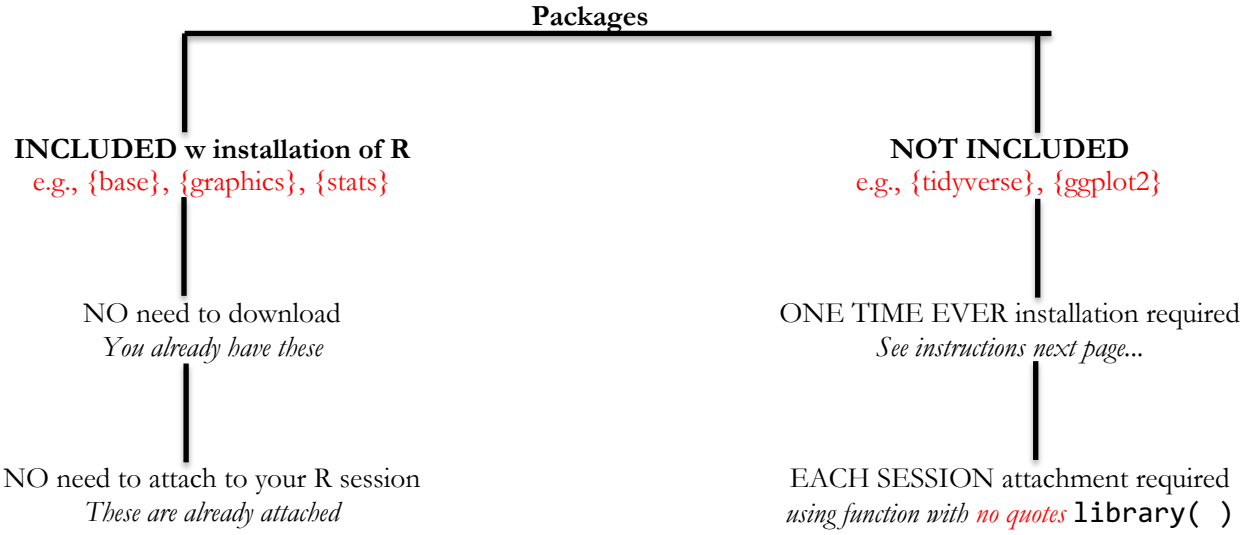
5. Introduction to Packages

Why do I need to know about packages?
Your installation of R came with some pre-installed commands that are contained within pre-installed packages. Lots of them. However, very often (and I mean VERY OFTEN), you will want to make use of commands and features of packages that are not pre-installed. To do so, you need to: 1) download and install the package (one time); and 2) attach the package to your R Studio session (one time/session).

A **package** is a collection of functions and datasets



There are two types of packages: included with installation and not included with installation.



How to Install a Package *(when you need to)*.

Install – One time!
Load – Each session: Loading a package is the attachment of that package to your R session as needed

	Packages Included with Base Installation	**Packages NOT included with Base Installation**
How to Install (one time)	*Not necessary* *You already have these packages installed*	Method 1 (issue command in console): `install.packages("PACKAGENAME")` **Example:** `install.packages("ggplot2")` Note. MUST enclose in quotes Method 2 (use drop down menu): Environment pane ➡ Packages tab ➡ Install Check "Include dependencies"
How to Load/Attach (each session)	`library(PACKAGENAME)` **Example:** `library(ggplot2)` Note. NO Quotes	`library(PACKAGENAME)` **Example:** `library(ggplot2)` Note. NO Quotes

6. A Brief Introduction to {tidyverse} and {dplyr}

6.1. What is tidyverse?

At the time of writing (9/13/2024), **tidyverse** is a bundle of 8 "core" packages and 11 "additional" packages. Quite possibly, additional packages will be added to **tidyverse** in the future

tidyverse includes 8 core packages + 11 additional packages.

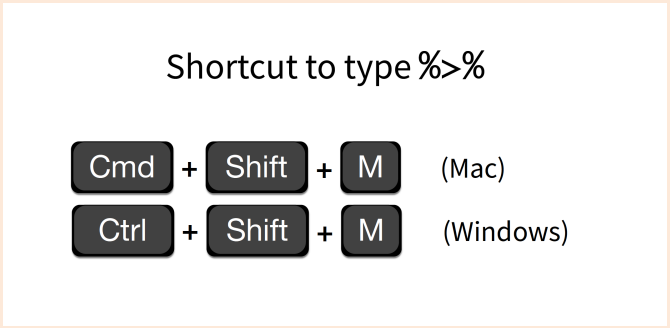
To use any of the 8 core packages:
`library(tidyverse)` one time to simultaneously attach all of 8 core packages.

