What to expect

- Understanding the basics of functions
- Get familiar with help documentation
- Working with packages
- R Scripts and loading files
- Data Wrangling with `tidyverse`

Agenda

- Understanding the basics of functions
- Get familiar with help documentation
- Working with packages
- R Scripts and loading files
- Data Wrangling with `tidyverse`

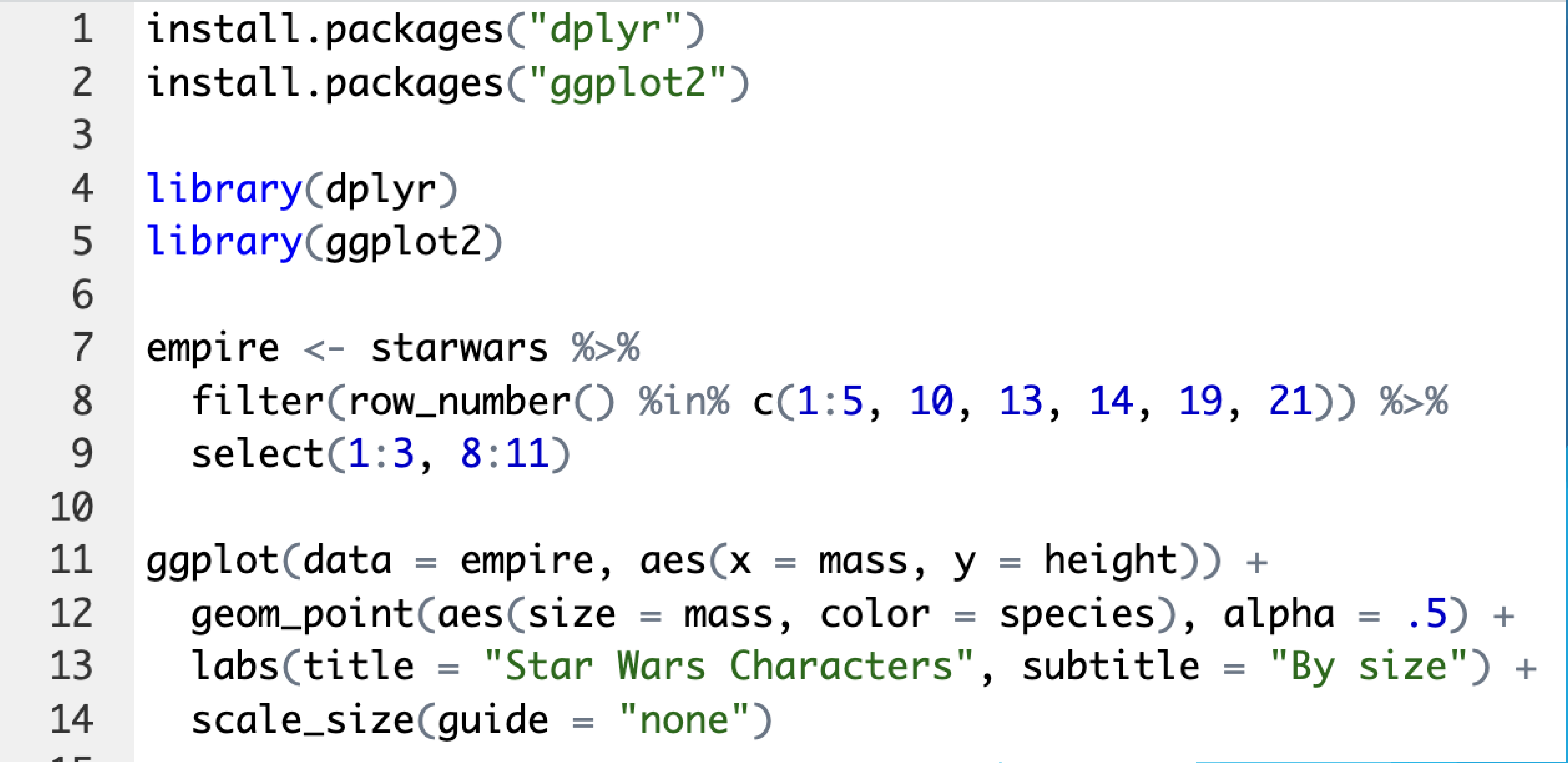
What to expect

- You will NOT become an expert overnight
- Learning to snowboard…
- Ask questions!
- **Make mistakes**
- When in doubt, Google

What you need

- R and RStudio installed *[raise hand if you have trouble with this]*
- An internet connection *[only for part of the time]*
- **Resiliency**

Type this code EXACTLY



```r
install.packages("dplyr")
install.packages("ggplot2")

library(dplyr)
library(ggplot2)

empire <- starwars %>%
  filter(row_number() %in% c(1:5, 10, 13, 14, 19, 21)) %>%
  select(1:3, 8:11)

ggplot(data = empire, aes(x = mass, y = height)) +
  geom_point(aes(size = mass, color = species), alpha = .5) +
  labs(title = "Star Wars Characters", subtitle = "By size") +
  scale_size(guide = "none")
```

Agenda

- Understanding the basics of **functions**
- Get familiar with help documentation
- Working with packages
- R Scripts and loading files
- Data Wrangling with `tidyverse`

Functions

- Sometimes, you want to do more than add or multiply variables.

- To perform more complicated actions, use ***functions***.
  - Functions are commands that describe, manipulate or analyze objects
  - This is why we use R! No one wants to calculate a regression by hand…

Functions have three parts

1. Function name
   - *Ex:* `log(10)`

   ```
   [1] 2.302
   ```

2. Arguments
   - *Ex:* `log(10)`

   ```
   [1] 2.302
   ```

3. Output
   - *Ex:* `log(10)`

   ```
   [1] 2.302
   ```

Each function has one and only one name.

Functions have three parts

1. Function name
   - Ex: `log(10)`

   ```
   [1] 2.302
   ```

2. Arguments
   - Ex: `log(10)`

   ```
   [1] 2.302
   ```

3. Output
   - Ex: `log(10)`

   ```
   [1] 2.302
   ```

One argument is always specified: the input. This is the object that the function acts on.

Other arguments control **how** the function acts. For example, do you want the natural log? Or log base 10?

Each function has defaults for its arguments. You should know what those are and how to change them.

Functions have three parts

1. Function name
   - Ex: `log(10)`

     ```
     [1] 2.302
     ```

2. Arguments
   - Ex: `log(10)`

     ```
     [1] 2.302
     ```

3. Output
   - Ex: `log(10)`

     ```
     [1] 2.302
     ```

Output can be a:
  number/integer
  a TRUE/FALSE statement
  a character value

Output can be a:
  single value
  vector
  data frame
  matrix
  list

You can store the output by assigning it to another object!

Mathematical functions

`sqrt()`	square root
`round()`	round a number
`log()`	logarithm
`exp()`	exponentiation
`abs()`	absolute value

```
sqrt(85)
[1] 9.219544
log(100)
[1] 4.60517
```

Functions you'll use a lot!

`c()` - combine or concatenate

`length()` - find out how long a vector is (this is the same as getting the last position)

`factor()` - change a character vector into a factor vector (is there meaning? Ex: Treatment vs. Control, Male vs. Female, Session 1 vs. Session 2)

`table()` - really nice for getting quick counts (Ex: how many males and females are there?)

`cbind()` and `rbind()` - add a vector to an existing data.frame. `cbind()` adds a new column. `rbind()` adds a new row.

Multiple arguments

Most functions take more than one argument.

Separate arguments with commas.

```
round (x = 5.86921, digits = 3)
[1] 5.869
```

Number that needs to be rounded.

Multiple arguments

Most functions take more than one argument.

Separate arguments with commas.

```
round (x = 5.86921, digits = 3)
[1] 5.869
```

Number of digits
to round to.

Arguments have Names

Most arguments in functions have names.

USE THE NAMES!!!

```
round (x = 5.86921, digits = 3)
[1] 5.869

round (digits = 3, x = 5.86921)
[1] 5.869

round (5.86921, 3)
[1] 5.869

round (3, 5.86921)XXXXXXXX
[1] 3
```

Exercise

1. Use the `seq()` function to list numbers 0 to 100.

   Arguments:

   - `from` = starting value of sequence
   - `to` = end value of sequence

2. Use the `seq()` function to list numbers 0 to 100, by intervals of 10.

   Arguments:

   - `from` = starting value of sequence
   - `to` = end value of sequence
   - `by` = increment of the sequence

Exercise

1. Use the `seq()` function to list numbers 0 to 100

   - `seq(from = 0, to = 100)`

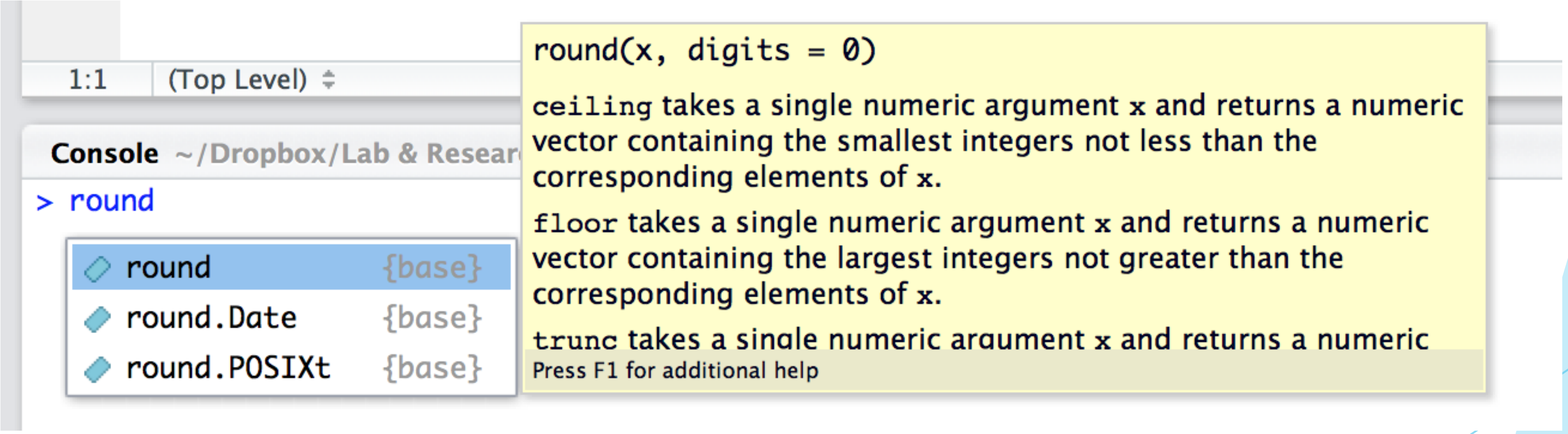
2. Use the `seq()` function to list numbers 0 to 100, by intervals of 10

   - `seq(from = 0, to = 100, by = 10)`

Great, but how do I know what the arguments are for a function?

Two ways:

1) In RStudio, press the **tab** key to see names of arguments and descriptions.



```
> round
round         {base}
round.Date    {base}
round.POSIXt  {base}
```

round(x, digits = 0)

ceiling takes a single numeric argument x and returns a numeric vector containing the smallest integers not less than the corresponding elements of x.

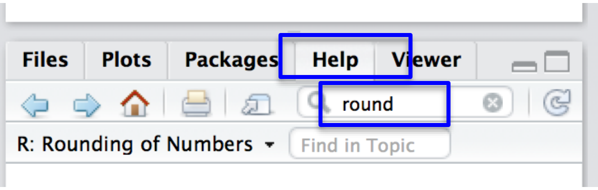
floor takes a single numeric argument x and returns a numeric vector containing the largest integers not greater than the corresponding elements of x.

trunc takes a single numeric argument x and returns a numeric

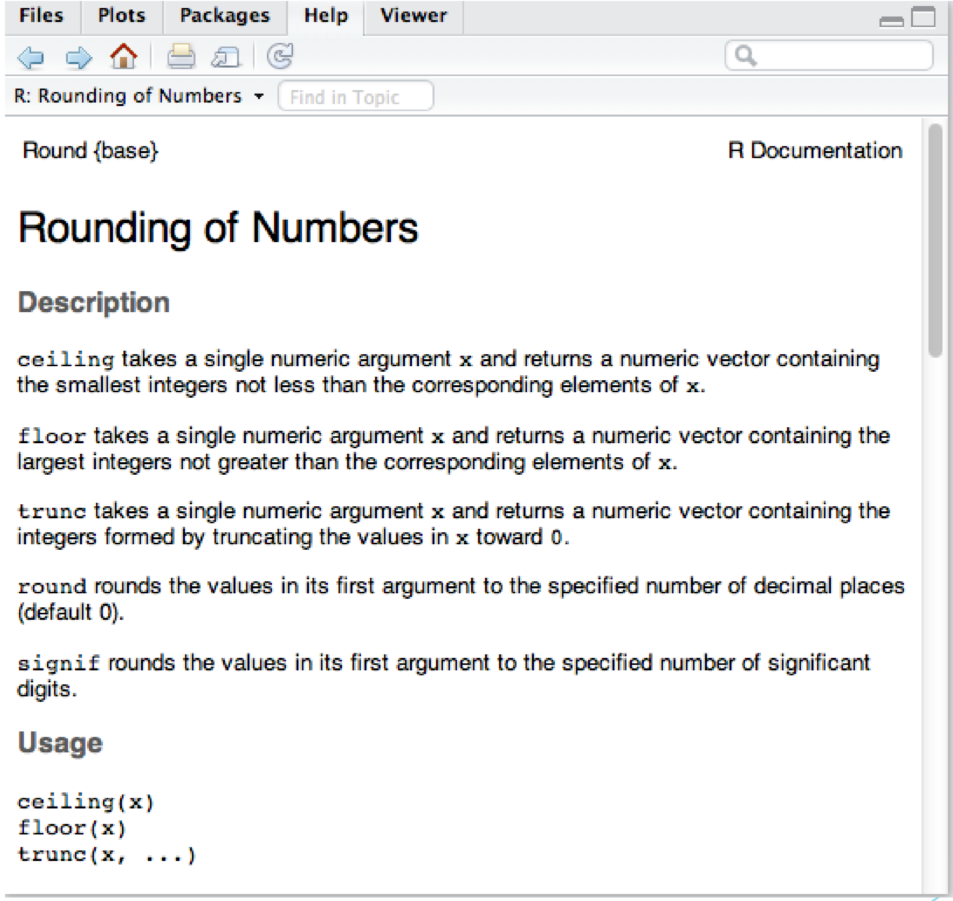
Press F1 for additional help

2) Look in the R documentation!

- Go to Help tab



- Or just type `?round` into the console



Round {base}

R Documentation

Rounding of Numbers

Description

`ceiling` takes a single numeric argument `x` and returns a numeric vector containing the smallest integers not less than the corresponding elements of `x`.

`floor` takes a single numeric argument `x` and returns a numeric vector containing the largest integers not greater than the corresponding elements of `x`.

`trunc` takes a single numeric argument `x` and returns a numeric vector containing the integers formed by truncating the values in `x` toward `0`.

`round` rounds the values in its first argument to the specified number of decimal places (default 0).

`signif` rounds the values in its first argument to the specified number of significant digits.

Usage

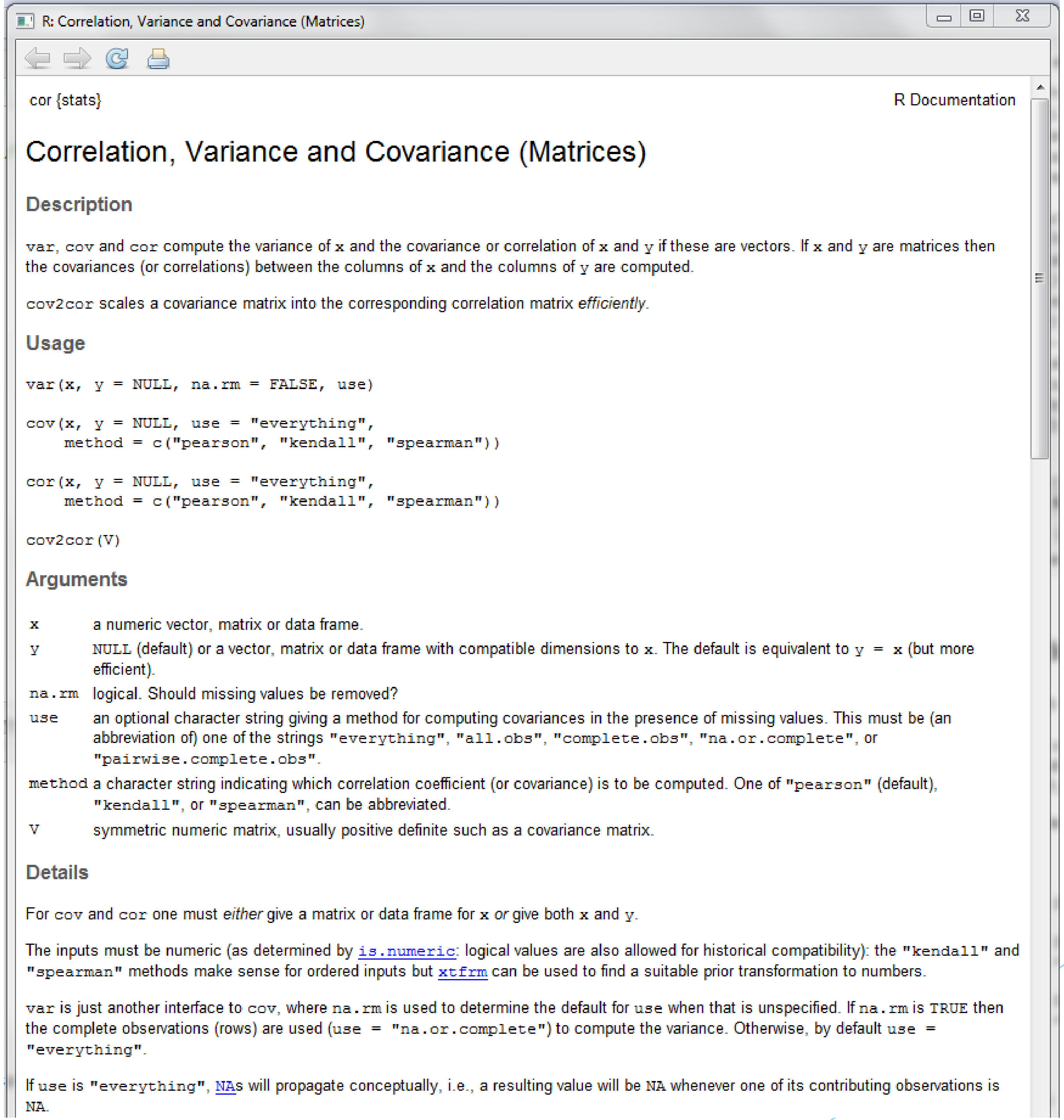
```
ceiling(x)
floor(x)
trunc(x, ...)
```

Agenda

- Understanding the basics of functions
- Get familiar with **help documentation**
- Working with packages
- R Scripts and loading files
- Data Wrangling with `tidyverse`

Help Documentation Format

Type code to look up the R documentation for the correlation function, `cor`



cor {stats}

R Documentation

Correlation, Variance and Covariance (Matrices)

Description

`var`, `cov` and `cor` compute the variance of `x` and the covariance or correlation of `x` and `y` if these are vectors. If `x` and `y` are matrices then the covariances (or correlations) between the columns of `x` and the columns of `y` are computed.

