Unit 4
Introduction to R and R Studio

"Data! Data! Data! I can't make bricks without clay."
- Sherlock Holmes

Why R?

Yes, there are reasons to cringe. The learning curve is steep and there exist already many other powerful data management and statistical analysis software with delightfully straightforward graphical user interface (GUI) systems (drop-down menus!). Examples are SAS, Stata, SPSS, etc. Learning R can also be frustrating when you cannot solve a simple programming task and the hours just tick on by.

Notwithstanding, there are many compelling reasons to learn R and R Studio. For one, it is increasingly the "industry standard" for all kinds of data-related activities: data management, statistical programming, statistical analyses, and data visualizations. R and R Studio enjoy a huge community of developers and maintainers, ensuring that you are always in the good company of experts as you make your way. Not unique to R and R Studio, but still an important feature, are the many utilities that ensure the production of transparent and reproducible research. Last but not least, it is free and open source.

We will be doing all our work R Studio.

R Studio sits "on top" of R. It is an "overlay" environment, within which R commands are written by you and then executed by R in the background. The R Studio "overlay" environment is an example of what is called an *integrated development environment (IDE)*.

Packages Used (one-time installation):

```
- stargazer
- summarytools
- Hmisc
- gmodels
- psych
- DescTools
- tidyverse
- gtools
```

Design ..... **Data Collection** ..... **Data Management** ..... **Data Summarization** ..... **Statistical Analysis** ..... **Reporting**

Table of Contents

- continued -

Table of Contents - *continued*

Learning Objectives

When you have finished this unit, you should be able to:

- Launch and exit R Studio, and obtain help**;**
- Use the R Studio console as a calculator;
- Explain the nature and utility of R objects;
- Create R Objects from "scratch";
- Load excel data into R Studio and save as an ".Rdata" dataset file;
- Create variable label and variable value labels;
- Create new variables;
- Assign missing value codes;
- Select subsets of observations for analysis;
- Screen a data set for a variety of "errors"; and
- Create **R Script and R Markdown** files.

1. Getting Started

1.1. Downloading and Installing R and R Studio

R and R Studio are separate packages!
You will need to install R first.

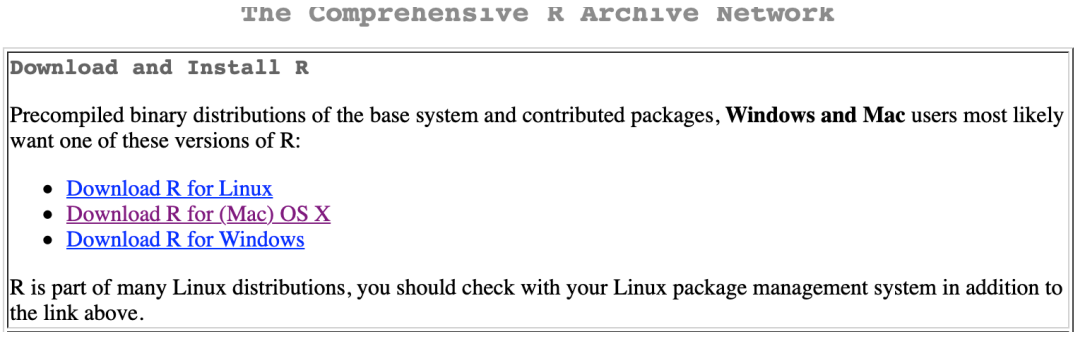
Windows Users *(v. Fall 2020)*

Before you begin:
Consider watching a video (duration, 3:23) here:
https://www.youtube.com/watch?v=VLWaED9jTiA

PART I – Download and Install R

___1. Go to https://www.r-project.org/
In "Getting Started", **Click on "Download R" or "CRAN mirror"**.

___2. Under **"CRAN mirrors"**, choose one of the URLs from **"USA"**.
Under **"The Comprehensive R Archive Network"**,
in the box "Download and Install R",
click on "Download R for Windows."


The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

___3. Under "R for Windows", **click on "install R for the first time".**

___4. Under R-4.0.2 for Windows (32/64 bit), **click on "Download R 4.0.2 for Windows"**

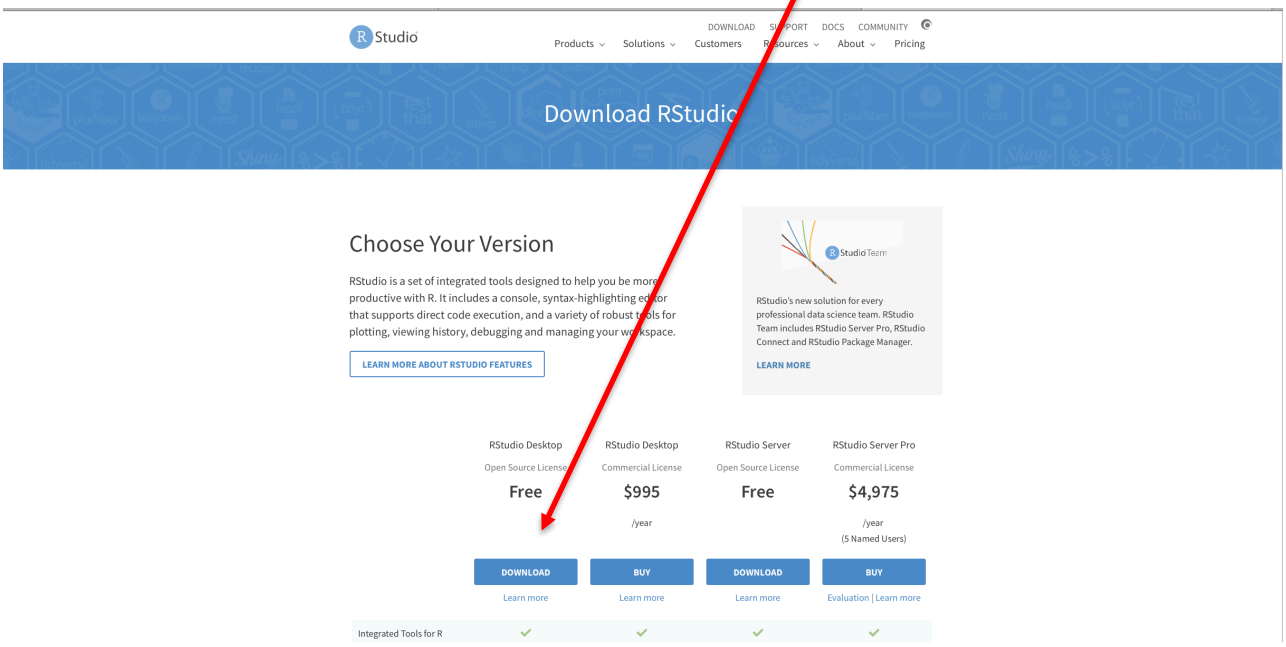
R-4.0.2 for Windows (32/64 bit)

Download R 4.0.2 for Windows (84 megabytes, 32/64 bit)
Installation and other instructions
New features in this version

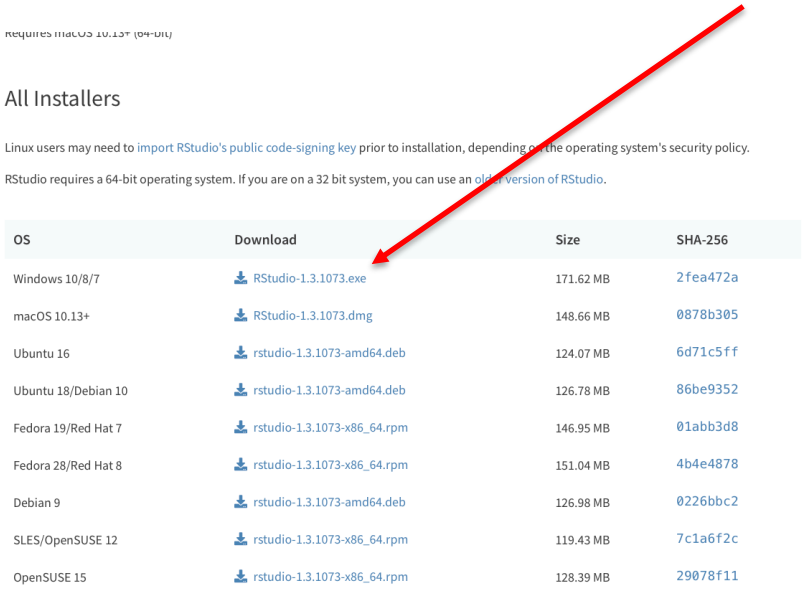
Notes -
* At the time of writing, 4.0.2 is the current version. However, you should use whatever version shows up as the most recent.

PART II – Download and Install R-Studio

___1. Go to https://rstudio.com/products/rstudio/download/
___2. At left, choose version RStudio Desktop. It is free. Click on DOWNLOAD



__3. At left, under Installers, click on the download corresponding to Windows 10/8/7



All Installers

Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.

RStudio requires a 64-bit operating system. If you are on a 32 bit system, you can use an older version of RStudio.

OS	Download	Size	SHA-256
Windows 10/8/7	RStudio-1.3.1073.exe	171.62 MB	2fea472a
macOS 10.13+	RStudio-1.3.1073.dmg	148.66 MB	0878b305
Ubuntu 16	rstudio-1.3.1073-amd64.deb	124.07 MB	6d71c5ff
Ubuntu 18/Debian 10	rstudio-1.3.1073-amd64.deb	126.78 MB	86be9352
Fedora 19/Red Hat 7	rstudio-1.3.1073-x86_64.rpm	146.95 MB	01abb3d8
Fedora 28/Red Hat 8	rstudio-1.3.1073-x86_64.rpm	151.04 MB	4b4e4878
Debian 9	rstudio-1.3.1073-amd64.deb	126.98 MB	0226bbc2
SLES/OpenSUSE 12	rstudio-1.3.1073-x86_64.rpm	119.43 MB	7c1a6f2c
OpenSUSE 15	rstudio-1.3.1073-x86_64.rpm	128.39 MB	29078f11

___4. Put the R-Studio shortcut onto your desktop

* From the **`Start`** menu:
* Scroll to find the shortcut for **RStudio**
* Click and hold on that shortcut and drag it to the desktop.

MAC Users *(v. Fall 2020)*

Before you begin.
Consider watching a video (duration, 3:01), here:
https://www.youtube.com/watch?v=EmZqlcKkJMM

Preliminary – Determine the version of your operating system *(E.g. – I have MAC OS Catalina)*
Why? If your operating system happens to be an older one, then in installing R you may have to choose an older version of R because your operating system may or may not be able to accommodate the most recent version of R.
How: From the top tool bar, just below the apple, **click About This Mac**

PART I – Download and Install R. Also (recommended) download and install XQuartz

___1. Go to https://www.r-project.org/
In "Getting Started", **Click on "Download R" or "CRAN mirror"**.

___2. Under **"CRAN mirrors"**, choose one of the URLs from **"USA"**.
Under **"The Comprehensive R Archive Network"**,
in the box "Download and Install R", **click on "Download R for (MAC) OS X"** .

___3. Under "R for MAC OS X", take care to read the first paragraph

___4. !!!! Under the heading "Latest release", check that the R package is suitable for your operating system. For most of you, the most recent release will be fine. But if you have an older MAC operating system, you may need to scroll down to find the appropriate R package for your operating system. **Click to download**.

___5. Install **XQuartz** by clicking on the link and downloading and installing the dmg file.

Latest release:

R-4.0.2.pkg (notarized and signed)
SHA1-hash: 7e4e1f0d407ccd475eeaeadd96a126ee9c83db3b
(ca. 84MB)

R 4.0.2 binary for macOS 10.13 (High Sierra) and higher, signed and notarized package. Contains R 4.0.0 framework, R.app GUI 1.72 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 6.7. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

Note: the use of X11 (including `tcltk`) requires XQuartz to be installed since it is no longer part of OS X. Always re-install XQuartz when upgrading your macOS to a new major version.

Important: this release uses Xcode 10.1 and GNU Fortran 8.2. If you wish to compile R packages from sources, you will need to download and GNU Fortran 8.2 - see the tools directory.

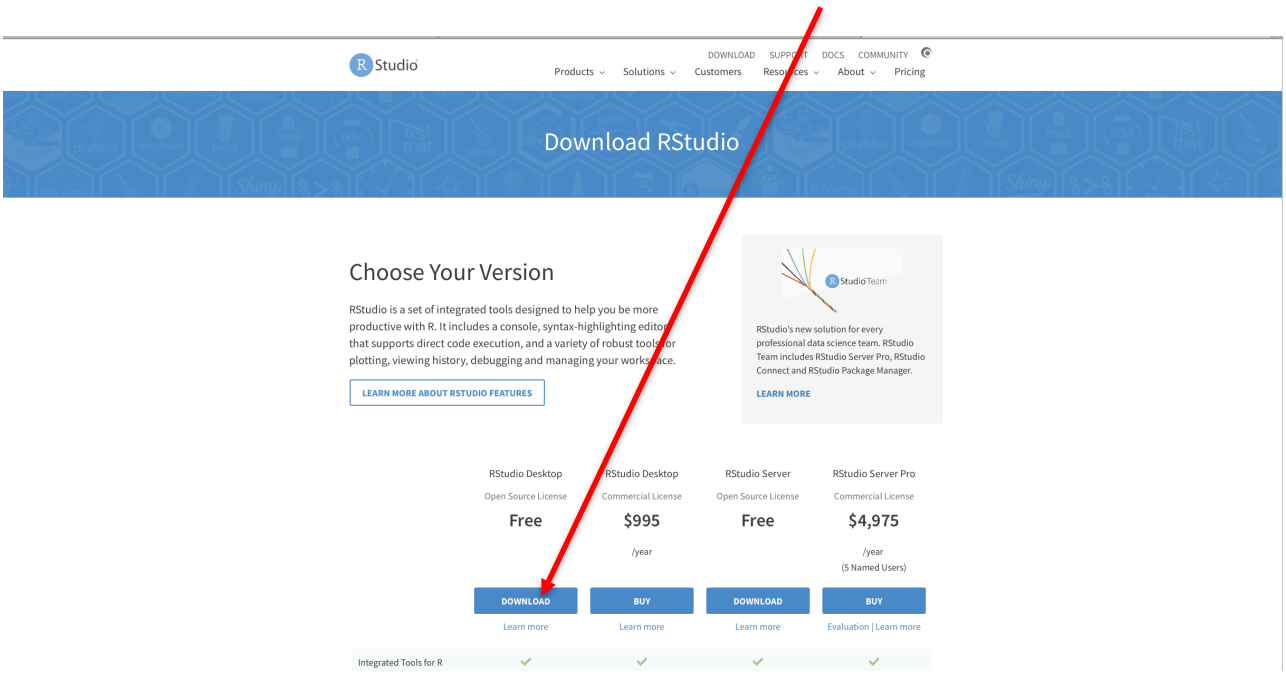
All done? At this point you should see the R icon on your dock:

Design ..... Data Collection ..... Data Management ..... Data Summarization ..... Statistical Analysis ..... Reporting

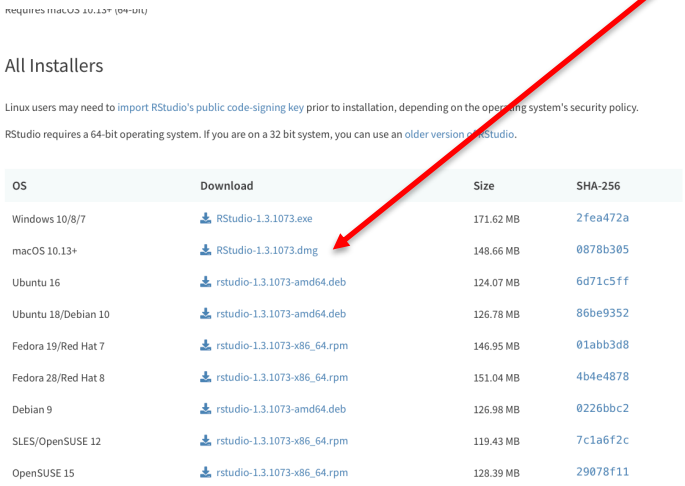
PART II – Download and Install R-Studio

___1. Go to https://rstudio.com/products/rstudio/download/

___2. At left, choose version RStudio Desktop. It is free. Click on DOWNLOAD



__3. At left, under Installers, click on the download corresponding to your MAC operating system



Put the R-Studio shortcut onto your dock, if it is not already there

* From the Applications folder:
* Scroll to find the icon for **RStudio**
* Click and drag it to your dock.

1.2. Launching and Exiting R Studio

We will be doing all our work in R Studio ONLY!	
R Studio Shortcut	**R Shortcut**
	

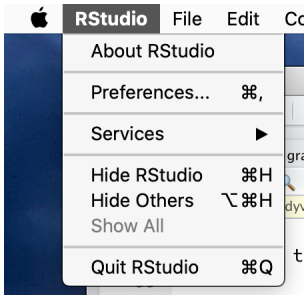
How to Launch R STUDIO

Note: Launching R Studio accomplishes launching BOTH R Studio and exiting R.