Package	Use for:
ggplot2	Data visualization
dplyr	Data manipulation
tidyr	Data cleaning
readr	Import/export rectangular data (e.g. ".xlsx", etc)
purr	Working with functions and vectors
tibble	Working with a reimagined dataframe
stringr	Working with strings
forcats	Working with factors

To use any of the 11 additional packages:
`library(packagename)` separately to attach each of these 11 packages.

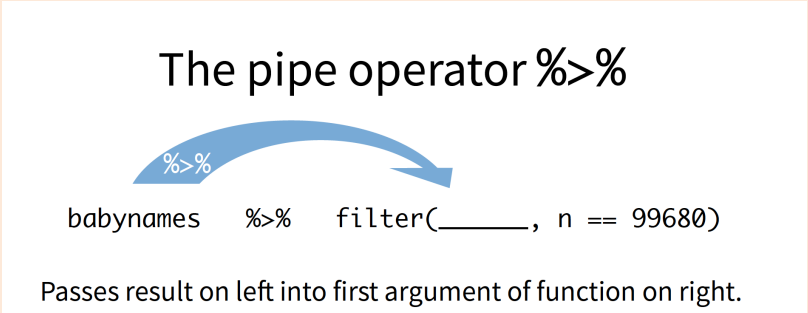
Package name	Use for:
haven	Importing SAS, SPSS, Stata
readxl	Importing .xls and .xlsx
lubridate	Dates and times
hms	Times
feather	Sharing with Python and other languages
httr	Working with web apis
jsonlite	Working with JSON
rvest	Web scraping
modelr	Modeling
broom	Modeling
Xml2	XML

6.2. Good to Know. The pipe operator %>%



The pipe operator %>% is a nifty way to create a chain of connected commands in an easy to read flow

The output of the function on the left is pipelined and becomes the input to the first argument on the right.



Source: Garrett Grolemund, R Studio

Example. Using the iris dataset, select the first 100 observations and produce a frequency table of Species

```
library(tidyverse)

iris %>%                          # start with dataframe iris.  THEN
    select(Species) %>%             # select variable (column)  THEN
    slice(1:100) %>%                  # select observations (rows) by position THEN
    table()                             # construct frequency table of counts and show

    setosa versicolor  virginica
        50         50          0
```

6.3. Introduction to `dplyr`

The **`dplyr`** package is one of the core component packages that comes bundled with **`tidyverse.`** **`dplyr`** is great for working with variables, observations, or combinations of variables and observations

`dplyr` commands for **observations** (ROWS):

`%>%`	Pipe operator (translation: "Then")
`slice( )`	Use to select observations by their position (rows)
`filter( )`	Use to select observations if they meet criteria defined by variables (rows)
`arrange( )`	Use to sort the observations (rows)

`dplyr` commands for **variables** (COLUMNS)

`%>%`	Pipe operator (translation: "Then")
`select( )`	Use to select variables (columns)
`rename( )`	Use to rename a variable (column)
`mutate( )`	Use to create new variable(column)
`relocate( )`	Use to rearrange the order of the columns

Example. Using the iris dataset, obtain the mean of Sepal.Length for subset of the data for which Species is setosa

```
# Note - the dataset iris came pre-installed with your installation of R

library(tidyverse)

iris %>%                            # DATASET: start with iris.  THEN
    filter(Species=="setosa") %>%     # ROWS: select observations that meet criterion THEN
    select(Sepal.Length) %>%           # COLUMNS: select variable THEN
    summarize(mean_length = mean(Sepal.Length,na.rm=TRUE))        # compute mean, save, and show

mean_length
[1]    5.006
```

7. Working with Observations - The Basics

Working with observations can be done using in at least two ways, EITHER:

1) Using commands in the **{base}** package that came pre-installed when you installed R and RStudio; or

2) using commands in the **{dplyr}** package in **{tidyverse}.**

If you are new to R, you may find it easier to use commands in the **{dplyr}** package in **{tidyverse}.** This approach has the added advantage that your code will be easier to read.

Recall. Comparison Operators

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Exactly equal to (Tip – the double equal sign is used for operations on conditions, not a calculation)
!=	Not equal to
!x	Not x
x|y	x or y
x&y	x and y
x%in%y	This is a logical operator for whether x matches y or not. If x matches y, then R returns TRUE If x does NOT match y, then R returns FALSE

Using {BASE}

Number observations `newvariable <- seq(from=1, to=nrow(dataframename,by=1)`	`# Example` `lung_demo$studyid <- seq(from=1, to=nrow(lung_demo), by=1)`
Select observations that meet condition `newdataframe <- subset(originaldataframe, condition)`	`# Example` `lung_demonew <- subset(lung_demo, area==1)`
Select observations and keep selected variables only `newdataframe <- subset(originaldataframe, condition, select=c(varname,varname))`	`# Example` `lung_demonew <- subset(lung_demo, area==1, select=c(id,area,mfvc))`
Draw a random sample of # observations, keep all variables `newdata <- original[sample(nrow((original), #, ]`	`# Select a random sample of 15 observations. Keep all variables` `lung_new <- lung_demo[sample(nrow(lung_demo), 15), ]`