`cov2cor` scales a covariance matrix into the corresponding correlation matrix *efficiently*.

Usage

```
var(x, y = NULL, na.rm = FALSE, use)

cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cov2cor(V)
```

Arguments

`x`	a numeric vector, matrix or data frame.
`y`	`NULL` (default) or a vector, matrix or data frame with compatible dimensions to `x`. The default is equivalent to `y = x` (but more efficient).
`na.rm`	logical. Should missing values be removed?
`use`	an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings `"everything"`, `"all.obs"`, `"complete.obs"`, `"na.or.complete"`, or `"pairwise.complete.obs"`.
`method`	a character string indicating which correlation coefficient (or covariance) is to be computed. One of `"pearson"` (default), `"kendall"`, or `"spearman"`, can be abbreviated.
`V`	symmetric numeric matrix, usually positive definite such as a covariance matrix.

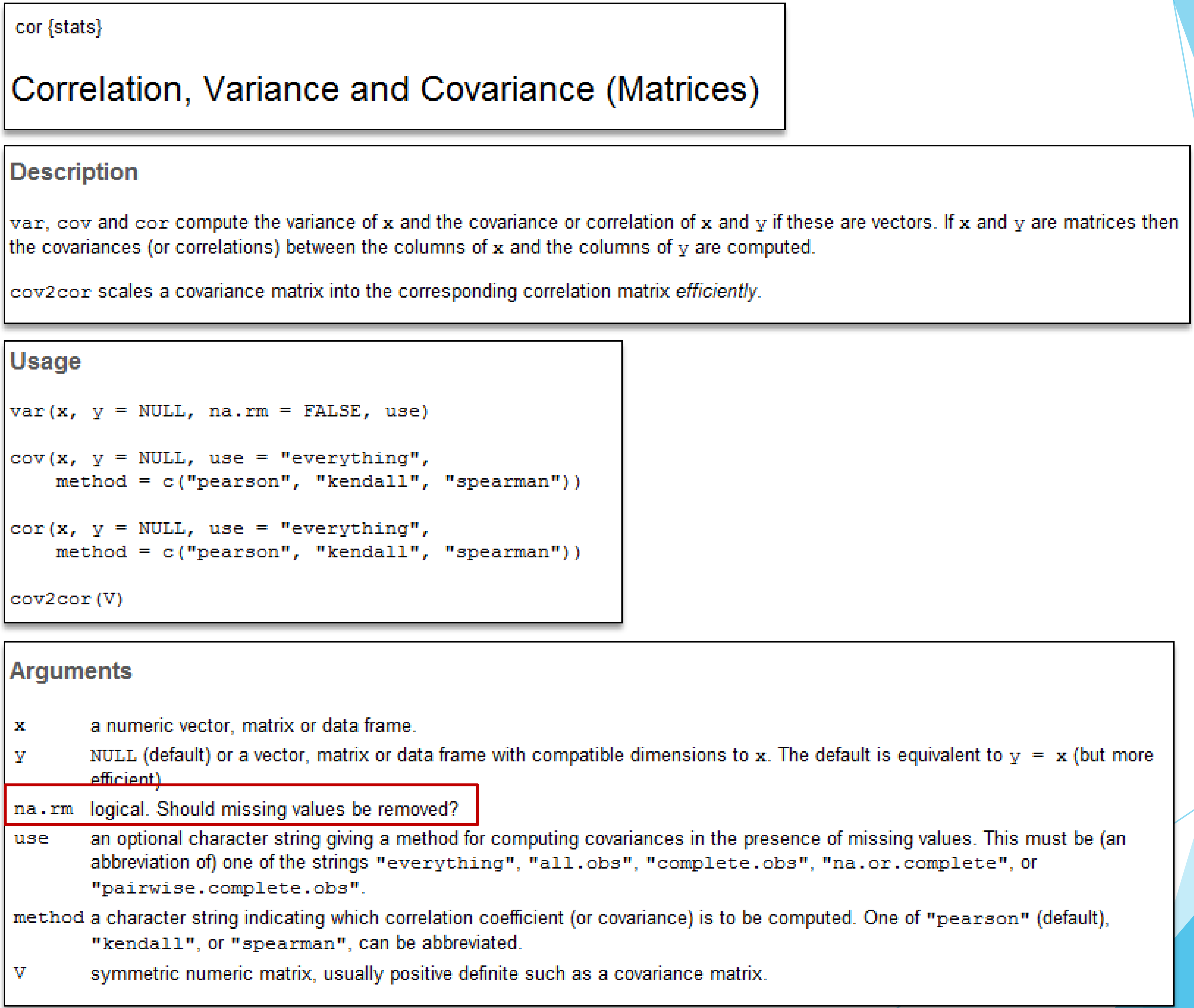
Details

For `cov` and `cor` one must *either* give a matrix or data frame for `x` *or* give both `x` and `y`.

The inputs must be numeric (as determined by `is.numeric`: logical values are also allowed for historical compatibility): the `"kendall"` and `"spearman"` methods make sense for ordered inputs but `xtfrm` can be used to find a suitable prior transformation to numbers.

`var` is just another interface to `cov`, where `na.rm` is used to determine the default for `use` when that is unspecified. If `na.rm` is `TRUE` then the complete observations (rows) are used (`use = "na.or.complete"`) to compute the variance. Otherwise, by default `use = "everything"`.

If `use` is `"everything"`, `NA`s will propagate conceptually, i.e., a resulting value will be `NA` whenever one of its contributing observations is `NA`.



cor {stats}

Correlation, Variance and Covariance (Matrices)

Description

`var`, `cov` and `cor` compute the variance of `x` and the covariance or correlation of `x` and `y` if these are vectors. If `x` and `y` are matrices then the covariances (or correlations) between the columns of `x` and the columns of `y` are computed.

`cov2cor` scales a covariance matrix into the corresponding correlation matrix *efficiently*.

Usage

```
var(x, y = NULL, na.rm = FALSE, use)

cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cov2cor(V)
```

Arguments

`x` a numeric vector, matrix or data frame.

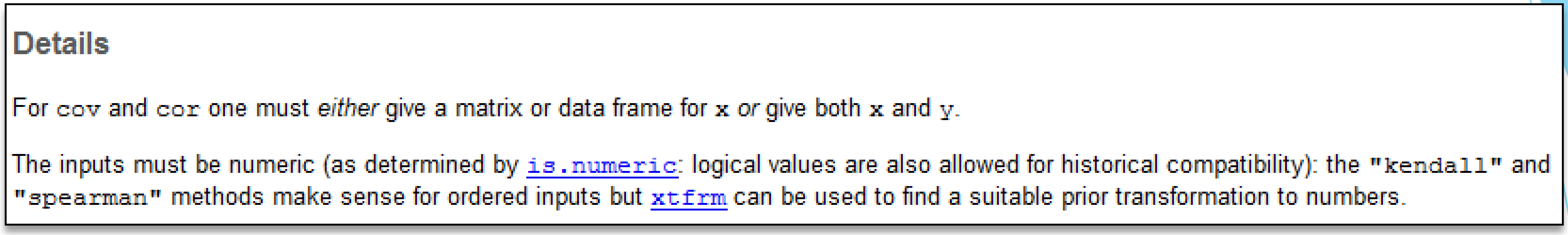
`y` `NULL` (default) or a vector, matrix or data frame with compatible dimensions to `x`. The default is equivalent to `y = x` (but more efficient).

`na.rm` logical. Should missing values be removed?

`use` an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings `"everything"`, `"all.obs"`, `"complete.obs"`, `"na.or.complete"`, or `"pairwise.complete.obs"`.

`method` a character string indicating which correlation coefficient (or covariance) is to be computed. One of `"pearson"` (default), `"kendall"`, or `"spearman"`, can be abbreviated.

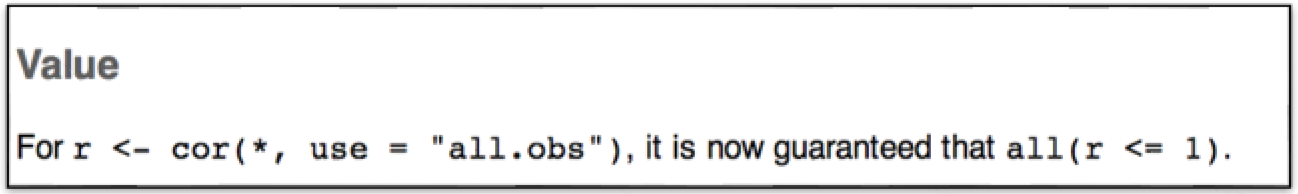
`V` symmetric numeric matrix, usually positive definite such as a covariance matrix.



Details

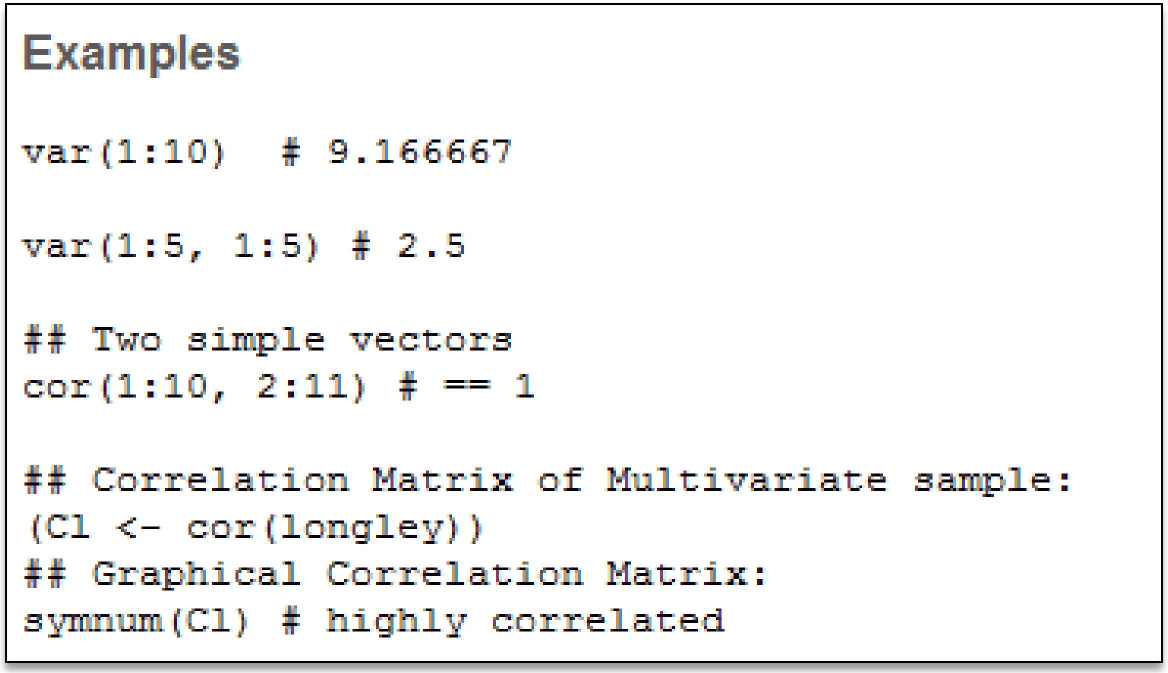
For cov and cor one must *either* give a matrix or data frame for x *or* give both x and y.

The inputs must be numeric (as determined by is.numeric: logical values are also allowed for historical compatibility): the "kendall" and "spearman" methods make sense for ordered inputs but xtfrm can be used to find a suitable prior transformation to numbers.



Value

For r <- cor(*, use = "all.obs"), it is now guaranteed that all(r <= 1).



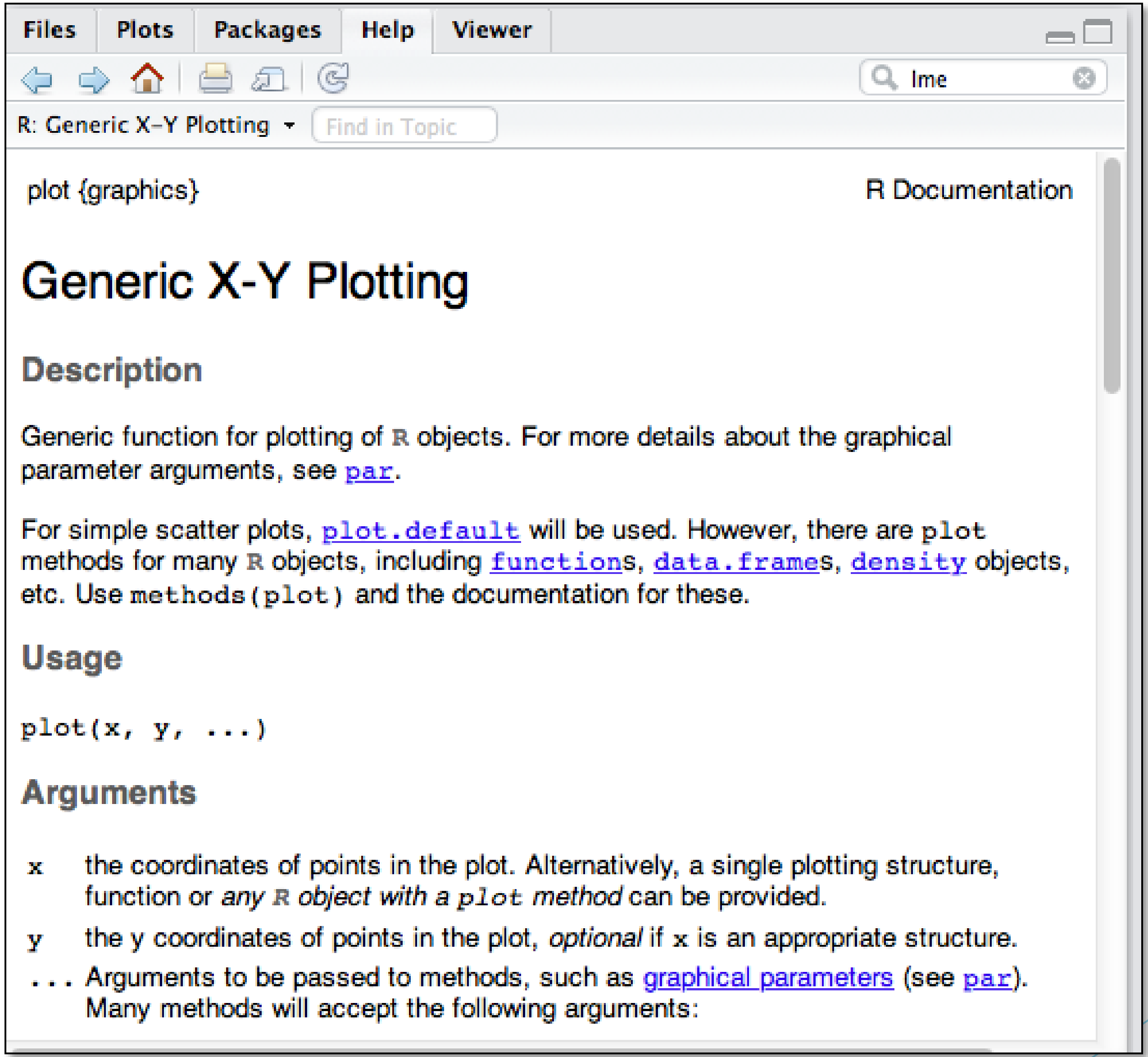
Examples

```
var(1:10)  # 9.166667

var(1:5, 1:5) # 2.5

## Two simple vectors
cor(1:10, 2:11) # == 1

## Correlation Matrix of Multivariate sample:
(Cl <- cor(longley))
## Graphical Correlation Matrix:
symnum(Cl) # highly correlated
```

Exercise

1. Look up documentation for `scale` and `plot`.
2. Using the `height` variable from our `empire` data.frame, make a new variable called `height_z`, using the `scale` function.
   - **Don't worry** if it appears directly underneath "empire_mini" – it is still correct. We will explain.
3. Do the same thing for `mass`.
4. Make a new `data.frame()` that only contains the `height_z` vector and `mass_z` vector. Store this new data.frame as `empire_z`.
5. Make a scatter plot of standardized height (*hint: y-axis*) by standardized mass, using the `plot` function.
6. Add a title to your plot.
7. Add labels to the x and y axes.

What happens if you add `type = "l"` ?
What is the default for type?



plot {graphics} R Documentation

Generic X-Y Plotting

Description

Generic function for plotting of `R` objects. For more details about the graphical parameter arguments, see `par`.

For simple scatter plots, `plot.default` will be used. However, there are `plot` methods for many `R` objects, including `function`s, `data.frame`s, `density` objects, etc. Use `methods(plot)` and the documentation for these.