Simply click on the R-Studio icon

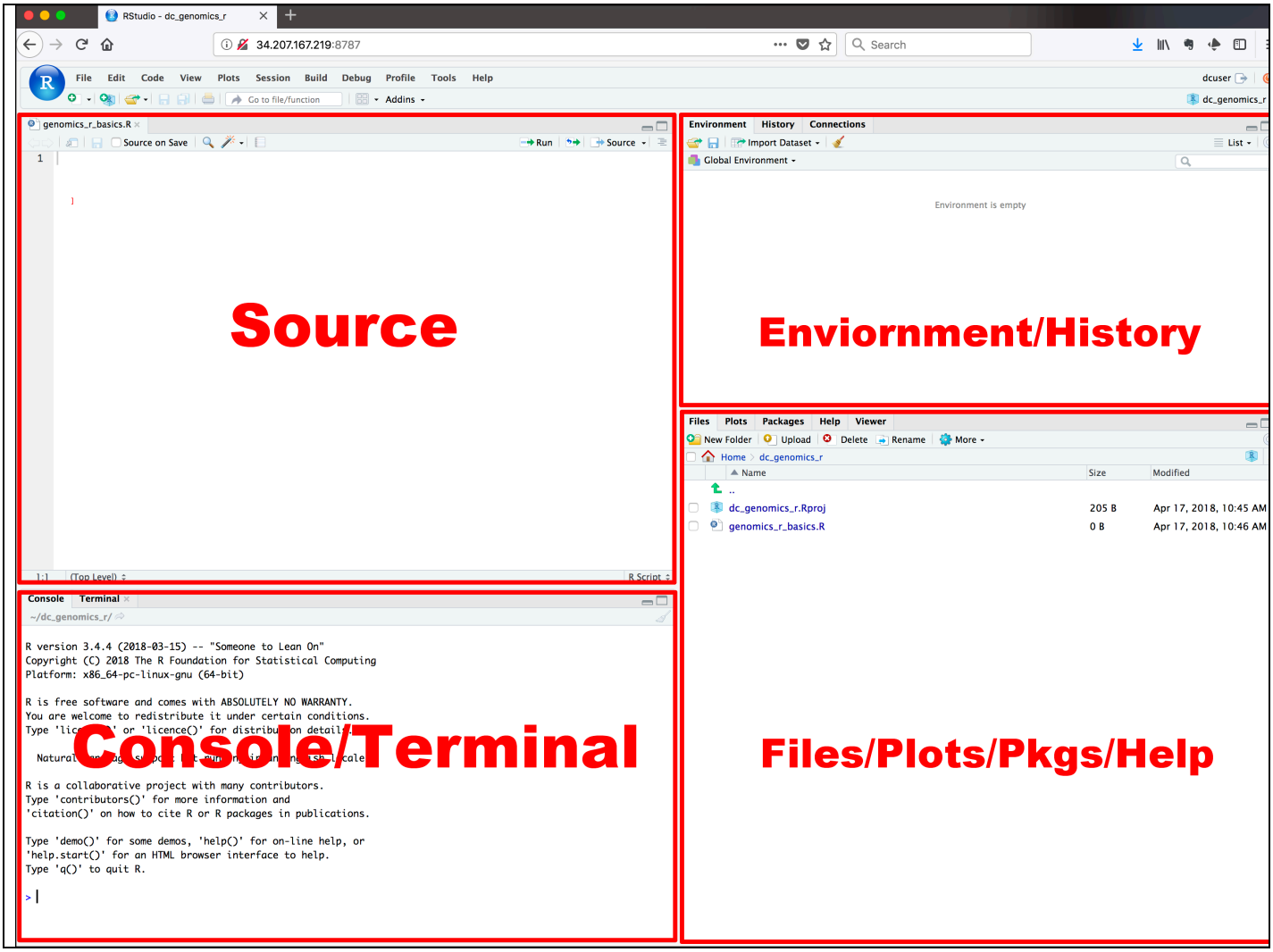
How to Exit R STUDIO

Note: Exiting R Studio accomplishes exiting BOTH R Studio and exiting R.



1.3. Get Acquainted with the R Studio Interface

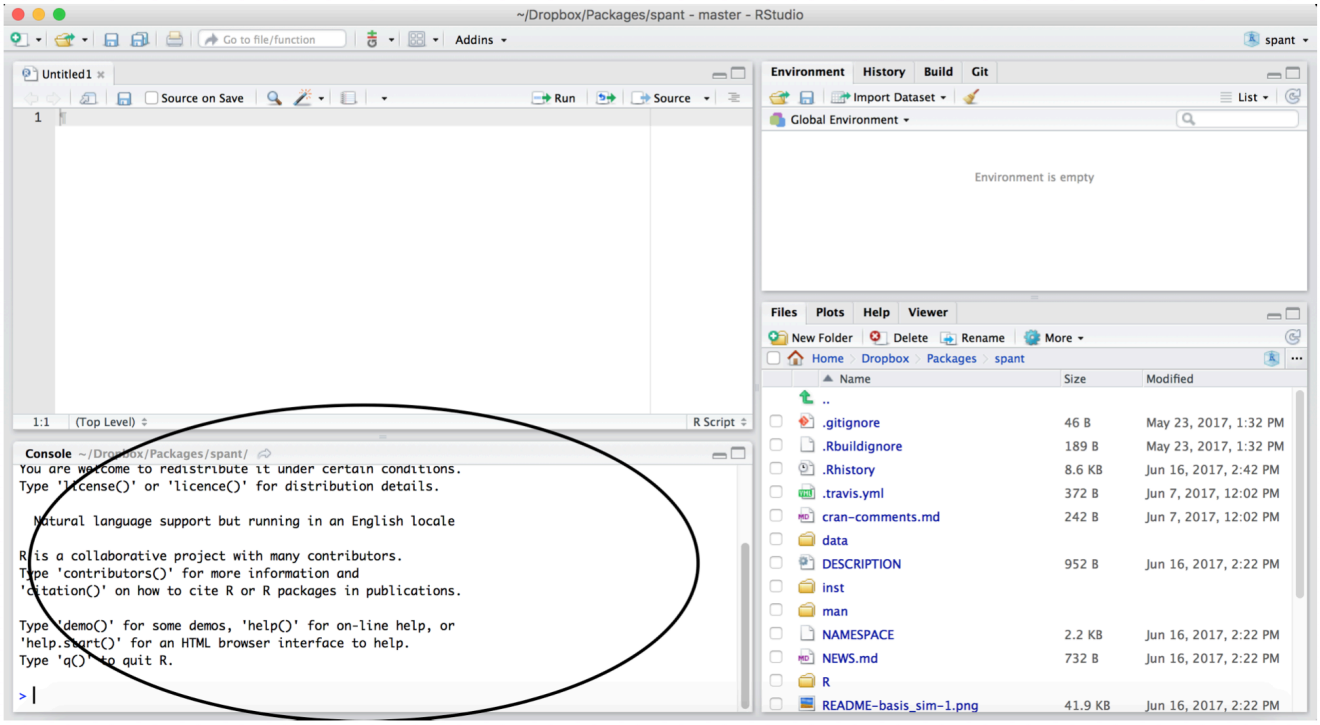
Before you begin.
Consider visiting this introduction, here:
https://ismayc.github.io/rbasics-book/3-rstudiobasics.html



(Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fdatacarpentry.org%2Fgenomics-r-intro%2F01-introduction%2Findex.html&psig=AOvVaw1hPui5N1bGL4CtSNUvA5Qi&ust=1600449355622000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCPiHvdLY8OsCFQAAAAAdAAAAABAs)

Design ..... Data Collection ..... Data Management ..... Data Summarization ..... Statistical Analysis ..... Reporting

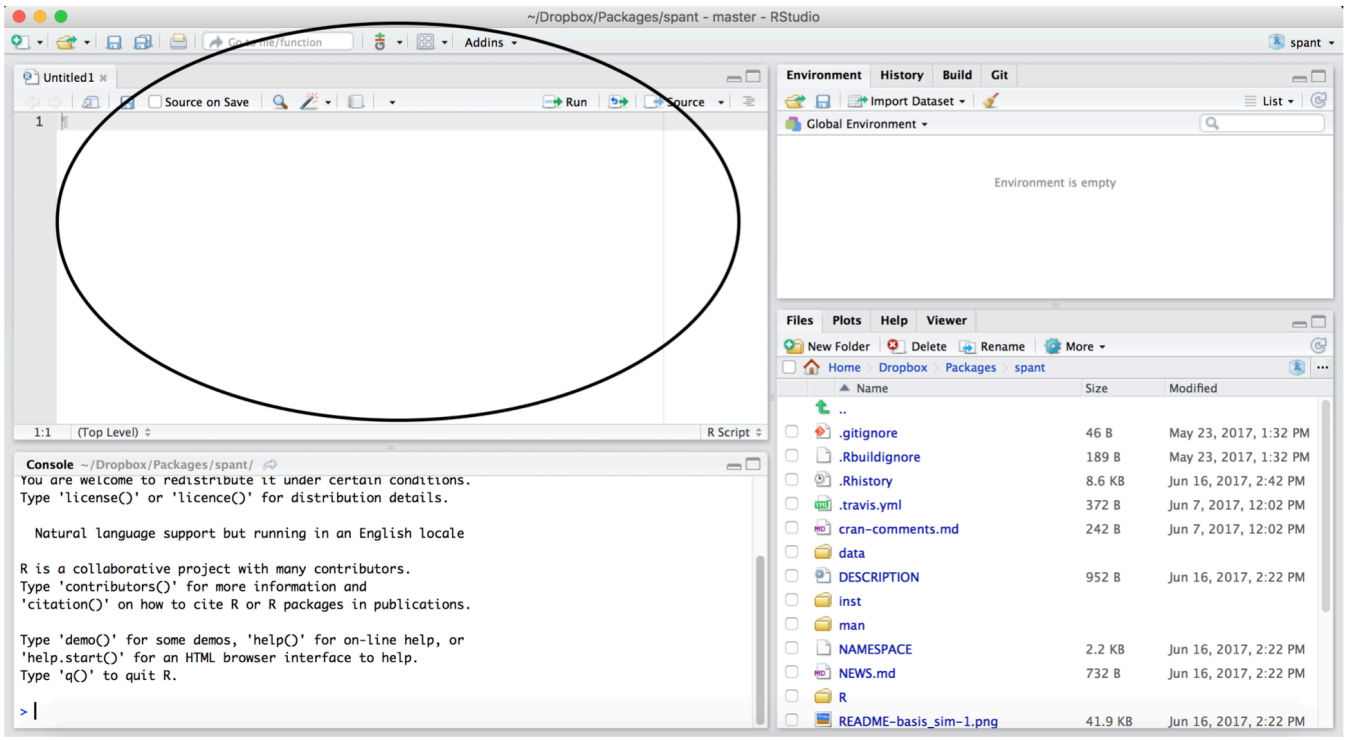
RStudio/R Console



(Source: https://johnmuschelli.com/intro_to_r/RStudio/RStudio.pdf)

- Typically located: **lower left** (see below for moving panes around)
- This is where code is executed (R Script and R Markdown will send code to here)
- Here you type code directly. R then executes it.
- The prompt is indicated by a "**>**"
- HACK: Code entered into the console is NOT SAVED
- HACK: To retrieve a previous command (*handy for editing!*): UP-arrow
- HACK: To clear contents of the console window: <control-l> *this is the letter "el"*

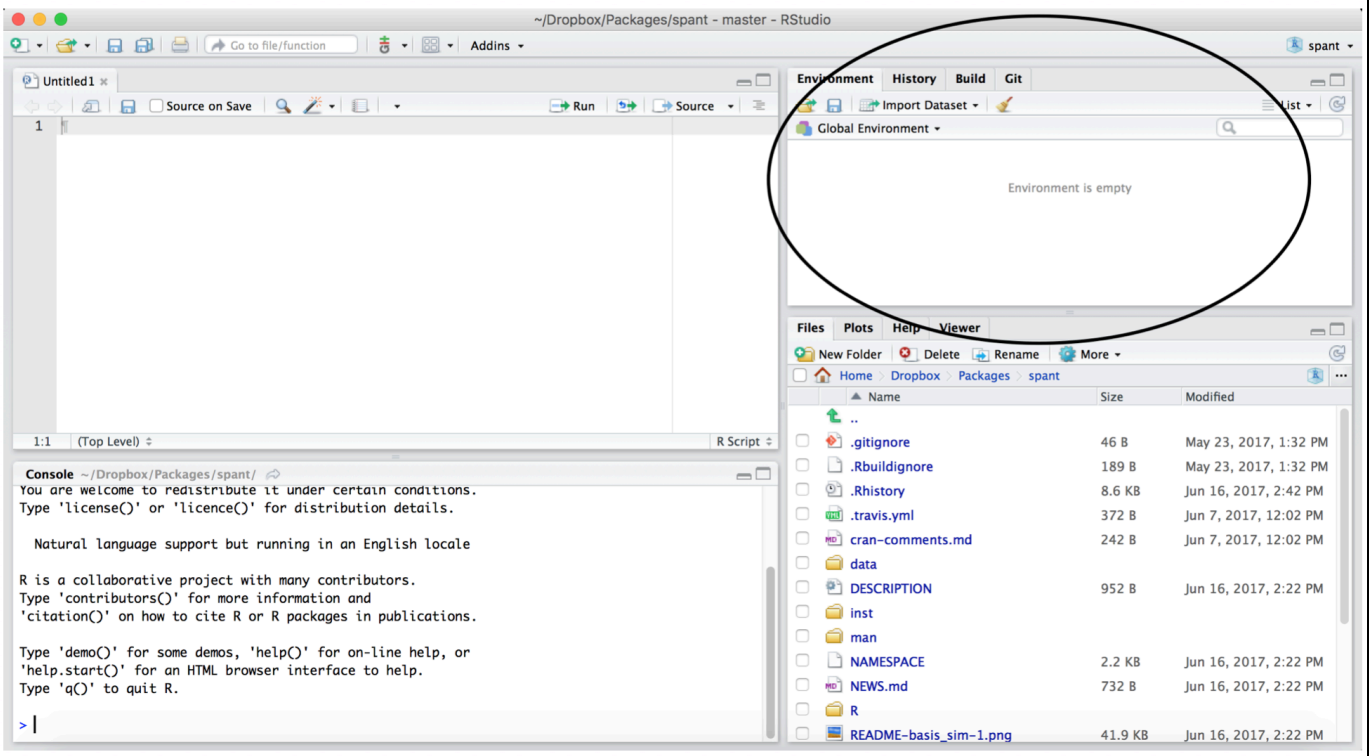
Source/Editor



(Source: https://johnmuschelli.com/intro_to_r/RStudio/RStudio.pdf)

- Typically located: **upper left** (see below for moving panes around)
- Here is where you will do your **R Script** and **R Markdown** work
- Use FILE > NEW FILE > *your choice* to begin a new R Script or R Markdown
- HACK: When to use R Script: To create and save files of R commands (more on this later)
- HACK: When to use R Markdown: To create archives of R sessions (R commands ***and*** output)
- HACK: In R Script, to execute a command (good for troubleshooting): <control-ENTER>
- Tip: Different R Script files can be saved in different tabs
- HACK: Create R Script files that you save as "boilers" for future work

Workspace/Environment



(Source: https://johnmuschelli.com/intro_to_r/RStudio/RStudio.pdf)

- Typically located: **upper right** (see below for moving panes around)
- Notice that there are <u>multiple tabs</u> here (yours might vary a little; that's okay)
- **Environment tab**: Here you see all your stuff, called "objects" – datasets, variables, etc
- **History tab:** Here you will see all your previous commands
- HACK: To view a "spreadsheet" version of your data: CLICK on its name in the workspace
- HACK: A better approach to scroll through previous commands for editing and re-use: In the console window, use the UP-ARROW

Other Panes

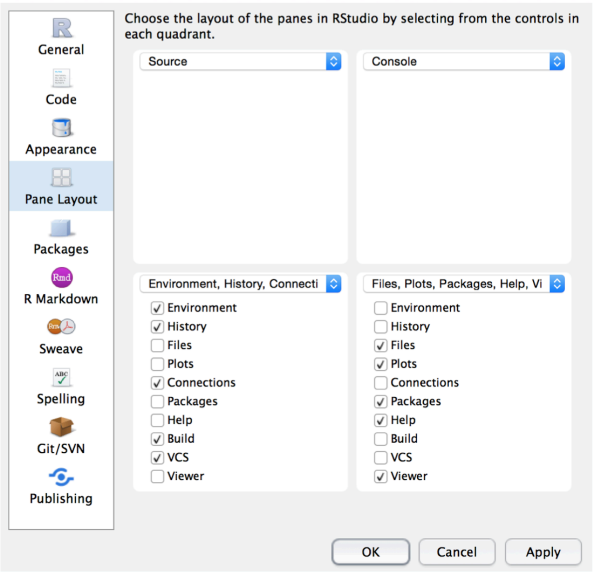
- **Files** - shows the files on your computer of the directory you are working in
- **Viewer** - can view data or R objects
- **Help** - shows help of R commands
- **Plots** - pretty pictures
- **Packages** - list of R packages that are loaded in memory

(Source: https://johnmuschelli.com/intro_to_r/RStudio/RStudio.pdf)

- Typically located: **lower right** (see below for moving panes around)
- Lots of useful tabs are here! Be sure to "toggle" about to acquaint yourself.
- For example, here is where you'll find: plots, help w packages, importing from your computer, help

How to Rearrange the Display of Your Panes

RStudio –> Preferences –> Pane Layout

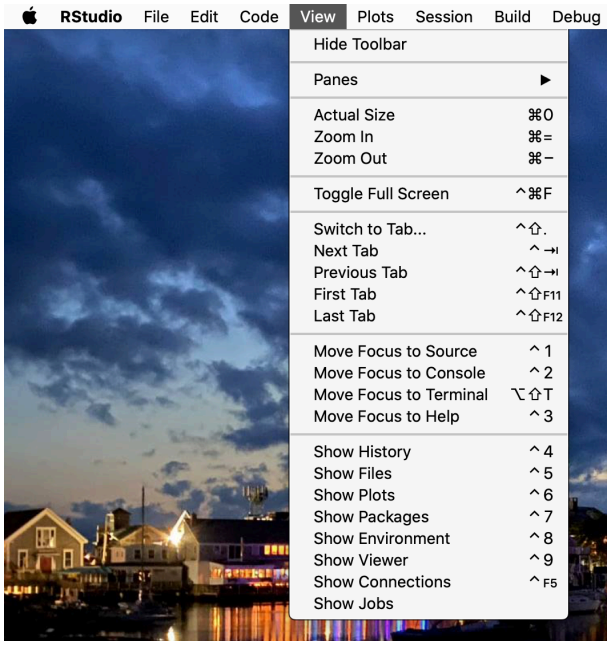


(Source: https://johnmuschelli.com/intro_to_r/RStudio/RStudio.pdf)

Shortcuts: How to Move Between Panes

Shortcut	Moves you to:
< control > 1	Source/Editor (your script file)
< control > 2	Console
< control > 3	Help
< control > 4	History
< control > 5	Files
< control > 6	Plots
< control > 7	Packages
< control > 8	Environment
< control > 9	Viewer
< control > SHIFT 0	Returns you to original 4 panel display

Forgot? You can also find these shortcuts from the top toolbar in R Studio. Click **VIEW**



For more, see
file:///Applications/RStudio.app/Contents/Resources/www/docs/keyboard.htm