Draw a random sample of # observations and keep selected variables only
`newdata <- original[sample(nrow((original),#,c("var1", "var2")]`
	`# Select a random sample of 15 observations, selected variables only`
	
	`# Example - Method 1:`
	`lung_demonew <- lung_demo[sample(nrow(lung_demo), 15),c("id", "area", "mfvc") ]`
	
	`# Example - Method 2 (sometimes clearer!)`
	`myvars <- c("id", "area", "mfvc")`
	`lung_demonew <- lung_demo[sample(nrow(lung_demo), 15), myvars ]`

Using {tidyverse}

Be sure you have attached {tidyverse} to your session using **library(tidyverse)**

Number observations from 1 to n dataframe %>% mutate(newvariable = row_number())	# Example lung_demo %>% mutate(studyid = row_number())

Select observations that meet condition
newdata <- sourcedata %>%
filter(condition)
	# Example
	lung_demonew <- lung_data %>%
	filter(lung_demo,area %in% c(1,2))

Select observations and keep selected variables only
newdata <- sourcedata %>%
filter(condition) %>%
select(varname, varname)
	# Example
	lung_demonew <- lung_data %>%
	filter(lung_demo,area %in% c(1,2) %>%
	select(id, area, mfvc)

Draw a random sample of # observations and keep selected variables only
newdata <- sourcedata %>%
sample_n(#) %>%
select(varname, varname)
	# Select a random sample of 15 observations. Keep
	# selected variables only
	lung_demonew <- lung_data %>%
	sample_n(lung_demo,15) %>%
	select(id, area, mfvc)

Draw a random sample that is a <u>percent</u> of the source data, selected variables only
newdata <- sourcedata %>%
sample_frac(%) %>%
select=c(varname,varname))
	# Select a 20% random sample, selected variables
	# only
	lung_demonew <- lung_data %>%
	sample_frac(.20) %>%
	select(id,area,mfvc)

8. Working with Variables - The Basics

Just as with working with observations, working with variables can be done EITHER:

1) Using commands in the **{base}** package that came pre-installed when you installed R and RStudio; or

2) using commands in the **{dplyr}** package in **{tidyverse}.**

Recall. Mathematical Functions

+	Addition
-	Subtraction
*****	Multiplication
/	Division
^	Exponentiation (raising to a power)
%/%	Integer part of division or quotient
%%	Remainder part of division or quotient
log()	logarithm to base e ("natural log"), also known as ln( )
log10()	Logarithm to base 10
exp()	Exponentiation of the constant e = 2.718 … (approx.)
sqrt()	Square root of
round(x,n)	Round x to the nth digit

Using {BASE}

Tip - When creating new variables, take care to handle missing values explicitly.
For example, creating a character variable involves multiple steps: 1) initializing to missing (NA); 2) coding the conditions *in square brackets* that define each level of your new character variable; and 3) remembering to use quotes (except for NA).

Create a Character Variable

```
dataframe$newvariable <- NA                                                    # Initialize to missing NA
dataframe$newvariable[condition_that_must_be_true] <- "charactervalue"
dataframe$newvariable[condition_that_must_be_true] <- "charactervalue"
etc.
```

```
# Example: Grouping age values to create a character variable
ivf$ageg <- NA                                                          # Initialize to missing
ivf$ageg[ivf$matage==23] <- "Youngest"
ivf$ageg[(ivf$matage>23) & (ivf$matage <30)] <- "Twenty Something"      # character values in quotes.
ivf$ageg[(ivf$matage>30) & (ivf$matage <43)] <- "Middles"
ivf$ageg[ivf$matage==43] <- "Oldest"
```

Create a Factor Variable

```
Create Factor From Character
dataframe$newvar <- factor(sourcedata$sourcevar,          # sourcevar is character
                      levels=c("string1", "string2"),         # source character values are strings
                      labels=c("level1","level2"))            # new factor level values 1="No", 1="Yes"
Create Factor From Numeric
dataframe$newvar <- factor(sourcedata$sourcevar,          # source var is numeric
                       levels=c(number1, number2),             # original numeric values
                       labels=c("level1","level2"))            # new factor level values 1="No", 1="Yes"
```

```
# Example:  Create Factor From Character
mydata$diabetesf <- factor(mydata$diabetes,               # source var diabetes is character
                    levels=c("no", "yes"),                    # source character values
                    labels=c("No","Yes"))                     # new factor level values 1="No", 1="Yes"

# Example:  Create Factor from From Numeric
mydata$exposedf <- factor(mydata$exposed,                 # original var exposed is numeric
                    levels=c(0,1),                            # original numeric
                    labels=c("Not Exposed","Exposed"))        # new factor level values
```