Usage

```
plot(x, y, ...)
```

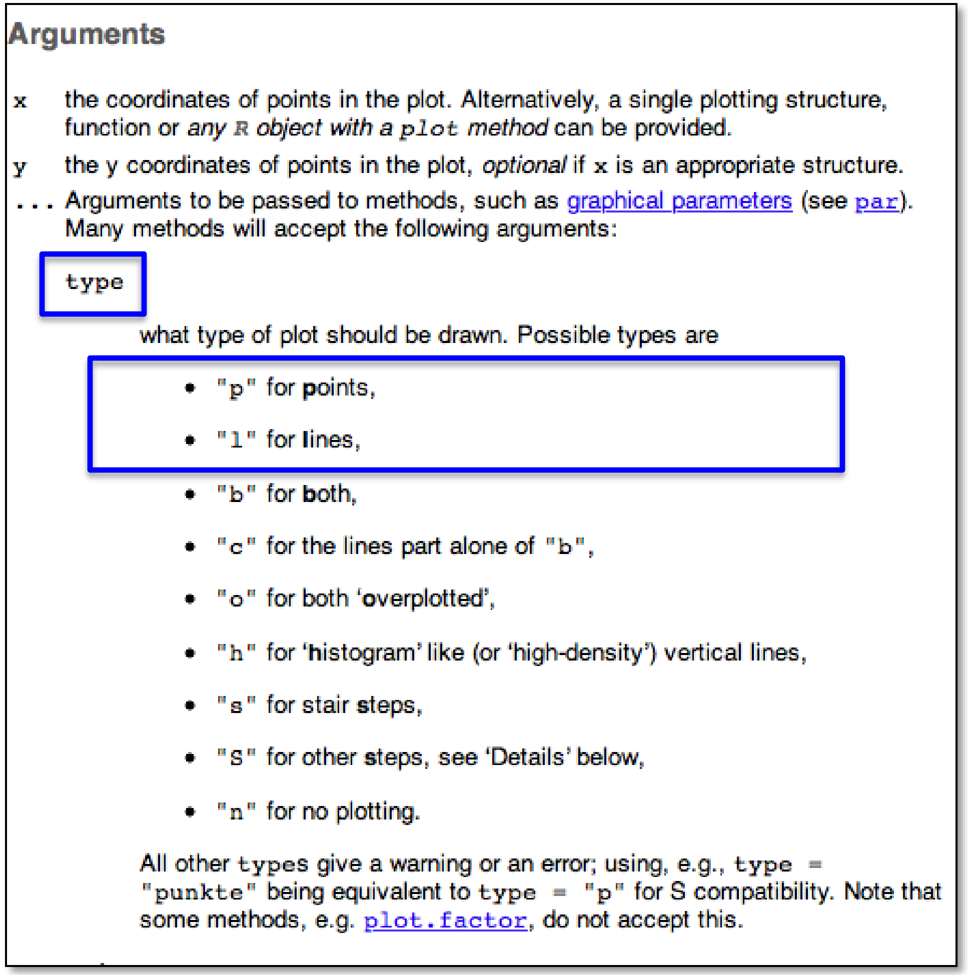
Arguments

`x` the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any* `R` *object with a* `plot` *method* can be provided.

`y` the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.

`...` Arguments to be passed to methods, such as graphical parameters (see `par`). Many methods will accept the following arguments:

Back to the Documentation!



Arguments

`x` the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any `R` object with a `plot` method* can be provided.

`y` the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.

`...` Arguments to be passed to methods, such as graphical parameters (see `par`). Many methods will accept the following arguments:

`type`

what type of plot should be drawn. Possible types are

- `"p"` for **p**oints,
- `"l"` for **l**ines,
- `"b"` for **b**oth,
- `"c"` for the lines part alone of `"b"`,
- `"o"` for both '**o**verplotted',
- `"h"` for '**h**istogram' like (or 'high-density') vertical lines,
- `"s"` for stair **s**teps,
- `"S"` for other **s**teps, see 'Details' below,
- `"n"` for no plotting.

All other `types` give a warning or an error; using, e.g., `type = "punkte"` being equivalent to `type = "p"` for S compatibility. Note that some methods, e.g. `plot.factor`, do not accept this.

Recap of Using Variables

- Most of the time, we want to do more than just add, subtract etc.
- We want to act on our variables. We do this with ***functions***.
  - *Each function* has a unique name
  - *Each function* requires some input, and the function can be modified using arguments
  - *Each function* will produce an output

Objects = subjects/nouns

Functions = verbs

Arguments = adverbs

Recap of Using Variables

- Where to find functions?
  - Some exist in R by default

```
t.test()
cor()
scale()
```

- Lots of people around the world write their own functions, and think it’s useful to share these with us!

Agenda

- Understanding the basics of functions
- Get familiar with help documentation
- Working with **packages**
- R Scripts and loading files
- Data Wrangling with `tidyverse`

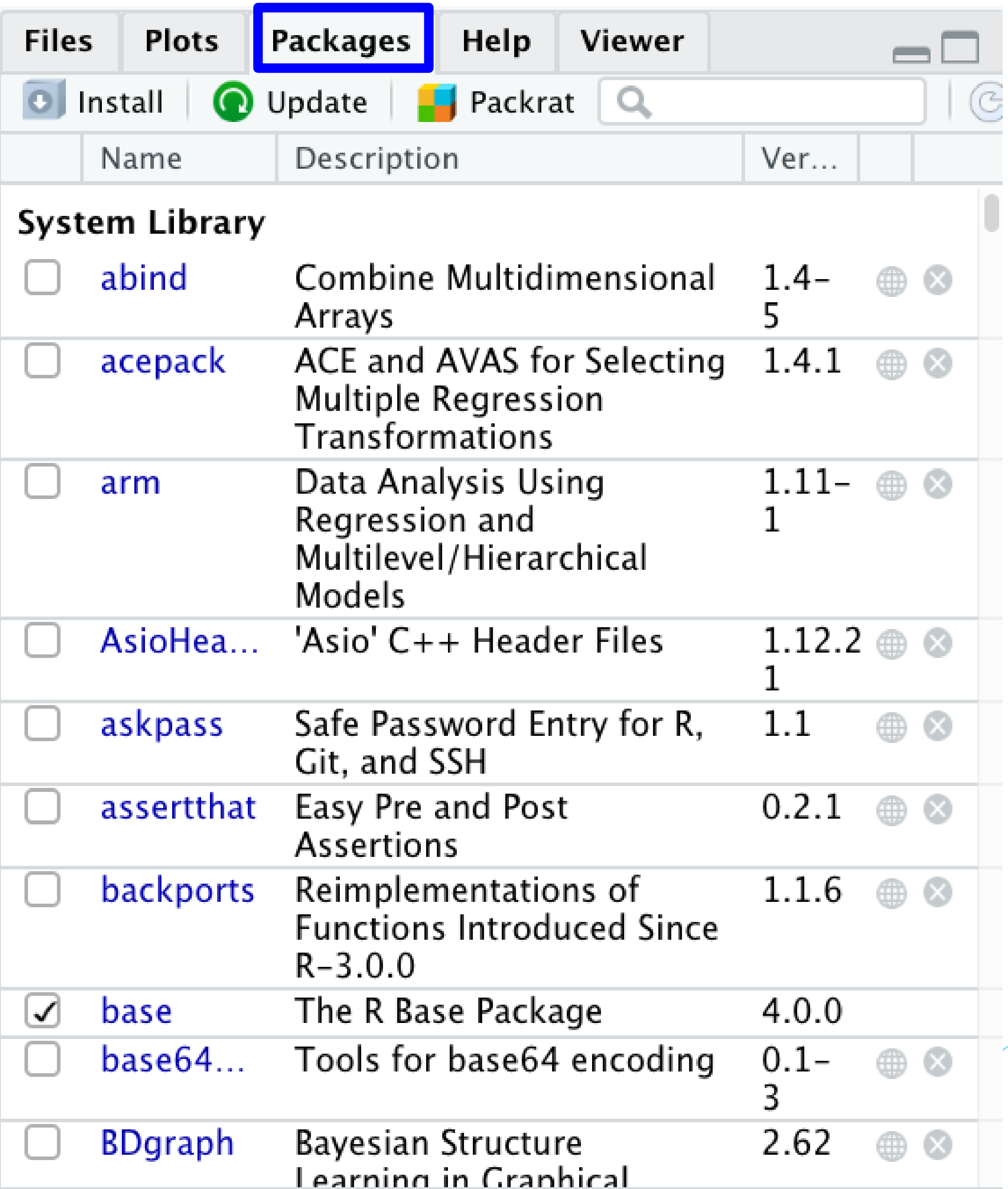
Packages

What is a package?

- A collection of functions and datasets.
- Open source (free!)

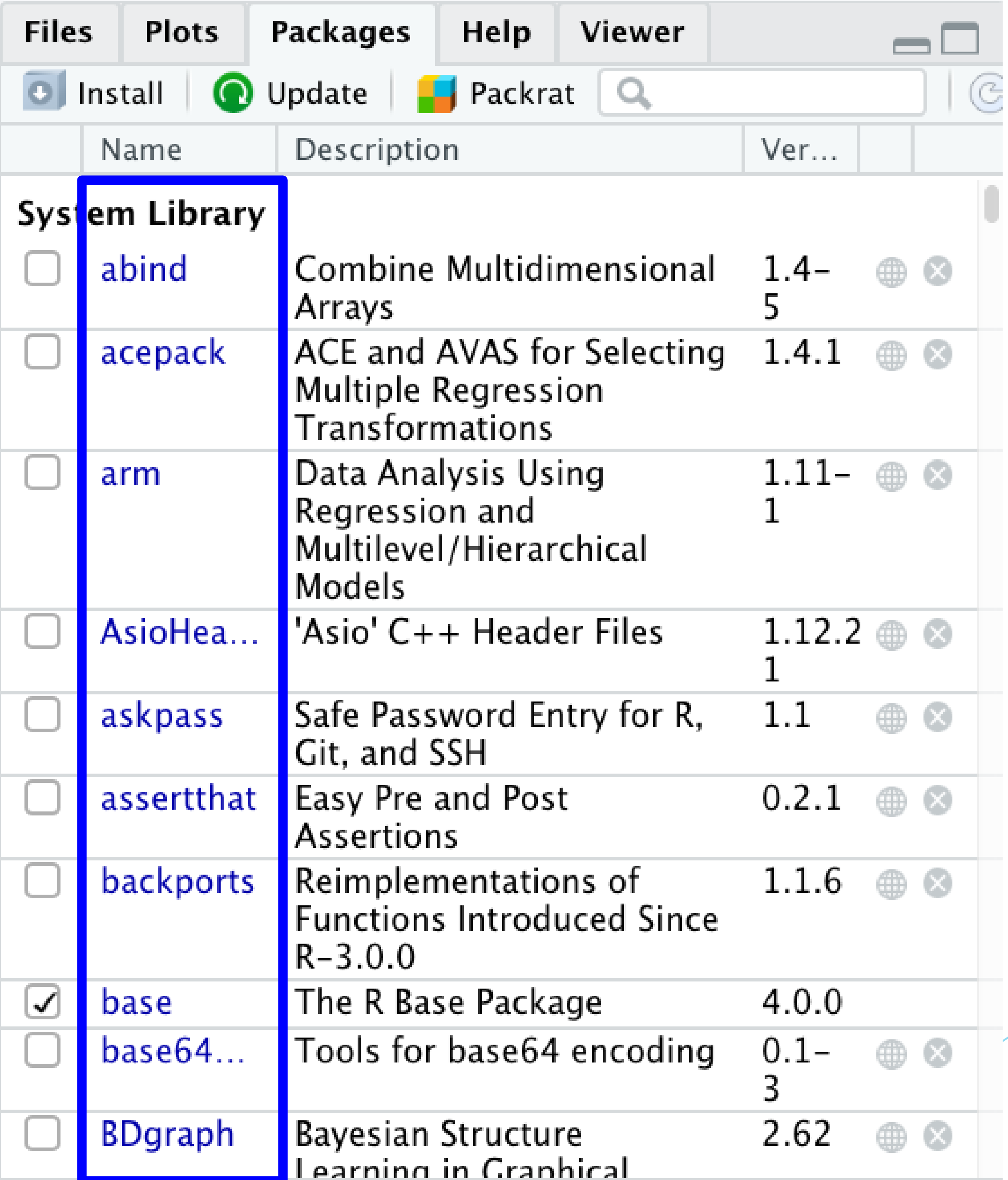
Packages are the reason R is so powerful.

- And why it will never be out-of-date.

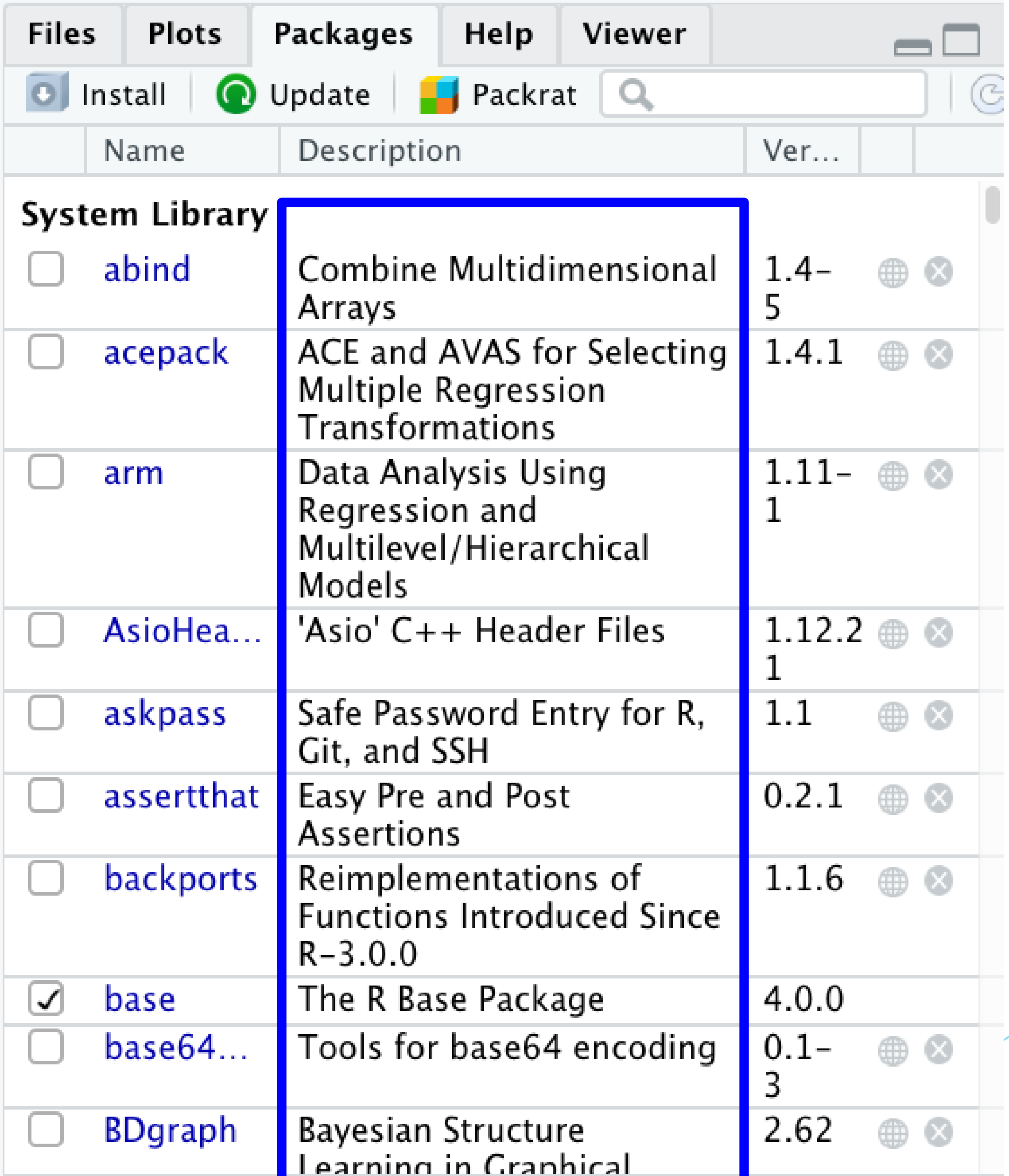


System Library

	Name	Description	Ver...
□	abind	Combine Multidimensional Arrays	1.4-5
□	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
□	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11-1
□	AsioHea...	'Asio' C++ Header Files	1.12.2 1
□	askpass	Safe Password Entry for R, Git, and SSH	1.1
□	assertthat	Easy Pre and Post Assertions	0.2.1
□	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.6
☑	base	The R Base Package	4.0.0
□	base64...	Tools for base64 encoding	0.1-3
□	BDgraph	Bayesian Structure Learning in Graphical	2.62