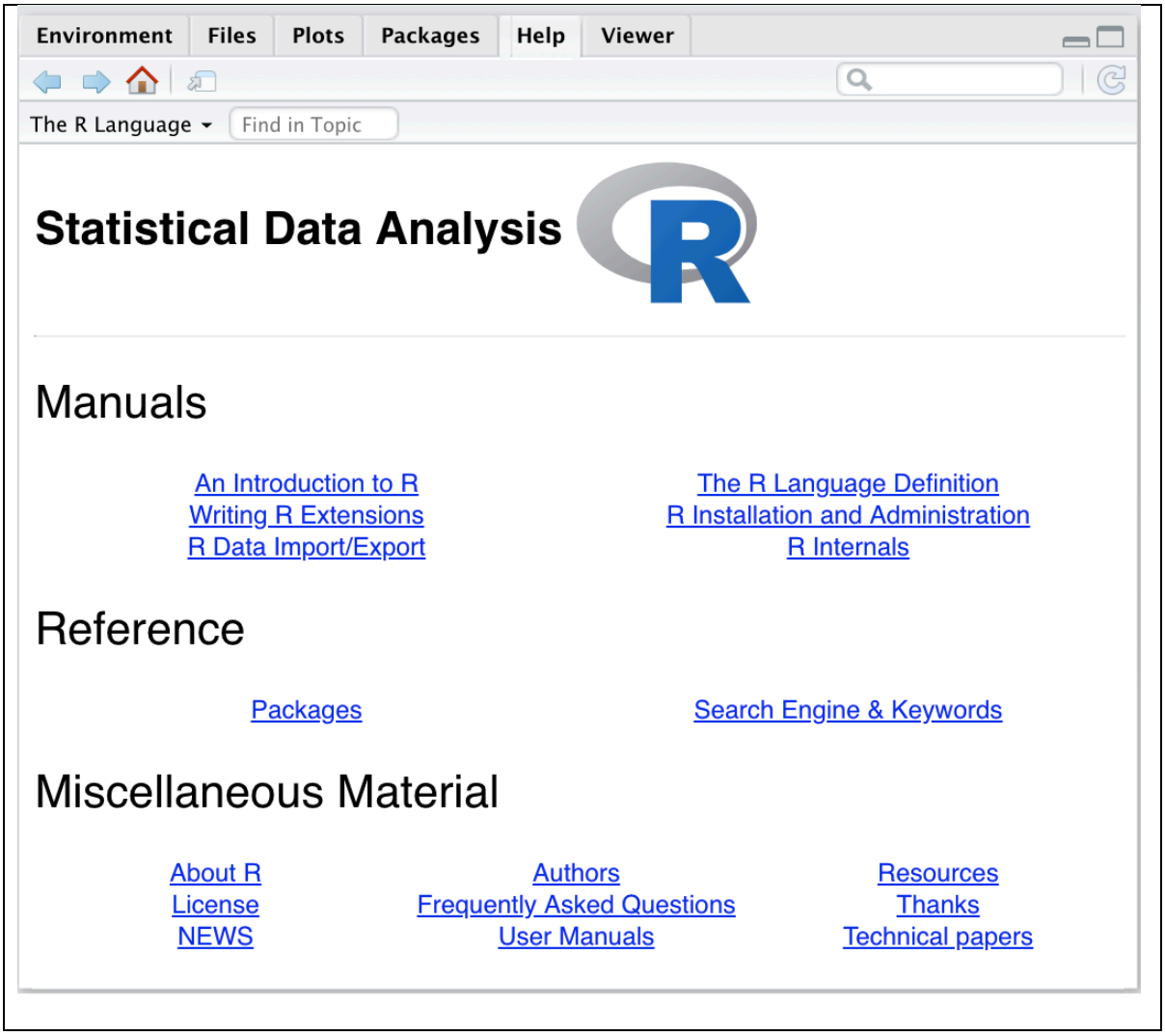
For yet more, see
At top, **TOOLS > KEYBOARD SHORTCUTS HELP**

1.4. Resources for Getting Help

There are lots of resources for getting help.

Within Your R Studio Session

In the console window, type: **help.start( )**
This will give you direct access to the R Help Pages. You'll see them at top right in the HELP tab:



Within Your R Studio Session

Note – the commands shown are typed into the console pane

```
#  Help with a function (two ways)

?functionname
help(functionname)
```

```
> # Example
> ?read.table
> help(read.table)
```

```
#  Help with a package

help(package="packagename")
```

```
> # Example
> help(package="readr")
```

```
#  Help with a topic

help.search("topicname")
```

```
> # Example
> help.search("regression")
```

The internet also has many good sites for help (with extra rows to add your own!)

Site	Description
Rseek.org	This is a search engine that you use to search for help on the official website, the CRAN
https://www.r-bloggers.com	Lots of articles by R-user bloggers
http://www.cookbook-r.com	Created by Winston Chang.
https://www.statmethods.net	Created by Rob Kabacoff
https://stackoverflow.com/questions/tagged/r	This is the R section of Stack Overflow and pertains solely to R questions
https://rstudio.com/resources/webinars/	R Studio sponsored webinars

2. Essential R Functionality: How to Work with Packages

Introduction

By default, when you downloaded and installed R and R Studio, the installer installed for you a set of standard packages. There are several of these; examples are the **base** and **stats** packages. For much of your work, you will want to make use of add-on packages. There are lots and lots of add-on packages!

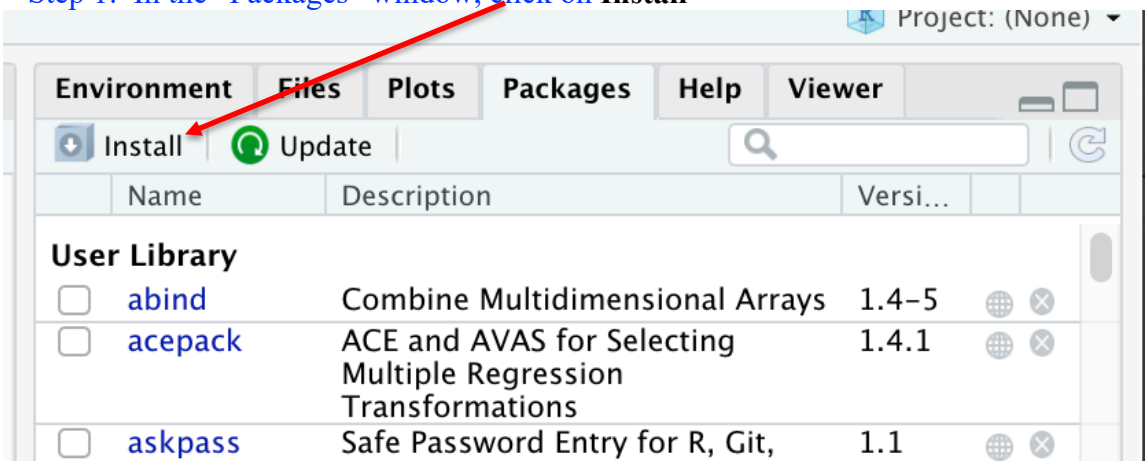
Working with Packages Requires TWO steps:

STEP 1: One-time installation, and;
STEP 2: Every session attachment of the package to your current R Studio session.

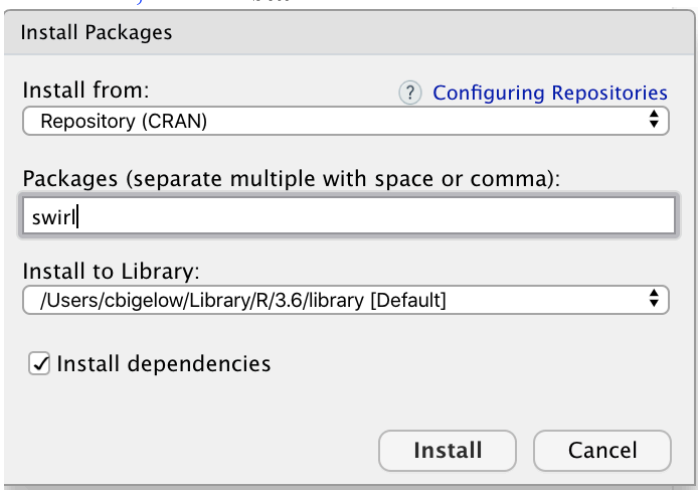
STEP 1: How to Install a Package
There's more than one way to download and install a package …

METHOD I: Menu Driven (Easy and recommended)

Step 1: In the "Packages" window, click on **Install**



Step 2: In the "Install Packages" window
Example: I want to install the package called swirl
In Install from: **default (Repository CRAN) is fine**
In Packages (separate multiple with space or comma:) **type in name of package (e.g. swirl)**
In Install to Library: **leave as is**
Check box for "Install dependencies": **check this**
At bottom, click **Install**



Step 3: Be patient. Wait. Then take a look at your console window. You should see:
At bottom you should see the prompt ">".

```
> install.packages("swirl")
Installing package into '/Users/cbigelow/Library/R/3.6/library'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/swirl_2.4.5.tgz'
Content type 'application/x-gzip' length 348039 bytes (339 KB)
==================================================
downloaded 339 KB


The downloaded binary packages are in
        /var/folders/rn/drwz6qbx3nb0ycr7vlppztt52nq3lw/T//RtmpSV27qs/downloaded_packages
>
```

METHOD II: Command Driven (From the console window)

How to Install a Package from the Console Window

install.packages("nameofpackageinquotes")

Example: install.packages("foreign")
HACK: Don't forget the period in install.packages
HACK: The name of the package MUST be enclosed in double quotes
HACK: Always install packages in the console window
HACK: Never install a package within a R Markdown file (more on this later)

STEP 2: Each Session - How to Attach a Package to Your Session

How to Attach a Package to Your R Studio Session

library(nameofpackageNOTinquotes)

Example: library(foreign)

Good to know. Some Useful help commands related to packages

- To see what packages you have installed now, from console: library()
- To see what packages are attached to your session, from console: search()
- To get location of library containing packages, from console: .libPaths()
- To get help, from console: help(package="foreign")
  NOTE: The help will appear in the help/plot window at bottom right

Good to know. How to Upgrade a Package

- Tip: Be sure to visit https://www.datacamp.com/community/tutorials/r-packages-guide
- To see what packages are old and need updating: old.packages()
- To update ALL packages: update.packages()

- To update JUST ONE package is a 2 step procedure:
  STEP 1: Remove the old package. Example: update.packages("vioplot")
  STEP 2: Re-install the same package. Example: install.packages("vioplot")

Good to know. How to Display a List of Your Installed Packages

Method I (point and click)
In the Other Panes click on the tab **Packages**

Method II (direct command)
In the console window, type: library( )

What can Go Wrong when Working with Packages

#1. You forgot to do the "one time" installation of that package.

#2. You have installed the package (hooray) but you forgot to issue the command **library(*packagename*)** to attach that package to your current R Studio session.

#3. *Really annoying.* Sometimes the command you are using has the same name in more than one package and you happen to have attached those packages to the same R Studio session. R Studio gets confused.

For example, if you want to use the command **freq** in the package **summarytools**

And the following does not work
freq(framingham$sex)

Then help R Studio to know what you want by prefacing your command with the package name and a double colon
summarytools::freq(framingham$sex)

3. Sample Session

Initialize R Studio Session.

```
# Set working directory (The following illustration works for Mac Users Only)
setwd("~/Desktop")

# Check working directory
getwd()

## [1] "/Users/cbigelow/Desktop"

# Turn off scientific notation
options(scipen=999)
```

Load Packages (Assumes previously installed)

```
library(stargazer)            # Command used:  stargazer()
library(summarytools)         # Command used:  freq()
library(Hmisc)                # Command used:  label()
library(gmodels)              # Command Used:  CrossTable()
library(psych)                # Command used:  describeBy()
```

R Studio Console as a giant calculator.

```
# Basic math - Result shown but NOT saved
4+6

## [1] 10

4-6

## [1] -2

# Basic math - Result saved as object and shown:  TWO lines of code
y <- 4+6
y

## [1] 10

# Basic math - Result saved as object and shown: ONE line of code using semi-colon
x<-5+8; x

## [1] 13

# Basic math - Result saved as object and shown:  ONE line of code using parentheses
(x<-5+8)

## [1] 13

# Basic math - Result saved as object and shown: ONE line, semi-colon, some explanatory text
z<-8+16; paste("z = 8+16 = ",z)

## [1] "z = 8+16 =  24"
```

Create some R Objects. Verify, echo, check their type

```
# Create a vector object using command c() - Do NOT save
c(1,2, 4, 8, 12, 13, 15)

## [1] 1  2  4  8 12 13 15

# Create a numeric vector object.  Save to v1.   Note - R Studio will not show it until you ask.
v1 <- c(1,2, 4, 8, 12, 13, 15)

# Create a character vector object of text. Note use of quotes (required) Save to v2
v2 <- c("Bob", "Jean", "Peg", "John", "Carol")


# List the object, by simply typing the name of the object
v1

## [1]  1  2  4  8 12 13 15

v2

## [1] "Bob"   "Jean"  "Peg"   "John"  "Carol"

# Use command class() to identify the type of object
class(v1)

## [1] "numeric"

class(v2)

## [1] "character"

# Create a random sample from a normal distribution
# Version 1 of command (recommended) is rnorm(n=, mean=, sd=)
y1 <- rnorm(n=5,mean=100,sd=15)
y1

## [1]  92.02998 105.79151 111.17030  84.75953  78.99063

# Version 2 of command (if you are an expert) is rnorm(n,mean,sd)
y2 <- rnorm(5,100,15)
y2

## [1]  94.27202 122.63773  86.98042  86.78760  87.76376
```

Load Data

```
# The following assumes that the data is located in working directory
# load(file="NAME.Rdata")
load(file="ivf.Rdata")

# What to do if the data is located in some other folder.
# Note the nasty looking path with icky blanks in it. I used the file.choose( ) followed by paste
load(file="/Volumes/users/cbigelow/1.  Teaching/web690C/data/toy.Rdata")
```

Review and Inspect Data

```
# Review data structures
# str(OBJECT)
paste("Quick look at ivf.Rdata")

## [1] "Quick look at ivf.Rdata"

str(ivf)

## 'data.frame':    641 obs. of  6 variables:
##  $ id     : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ matage : int  33 34 34 30 35 37 31 31 33 33 ...
##  $ hyp    : int  0 0 0 0 0 0 0 1 1 0 ...
##  $ gestwks: num  37.7 39.2 35.7 39.3 38.4 ...
##  $ sex    : Factor w/ 2 levels "male","female": 2 2 2 1 2 1 1 2 1 2 ...
##  $ bweight: int  2410 2977 2100 3270 2620 3260 3750 1450 3200 3675 ...
##  - attr(*, "datalabel")= chr "In Vitro Fertilization data"
##  - attr(*, "time.stamp")= chr "14 Feb 2017 09:55"
##  - attr(*, "formats")= chr  "%9.0g" "%8.0g" "%8.0g" "%9.0g" ...
##  - attr(*, "types")= int  254 251 251 254 251 252
##  - attr(*, "val.labels")= chr  "" "" "" "" ...
##  - attr(*, "var.labels")= chr  "identity number" "maternal age (years)" "hypertension (1=yes, 0=no)" "ge
stational age (weeks)" ...
##  - attr(*, "version")= int 12
##  - attr(*, "label.table")=List of 1
##   ..$ sex: Named int  1 2
##   .. ..- attr(*, "names")= chr  "male" "female"

# List first 6 observations - all variables
# head(OBJECT)
head(ivf)

##   id matage hyp gestwks    sex bweight
## 1  1     33   0   37.74 female    2410
## 2  2     34   0   39.15 female    2977
## 3  3     34   0   35.72 female    2100
## 4  4     30   0   39.29   male    3270
## 5  5     35   0   38.38 female    2620
## 6  6     37   0   37.86   male    3260
```

```
# List last 6 observations - all variables
# tail(OBJECT)
tail(ivf)

##      id matage hyp gestwks    sex bweight
## 636 636     34   0   41.15   male    2972
## 637 637     28   0   38.58 female    2850
## 638 638     38   1   38.44   male    3182
## 639 639     26   0   38.94 female    3048
## 640 640     31   0   40.43 female    3183
## 641 641     31   0   38.15   male    2920

# List first 6 observations (rows) - all variables (columns).  Don't forget comma.
# OBJECT[ROWFIRST:ROWLAST,]
ivf[1:6,]

##   id matage hyp gestwks    sex bweight
## 1  1     33   0   37.74 female    2410
## 2  2     34   0   39.15 female    2977
## 3  3     34   0   35.72 female    2100
## 4  4     30   0   39.29   male    3270
## 5  5     35   0   38.38 female    2620
## 6  6     37   0   37.86   male    3260

# List first 6 observations (rows) - ONLY variables 1:5 (columns)
# OBJECT[ROWFIRST:ROWLAST,COLUMNFIRST:COLUMNLAST]
ivf[1:6,1:5]

##   id matage hyp gestwks    sex
## 1  1     33   0   37.74 female
## 2  2     34   0   39.15 female
## 3  3     34   0   35.72 female
## 4  4     30   0   39.29   male
## 5  5     35   0   38.38 female
## 6  6     37   0   37.86   male

# List first 6 observations (rows) - Specific selection of variables (columns)
# OBJECT[ROWFIRST:ROWLAST, c("VARIABLENAME", "VARIABLENAME", "VARIABLENAME")]
ivf[1:6,c("matage", "gestwks", "sex")]

##   matage gestwks    sex
## 1     33   37.74 female
## 2     34   39.15 female
## 3     34   35.72 female
## 4     30   39.29   male
## 5     35   38.38 female
## 6     37   37.86   male
```

Label variables. Label discrete variable values.

```
# Label variable using package = Hmisc and command label( )
# label(OBJECT$VARIABLE) <- "Variable label"
label(ivf$hyp) <- "Hypertension"
label(ivf$sex) <- "Sex at Birth"

# Label discrete variable values.  Tip - I saved to a new variable as type=factor for nicer descriptives
ivf$hypF <- factor(ivf$hyp,
                   levels = c(0,1),
                   labels = c("0 = no", "1 = hypertension"))
label(ivf$hypF) <- "Hypertension"
```