Create a numeric 0/1 Indicator Variable, 2 ways

```
dataframe$new_01var <- with(data=dataframe, ifelse(CONDITION,1,0) na.rm=TRUE)    # Method 1

dataframe$new_01var <- as.numeric(CONDITION, na.rm=TRUE)                         # Method 2
```

```
# Example
ivf$female01 <- with(data=ivf, ifelse(sex=="female",1,0) na.rm=TRUE)             # Method 1

# Example
ivf$female01 <- as.numeric(ivf$sex=="female", na.rm=TRUE)                        # Method 2
```

Using {tidyverse}

Be sure you have attached {tidyverse} to your session using **library(tidyverse)**

Create a Factor Variable

```
Create Factor From Character
dataframe %>%
       mutate(newvar = factor(sourcevar),            # source var is character
       levels = c("string1", "string2", "string3"),     # original character string values
       labels = c("label1", "level2", "level3"))        # new factor level labels

Create Factor From Numeric
dataframe %>%
        mutate(newvar = case_when(
                            (condition) ~ “factor value”,
                            (condition) ~ “factor value”),
                       na.rm=TRUE)
```

```
# Example:  From Character to Factor
mydata %>%
      mutate(diabetesf = factor(mydata$diabetes,              # source var diabetes is character
                    levels=c("no", "yes"),                    # original character values
                    labels=c("No","Yes"))                     # new factor level values 1="No", 1="Yes"

# Example:  From Numeric to Factor
ivf %>%
    mutate(age_group = case_when(
                          ( (age >= 18) & (age < 65) ) ~ “Young”,
                                          (age >= 65) ~ “Old” ), na.rm=TRUE) )
```

Create a numeric 0/1 Indicator Variable

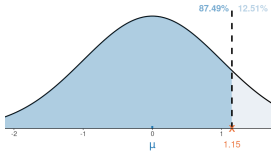
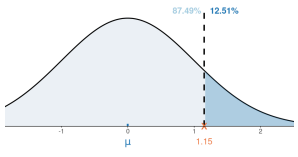
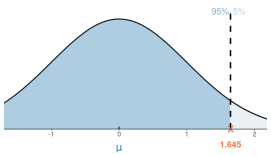
```
dataframe %>%
          mutate(new01var = ifelse(CONDITION,1,0) na.rm=TRUE)                  # Method 1

dataframe %>%
          mutate(new01var = as.numeric(CONDITION), na.rm=TRUE)                  # Method 2
```

```
# Example
ivf %>%
    mutate(female01 = ifelse(sex=="female",1,0), na.rm=TRUE)            # Method 1

# Example
ivf %>%
   mutate(female01 = as.numeric(ivf$sex=="female", na.rm=TRUE))                      # Method 2
```

9. Probability Distribution Calculations

	Density (d) Pr [X = x]	Left tail (p) Pr[X <= x]	Right tail Pr [X >= x]	Quantile (q) (e.g. - 95th)
				
Binomial(n=20, p=.20)	Pr[X=3] is dbinom(x=3,20,.20)	Pr X <= 3] is pbinom(x=3,20,.20)	Pr [X >= 3] is 1 – pbinom(x=2,20,.20)	95th percentile is qbinom(p=.95,20,.20) To obtain 2.5th and 97.5th: quantiles <- c(0.025,0.975) qbinom(quantiles,20.20)
Poisson Distribution mean = 2	Pr[X=3] is dpois(lambda=2, x=3)	Pr [X <= 3] is ppois(lambda=2, x=3)	Pr [X >= 3] is 1 - ppois(lambda=2, x=2)	95th percentile is qpois(.95,lambda=2)
Normal Distribution mean = 100, sd=15	-	Pr [X <= 87] is pnorm(87,mean=100, sd=15)	Pr [X >= 87] is pnorm(87,mean=100, sd=15, lower.tail=F)	95th percentile is qnorm(.95,mean=100, sd=15)
Student-t Distribution, df=13	-	Pr [T <= 1.57] is pt(1.57,df=13)	Pr [T >= 1.57] is pt(1.57, df=13, lower.tail=F)	95th percentile is qt(.95,df=13)
Chi Square Distribution, df=13	-	Pr [Y <= 1.57] is pchisq(1.57,df=13)	Pr [Y >= 1.57] is pchisq(1.57, df=13, lower.tail=F)	95th percentile is qchisq(.95,df=13)
F Distribution df1=7 and df2=13	-	Pr [F <= 1.57] is pf(1.57, df1=7,df2=13)	Pr [F >= 1.57] is pf(1.57, df1=7, df2=13, lower.tail=F)	95th percentile is qf(.95,df1=7,df2=13)