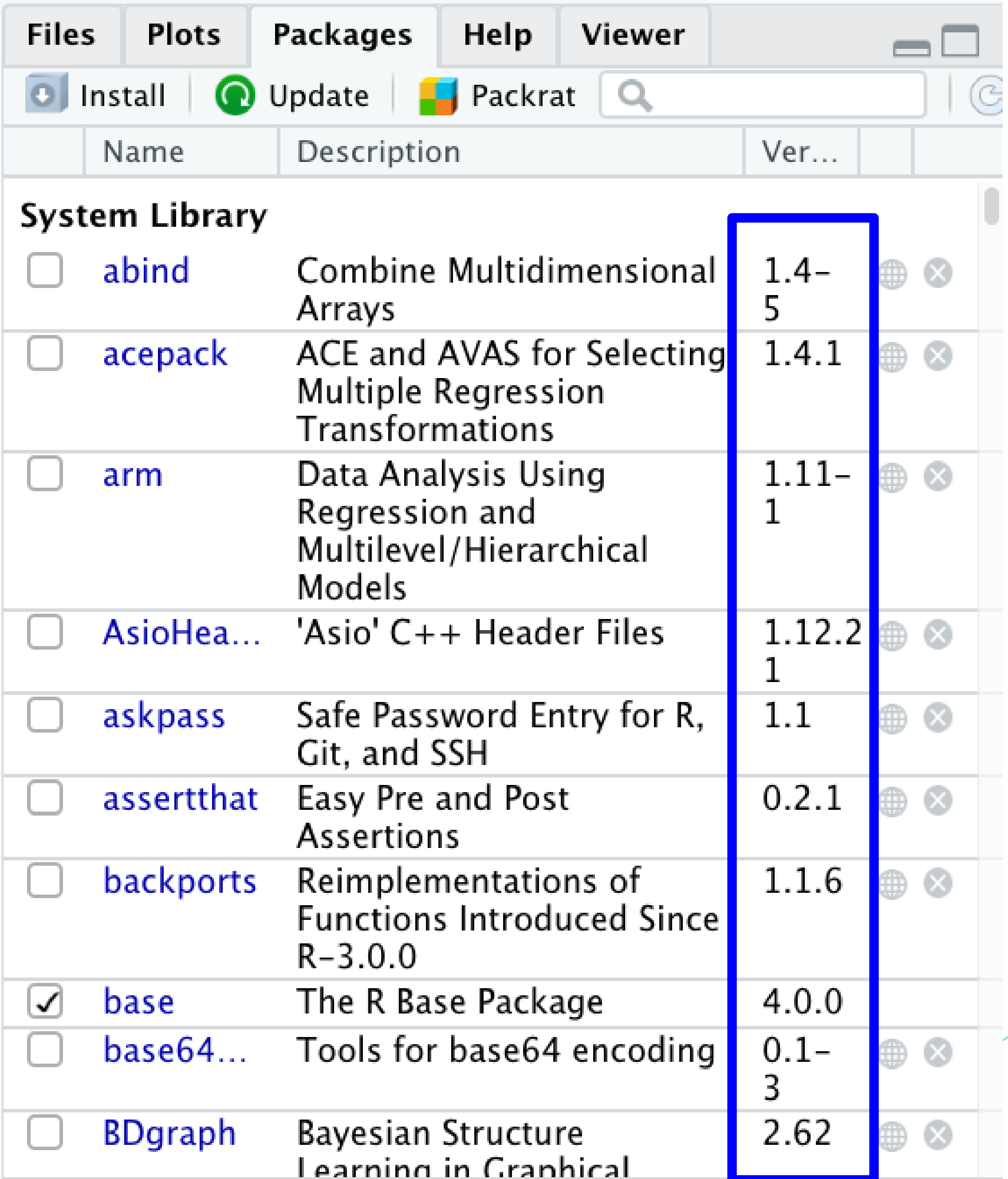


	Name	Description	Ver...
System Library			
☐	abind	Combine Multidimensional Arrays	1.4-5
☐	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
☐	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11-1
☐	AsioHea...	'Asio' C++ Header Files	1.12.2 1
☐	askpass	Safe Password Entry for R, Git, and SSH	1.1
☐	assertthat	Easy Pre and Post Assertions	0.2.1
☐	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.6
☑	base	The R Base Package	4.0.0
☐	base64...	Tools for base64 encoding	0.1-3
☐	BDgraph	Bayesian Structure Learning in Graphical	2.62



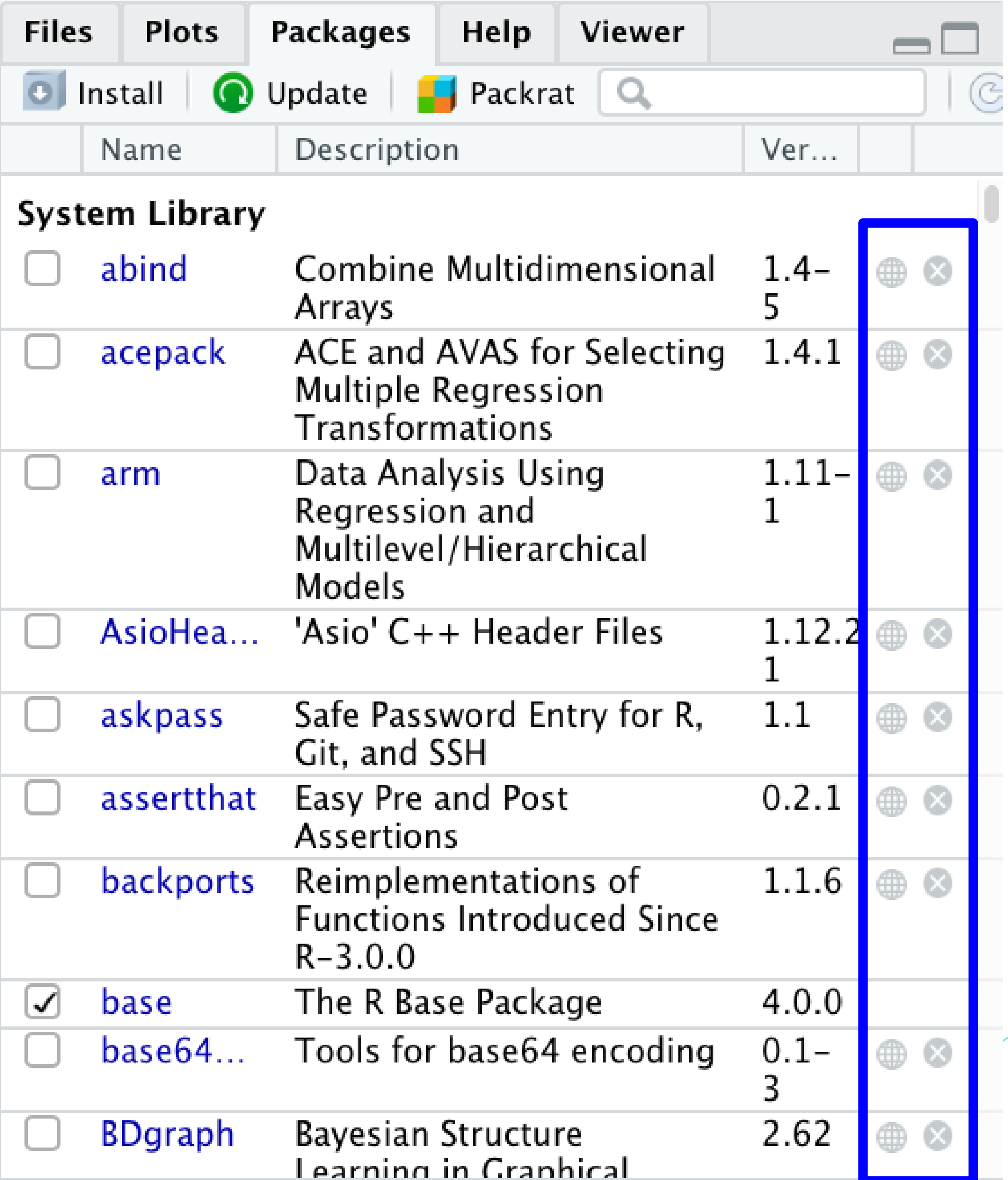
System Library

	Name	Description	Ver…
☐	abind	Combine Multidimensional Arrays	1.4-5
☐	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
☐	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11-1
☐	AsioHea…	'Asio' C++ Header Files	1.12.2-1
☐	askpass	Safe Password Entry for R, Git, and SSH	1.1
☐	assertthat	Easy Pre and Post Assertions	0.2.1
☐	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.6
☑	base	The R Base Package	4.0.0
☐	base64…	Tools for base64 encoding	0.1-3
☐	BDgraph	Bayesian Structure Learning in Graphical	2.62



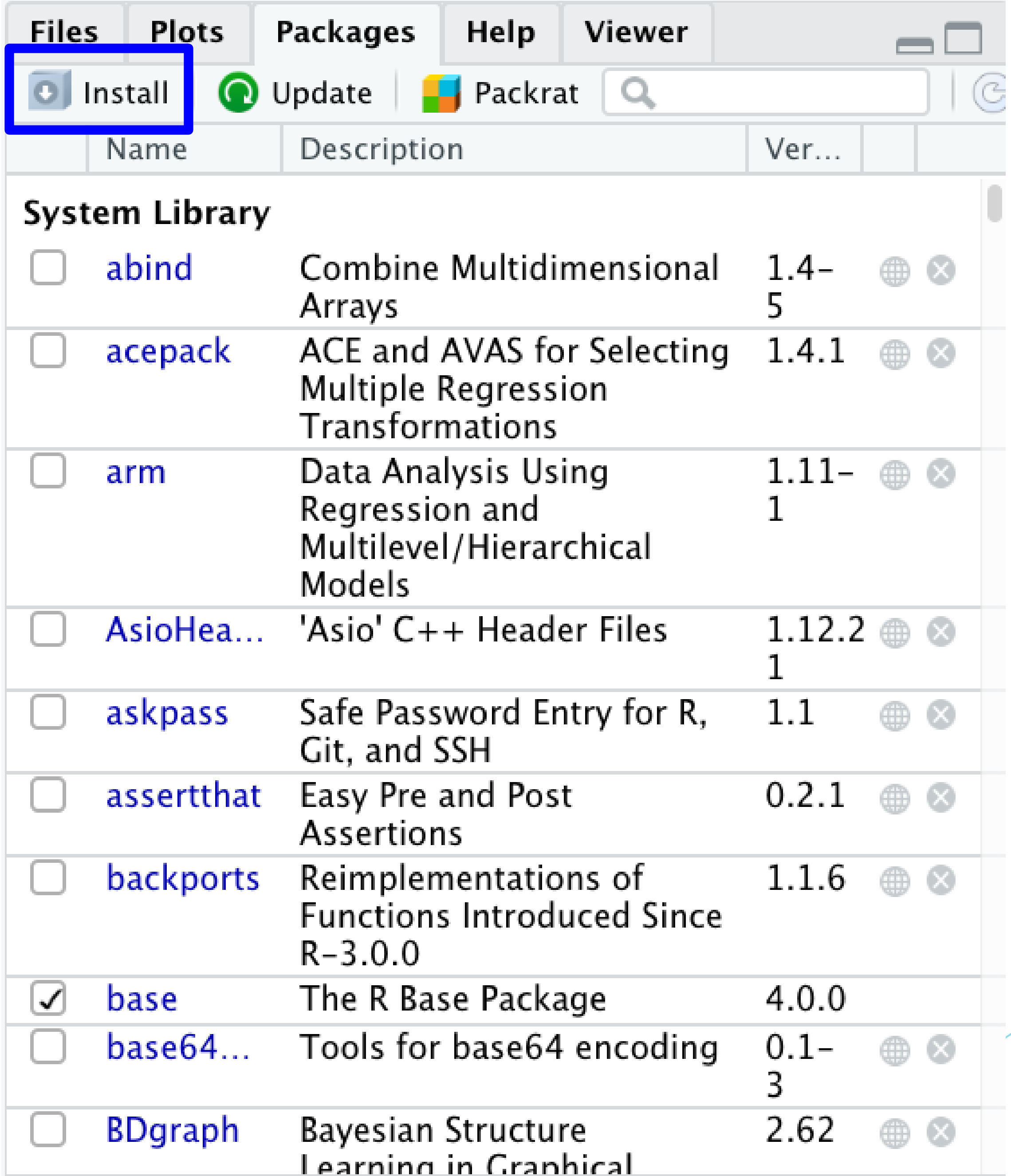
System Library

	Name	Description	Ver...
□	abind	Combine Multidimensional Arrays	1.4-5
□	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
□	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11-1
□	AsioHea...	'Asio' C++ Header Files	1.12.2 1
□	askpass	Safe Password Entry for R, Git, and SSH	1.1
□	assertthat	Easy Pre and Post Assertions	0.2.1
□	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.6
☑	base	The R Base Package	4.0.0
□	base64...	Tools for base64 encoding	0.1-3
□	BDgraph	Bayesian Structure Learning in Graphical	2.62

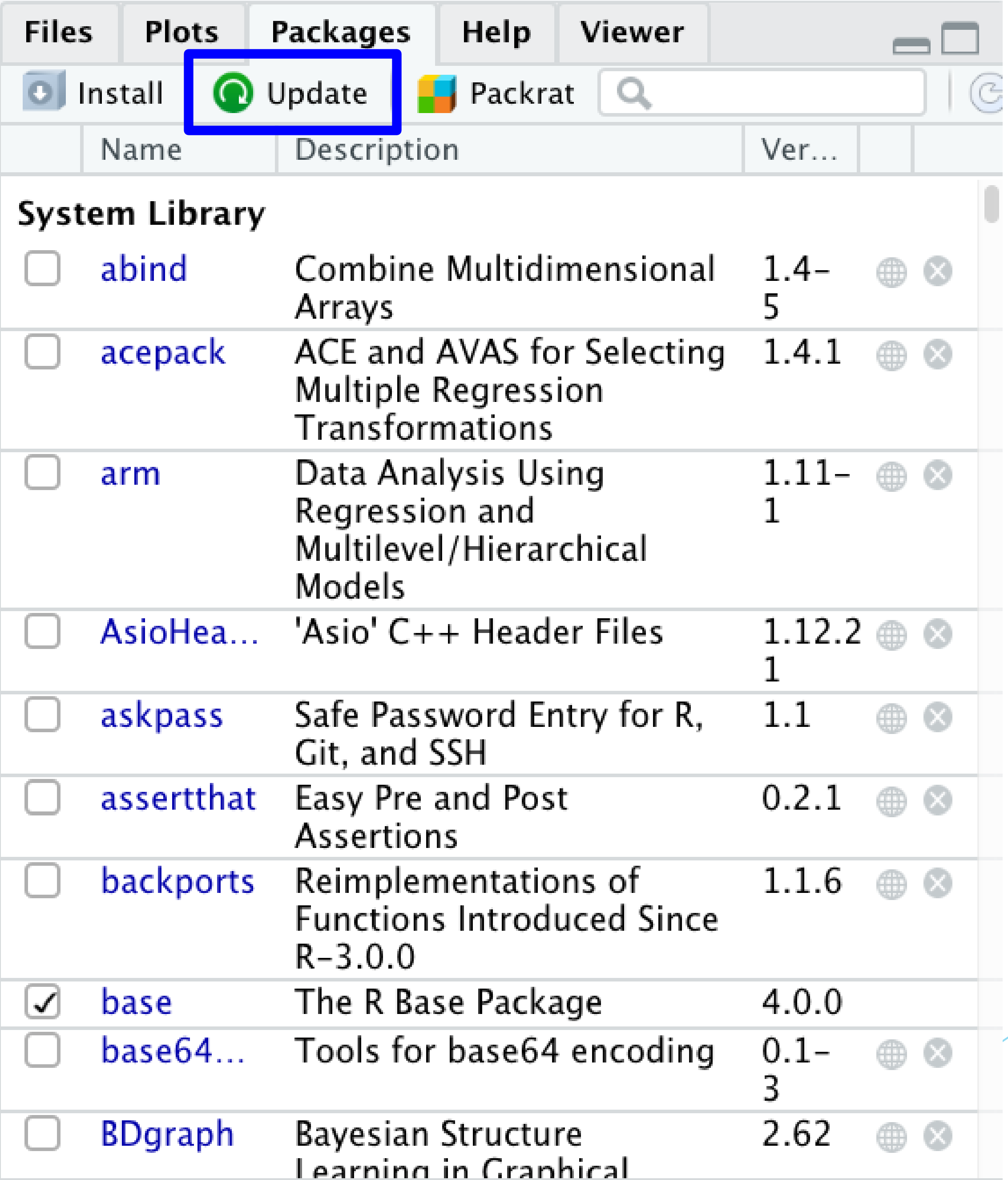


System Library

	Name	Description	Ver...	
☐	abind	Combine Multidimensional Arrays	1.4-5	🌐 ⊗
☐	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1	🌐 ⊗
☐	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11-1	🌐 ⊗
☐	AsioHea...	'Asio' C++ Header Files	1.12.2 1	🌐 ⊗
☐	askpass	Safe Password Entry for R, Git, and SSH	1.1	🌐 ⊗
☐	assertthat	Easy Pre and Post Assertions	0.2.1	🌐 ⊗
☐	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.6	🌐 ⊗
☑	base	The R Base Package	4.0.0	
☐	base64...	Tools for base64 encoding	0.1-3	🌐 ⊗
☐	BDgraph	Bayesian Structure Learning in Graphical	2.62	🌐 ⊗