Produce Numerical Summaries

```
# Quick summarization - all variables using package=stargazer and command stargazer( )
# stargazer(data=OBJECT,type="text",median=TRUE, title="FILLIN")
stargazer(data=ivf,type="text", title="Quick look at ivf.Rata")

##
## Quick look at ivf.Rata
## ===========================================================================
## Statistic  N    Mean    St. Dev.  Min   Pctl(25) Pctl(75)  Max
## ---------------------------------------------------------------------------
## id        641  321.000  185.185    1      161      481     641
## matage    641  33.972    3.870     23      31       37      43
## hyp       639   0.139    0.347   0.000   0.000    0.000   1.000
## gestwks   641  38.687    2.330   24.690  38.010   40.150  42.350
## bweight   641 3,129.137 652.783   630    2,850    3,550   4,650
## ---------------------------------------------------------------------------

# Single variable - discrete using package=summarytools and command freq( )
# What can go wrong:  variable must be type FACTOR
# summarytools::freq(OBJECT$FACTORVARIABLE)
freq(ivf$hypF, order="freq")

## Frequencies
## ivf$hypF
## Label: Hypertension
## Type: Factor
##
##                         Freq   % Valid   % Valid Cum.   % Total   % Total Cum.
## --------------------- ------ --------- -------------- --------- --------------
##                0 = no    550     86.07          86.07     85.80          85.80
##       1 = hypertension     89     13.93         100.00     13.88          99.69
##                   <NA>      2                                0.31         100.00
##                  Total    641    100.00         100.00    100.00         100.00
```

```
# Two variables - discrete crosstab using package = gmodels and command CrossTable( )
# gmodels::CrossTable(OBJECT$VAR1, OBJECT$VAR2, prop.t=FALSE, prop.r=TRUE, prop.c=FALSE)
# What can go wrong:  variables must be type FACTOR
gmodels::CrossTable(ivf$sex,ivf$hypF,digits=2,prop.r=TRUE,prop.c=FALSE,prop.t=FALSE,prop.chisq=FALSE,dnn=c(
"Sex at Birth","Hypertension"))

##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |-------------------------|
##
##
## Total Observations in Table:  639
##
##
##              | Hypertension
## Sex at Birth |            0 = no | 1 = hypertension |        Row Total |
## -------------|-------------------|------------------|------------------|
##         male |               273 |               52 |              325 |
##              |              0.84 |             0.16 |             0.51 |
## -------------|-------------------|------------------|------------------|
##       female |               277 |               37 |              314 |
##              |              0.88 |             0.12 |             0.49 |
## -------------|-------------------|------------------|------------------|
## Column Total |               550 |               89 |              639 |
## -------------|-------------------|------------------|------------------|


# One or multiple variables - continuous using package=stargazer and option summary.stat( )
# stargazer(OBJECT[c("VAR1","VAR2")],type="text",
#         summary.stat=c("n","mean", "sd","min","p25","median","p75","max"))
stargazer(ivf[c("matage","gestwks","bweight")],type="text",
          summary.stat=c("n","mean", "sd","min","p25","median","p75","max"),
          title="IVF Data")

##
## IVF Data
## ==========================================================================
## Statistic  N    Mean    St. Dev.  Min   Pctl(25) Median Pctl(75)  Max
## --------------------------------------------------------------------------
## matage    641  33.972    3.870     23      31      34      37      43
## gestwks   641  38.687    2.330   24.690  38.010  39.150  40.150  42.350
## bweight   641 3,129.137 652.783   630    2,850   3,200   3,550   4,650
## --------------------------------------------------------------------------
```

```
# Continuous - by Group (discrete):  Method I (lots of detail, but a bit messy looking in my opinion)
# Uses package = psych and command describeBy( )
# describeBy(OBJECT$CONTINUOUS, OBJCECT$GROUPVAR)
describeBy(ivf$bweight,group=ivf$hypF)

##
##  Descriptive statistics by group
## group: 0 = no
##    vars   n    mean     sd median trimmed    mad min  max range  skew
## X1    1 550 3191.74 601.35   3240 3227.22 491.48 630 4650  4020 -0.85
##    kurtosis    se
## X1     1.96 25.64
## ------------------------------------------------------------
## group: 1 = hypertension
##    vars  n    mean     sd median trimmed    mad min  max range  skew
## X1    1 89 2742.16 812.95   3000 2802.97 563.39 700 4425  3725 -0.72
##    kurtosis    se
## X1     -0.1 86.17



# Continuous - by Group(discrete):  Method II (less detail but much nicer to look at)
# Uses package=summarytools and command stby( )
# with(OBJECT,stby(data=CONTINUOUS,INDICES=GROUPVAR,
#                  FUN=descr,stats=c("statistic", "statistic", "statistic")))
with(ivf, stby(data = bweight, INDICES = hypF,
                  FUN = descr, stats = c("mean", "sd", "min", "med", "max")))

## Descriptive Statistics
## bweight by hypF
## Data Frame: ivf
## N: 550
##
##                  0 = no   1 = hypertension
## ------------- --------- ------------------
##          Mean   3191.74            2742.16
##       Std.Dev    601.35             812.95
##           Min    630.00             700.00
##        Median   3240.00            3000.00
##           Max   4650.00            4425.00
```

4. R Basics

4.1. R Command Conventions

Most R commands end with a set of parentheses ( )

- The parentheses ( ) usually contains what R is to work on
- **Note -** Sometimes the parentheses ( ) is empty; that's okay!. For example: **getwd( )**

The best assignment operator to use is < -

- For example: **y < - 10**
- Translation: What appears on the right is assigned to the left
- You could also use = as an assignment operator, but this is NOT recommended
- You could also assign from left to right using **- >** Sometimes this is a good idea, but not often.
  When R has a response, or multiple responses, they are provided numbered, beginning with [1]
- If there is more than one response, these will be numbered **[1], [2]**, and so on.

The comment character in R is the hashtag, #

- R ignores everything after # on that line
- **Most R commands have options. These are separated by commas and they must be in the correct order.**
- For example: **rnorm(5,100,15)** tells R to do n=5 random draws from a normal distribution with mean = 100 and standard deviation = 15
- **Tip for newcomers to R!** As you are learning R functions, consider providing the option names, followed by an equal sign, then the value. For example: **rnorm(n=5, mean=100, sd=15)**
- **Tip for experts:** Once you are expert, it's fine to leave off the function names. For example: **rnorm(5, 100, 15)**

4.2. R Objects

Previously….

You may have worked with **ONE dataset ("spreadsheet") at a time**.

For example – Until recently, SAS, Stata, SPSS, Minitab, etc. stored ONLY ONE dataset in its workspace at a time.

Example of a single Excel Spreadsheet

studyid	favorite	mpg	date_birth	date_today	age
1	chocolate	21.65	6/9/1956	9/16/2020	64.27
2	avocado	31.00	10/1/1953	7/4/2020	66.76
3	shrimp	50.62	10/14/2002	4/11/2020	17.49
4	pistachios	28.76	11/4/1951	6/18/1999	47.62
5	gruyere	22.40	7/5/2015	5/15/2020	4.86
6	oj	18.90	1/12/1968	9/12/2020	52.67

- Rows define observations (one unit per row)
- Columns define variables (column headings are variable names)
- Each cell contains a single entry ("tidy data", more on that later)
- Tip! Once entered into R and saved, R calls this spreadsheet object a **dataframe**
- In this example, the sample size is n=6 and there are 6 variables.

Welcome to R!

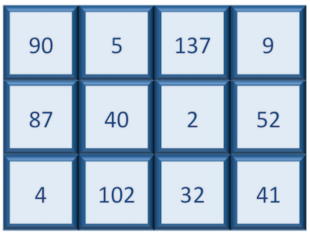
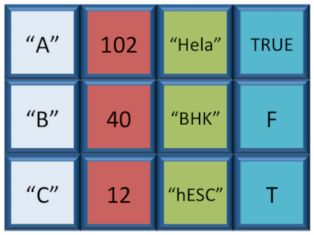
R lets you work with **more than one "data object" at a time**.

An R object (also called data structure) is simply a **"container"** for holding data. There are four (4) types of R Data Objects

- **Vector**
- **Matrix**
- **Data Frame**
- **List**

Design …… **Data Collection** …… **Data Management** …… **Data Summarization** …… **Statistical Analysis** …… **Reporting**

The 4 Types of R Objects (Data Structure) – Additional Detail.

Vector	Matrix	Data Frame	List
 *Note: This is a row vector. Column vectors are possible, too.*			
Definition and notes: - A set of data elements, all of the same storage type. - Separate entries (elements) are separated by a comma. - There can be as many elements as you like. - R is a command line processor that (for the most part) works on vectors or collections of vectors (matrices).	Definition and notes: - A two dimensional array, indexed by rows and columns. - **Every element must be of the same storage mode** (numeric, character, logical) - There can be as many rows, column as you like, but only ONE element per "cell". - Much of statistical programming operates on matrices.	Definition and notes: - Also a two dimensional array, indexed by rows and columns. - **Different columns (variables) can be of different storage modes.** **However, within a column (variable) every element must be of the same storage mode** - The **excel spreadsheet** you load into R Studio is often input as a **dataframe** object.	Definition and notes: - A collection of R Data Objects - A list can be a **mix of different types of R data objects.** - Distinct objects can be of **different storage modes.**
Depending on the ***storage mode (numeric, character, logical)***, we might have: A numeric vector: (21, 34, 16.8, 19.2) A character vector (note quotes): ("Bob", "Carol", "Jean") A logical vector: (TRUE,FALSE, FALSE)	The example shown is a matrix of **storage mode numeric.**	In this example: Columns 1 & 3: These are character variables. Entries have quotes. Column 2: This is a numeric variable. It could be either storage type integer or double. Column 4: This is a logical variable. Entries are either: TRUE, T, FALSE, or F.	This list is a mix of: - One column vector; - Two vectors of length =1; - One matrix; and - One data frame.

Source of pictures: https://hbctraining.github.io/Intro-to-R-flipped/lessons/02_introR-syntax-and-data-structures.html

Storage mode refers to the way that R stores data*. Here are the ones you are most likely to need.

Storage Mode	Sub-category	Description
Numeric	**Integer** (whole numbers only)	Used for performing mathematical operations. Examples: 5, 3, 2.89, 100.6892
	Double (with decimals)	
Character	None. Character data are text or string.	Always enclosed in quotes. Cannot be used in mathematical operations. Examples are anything enclosed in quotes: "Dog", "Cat", "5", "2.89"
Logical		Used for logical comparisons. Allowed entries are: FALSE, F, TRUE, and T.
Date *R stores dates as numeric and = # days since January 1, 1970*	**Date**	Calendar dates
	POSIX	Dates and Times
Factor *R stores your categorical data as "factors" Factors are special vectors where each value is a "level"*	**Unordered ("nominal")**	Example: ("dog", "cat", "rabbit")
	Ordered ("ordinal")	Example: ("low", "medium", "high")

* Not included here are complex and raw

Why do Object Types (and storage modes) matter?

Often, when something goes wrong, **the problem may be the object type**

4.3. Good to Know. More on Factors

factor( ) with no options stores levels as integers ***alphabetically.*** This is often NOT you want!

Example: We want to a categorical variable called **season** with: 1=winter, 2=spring, 3=summer, and 4=fall)

```
What could go wrong -  you get an alphabetic correspondence that you don't want
> season <- factor(c("winter", "summer", "summer", "fall", "fall", "spring"))
> table(season)
  fall spring summer winter
     2      1      2      1
 This says the following (alphabetic) which we do NOT want:  1=fall, 2=spring, 3=summer, 4=winter

       Solution I:  Set factor levels explicitly using option ordered=TRUE
       > season <- factor(c("winter", "summer", "summer", "fall", "fall", "spring"))
       > season <- factor(season,levels=c("winter","spring","summer", "fall"),ordered=TRUE)
       > table(season)
       winter spring summer   fall
            1      1      2      2
       NOW WE HAVE THE ORDERING WE WANT: 1=winter, 2=spring, 3=summer, 4=fall

       Solution II: Set factor levels explicitly using a vector of levels that you create first
       > season <- factor(c("winter", "summer", "summer", "fall", "fall", "spring"))
       > mylevels <- c("winter", "spring", "summer", "fall")
       > season <- factor(season,levels=mylevels)
       > table(season)
       winter spring summer   fall
            1      1      2      2
       AGAIN - NOW WE HAVE THE ORDERING WE WANT: 1=winter, 2=spring, 3=summer, 4=fall
```

```
# How to set integer storage to factor levels explicitly – Method I
factor(factorvariable, levels=c("level","level","level"),ordered=TRUE)

# How to set integer storage to factor levels explicitly – Method II
namelevels <- c("level","level","level")
factor(factorvariable, levels=namelevels)

# To check: table(factorvariable)
```

```
> # Example – Solution I
> season <- factor(c("winter", "summer", "summer", "fall", "fall", "spring"))
> season <- factor(season,levels=c("winter","spring","summer", "fall"),ordered=TRUE)
> table(season)

> # Example – Solution
> season <- factor(c("winter", "summer", "summer", "fall", "fall", "spring"))
> mylevels <- c("winter", "spring", "summer", "fall")
> season <- factor(season,levels=mylevels)
> table(season)
```

4.4. Parentheses ( ), Brackets [ ], and Braces { }

Take care! They are not interchangeable!

Parentheses ( ): Used in commands to execute functions

- Parentheses demarcate functions
- Parentheses demarcate options of functions
- **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 to subset/select particular row and column entries of an R data object ("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 { }: Use to begin and end a group of commands (often used in loops)

- **Example**:

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

4.4. 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 making a comparison 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

4.5. Mathematical Functions

Function	Definition	Example
`+`	Addition	`> 2+2` `[1] 4`
`-`	Subtraction	`> 5-3` `[1] 2`
`*`	Multiplication	`> 5*4` `[1] 20`
`/`	Division	`> 20/4` `[1] 5`
`^`	Exponentiation (raising to a power)	`> 6^2` `[1] 36`
`%/%`	Integer part of division or quotient	`> 48 %/% 5` What is whole number of 48/5? `[1] 9`
`%%`	Remainder part of division or quotient	`> 48 %% 5` What is the remainder of 48/5? `[1] 3`
`log()`	logarithm to base e ("natural log") You may know this as ln( )	`> log(34)` `[1] 3.526361` $e^{3.526361} = 34$
`log10()`	Logarithm to base 10	`> log10(100)` `[1] 2` $10^2 = 100$
`exp()`	Exponentiation of the constant e Recall: e = 2.718 … (approx.)	`> exp(4)` `[1] 54.59815` $e^4 = 54.59815$
`sqrt()`	Square root of	`> sqrt(100)` `[1] 10` $\sqrt{100} = 10$
`round(x,n)`	Round x to the nth digit	

4.6. Statistical Functions (fyi …. you might apply these to a column (a variable) in your data frame)

Tip!!! Want to remove missing values? Use the following option, after a comma: (na.rm=TRUE)

Function	Definition	Example

`length(x)`	Number of values in vector x	`x <- c(3,1,6,0,6)`
		`> length(x)`
		`[1] 5`
		
		`… alternatively, you could do ..`
		
		`> length(c(3,1,6,0,6))`
		`[1] 5`

`max(x)`	Maximum of values in vector x	`> x <- c(3,1,6,0,6)`
		`> max(x)`
		`[1] 6`

`min(x)`	Minimum of values in vector x	`> x <- c(3,1,6,0,6)`
		`> min(x)`
		`[1] 0`

`mean(x)`	Mean of values in vector x	`> x <- c(3,1,6,0,6)`
		`> mean(x)`
		`[1] 3.2`
		`> x <- c(3,1,NA,0,6)` *Oops a missing!*
		`> mean(x,na.rm=TRUE)`
		`[1] 2.5`

`median(x)`	Median of values in vector x	`> x <- c(3,1,6,0,6)`
		`> median(x)`
		`[1] 3`

`quantile(x,c(.25,.75))`	Obtain 25th and 75th quantile values in vector x	`> x <- c(3,1,6,0,6)`
		`> quantile(x,c(0.25,0.75))`
		`25% 75%`
		`1 6`

`range(x)`	Display minimum and maximum values in vector x	`> x <- c(3,1,6,0,6)`
		`> range(x)`
		`[1] 0 6`

`sd(x)`	Standard deviation of values in vector x	`> x <- c(3,1,6,0,6)`
		`> sd(x)`
		`[1] 2.774887`

`sum(x)`	Total of values in vector x	`> x <- c(3,1,6,0,6)`
		`> sum(x)`
		`[1] 16`

`var(x)`	Variance of values in vector x	`> x <- c(3,1,6,0,6)`
		`> var(x)`
		`[1] 7.7`

`abs(x)`	Absolute values of values in vector x	`> abs(2-10)` `[1] 8`
`factorial(x)`	Calculate x! = x(x-1)(x-2)…(2)(1)	`> factorial(4)` `[1] 24` 4! = 4*3*2*1 = 24

`rank(x)`	Ranks of values in vector x	`> x <- c(3,1,6,0,6)`
		`> rank(x)`
		`[1] 3.0 2.0 4.5 1.0 4.5`

4.7. Probability Distribution Calculators

Distribution of X	Pr [X = x]	Pr [X <= x]	Pr [X >= x]	Quantile (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)
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)
F Distribution df1=7 and df2=13	-	Pr [F <= 1.57] is pf(1.57, df1=7,df2=13)	Pr [F >= 1.57] is pt(1.57, df1=7, df2=13, lower.tail=F)	95th percentile is qf(.95,df1=7,df2=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)

5. Getting Data In and Out of R Studio

See also:
(source: R Studio Support) Importing Data with R Studio
https://support.rstudio.com/hc/en-us/articles/218611977-Importing-Data-with-RStudio

5.1. file.choose( )

Tip! Avoid typing out full paths "by hand." Invariably, you'll make a typo. Instead, PRIOR to writing your script file, in the CONSOLE, enter **file.choose()**. Browse and navigate to your data set. Click OPEN. R Studio will return the full path result in the console pane. Now you can EDIT/COPY/PASTE the full path into your R Script or R Markdown.

```
file.choose()
```

```
> # Example.
> file.choose()
[1] "/Volumes/users/cbigelow/1.  Teaching/web690C/data/relate100obs.Rdata"
```

5.2 Input/Output R Dataset (".Rdata")

```
# Input
load(file="FILLIN")

# Output
save(OBJECT, file="FILLIN")
```

```
> # Example
> load(file="ivf.Rdata")
> save(ivf,file="ivfnew.Rdata")
```

What could go wrong:
- You used the wrong name of "OBJECT". You can find this in the ENVIRONMENT pane
- You forgot the quotes.