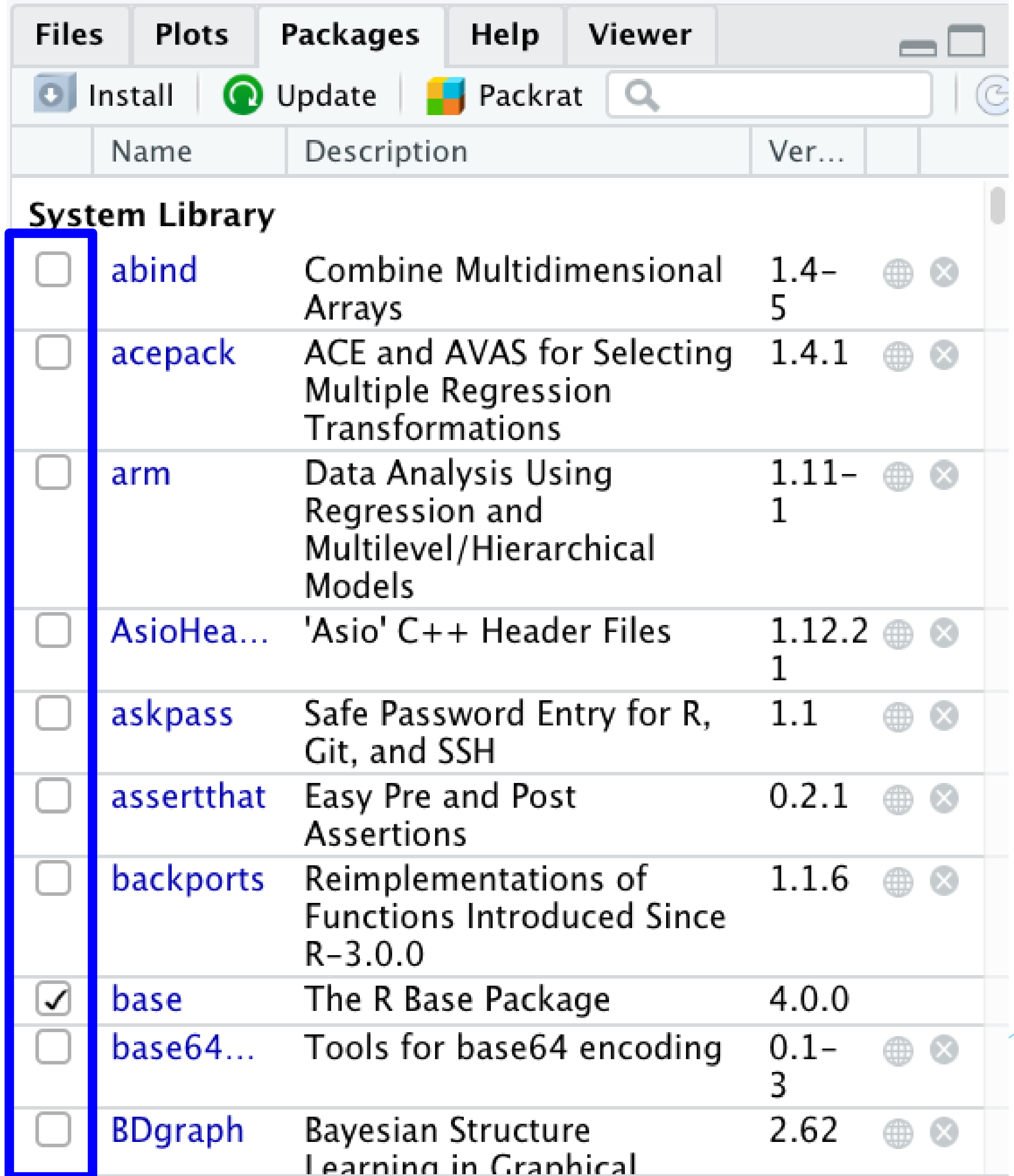


	Name	Description	Ver...		
System Library					
□	abind	Combine Multidimensional Arrays	1.4-5	🌐	⊗
□	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1	🌐	⊗
□	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11-1	🌐	⊗
□	AsioHea...	'Asio' C++ Header Files	1.12.2 1	🌐	⊗
□	askpass	Safe Password Entry for R, Git, and SSH	1.1	🌐	⊗
□	assertthat	Easy Pre and Post Assertions	0.2.1	🌐	⊗
□	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.6	🌐	⊗
☑	base	The R Base Package	4.0.0		
□	base64...	Tools for base64 encoding	0.1-3	🌐	⊗
□	BDgraph	Bayesian Structure Learning in Graphical	2.62	🌐	⊗



System Library

	Name	Description	Ver...
□	abind	Combine Multidimensional Arrays	1.4-5
□	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
□	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11-1
□	AsioHea...	'Asio' C++ Header Files	1.12.2 1
□	askpass	Safe Password Entry for R, Git, and SSH	1.1
□	assertthat	Easy Pre and Post Assertions	0.2.1
□	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.6
☑	base	The R Base Package	4.0.0
□	base64...	Tools for base64 encoding	0.1-3
□	BDgraph	Bayesian Structure Learning in Graphical	2.62



System Library

	Name	Description	Ver…
□	abind	Combine Multidimensional Arrays	1.4–5
□	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
□	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11–1
□	AsioHea…	'Asio' C++ Header Files	1.12.2 1
□	askpass	Safe Password Entry for R, Git, and SSH	1.1
□	assertthat	Easy Pre and Post Assertions	0.2.1
□	backports	Reimplementations of Functions Introduced Since R–3.0.0	1.1.6
☑	base	The R Base Package	4.0.0
□	base64…	Tools for base64 encoding	0.1–3
□	BDgraph	Bayesian Structure Learning in Graphical	2.62

How do I get packages?

Packages can be downloaded from the CRAN (**C**omprehensive **R** **A**rchive **N**etwork).

You do this from inside R!

- Need to be connected to the internet

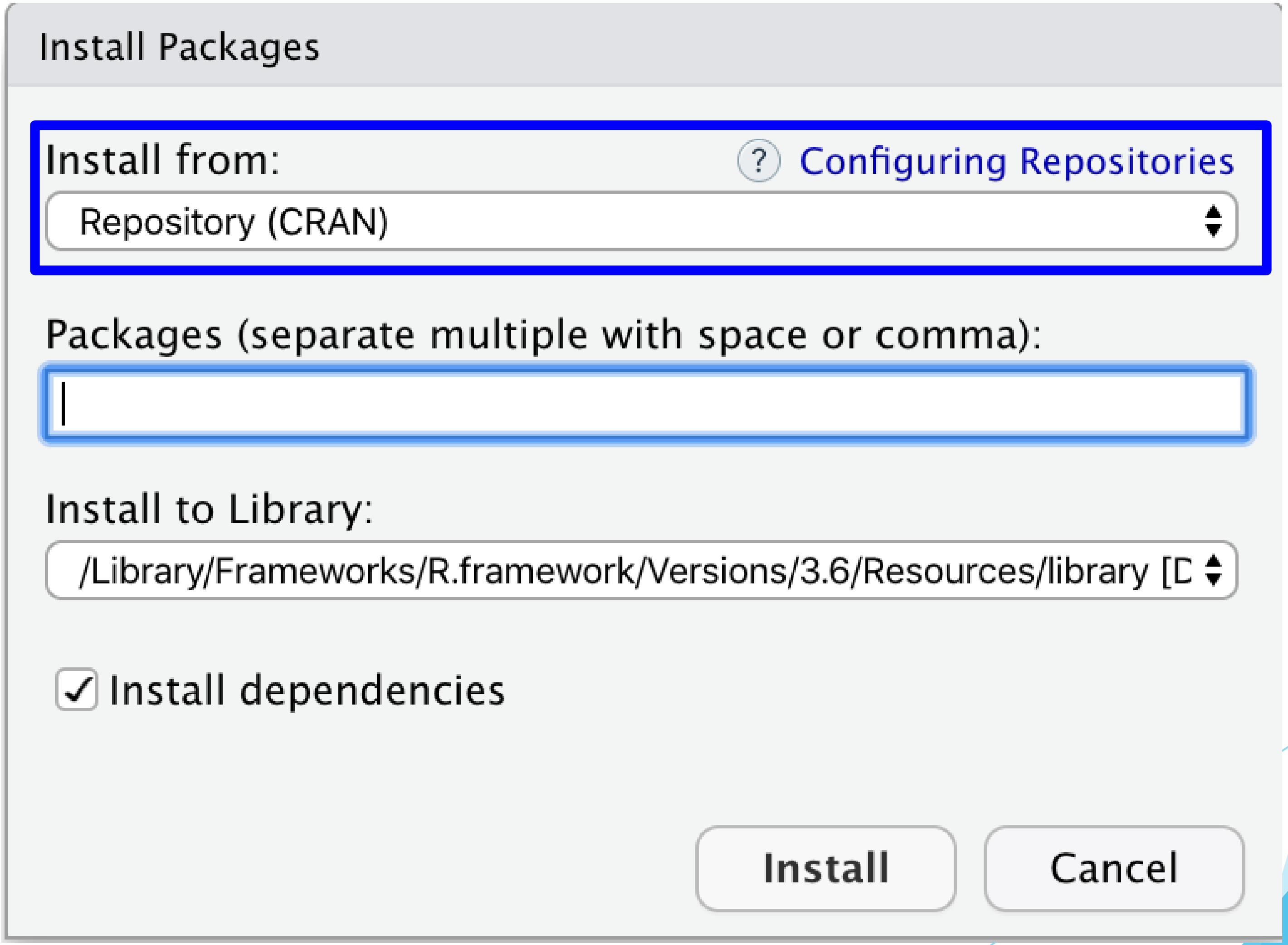
2 ways to install packages

1) Install button in the Packages window

2) R Code

Either way, you'll need to know the name of the package.

Install Button



Install Packages

Install from: ? Configuring Repositories

Repository (CRAN)

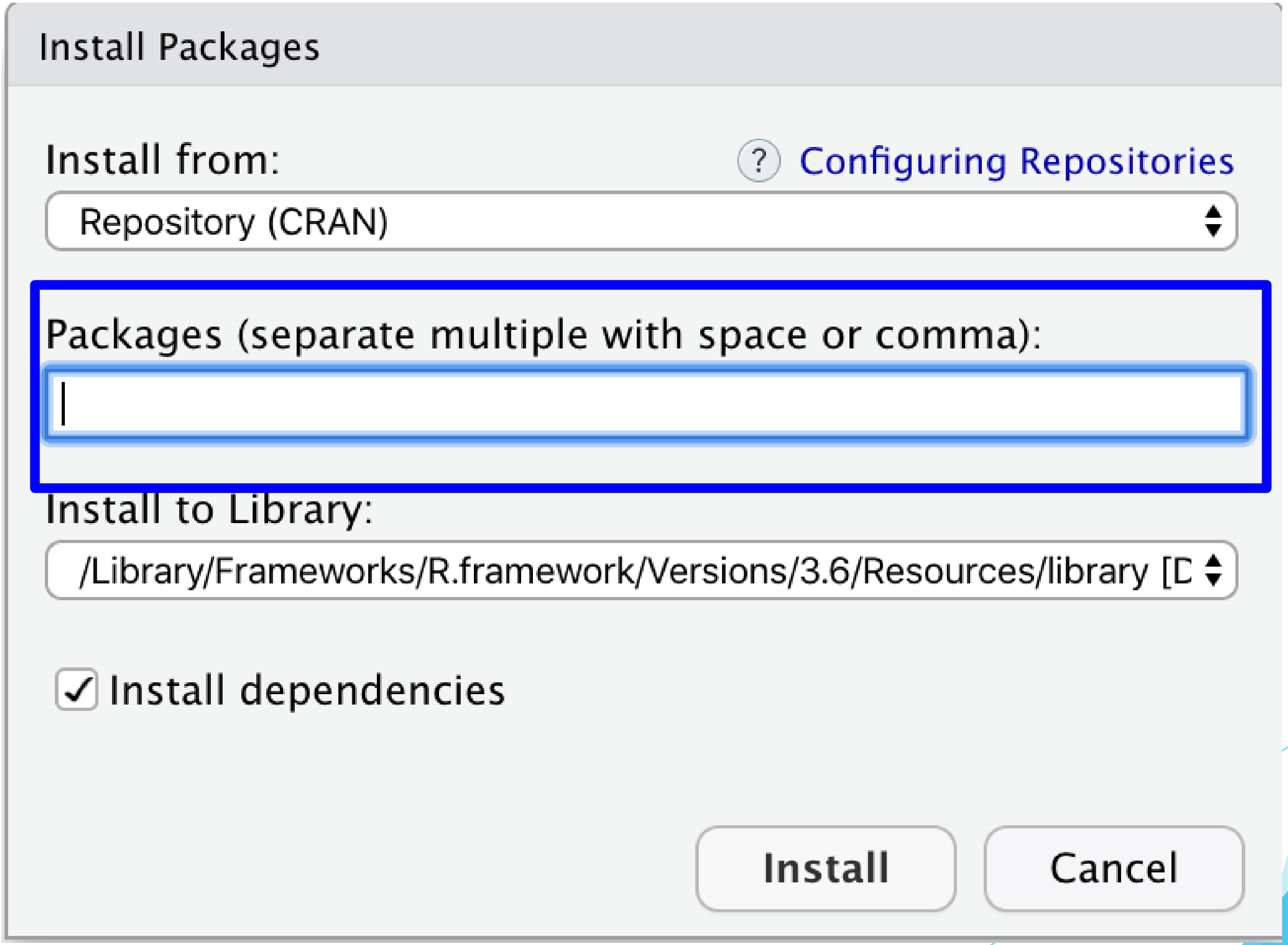
Packages (separate multiple with space or comma):

Install to Library:

/Library/Frameworks/R.framework/Versions/3.6/Resources/library [D

☑ Install dependencies

Install Button



Install Packages

Install from:

Repository (CRAN)

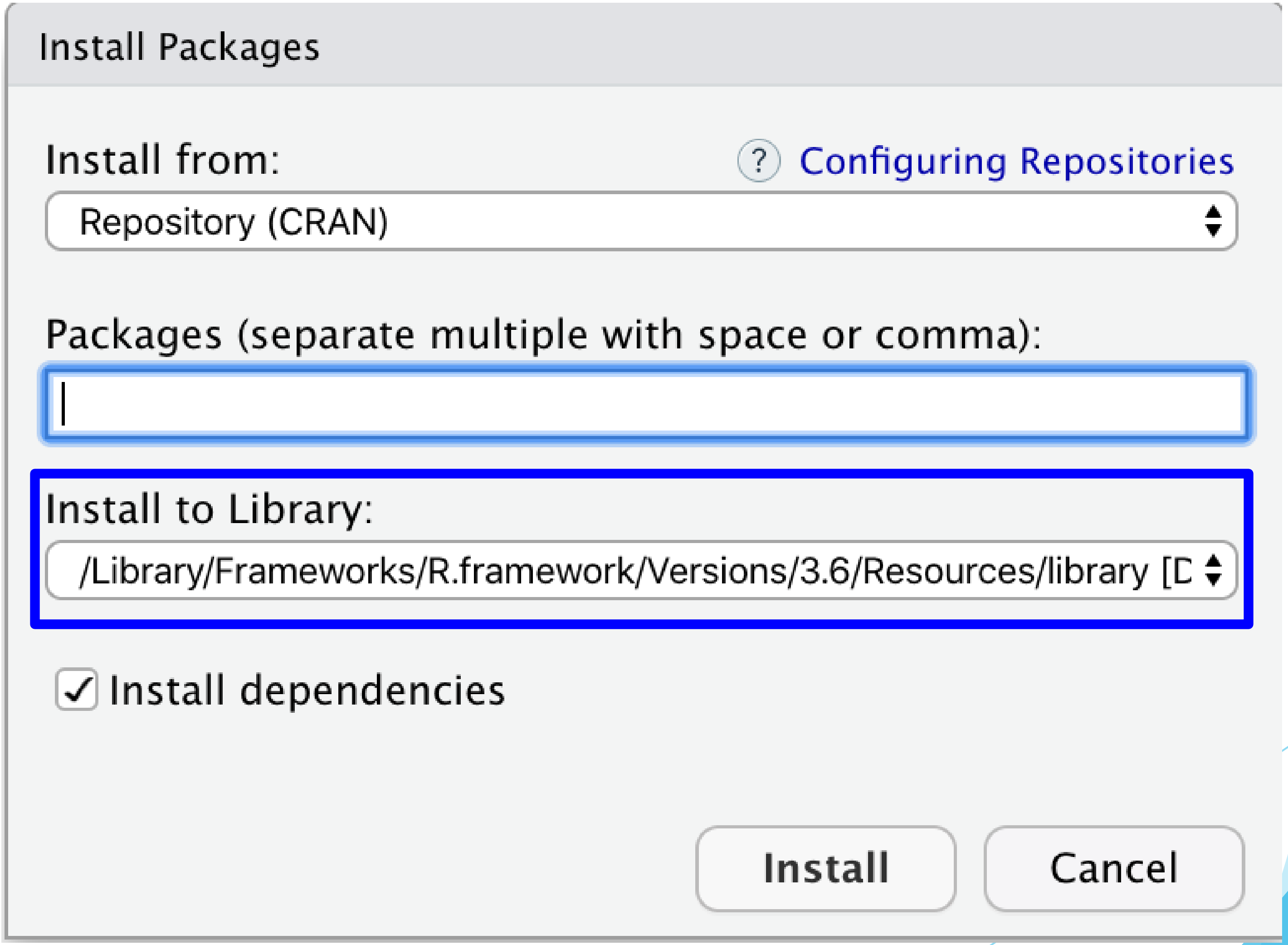
Packages (separate multiple with space or comma):

Install to Library:

/Library/Frameworks/R.framework/Versions/3.6/Resources/library [D

☑ Install dependencies

Install Button



Install Packages

Install from:

Repository (CRAN)

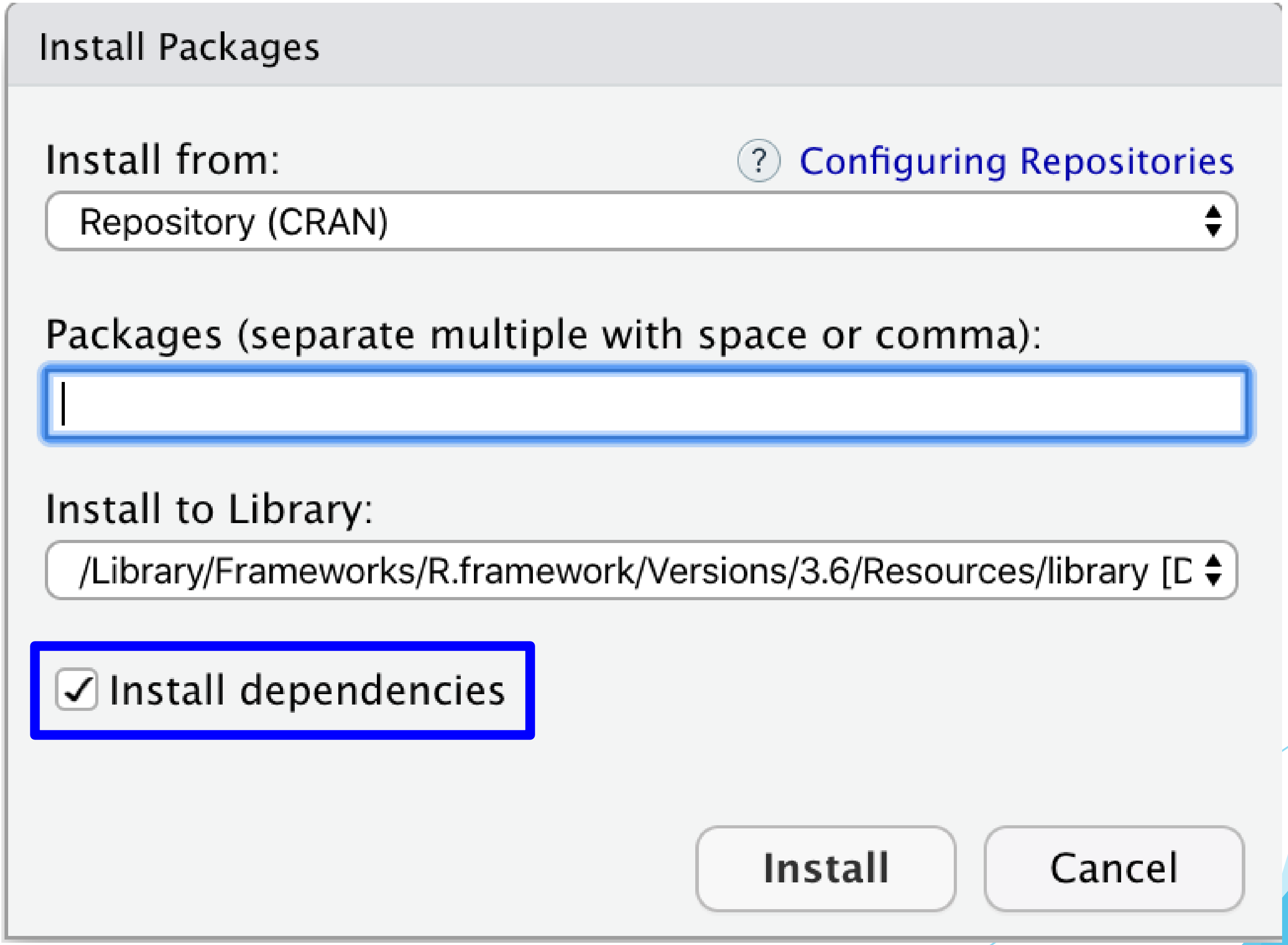
Packages (separate multiple with space or comma):