Design ..... Data Collection ..... Data Management ..... Data Summarization ..... Statistical Analysis ..... Reporting

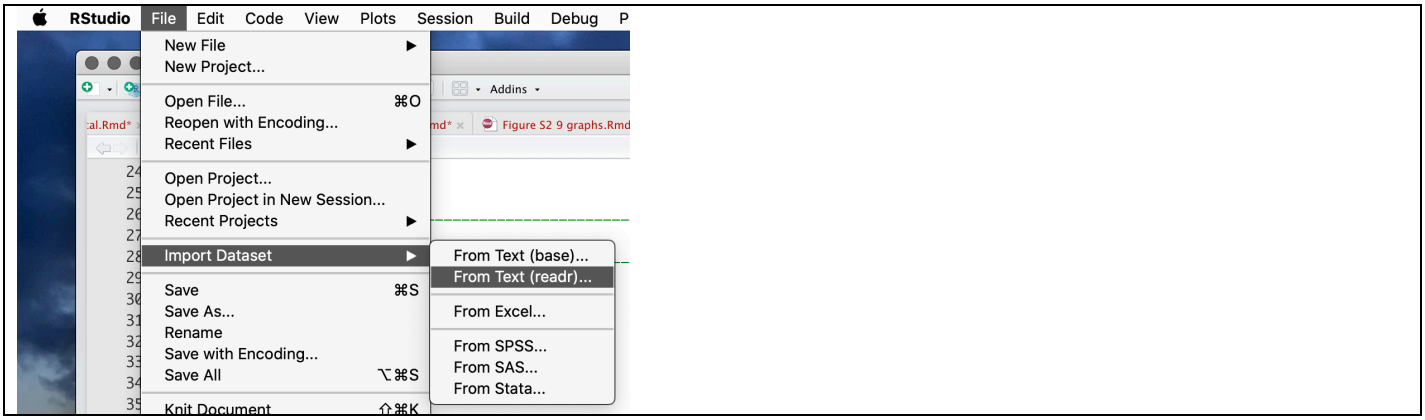
5.3 Input Text and Excel (".csv")

R Studio makes this easy. Use the utilities and menus provided in R Studio, rather than typing commands into the console window. You can access R Studio's import utilities using either **FILE > IMPORT DATA** or from the **Environment pane**, click on the tab **Import Dataset**

Step 1: Access R Studio's import utilities

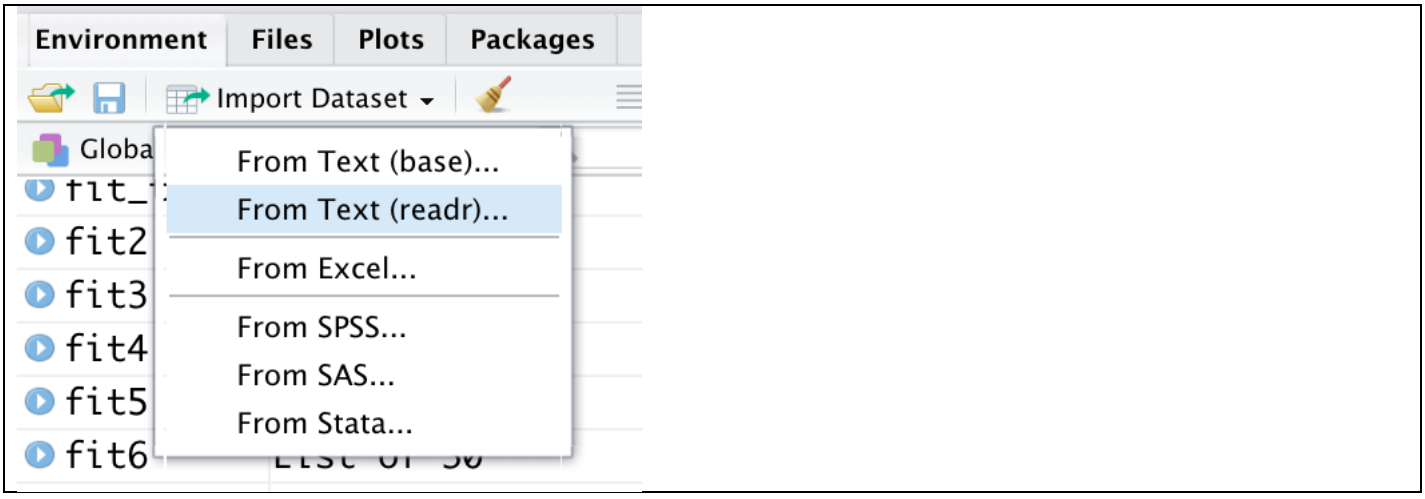
Method 1.

File > Import Dataset >

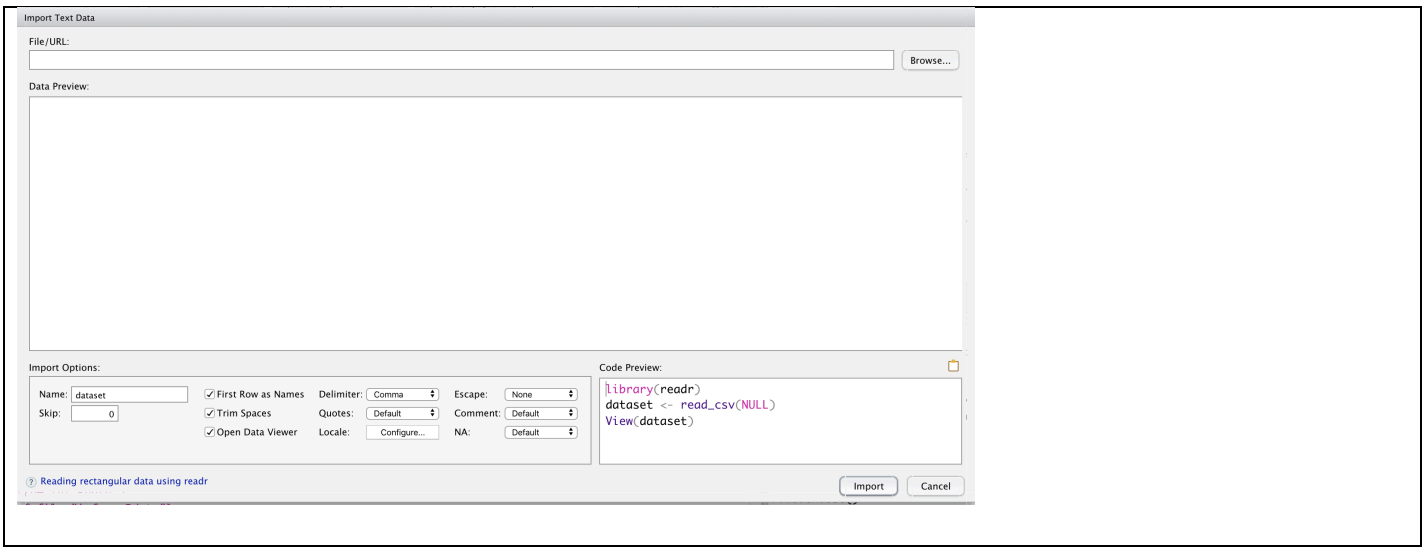


Method 2.

From the Environment Pane, click on the tab: Import Dataset

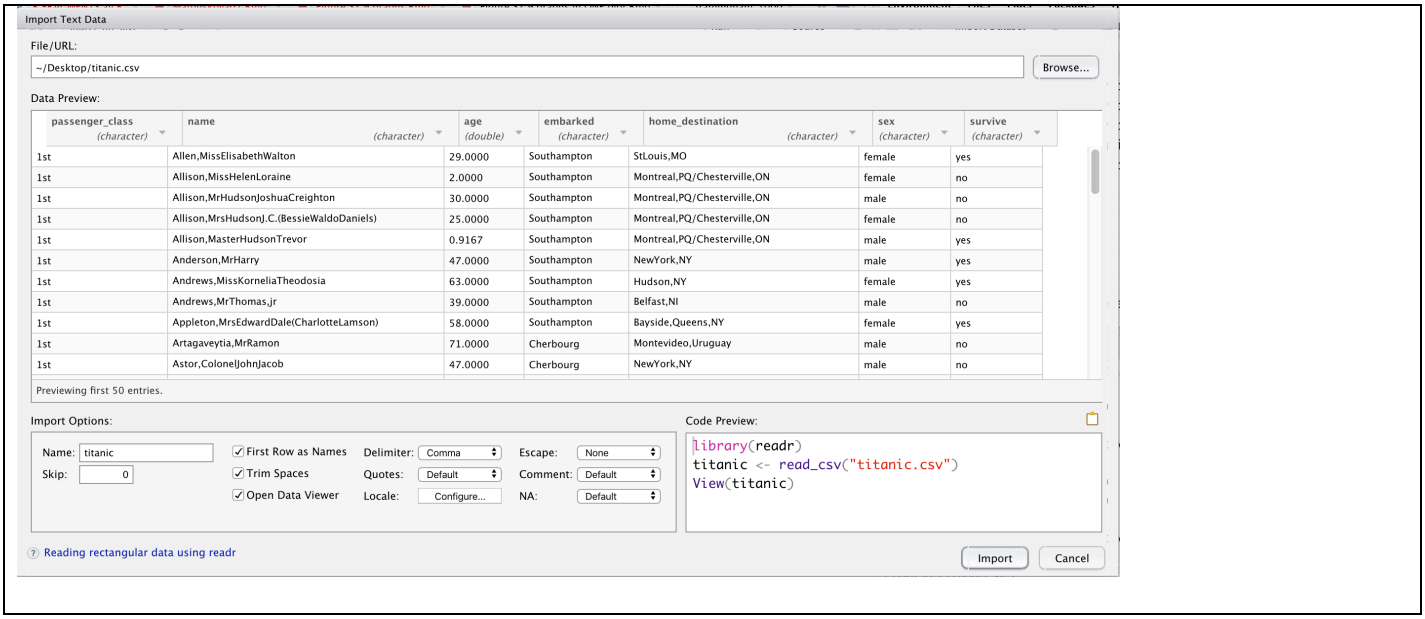


Step 2: Click **From Text (readr) …**
This will produce a screen where you navigate to select your data.

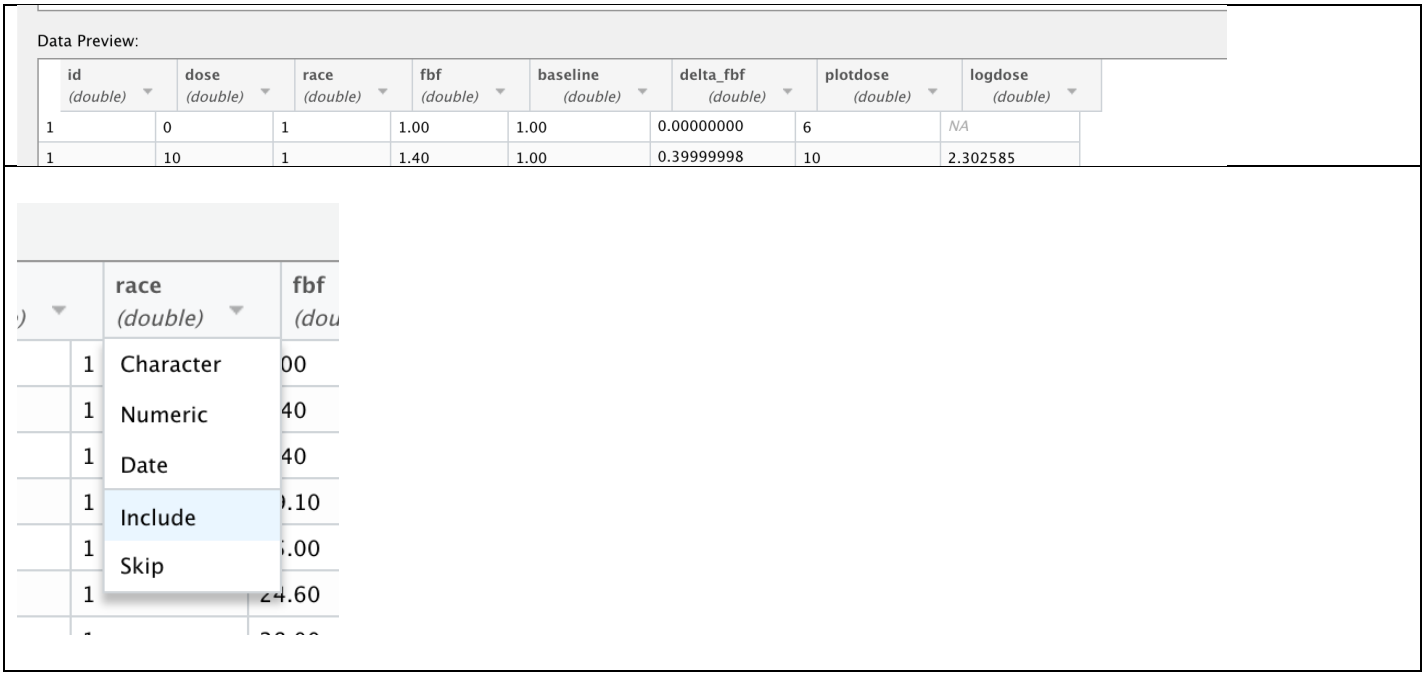


At top right, click **browse** to navigate and to select your data. Click **OPEN.**

Step 3: At bottom left, review your data under *Data Preview* and make your IMPORT OPTIONS selections.



Tip. It is possible to set the storage type of each variable before clicking on IMPORT. In reviewing your data under *Data Preview,* you may want to change its storage type (for example: sometimes codes of 1, 2, 3 are codes for character variables). For each variable, there is a drop- down menu. Use this drop-down menu to change storage type, as appropriate.



Step 4: All done with your selections?
- Copy and paste the code at lower right into your R Script file.
- Click **Import.**

5.4 Recommended. Output Excel (".csv")

Why ".csv"? In a nutshell, it's **good data management practice**: 1) ".csv" files can be read simply and easily by a variety of software programs (regardless of their version); and 2) ".csv" files are sustainable; they are likely to be readable by future software programs (including ones not yet invented)!

Step 1: Preliminary: Install (one time) the package **tidyverse** (see again, Section 2, pp 20-23)

Outputting an R dataset to an Excel ".cvs" file is done using the command **write_csv( )** which is actually a command in the package **readr.** However, the package **readr** is one of the packages that come bundled in the package **tidyverse**. As we will be using **tidyverse** for several commands, I recommend installing **tidyverse**.

Step 2: Output to Excel (".csv")

```
library(tidyverse)
write_csv(OBJECT, path="FILLIN.csv")
```

```
> # Example
> library(tidyverse)
> write_csv(ivf, path="ivfnew.csv")
```

What could go wrong:
- You have not done a one-time installation of the package **tidyverse**
- You forgot to issue the command **library(tidyverse)**
- You used the wrong name of "OBJECT". You can find this in the ENVIRONMENT pane
- You forgot the quotes.

5.5 Input Excel (".xlsx")

Again, R Studio makes this easy. Use the utilities and menus provided in R Studio, rather than typing commands into the console window. You can access R Studio's import utilities using either: 1) **FILE > IMPORT DATA** or 2) from the **Environment pane**, click on the tab **Import Dataset**

Step 1: Access R Studio's import utilities

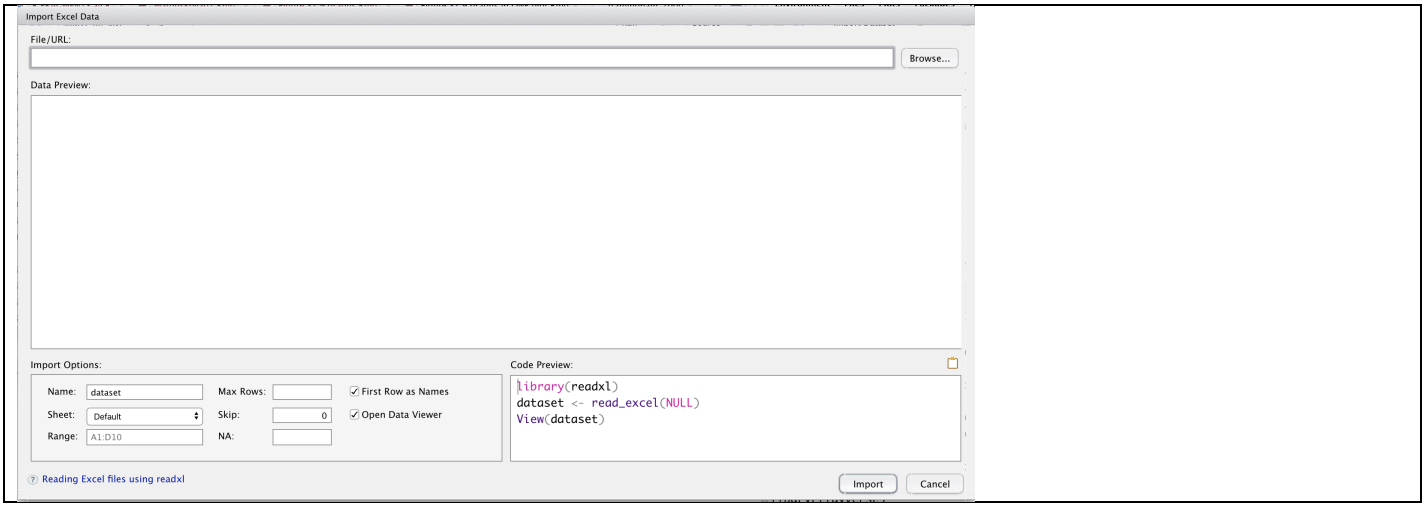
Method 1.

File > Import Dataset >

Method 2.