Install to Library:

/Library/Frameworks/R.framework/Versions/3.6/Resources/library [D

☑ Install dependencies

Install Button



Install Packages

Install from:

Configuring Repositories

Repository (CRAN)

Packages (separate multiple with space or comma):

Install to Library:

/Library/Frameworks/R.framework/Versions/3.6/Resources/library [D

☑ Install dependencies

R Code to Install Packages

Note the quotation marks!

```
install.packages(“psych”)
```

Packages

INSTALLING

- Downloading the package and saving it to your computer.
- Like installing Microsoft Word on your computer.
- Do this **ONCE**

LOADING

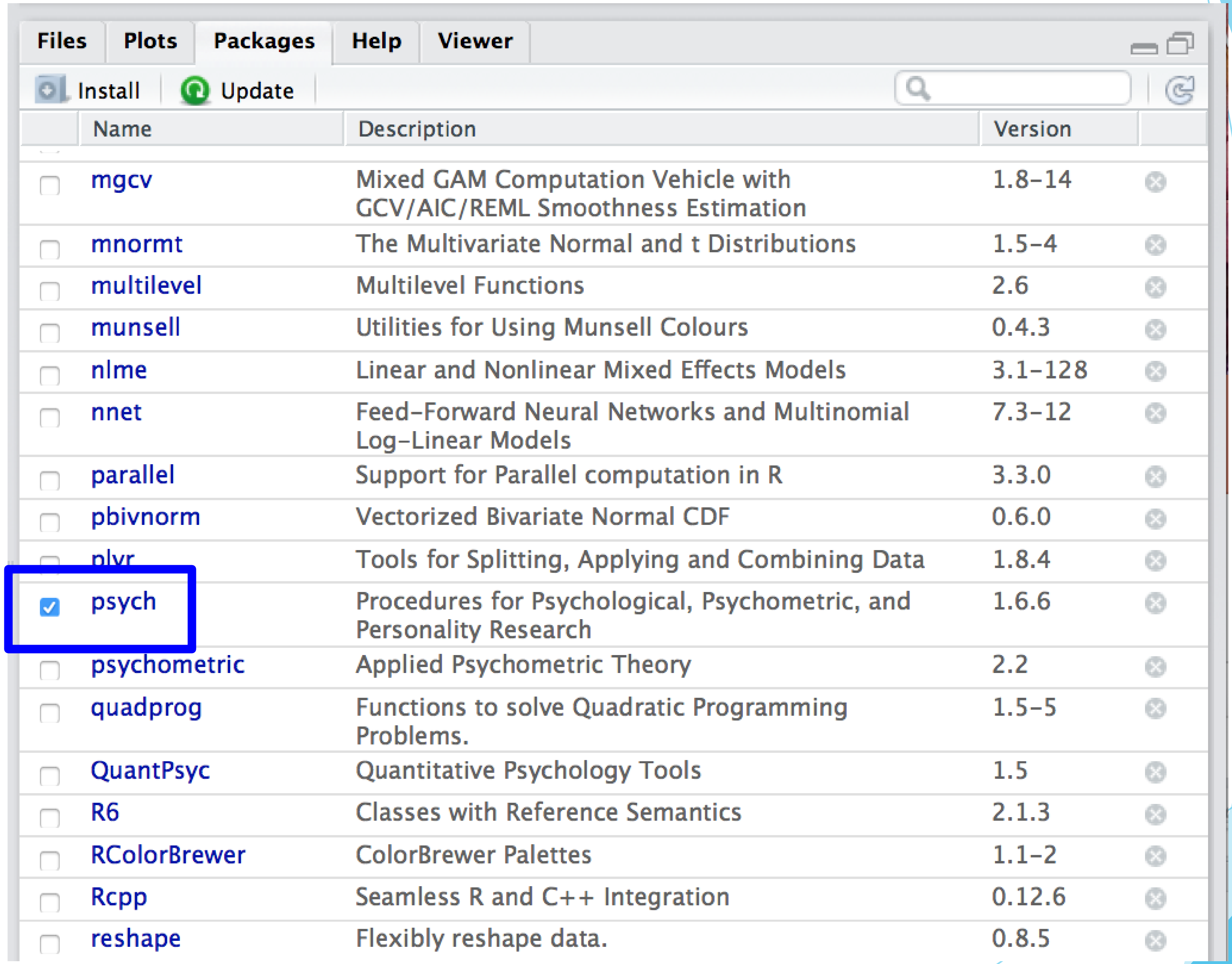
- Like opening Microsoft Word to use now.
- Once a package is loaded in R, all of its functions are ready to use now.
- Do this **EVERY TIME** you open an R session.

2 ways to load packages

1) Checkbox in the Packages window

2) R Code

Either way, you’ll need to know the name of the package.



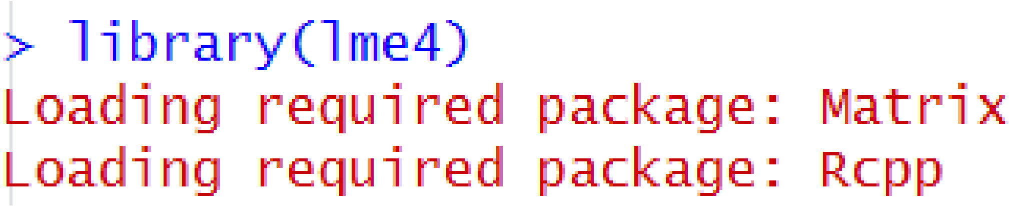
	Name	Description	Version
☐	mgcv	Mixed GAM Computation Vehicle with GCV/AIC/REML Smoothness Estimation	1.8-14
☐	mnormt	The Multivariate Normal and t Distributions	1.5-4
☐	multilevel	Multilevel Functions	2.6
☐	munsell	Utilities for Using Munsell Colours	0.4.3
☐	nlme	Linear and Nonlinear Mixed Effects Models	3.1-128
☐	nnet	Feed-Forward Neural Networks and Multinomial Log-Linear Models	7.3-12
☐	parallel	Support for Parallel computation in R	3.3.0
☐	pbivnorm	Vectorized Bivariate Normal CDF	0.6.0
☐	plyr	Tools for Splitting, Applying and Combining Data	1.8.4
☑	psych	Procedures for Psychological, Psychometric, and Personality Research	1.6.6
☐	psychometric	Applied Psychometric Theory	2.2
☐	quadprog	Functions to solve Quadratic Programming Problems.	1.5-5
☐	QuantPsyc	Quantitative Psychology Tools	1.5
☐	R6	Classes with Reference Semantics	2.1.3
☐	RColorBrewer	ColorBrewer Palettes	1.1-2
☐	Rcpp	Seamless R and C++ Integration	0.12.6
☐	reshape	Flexibly reshape data.	0.8.5

R Code to Load Packages

Note: NO quotation marks!

```
library(psych)
```

Dependencies



```
> library(lme4)
Loading required package: Matrix
Loading required package: Rcpp
```

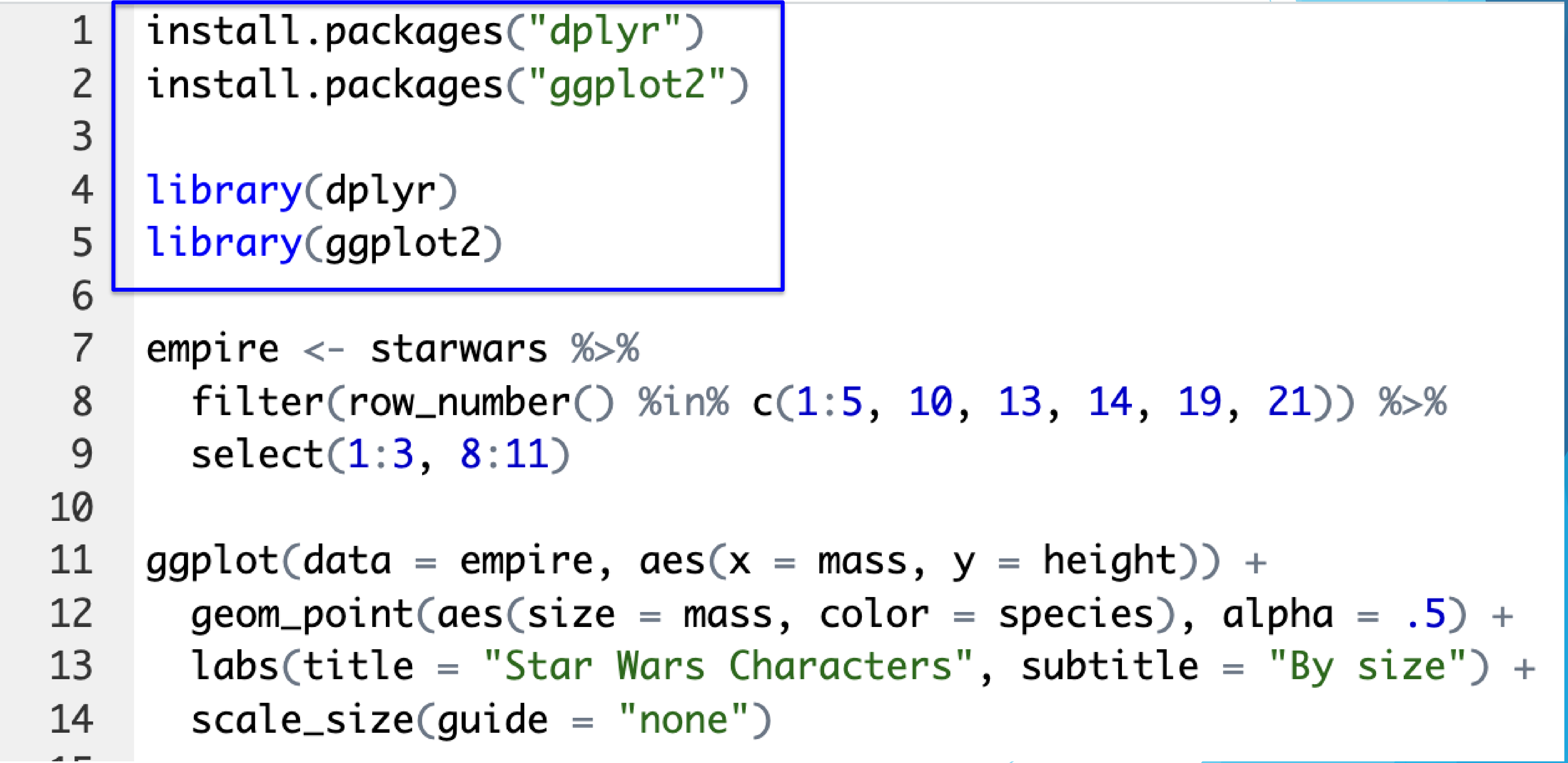
Uses functions from other packages.

Installed automatically.

Loaded automatically.

Exercise

- **Install & Load** these three packages:
  - tidyverse
  - gghalves
  - psych



```r
install.packages("dplyr")
install.packages("ggplot2")

library(dplyr)
library(ggplot2)

empire <- starwars %>%
  filter(row_number() %in% c(1:5, 10, 13, 14, 19, 21)) %>%
  select(1:3, 8:11)

ggplot(data = empire, aes(x = mass, y = height)) +
  geom_point(aes(size = mass, color = species), alpha = .5) +
  labs(title = "Star Wars Characters", subtitle = "By size") +
  scale_size(guide = "none")
```

Help! (again)

Ways to find documentation:

`?psych` – opens documentation specific to that package or function

`??psych` – searches for this in all documentation

**Only looks in documentation for packages you have installed and loaded.

To find a package that does what you need: Google ☺

Help! (again)

To find functions available in a package:

In the Packages tab, click on the name of the package to see what functions are available!

Agenda

- Understanding the basics of functions
- Get familiar with help documentation
- Working with packages
- R **Scripts** and **loading files**
- Data Wrangling with `tidyverse`

.R files

.R files are text files.

- They contain the code that you’ve written – the commands that you want R to run.

Equivalent to syntax files in SPSS.

Also called **scripts**.

.R files – why use them?

Keep track of what commands you use.

Save only the commands that are useful.

Make notes to yourself!

- `#Updated code for R workshop!`
- `#reliability estimates for depression scale`
- `#scatter plot for BMI predicting diabetes diagnosis`

Share your analyses with collaborators and readers.

Your Data

Original data files

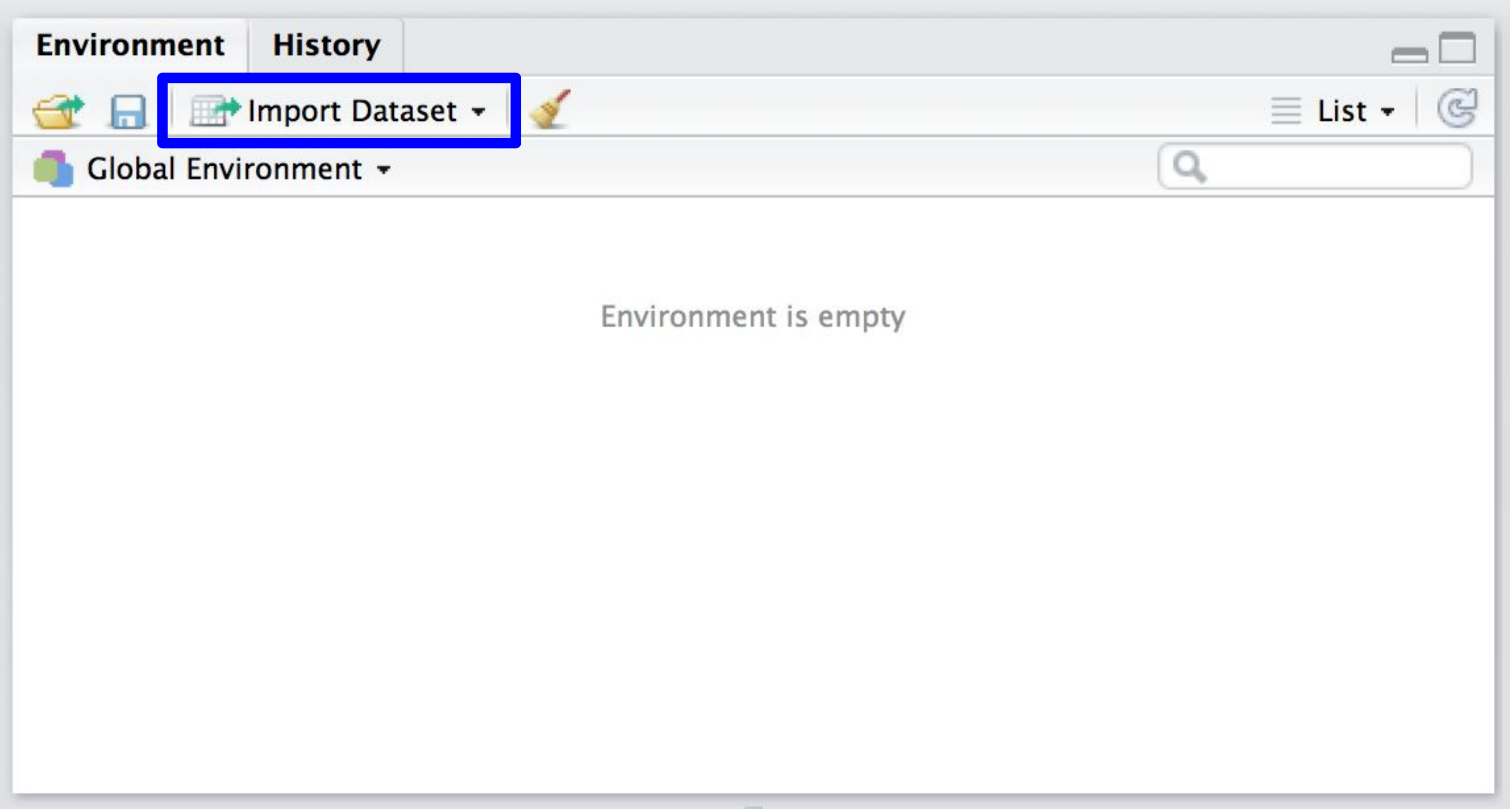
- Most of the time, these are going to be either .csv or .txt, depending on how you collect data.

These are not altered by R! ***(different from SPSS!)***

If your data is not a **.csv** or **.txt** file, don’t worry! R can do a lot of stuff!