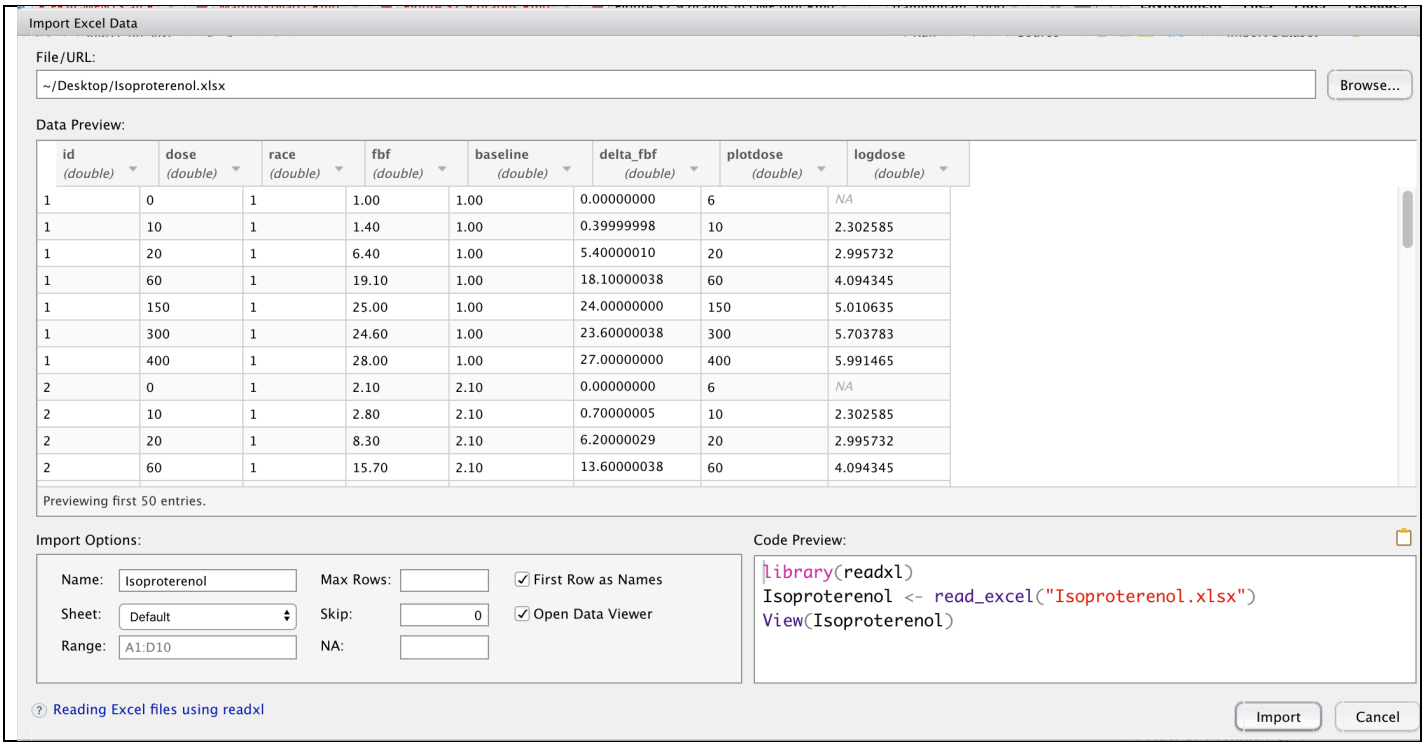
From the Environment Pane, click on the tab: Import Dataset

Step 2: Click **From Excel …**
This will produce a screen where you navigate to select your data.

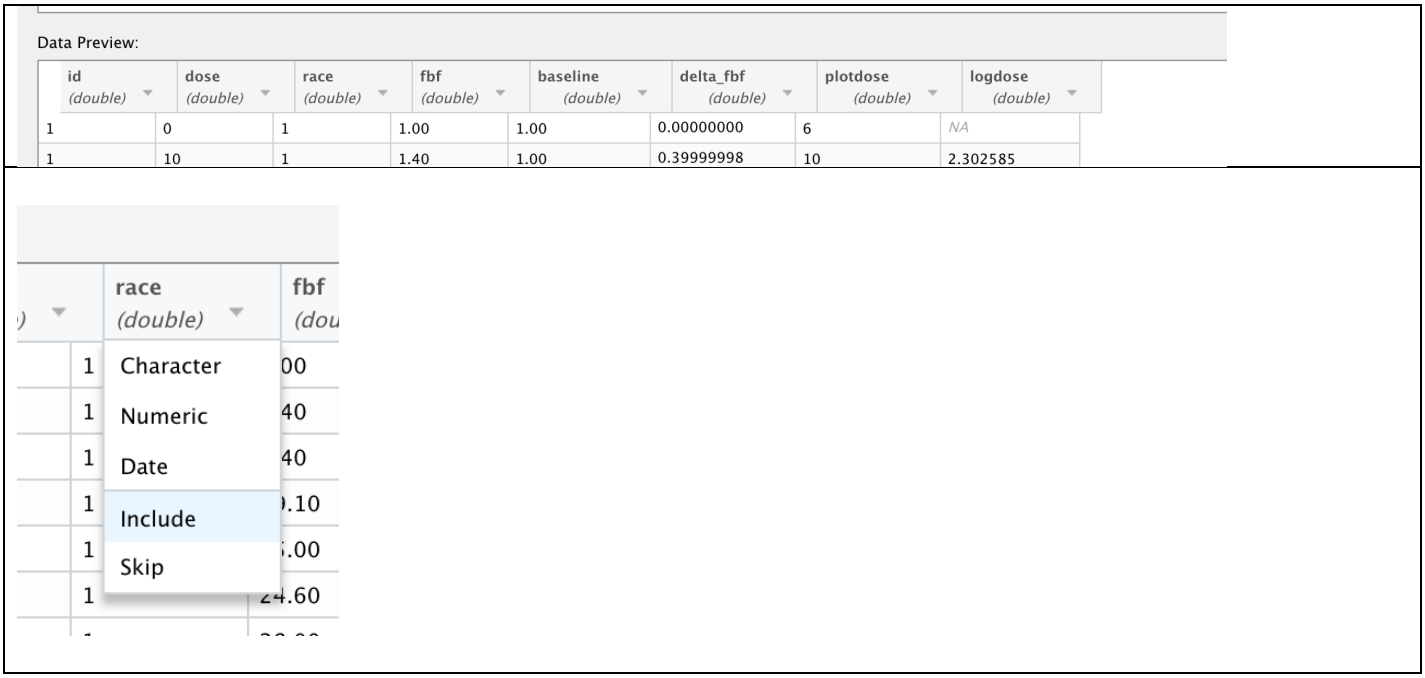


At top right, click **browse** to navigate and to select your data. Click **OPEN.**

Step 3: At bottom left, review your data under *Data Preview* and make your IMPORT OPTIONS selections.



Tip. **Review (and reset if need be) the storage type of each variable before clicking on IMPORT.** In reviewing your data under *Data Preview,* you may want to change its storage type (for example: sometimes codes of 1, 2, 3 are codes for character variables). For each variable, there is a drop- down menu. Use this drop-down menu to change storage type, as appropriate.



Step 4: All done with your selections?
- Copy and paste the code at lower right into your R Script or R Markdown file.
- Click **Import.**

What could go wrong:
- You forgot to click (or unclick) the box next to ***"First Row as Names"***

5.6 Create Your Own R Dataset

5.6.1. R Dataset (dataframe) from vectors

```
data.frame(vector, vector, vector, …)
```

```
> # Example.

> studyid <- as.integer(seq(from=1, to=3, by=1))
> name <- as.character(c("Bob", "Jean", "Peg"))
> age <- as.numeric(c(87.8,92.0,53.5))
> opinion <- as.factor(c("high", "low", "medium"))

> scratchdf <- data.frame(studyid,name,age,opinion)

> scratchdf

  studyid name  age opinion
1       1  Bob 87.8    high
2       2 Jean 92.0     low
3       3  Peg 53.5  medium
```

What could go wrong:
- Your vectors are not the same length

5.6.2. Two Way Table

```
tablename <- as.table(rbind(c(##,##),c(##,##)))
dimnames(tablename) <- list(
    rowvariablename=c("row1label","row2label"),
    columnvariablename=c("column1label","column2label"))
tablename
```

Exposure		Culture: Positive	Culture: Negative
	Yes	84	43
	No	10	92

```
> # Example.
> table1 <- as.table(rbind(c(84,43),c(10,92)))
> dimnames(table1) <- list(
    Exposure=c("Yes","No"),
    Culture=c("Positive","Negative"))
> table1

       Culture
Exposure Positive Negative
    Yes       84       43
    No        10       92
```

What could go wrong:

- You entered the data by columns instead of by rows (in which case you could have used **cbind( )**)

5.6.3. K 2x2 Tables

```
# Enter K=2 strata of 2x2 tables using command array()
# TIP:  Each row is 2x2, entered by COLUMN:  a,c,b,d
# dimension=c(#rows,#columns,#strata)

tablename <- array(c(a,c,b,d,
                     a,c,b,d),
                   dim=c(k,2,2),
                   dimnames=list(
                     rowvariablename=c("row1name","row2name"),
                     columnvariablename=c("column1name","columnname2"),
                     stratavariablename=c("stratum1name","stratum1name")))
```

Stratum = Smokers:

Coffee	MI = Yes	MI = No
Lots	a = 1011	b = 390
None	c = 81	d = 77

Stratum = Non-Smokers:

Coffee	MI = Yes	MI = No
Lots	a = 383	b = 365
None	c = 66	d = 1232

```
> # Example.
> table2 <- array(c(1011,81,390,77,
                    383,66,365,123),
                  dim=c(2,2,2),
                  dimnames=list(
                     Coffee=c("Lots","None"),
                     MI=c("Yes","No"),
                     Stratum=c("Smokers","NON-smokers")))

> table2
Stratum = Smokers

      MI
Coffee  Yes  No
  Lots 1011 390
  None   81  77

Stratum = NON-smokers

      MI
Coffee Yes  No
  Lots 383 365
  None  66 123
```

What could go wrong:

- You entered the data by rows instead of by columns

5.6.4. Individual Observations from a Table

You want to convert a table of counts to a dataset with individual observations, especially when making graphs. To do this involves two steps: 1) Use package **DescTools** includes a command **Untable( ),** to create your dataframe of individual observations, followed by (as you wish); 2) Convert factor → numeric

```
library(DescTools)

 # STEP 1:  Individual Observations from a Table using Untable( )
 dataframename <- DescTools::Untable(sourcetable)

#  STEP 2:  In new dataframe, convert factors to numeric
 dataframename$rowvariablename <- as.numeric(dataframename$rowvariablename)
 dataframename$columnvariablename <- as.numeric(dataframename$columnvariablename)

# Check:  Compare original table to 2x2 table of individuals
sourcetable
table(dataframename$rowvariablename, dataframename$columnvariablename)
```

Exposure	Culture: Positive	Culture: Negative
Yes	84	43
No	10	92

```
library(DescTools)
# Preliminary – Enter source table
table1 <- as.table(rbind(c(84,43),c(10,92)))
dimnames(table1) <- list(
     Exposure=c("Yes","No"),
     Culture=c("Positive","Negative"))

# STEP 1:  Untable( ) to create dataframe of individual observations called data1
data1 <- DescTools::Untable(table1)
# STEP 2: Convert factor to numeric
data1$Exposure <- as.numeric(data1$Exposure)
data1$Culture <- as.numeric(data1$Culture)
# Check
table1
table(data1$Exposure,data1$Culture)

        Culture
Exposure Positive Negative
     Yes       84       43
     No        10       92

     1  2
  1 84 43
  2 10 92
```

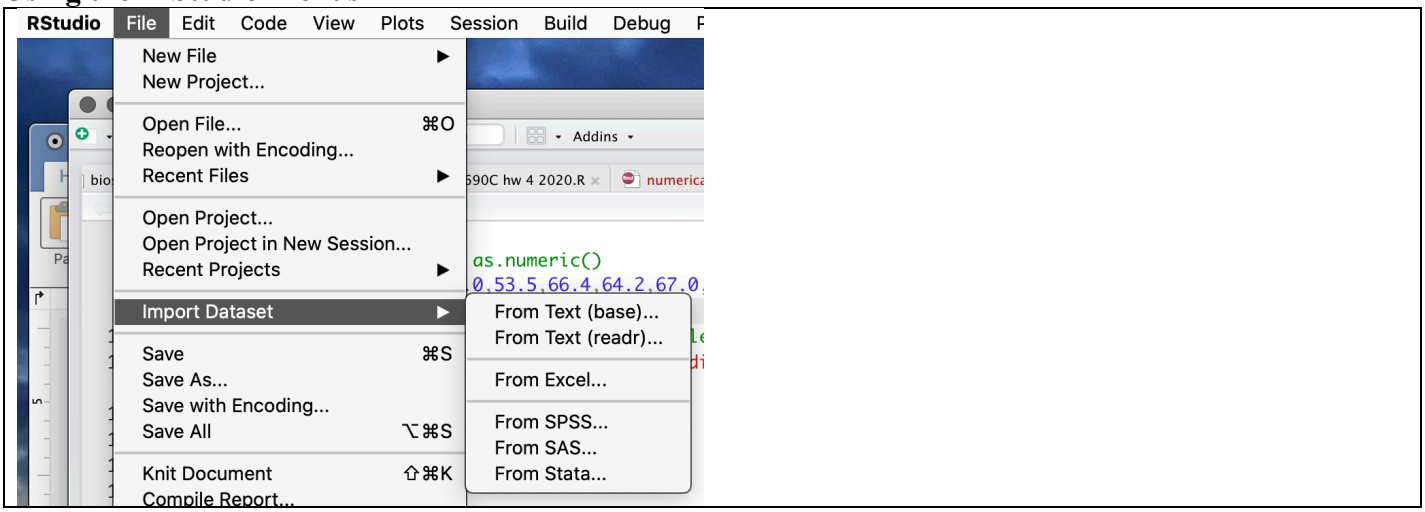
5.7. Input/Output SAS, SPSS, or Stata

Preliminary:
Be sure you have installed (one time) the package **`tidyverse`.** Importing SAS, SPSS or Stata is done using the package **haven** which is one of the packages bundled in **`tidyverse`**.

From the console pane

```
library(haven)

# SAS
read_sas("name.sas7bdat")
write_sas(object, "name.sas7bdat")

# SPSS
read_sav("name.sav")
write_sav(object, "name.sav")

# Stata
read_dta("name.dta")
write_dta(object, "name.dta")
```

Using the R Studio Menus



6. A Very Brief Introduction to `tidyverse`

https://tidyverse.tidyverse.org

6.1. What is `tidyverse`?

`tidyverse` is a collection of packages.
"Collection" means that when you install **`tidyverse`**, you are simultaneously installing all the packages that are in this collection. The collection is designed specifically to integrate and streamline several interrelated tasks in R:

- Input/Output (*importing/exporting*);
- Data Cleaning (*tidying*);
- Working with variables (*data wrangling*);
- Working with observations (*data wrangling*);
- Data visualization; and
- Programming generally

`tidyverse` includes 8 core packages.

Hack: The **`library(tidyverse)`** command attaches only some of the tidyverse component packages (the "core" ones). You will need to issue specific **`library()`** commands to attach the others.

`library(tidyverse)` simultaneously attaches all of 8 core packages; no other **`library( )`** required.

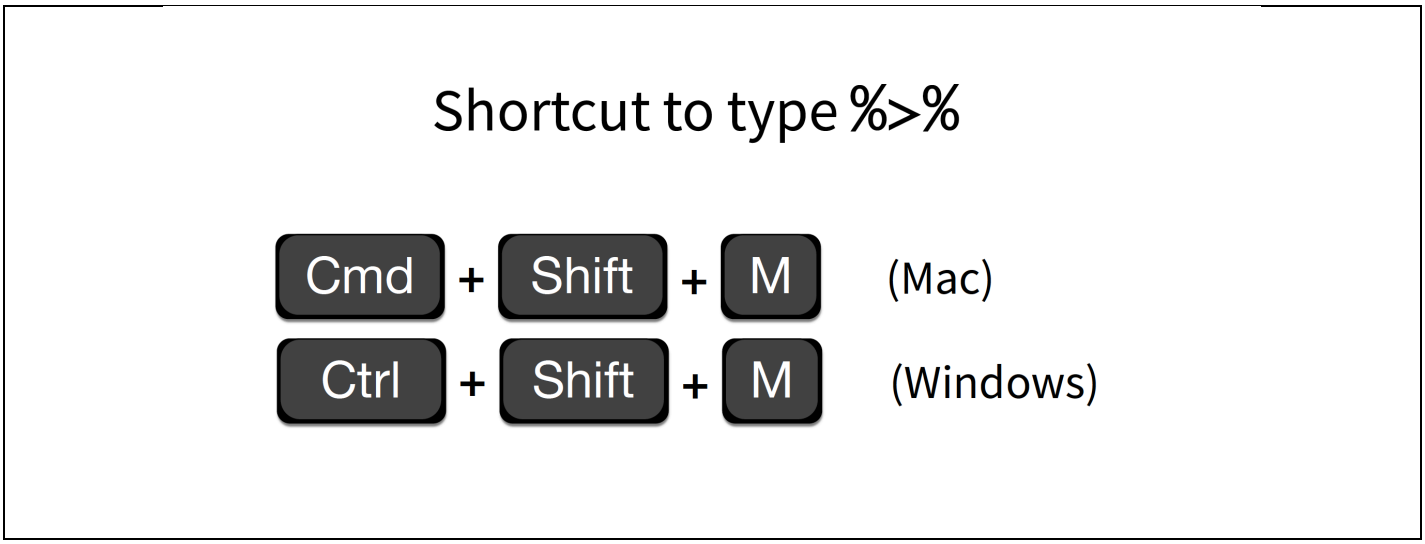
	Use for:
`ggplot2`	Data visualization
`dplyr`	Data manipulation
`tidyr`	Data cleaning
`readr`	Import/export rectangular data (e.g. ".xlsx", etc)
`purr`	Working with functions and vectors
`tibble`	Working with a reimagined dataframe
`stringr`	Working with strings
`forcats`	Working with factors

Design ..... **Data Collection** ..... **Data Management** ..... **Data Summarization** ..... **Statistical Analysis** ..... **Reporting**

`tidyverse` includes 11 <u>additional</u> packages. For any of these, you DO NEED to issue its own **`library( )`**
Example: **`library(haven)`**.

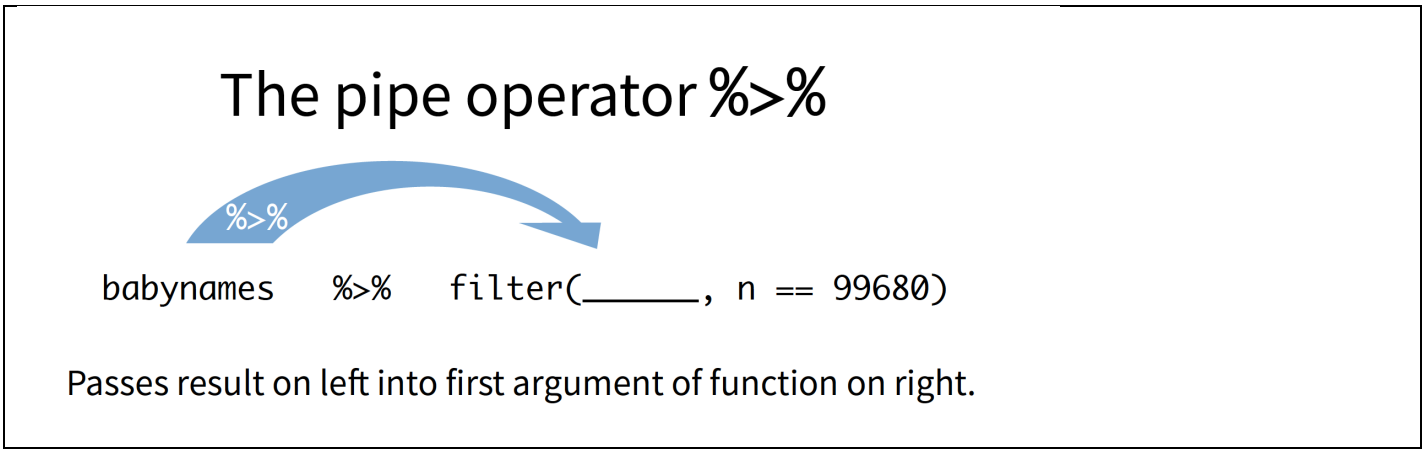
	Use for:
`haven`	Importing SAS, SPSS, Stata
`readxl`	Importing .xls and .xlsx
`lubridate`	Dates and times
`hms`	Times
`feather`	Sharing with Python and other languages
`httr`	Working with web apis
`jsonlite`	Working with JSON
`rvest`	Web scraping
`modelr`	Modeling
`broom`	Modeling
`Xml2`	XML

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



Introduction. The pipe operator **%>%** is a function in the package **dplyr** (recall – this is a component package of the **tidyverse** collection). The pipe operator creates a connected set of multiple operations in R into a single, ordered, *chain* of operations.