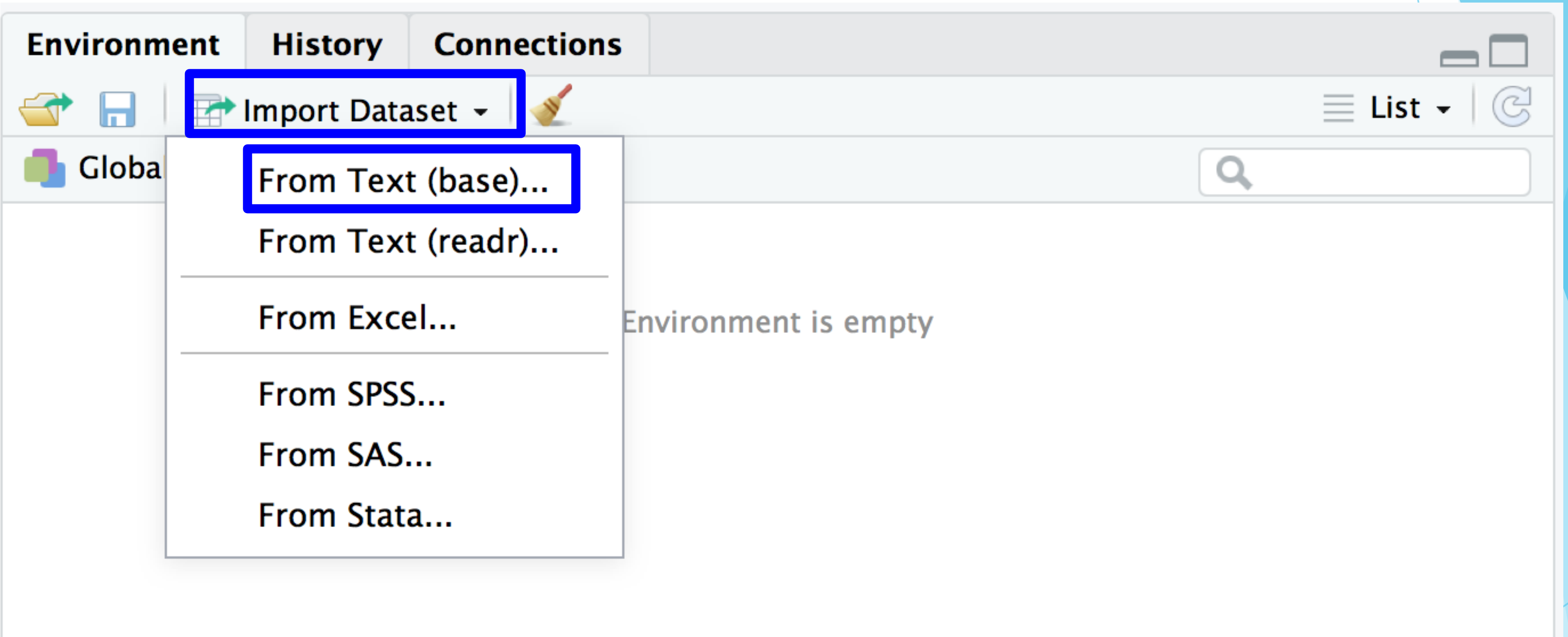
We will work with .csv, just to keep things simple.

Loading .csv files

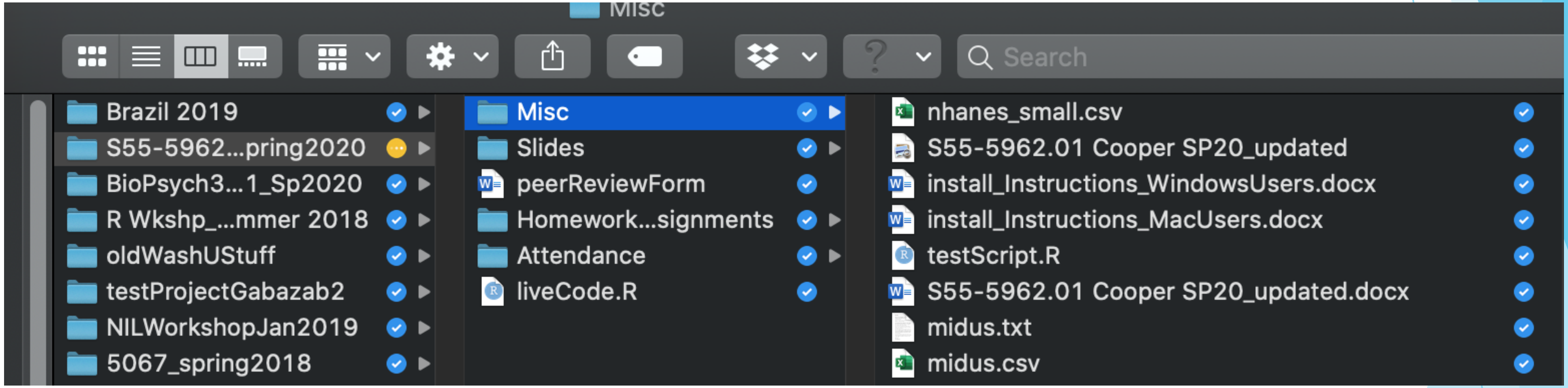


Environment is empty

Loading .csv files



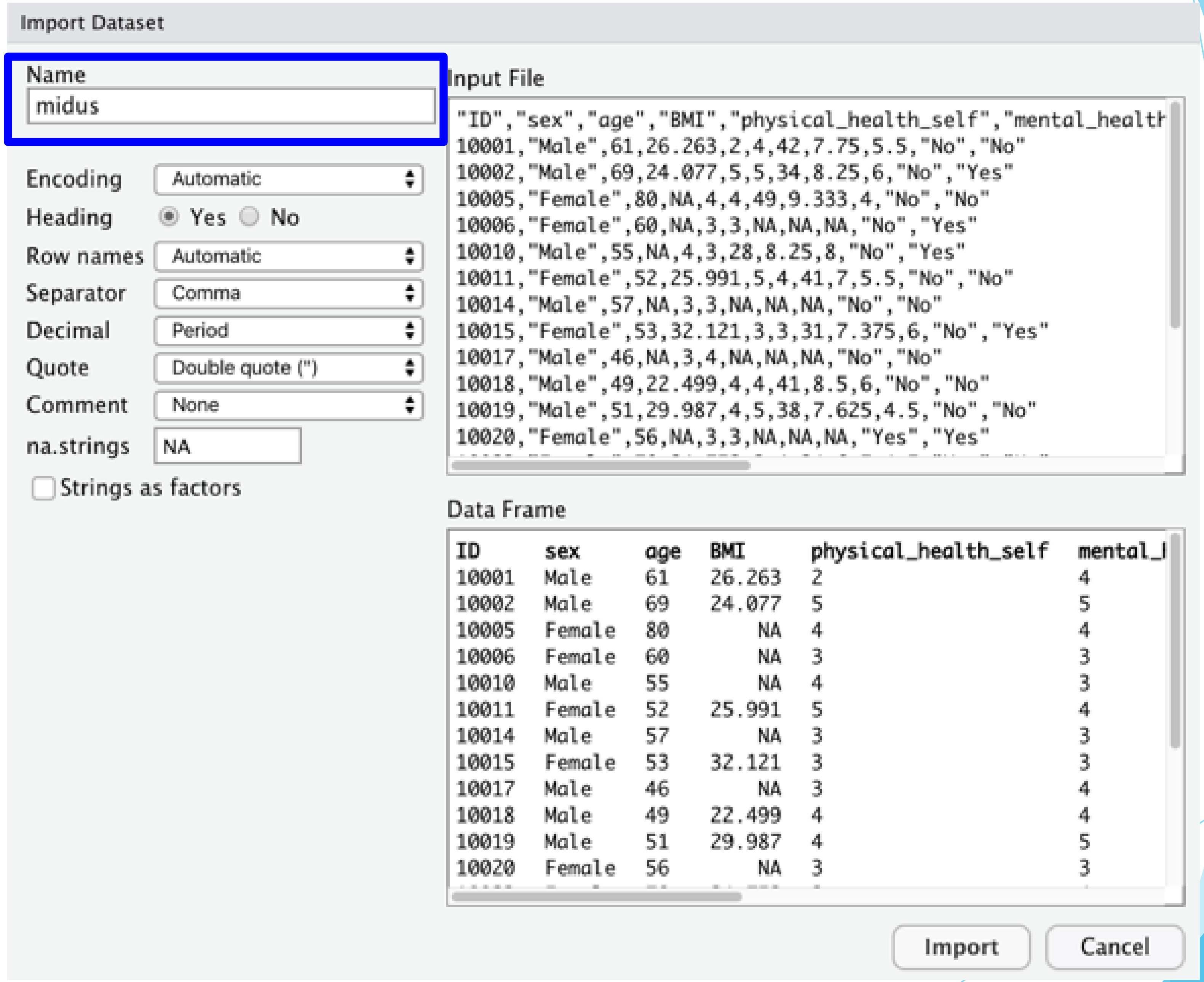
Loading .csv files



Brazil 2019
S55-5962…pring2020
BioPsych3…1_Sp2020
R Wkshp_…mmer 2018
oldWashUStuff
testProjectGabazab2
NILWorkshopJan2019
5067_spring2018

Misc
Slides
peerReviewForm
Homework…signments
Attendance
liveCode.R

nhanes_small.csv
S55-5962.01 Cooper SP20_updated
install_Instructions_WindowsUsers.docx
install_Instructions_MacUsers.docx
testScript.R
S55-5962.01 Cooper SP20_updated.docx
midus.txt
midus.csv



Name: midus

Encoding: Automatic

Heading: ● Yes ○ No

Row names: Automatic

Separator: Comma

Decimal: Period

Quote: Double quote (")

Comment: None

na.strings: NA

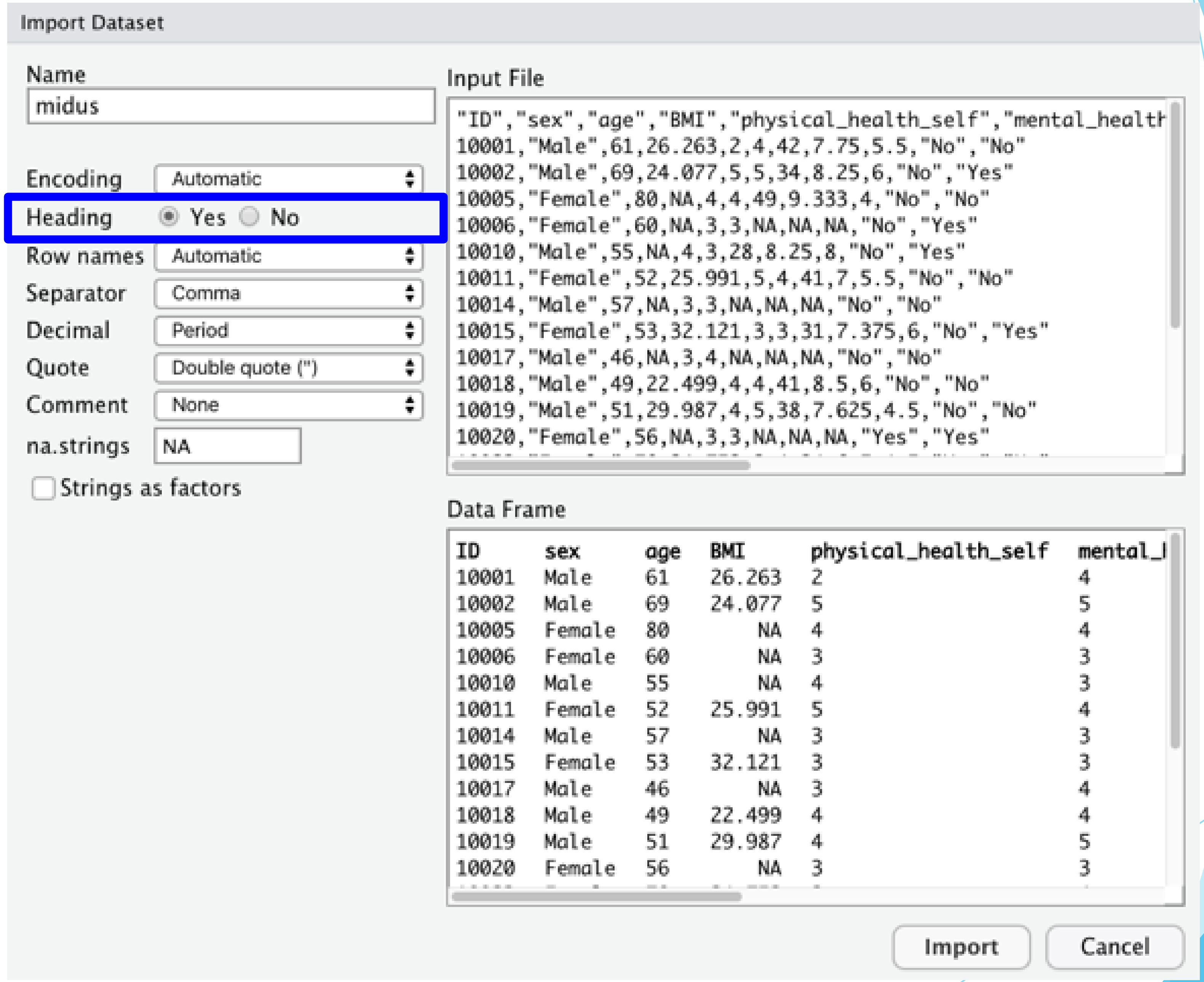
□ Strings as factors

Input File

```
"ID","sex","age","BMI","physical_health_self","mental_health
10001,"Male",61,26.263,2,4,42,7.75,5.5,"No","No"
10002,"Male",69,24.077,5,5,34,8.25,6,"No","Yes"
10005,"Female",80,NA,4,4,49,9.333,4,"No","No"
10006,"Female",60,NA,3,3,NA,NA,NA,"No","Yes"
10010,"Male",55,NA,4,3,28,8.25,8,"No","Yes"
10011,"Female",52,25.991,5,4,41,7,5.5,"No","No"
10014,"Male",57,NA,3,3,NA,NA,NA,"No","No"
10015,"Female",53,32.121,3,3,31,7.375,6,"No","Yes"
10017,"Male",46,NA,3,4,NA,NA,NA,"No","No"
10018,"Male",49,22.499,4,4,41,8.5,6,"No","No"
10019,"Male",51,29.987,4,5,38,7.625,4.5,"No","No"
10020,"Female",56,NA,3,3,NA,NA,NA,"Yes","Yes"
```

Data Frame

```
ID      sex     age   BMI     physical_health_self   mental_
10001   Male    61    26.263  2                      4
10002   Male    69    24.077  5                      5
10005   Female  80        NA  4                      4
10006   Female  60        NA  3                      3
10010   Male    55        NA  4                      3
10011   Female  52    25.991  5                      4
10014   Male    57        NA  3                      3
10015   Female  53    32.121  3                      3
10017   Male    46        NA  3                      4
10018   Male    49    22.499  4                      4
10019   Male    51    29.987  4                      5
10020   Female  56        NA  3                      3
```



Name: midus

Encoding: Automatic

Heading: ● Yes ○ No

Row names: Automatic

Separator: Comma

Decimal: Period

Quote: Double quote (")

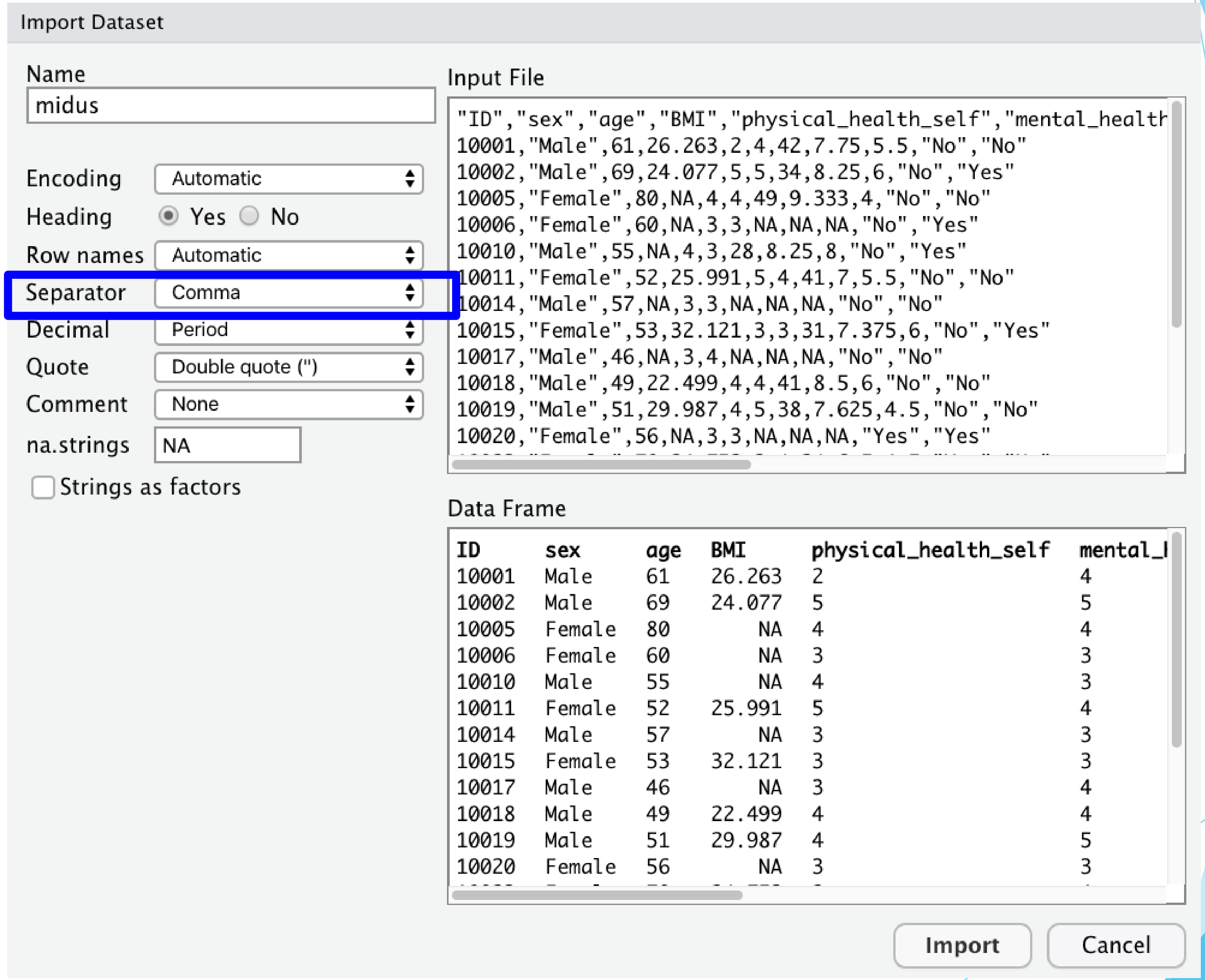
Comment: None

na.strings: NA

□ Strings as factors

```
"ID","sex","age","BMI","physical_health_self","mental_health
10001,"Male",61,26.263,2,4,42,7.75,5.5,"No","No"
10002,"Male",69,24.077,5,5,34,8.25,6,"No","Yes"
10005,"Female",80,NA,4,4,49,9.333,4,"No","No"
10006,"Female",60,NA,3,3,NA,NA,NA,"No","Yes"
10010,"Male",55,NA,4,3,28,8.25,8,"No","Yes"
10011,"Female",52,25.991,5,4,41,7,5.5,"No","No"
10014,"Male",57,NA,3,3,NA,NA,NA,"No","No"
10015,"Female",53,32.121,3,3,31,7.375,6,"No","Yes"
10017,"Male",46,NA,3,4,NA,NA,NA,"No","No"
10018,"Male",49,22.499,4,4,41,8.5,6,"No","No"
10019,"Male",51,29.987,4,5,38,7.625,4.5,"No","No"
10020,"Female",56,NA,3,3,NA,NA,NA,"Yes","Yes"
```

```
ID      sex     age   BMI     physical_health_self   mental_l
10001   Male    61    26.263  2                      4
10002   Male    69    24.077  5                      5
10005   Female  80        NA  4                      4
10006   Female  60        NA  3                      3
10010   Male    55        NA  4                      3
10011   Female  52    25.991  5                      4
10014   Male    57        NA  3                      3
10015   Female  53    32.121  3                      3
10017   Male    46        NA  3                      4
10018   Male    49    22.499  4                      4
10019   Male    51    29.987  4                      5
10020   Female  56        NA  3                      3
```