How it works. 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 #1: For x=3, compute $(x^2 + 15)/2$. Correct answer should be 12.

3 %>% raise_to_power(.,2) %>% add(.,15) %>% divide_by(.,2)	Think of **%>%** as saying to R ***"Then"*** Start with 3 as output. *Then* raise to power 2. *Then* add 15. *Then* Divide by 2.

```
> library(magrittr)
> 3 %>%
    raise_to_power(.,2) %>%
    add(.,15) %>%
    divide_by(.,2)

[1] 12

> # Check.
> answer <- (3^2 + 15)/2
> answer

[1] 12
```

Example #2: Obtain the mean value of the first 6 observations of variable `matage` in dataset = ivf

ivf$matage %>% **head() %>%** **mean() %>%**	Start with vector of data for matage. *Then* obtain the first 6 observations. *Then* compute the sample mean

```
> library(magrittr)
> ivf$matage %>%
    head() %>%
    mean()

[1] 33.83333

> # Check.
> answer <- mean(head(ivf$matage))
> answer

[1] 33.83333
```

6.3. Introduction to `dplyr`

The **`dplyr`** package is one of the component packages that comes bundled with **`tidyverse.`**

`dplyr` commands are terrific for some basics of data management:

`%>%`	Pipe operator (translation: "Then")
`filter( )`	Use to select observations (rows)
`select( )`	Use to select variables (columns)
`mutate( )`	Use to create new variables
`arrange( )`	Use to sort
`summary( )`	Use to collapse individual records into summaries

… stay tuned …

7. Working with Variables

7.1. Missing Values (NA)

Missing data, indicated by `NA`, means *not applicable.*
What can go wrong: R will not do calculations on vectors (variables) with missing data.

How to Work with Missing Values

IF YOU WANT:
Produce an error message if that tells me I have missing values
	`na.fail(dataframe)`
	
	Example:
	`> na.fail(ivf)`
	`Error in na.fail.default(ivf) : missing values in object`

IF YOU WANT:
Display observation #'s of observations with missing values on a particular variable
	`which(is.na(dataframe$variablename)`
	
	Example:
	`> which(is.na(ivf$hyp))`
	`[1]  21 167`

IF YOU WANT:
Compute function() with missing values removed
 	
Note: This works functions such as mean(), sd(), max(), etc.
	# Three Ways
	
	`mean(dataframe$variablename,na.rm=TRUE)`
	`mean(dataframe$variablename,na.rm=T)`
	`mean(na.omit(dataframe$variablename))`
	
	Example:
	`> mean(ivf$hyp,na.rm=TRUE)`
	`[1] 0.1392801`
	
	`> mean(ivf$hyp,na.rm=T)`
	`[1] 0.1392801`
	
	`> mean(na.omit(ivf$hyp))`
	`[1] 0.1392801`

- continued -

How to Work with Missing Values - continued

For single variable - CHARACTER: Replace every occurrence of the string missing value code "missing" with the missing value indicator NA	`myvariable[myvariable=="missing"] <- NA`

For single variable - NUMERIC:
Replace every occurrence of the numeric missing value code 9999 with the missing value indicator NA
	`for (i in 1:nrow(mydata$variable) {`
	`  if(mydata$variable[i]==9999 {`
	`  mydata$variable[i]=NA}`
	`  }`

For entire data set -NUMERIC:
Replace every occurrence of the numeric missing value code 9999 with the missing value indicator NA
	`for (i in 1:nrow(mydataframe) {`
	`  for (k in 1:ncol(mydataframe) {`
	`  if(mydataframe[i,k]==9999 {`
	`  mydataframe[i,k]=NA}`
	`  }`
	`  }`

7.2. Create a Numeric Variable

```
# Using Base Package
dataframe$newvariable <- function of dataframe$oldvariables

# Using command mutate() in package dplyr
library(tidyverse)
dataframe <- mutate(dataframe, newvar = function of oldvar)
```

```
> # Example
> library(tidyverse)
> library(stargazer)
> load(file="ivf.Rdata")

> # Using base package
> ivf$months <- ivf$gestwks/30

> # Using mutate() in package dplyr
> ivf <- mutate(ivf,months2=gestwks/30)

> # Check
> stargazer(ivf[c("months","months2")],type="text",summary.stat=c("n", "mean", "sd"))

============================
Statistic  N  Mean  St. Dev.
----------------------------
months    641 1.290  0.078
months2   641 1.290  0.078
----------------------------
```

7.3. Create a Character Variable

Creating a character variable involves: 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)

Recall this very handy chart - **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 making a comparison 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

```
dataframe$newvariable <- NA
dataframe$newvariable[condition_that_must_be_true] <- “charactervalue”
dataframe$newvariable[condition_that_must_be_true] <- “charactervalue”
etc.
```

```
> # Example

> ivf$ageg <- NA
> ivf$ageg[ivf$matage==23] <- "Youngest"
> ivf$ageg[(ivf$matage>23) & (ivf$matage <30)] <- "Twenty Something"
> ivf$ageg[(ivf$matage>30) & (ivf$matage <43)] <- "Middles"
> ivf$ageg[ivf$matage==43] <- "Oldest"

> # Check
> table(ivf$ageg)
        Middles          Oldest Twenty Something         Youngest
            510               3              90                2
```

7.4. Create a Factor Variable

Creating a factor variable is similar to creating a character variable. However, depending, ***it also involves setting explicitly the correspondence the integer storage that R uses to store your factor variable "under the hood".*** Thus, the steps now involve: 1) initializing to missing (NA); 2) coding the conditions ***in square brackets*** that define each level of your new character variable; 3) remembering to use quotes (except for NA); and finally 4) defining explicitly the integer storage that you want R to use.

```
newvariable <- NA

newvariable[condition_that_must_be_true] <- "charactervalue"
newvariable[condition_that_must_be_true] <- "charactervalue"

newvariable <- as.factor(newvariable)

namelevels <- c("level","level","level")
factor(newvariable, levels=namelevels)

table(newvariable)
```

```
> # Example
> ivf$ageg <- NA
> ivf$ageg[ivf$matage==23] <- "Youngest"
> ivf$ageg[(ivf$matage>23) & (ivf$matage <30)] <- "Twenty Something"
> ivf$ageg[(ivf$matage>30) & (ivf$matage <43)] <- "Middles"
> ivf$ageg[ivf$matage==43] <- "Oldest"
> ivf$ageg <- as.factor(ivf$ageg)

> # Default factor level assignment is ALPHABETIC
> table(ivf$ageg)

        Middles           Oldest Twenty Something         Youngest
            510                3               90                2

> # SOLUTION:  Set factor level assignment explicitly using created vector of labels
> mylevels <- c("Youngest","Twenty Something","Middles", "Oldest")
> ivf$ageg <- factor(ivf$ageg,levels=mylevels)
> table(ivf$ageg)

       Youngest Twenty Something          Middles           Oldest
              2               90             510                3
```

7.5. Create a 0/1 Variable

There are at least three (3) ways to do this. Caution! – Take care that you handle missing values correctly.

```
# METHOD I – Most explicit and easiest to understand
newvariable <- NA
newvariable[conditionfor0] <- 0
newvariable[conditionfor1] <- 1
newvariable <- as.numeric(newvariable)

# Method II – Utilizes 0/1 logical operator. Condition true is coded 1. Code false 0.
newvariable <- as.numeric(conditionfor1, na.rm=TRUE)


#  Method III – Utilizes "if/else".  If true is coded 1.   Else is coded 0
newvariable <- with(data=dataframe,ifelse(conditionfor1,1,0), na.rm=TRUE)
```

```
> # EXAMPLE:  Method I
> ivf$female01 <- NA
> ivf$female01[ivf$sex=="male"] <- 0
> ivf$female01[ivf$sex=="female"] <- 1
> ivf$female01 <- as.numeric(ivf$female01)
> table(ivf$sex,ivf$female01)

          0   1
    male 326   0
  female   0 315

> # EXAMPLE:  Method II
> ivf$female01 <- as.numeric(ivf$sex == "female", na.rm=TRUE)
> table(ivf$sex,ivf$female01)

          0   1
    male  326   0
  female   0 315

> # EXAMPLE:  Method III
> # KEY:  ifelse(CONDITION,VALUETRUE,VALUEFALSE)
> ivf$female01 <- with(data=ivf,ifelse(sex=="female",1,0), na.rm=TRUE)
> table(ivf$sex,ivf$female01)

          0   1
    male 326   0
  female   0 315
```

7.6. Create a Grouped Numeric Variable

Creating a grouped numeric variable is similar to creating a character variable.

```
newvariable <- NA
newvariable[condition_that_must_be_true] <- value
newvariable[condition_that_must_be_true] <- value
etc.
```

```
> # Example

> ivf$agen <- NA
> ivf$agen[ivf$matage==23] <- 1
> ivf$agen[(ivf$matage>23) & (ivf$matage <30)] <- 2
> ivf$agen[(ivf$matage>30) & (ivf$matage <43)] <- 3
> ivf$agen[ivf$matage==43] <- 4

> # Check
> class(ivf$agen)
[1] "numeric"

> table(ivf$agen)
  1   2   3   4
  2  90 510   3
```

7.7. Create a Quantiles Variable

Perhaps the simplest is to install (one time) the package **gtools** and use the function **quantcut( )**. But you could also do it using the base package. Here, however, you need to be sure to tell R to put the smallest observation into the first bin by including the option **include.lowest=TRUE**.

```
library(gtools)

# METHOD I:  With gtools package and command quantcut( ) and option labels=.
newvariable <- quantcut(sourcevariable, q=#, labels=c("label","label"),na.rm=TRUE)


# METHOD II:  With base package using seq( ) and labeling
newvariable <- with(data=dataframe, cut(sourcevariable,
                                breaks=quantile(sourcevariable, probs=seq(0,1, by=0.2),
na.rm=TRUE),
                                labels=c("Q1","Q2", "Q3", "Q4","Q5"),
                                include.lowest=TRUE))

# METHOD III:  With base package specifying cutoffs explicitly and option labels=
newvariable <- with(data=dataframe, cut(sourcevariable,
                                breaks=quantile(sourcevariable, probs=(c(0, .2, .4, .6, .8,
1)),
                                na.rm=TRUE),
                                include.lowest=TRUE))
```

```
> # Method I
> library(gtools)
> ivf$qage <- quantcut(ivf$matage, q=5, na.rm=TRUE)
> table(ivf$qage)
[23,31] (31,33] (33,35] (35,38] (38,43]
    166     112     123     163      77

> # Method II
> ivf$qage <- with(data=ivf, cut(matage,
                               breaks=quantile(matage, probs=seq(0,1, by=0.2), na.rm=TRUE),
                               labels=c("Q1","Q2", "Q3", "Q4","Q5"),
                               include.lowest=TRUE))

> table(ivf$qage)
 Q1  Q2  Q3  Q4  Q5
166 112 123 163  77

> # Method III
>  ivf$qage <- with(data=ivf, cut(matage,
                                 breaks=quantile(matage, probs=(c(0, .2, .4, .6, .8, 1)), na.rm=TRUE),
                                 include.lowest=TRUE))

> table(ivf$qage)
[23,31] (31,33] (33,35] (35,38] (38,43]
    166     112     123     163      77
```

7.8. Label a Variable

```
# METHOD I:  With base package using command names( )
names(dataframe)[columnnumber] <- "This is the label"

# METHOD II: with package Hmisc using command label( )
library(Hmisc)
label(variable) <- "This is the label"
```

```
> # Method I
> names(ivf)[5] <- “bmi: Body Mass Index”

> # Method II
> library(Hmisc)
> label(ivf$bmi) <- “bmi: Body Mass Index”
```

7.9. Label Factor Variable Values

```
variablename <- factor(variablename,
                                levels = c(value, value),
                                labels = c("label", "label"))
```

```
> surveydf$m_praise <- factor(surveydf$m_praise,
                              levels = c(0,1,2,3,4,".",".s"),
                              labels = c("never", "rarely", "sometimes","usually","always",".",".s"))
```

7.10. Delete A Variable

What could go wrong. In the option **`select =`**

1) Do ***NOT*** forget the minus sign
2) Do ***NOT*** enclose variable names in quotes!

```
newdataframe <- subset(sourcedataframe, select = -c(var1, var2))
```

```
> # Example
> ivfnew <- subset(ivf, select = -c(female01,qage)
```

8. Working with Observations

Working with observations (and variables, too, for that matter) can be done using the base package. Sometimes, however, it is easier (and easier to read!) to use the **`dplyr`** package in **`tidyverse`**

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 making a comparison 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

Recall. Key functions in `dplyr` package

`%>%`	Pipe operator (think "and then")
`filter( )`	Use to select observations (rows)
`select( )`	Use to select variables (columns)
`mutate( )`	Use to create new variables
`arrange( )`	Use to sort
`summary( )`	Use to collapse individual records into summaries

8.1. Numbering Observations

Suppose you want to number the observations in your data from 1 to n where n is your sample size.

Good to know about dataframes

nrow( ) – tells you the number of rows of your dataframe (translation: your sample size n).
ncol( ) – tells you the number of columns of your dataframe (translation: the # variables you have).

How to number observations from 1 to n:

```
newvariable <- seq(from=1, to=nrow(dataframename,by=1)
```

```
> lung_demo$studyid <- seq(from=1, to=nrow(lung_demo), by=1)
```

8.2. Selecting Observations

To select a subset of observations, you must tell R the condition that must be true for the observations to be included in the selection. To select a subset of the variables is easier; you simply tell R which variables to keep. **Encouraged –** Consider using the package **`dplyr`** and the pipe operator **`%>%`** if you have a complicated set of selections (the coding is much cleaner).

Using BASE package

How to select a subset of observations.

```
newdataframe <- subset(originaldataframe, condition)
```

How to select a subset of observations and select subset of variables.

```
newdataframe <- subset(originaldataframe, condition, select=c(varname,varname))
```

```
> # Select subset of observations from lung_demo.   Include if area=1
> lung_demonew <- subset(lung_demo, area==1)

> # Select subset of observations from lung_demo.  Include if area=1.  Keep variables id, area, mfvc
> lung_demonew <- subset(lung_demo, area==1, select=c(id,area,mfvc))
```

Using `dplyr` package

```
library(tidyverse)
```

How to select a subset of observations.

```
newdataframe <- filter(originaldataframe, condition)
```

How to select a subset of observations (first) and select subset of variables (second).

```
newdataframe <- filter(originaldataframe, condition, condition)) %>%
                select=c(varname,varname))
```

```
library(tidyverse)

> # Select a subset of observations from lung_demo.  Include if area=1 or area=2
lung_demonew <- filter(lung_demo,area %in% c(1,2))

# Select subset of observations.  Keep if (area=1 or 2) and if mfvc <300.  Keep variables id, area, mfvc
lung_demonew <- filter(lung_demo, area %in% c(1,2), mfvc <300) %>%
                select(id,area,mfvc)
```

8.3. Random Sampling of Your Data (# or %)

Using BASE package: Draw a random sample of size # using function `sample( )`

Random sample of # observations. Keep all variables.

```
newdataframe <- originaldataframe[sample(nrow((originaldataframe), #,  ]
```

Random sample of # Observations. Keep selected variables only.

```
newdataframe <- originaldataframe[sample(nrow((originaldataframe),#,c("var1", "var2")]
```

```
# Select a random sample of 15 observations. Keep all variables
lung_demonew <- lung_demo[sample(nrow(lung_demo), 15), ]

# Select a random sample of 15 observations. Keep a subset of variables:  Version 1
lung_demonew <- lung_demo[sample(nrow(lung_demo), 15),c("id", "area", "mfvc") ]

# Select a random sample of 15 observations. Keep a subset of variables:  Version 2 (less error prone?)
myvars <- c("id", "area", "mfvc")
lung_demonew <- lung_demo[sample(nrow(lung_demo), 15), myvars ]
```

Using `dplyr` package. Draw a random sample using function `sample_n( )` or `sample_frac( )`

```
library(tidyverse)
```

Random sample of # observations. Use option `replace=TRUE` to sample with replacement

```
newdataframe <- sample_n(originaldataframe, #)
```

Random sample of % of observations (first) and select subset of variables (second).

```
newdataframe <- sample_frac(originaldataframe, %) %>%
                select=c(varname,varname))
```

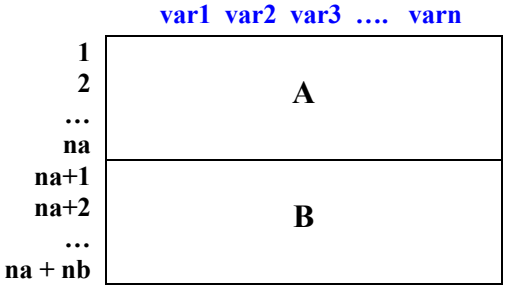
```
library(tidyverse)

# Select a random sample of 15 observations. Keep all variables
lung_demonew <- sample_n(lung_demo,15)

# Select a random sample of 20% of the obesrvations.  Keep a subset of variables
lung_demonew <- sample_frac(lung_demo, .20) %>%
                select(id,area,mfvc)
```

9. Working with Dataframes (Datasets)

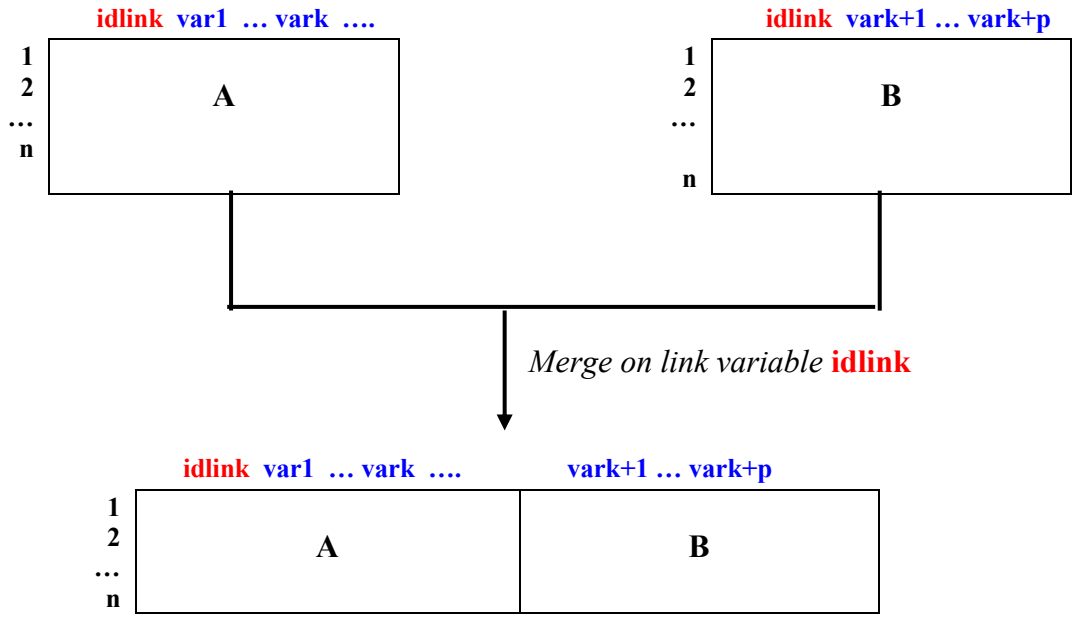
Append or Merge?

If you want …	Then what you want to do is to:
More subjects **More observations** of the same variables, this is a concatenation	**APPEND**
More variables **More data** on the same subjects, this is a merge	**MERGE**

Schematic of Append (sample size increases from na to (na + nb), number of variables is the same)



Schematic of Merge (sample size is the same, number of variables increases from k to (k+p)



Design ..... Data Collection ..... Data Management ..... Data Summarization ..... Statistical Analysis ..... Reporting

9.1. How to Append

Suppose you want to append ***additional observations*** of the ***same variables*** to a data set.

Use `dplyr` package and the command `bind_rows( )`

```
library(tidyverse)

newdataframe <- bind_rows(dataframe1, dataframe2)
```

```
library(tidyverse)

> dfa
  studyid last       age
  <chr>   <chr>    <dbl>
1 1       rice        45
2 2       stoddard    26
3 3       sulsky      50

> dfb
   studyid last       age
   <chr>   <chr>    <dbl>
1 4       crossley    31
2 5       wilson      18
3 6       piccini     39
4 7       valois      16

> newdata <- bind_rows(dfa,dfb)
> newdata
  studyid last       age
  <chr>   <chr>    <dbl>
1 1       rice        45
2 2       stoddard    26
3 3       sulsky      50
4 4       crossley    31
5 5       wilson      18
6 6       piccini     39
7 7       valois      16
```

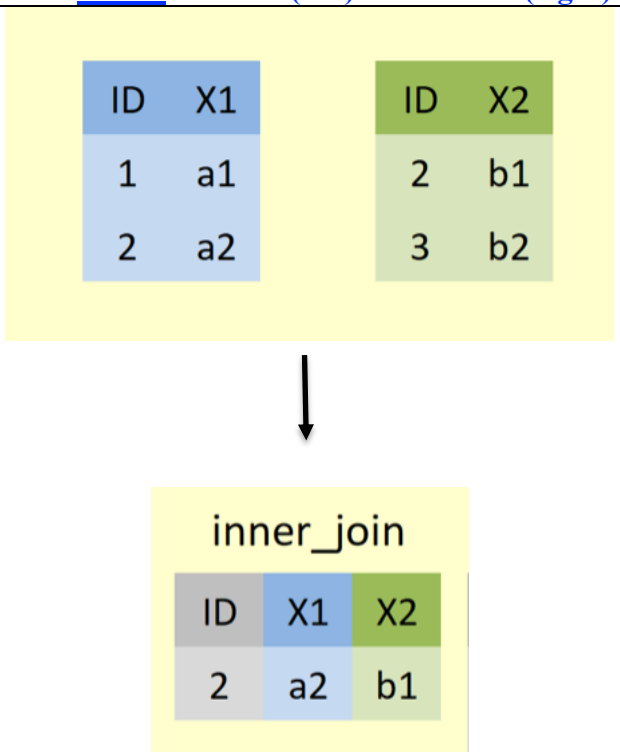
9.2. How to Merge

Suppose you want to add ***additional data (new variable values)*** of the ***same sample of n observations***.
Beware. Merging is more involved:

- Merging utilizes a linking variable (for example **`idlink`**, as on page 71). The idling variable must be a column in each data frame that is being merged; and
- There are multiple ways to merge and the choice depends on what you want in the merged result.
- Here we consider: **`inner_join( ), left_join( ), right_join( ),`** and **`full_join( )`**
- Not considered here are: **`semi_join( ),`** and **`anti_join( )`**

`inner_join( )`

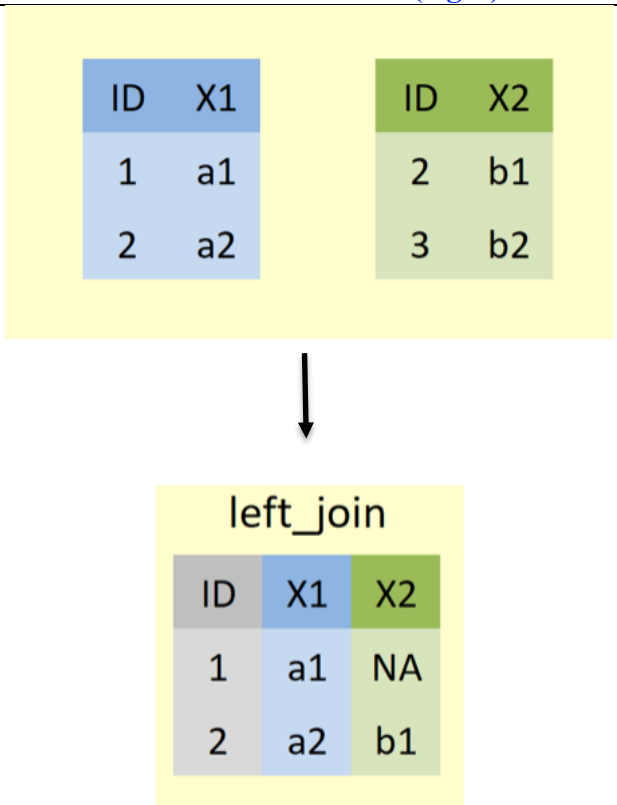
Returns: Observations must be in *BOTH* the first (left) and second (right) dataframes




```
inner_join(leftdataframe,rightdataframe, by="linkingvariable")
```

left_join( )

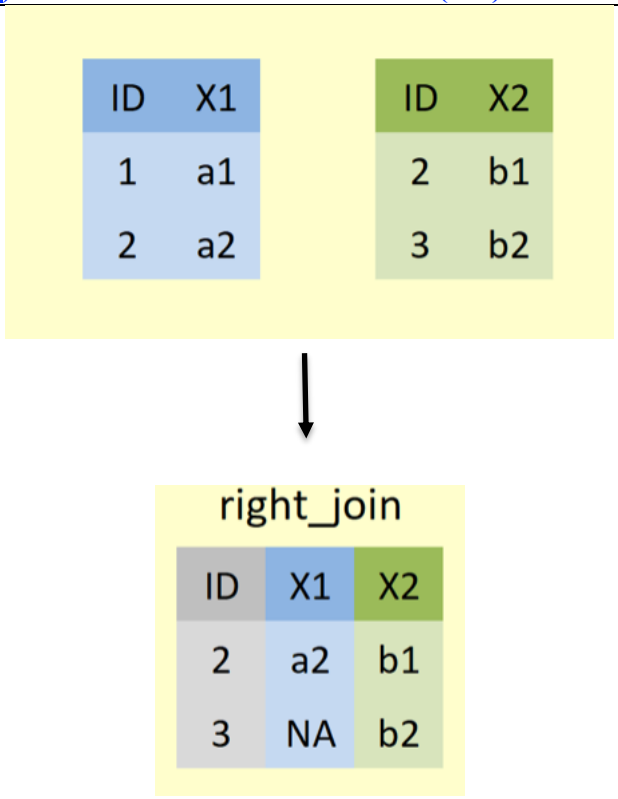
Returns: Every first (*left*) dataframe observation. Second (right) contributes MATCHES ONLY




```
left_join(leftdataframe,rightdataframe, by="linkingvariable")
```

right_join( )

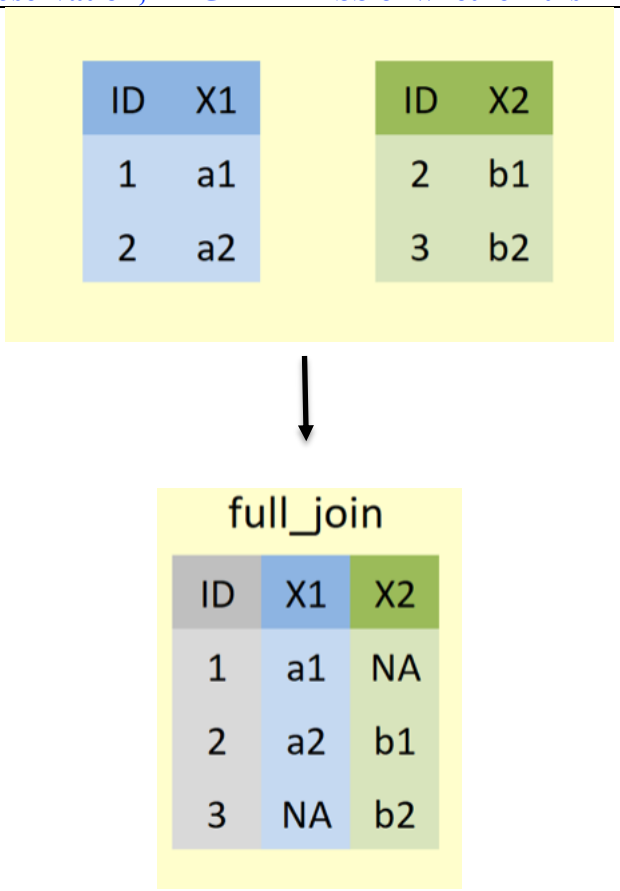
Returns: Every second (*right*) dataframe observation. First (left) contributes MATCHES ONLY




```
right_join(leftdataframe,rightdataframe, by="linkingvariable")
```

full_join( )

Returns: Every dataframe observation, REGARDLESS of whether it is in one or both dataframes




```
full_join(leftdataframe,rightdataframe, by="linkingvariable")
```

10. Some Basics of Data Screening

10.1. EDA: Exploratory Data Analysis – Look at Your Data!

Before doing anything, ***look at your data!!!*** Are the variables all there? Is the sample size right? Are the variable types what they should be? ***Look at the variable distributions.*** Do they make sense?

Commands for Looking at Your Data

{ Package } command	Example
Check structure of dataframe {base} str(dataframename)	str(Isoproterenol)
Obtain summary statistics of every variable {base} summary(dataframename)	summary(Isoproterenol)
Nicer looking summary stats of every variable {stargazer} stargazer(dataframename,type="text", median=TRUE)	library(stargazer) stargazer(Isoproterenol,type="text", median=TRUE) What could go wrong: If the input is not a dataframe (for example, it is a tibble), you will only get a header. Solution: dataframename <- as.data.frame(dataframename)
Show n and variable type for every variable {psych} print(describeFast(dataframename),short=FALSE)	library(psych) print(describeFast(Isoproterenol,short=FALSE)
Consider this really nice dataset summary. {summarytools} print(dfSummary(dataframename))	library(summarytools) print(dfSummary(Isoproterenol))

10.2. Assess Missing Values

{ Package } command	Example
Obtain number of missing values for each variable {base} colSums(is.na(dataframename))	colSums(is.na(Isoproterenol))
List idvar for observations that are missing on a particular variable {base} dataframe[!complete.cases(dataframename$varname),"idvar"]	Isoproterenol[!complete.cases(Isoproterenol$logdose),"id"]
List rows that have missing values on any variable {base} dataframe[!complete.cases(dataframe),]	Isoproterenol[!complete.cases(Isoproterenol),]
Create a new dataset that contains observations that are complete on EVERY variable newdata <- na.omit(sourcedata) Check newdata[!complete.cases(newdata),]	complete <- na.omit(Isoproterenol) complete[!complete.cases(complete),]

10.3. Range Checks

Suppose you want to obtain the minimum and maximum value for each variable and the number of missing values for each variable.

{ Package } command	Example
Obtain n, min, and max for each variable {stargazer} stargazer(dataframe, type="text", summary.stat=c("n", "min", "max")	library(stargazer) stargazer(lung_demo,type="text",summary.stat=c("n", "min", "max"))

10.4. Screen for Duplicates

{ Package } command	Example
SCREEN: Identify which rows (TRUE)are duplicates of earlier rows {base} duplicated(dataframe)	duplicated(toomuch)
FIX: Create a new dataframe that contains unique observations only (no duplicates) {base} datanew <- dataframe[!duplicated(dataframe),]	unique <- toomuch[!duplicated(toomuch),]

```
> # EXAMPLE
> toomuch
  id    name    age
  <chr> <chr> <dbl>
1 1     john     46
2 2     carol    31
3 3     bill     27
4 1     john     46
5 2     carol    31

> duplicated(toomuch)
[1] FALSE FALSE FALSE  TRUE  TRUE

> unique <- toomuch[!duplicated(toomuch), ]
> unique
  id    name    age
  <chr> <chr> <dbl>
1 1     john     46
2 2     carol    31
3 3     bill     27
```

10.5. Screen for Errors

There are lots of ways to do this *(I'm still learning…).*

Preliminary - Prior to your R session, make a plan/list of the error screens you want to do.

Examples -

#1: Print out **studyid** for specific variable whenever it has a **missing value**

#2. Print out the **studyid** and the **variable value** whenever the variable **value should be checked**

and so on …

Screen for Error
Suggested Approach
(there may be better approaches)

```
#1.  Initialize results of error screen to vector(s) with initial value = NA

#2.  Code an outer for loop that will processes your data, one observation at a time

#3.  Within the outer for loop, code an if-then loop that processes each observation

         -   If condition for error is NOT true, do nothing
         -   If condition for error IS true, append message to error screen vector(s)

#4.  All done processing?  Create dataframe that binds error screen vector(s).

#5.  Print.


errorvector1 <- NA
errorvector2 <- NA

for (i in 1:nrow(sourcedata))
   { if (errorcondition is TRUE)
         {    errorvector1 <- c(errorvector1, message1)
              errorvector2 <- c(errorvector2, message2)
         }
   {

errordf <- data.frame(errorvector1, errorvector2)

(errordf)
```

Example -

Single variable – Identify and display study id for observations with variable value = NA

Yields single row (missing_age) of study ids

Condition used to detect missing: `!complete.cases( ) = TRUE`

```
# Input source data.  Echo.
playdata <- read_excel("playdata.xlsx")
playdata

     id   age
  <dbl> <dbl>
1     1    21
2     2    NA
3     3    54
4     4    33
5     5    17
6     6    82
7     7    91
8     8    NA

# Initialize error screen vector to missing
missing_age <- NA

#  Outer for loop processes observations.  Inner if loop assesses and appends to error screen vector.

for (i in 1:nrow(playdata))
{  if (!complete.cases(playdata$age[i]))
     { missing_age <- c(missing_age, playdata$id[i])
     }
}

#  Display results of error check.
print("Error Check:  Study ID's for observations with age = NA")
[1] "Error Check:  Study ID's for observations with age = NA"

print(missing_age)
[1] NA  2  8
```

Example - Single variable – Identify and display study id for observations with variable value = NA
Yields listing of study ids, one row at a time.
Condition used to detect missing: `!complete.cases( ) = TRUE`

```
# Input source data.  Echo.
playdata <- read_excel("playdata.xlsx")
playdata

     id   age
  <dbl> <dbl>
1     1    21
2     2    NA
3     3    54
4     4    33
5     5    17
6     6    82
7     7    91
8     8    NA

# No error vector created and initialized to NA in this example

# For readability, print out a title
print("Error Check:  Study ID's for observations with age = NA")

#  Outer for loop processes observations.
#  Inner if loop evaluates error condition.  If error condition TRUE, print id

for (i in 1:nrow(playdata))
{    if (!complete.cases(playdata$age[i]))
      { print(playdata$id[i])
      }
}

[1] "Error Check:  Study ID's for observations with age = NA"
[1] 2
[1] 8
```

Example - Single variable – Identify and display study id *and* variable value when variable value is problem

Note - This works on NON missing observations ONLY

Therefore, must code 2 conditions:

(1) Observation of age must be non-missing: `complete.cases( ) = TRUE` ***and***

(2) Value of variable is a problem: `age > 65 = TRUE`

```
# Input source data.  Echo.
playdata <- read_excel("playdata.xlsx")
playdata

     id   age
  <dbl> <dbl>
1     1    21
2     2    NA
3     3    54
4     4    33
5     5    17
6     6    82
7     7    91
8     8    NA

#  Initialize 2 vectors in this example
id_extreme <- NA
age_extreme <- NA

#  Outer for loop processes observations.
#  Inner if loop evaluates error condition.  If TRUE, append to both id_extreme and age_extreme.
for (i in 1:nrow(playdata))
{  if (complete.cases(playdata$age[i]) & playdata$age[i] > 65)
   { id_extreme <- c(id_extreme,playdata$id[i])
     age_extreme <- c(age_extreme, playdata$age[i])
   }
}

error_age <- data.frame(id_extreme,age_extreme)
print("Error Check:  Study ID's and values of age for observations with age > 65")
[1] "Error Check:  Study ID's and values of age for observations with age > 65"

print(error_age)

  id_extreme age_extreme
1         NA          NA
2          6          82
3          7          91
```