Name: midus

Encoding: Automatic

Heading: ● Yes ○ No

Row names: Automatic

Separator: Comma

Decimal: Period

Quote: Double quote (")

Comment: None

na.strings: NA

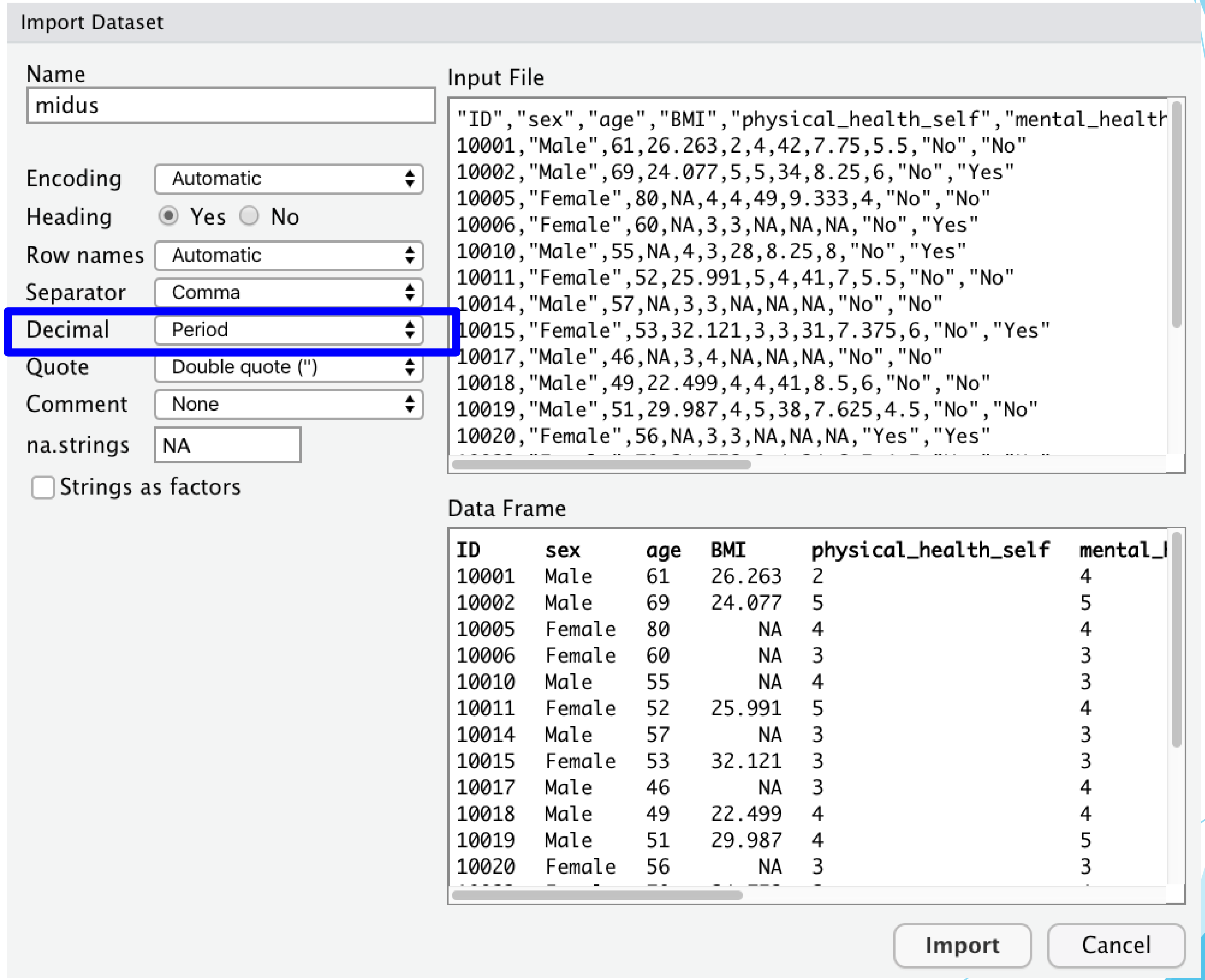
□ Strings as factors

Input File

```
"ID","sex","age","BMI","physical_health_self","mental_health
10001,"Male",61,26.263,2,4,42,7.75,5.5,"No","No"
10002,"Male",69,24.077,5,5,34,8.25,6,"No","Yes"
10005,"Female",80,NA,4,4,49,9.333,4,"No","No"
10006,"Female",60,NA,3,3,NA,NA,NA,"No","Yes"
10010,"Male",55,NA,4,3,28,8.25,8,"No","Yes"
10011,"Female",52,25.991,5,4,41,7,5.5,"No","No"
10014,"Male",57,NA,3,3,NA,NA,NA,"No","No"
10015,"Female",53,32.121,3,3,31,7.375,6,"No","Yes"
10017,"Male",46,NA,3,4,NA,NA,NA,"No","No"
10018,"Male",49,22.499,4,4,41,8.5,6,"No","No"
10019,"Male",51,29.987,4,5,38,7.625,4.5,"No","No"
10020,"Female",56,NA,3,3,NA,NA,NA,"Yes","Yes"
```

Data Frame

```
ID     sex     age  BMI     physical_health_self  mental_h
10001  Male    61   26.263  2                     4
10002  Male    69   24.077  5                     5
10005  Female  80       NA  4                     4
10006  Female  60       NA  3                     3
10010  Male    55       NA  4                     3
10011  Female  52   25.991  5                     4
10014  Male    57       NA  3                     3
10015  Female  53   32.121  3                     3
10017  Male    46       NA  3                     4
10018  Male    49   22.499  4                     4
10019  Male    51   29.987  4                     5
10020  Female  56       NA  3                     3
```



Name: midus

Encoding: Automatic

Heading: ● Yes ○ No

Row names: Automatic

Separator: Comma

Decimal: Period

Quote: Double quote (")

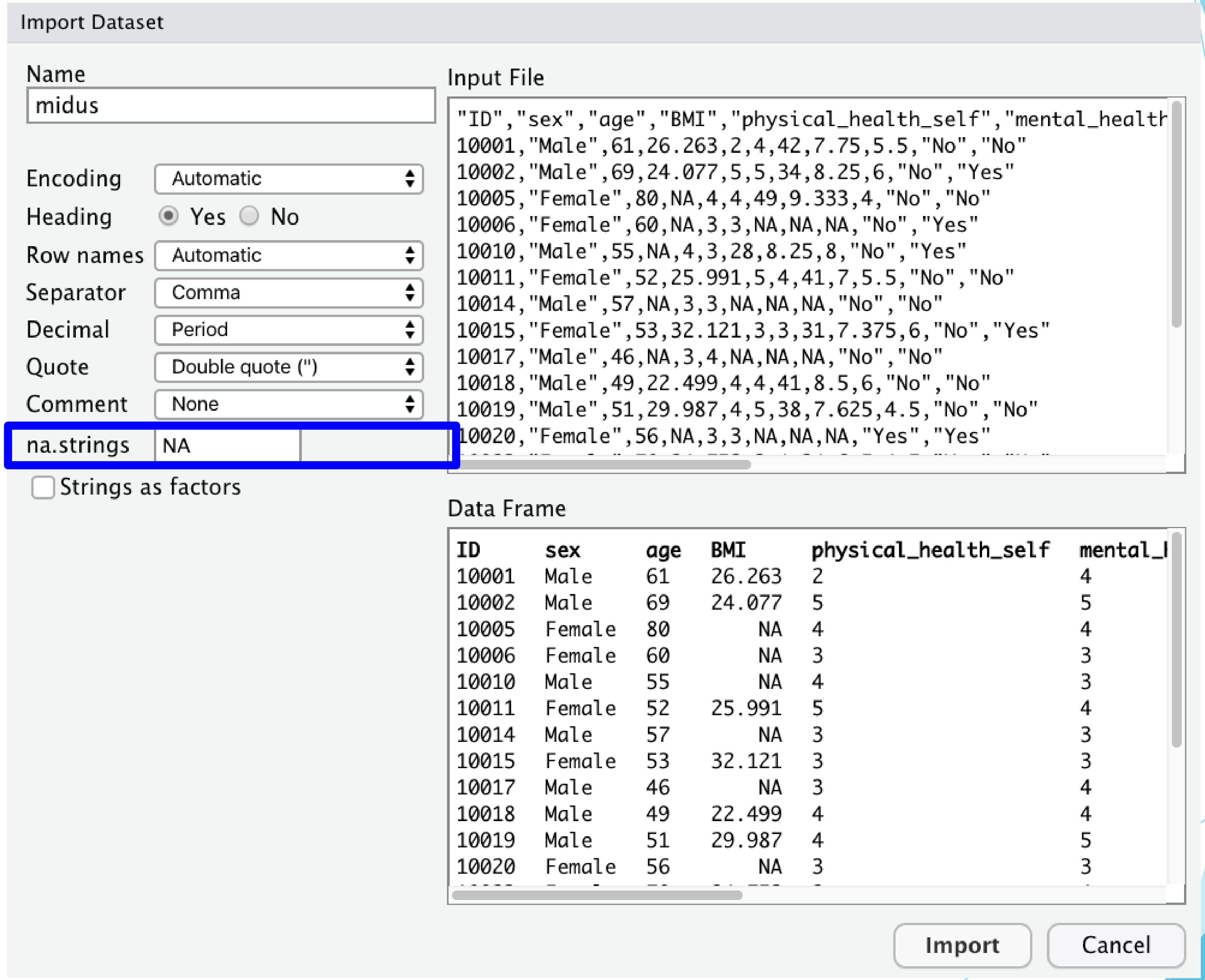
Comment: None

na.strings: NA

□ Strings as factors

```
"ID","sex","age","BMI","physical_health_self","mental_health
10001,"Male",61,26.263,2,4,42,7.75,5.5,"No","No"
10002,"Male",69,24.077,5,5,34,8.25,6,"No","Yes"
10005,"Female",80,NA,4,4,49,9.333,4,"No","No"
10006,"Female",60,NA,3,3,NA,NA,NA,"No","Yes"
10010,"Male",55,NA,4,3,28,8.25,8,"No","Yes"
10011,"Female",52,25.991,5,4,41,7,5.5,"No","No"
10014,"Male",57,NA,3,3,NA,NA,NA,"No","No"
10015,"Female",53,32.121,3,3,31,7.375,6,"No","Yes"
10017,"Male",46,NA,3,4,NA,NA,NA,"No","No"
10018,"Male",49,22.499,4,4,41,8.5,6,"No","No"
10019,"Male",51,29.987,4,5,38,7.625,4.5,"No","No"
10020,"Female",56,NA,3,3,NA,NA,NA,"Yes","Yes"
```

```
ID      sex     age   BMI     physical_health_self  mental_
10001   Male    61    26.263  2                     4
10002   Male    69    24.077  5                     5
10005   Female  80        NA  4                     4
10006   Female  60        NA  3                     3
10010   Male    55        NA  4                     3
10011   Female  52    25.991  5                     4
10014   Male    57        NA  3                     3
10015   Female  53    32.121  3                     3
10017   Male    46        NA  3                     4
10018   Male    49    22.499  4                     4
10019   Male    51    29.987  4                     5
10020   Female  56        NA  3                     3
```



Import Dataset

Name
midus

Encoding: Automatic
Heading: ● Yes ○ No
Row names: Automatic
Separator: Comma
Decimal: Period
Quote: Double quote (")
Comment: None
na.strings: NA

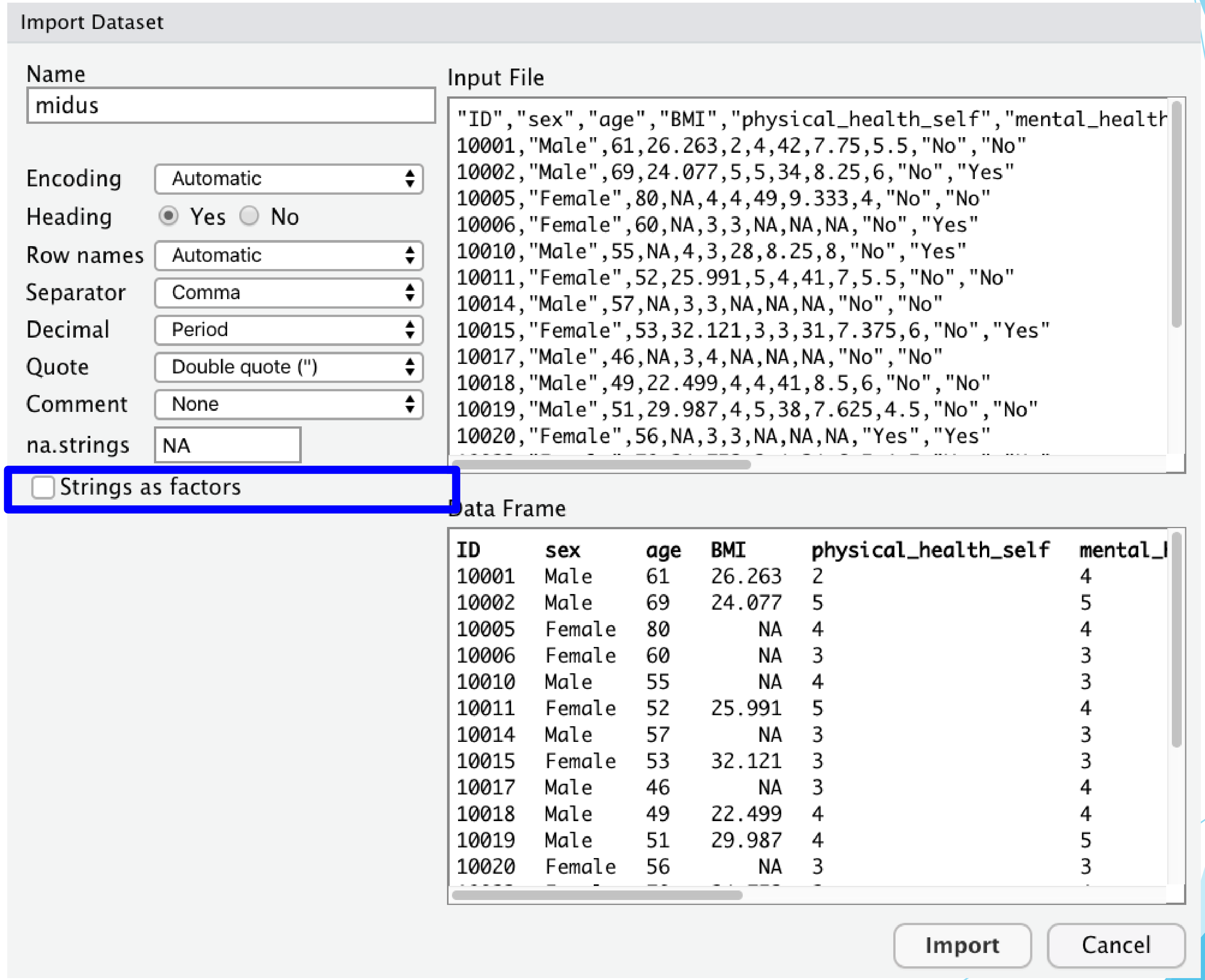
□ Strings as factors

Input File

```
"ID","sex","age","BMI","physical_health_self","mental_healt
10001,"Male",61,26.263,2,4,42,7.75,5.5,"No","No"
10002,"Male",69,24.077,5,5,34,8.25,6,"No","Yes"
10005,"Female",80,NA,4,4,49,9.333,4,"No","No"
10006,"Female",60,NA,3,3,NA,NA,NA,"No","Yes"
10010,"Male",55,NA,4,3,28,8.25,8,"No","Yes"
10011,"Female",52,25.991,5,4,41,7,5.5,"No","No"
10014,"Male",57,NA,3,3,NA,NA,NA,"No","No"
10015,"Female",53,32.121,3,3,31,7.375,6,"No","Yes"
10017,"Male",46,NA,3,4,NA,NA,NA,"No","No"
10018,"Male",49,22.499,4,4,41,8.5,6,"No","No"
10019,"Male",51,29.987,4,5,38,7.625,4.5,"No","No"
10020,"Female",56,NA,3,3,NA,NA,NA,"Yes","Yes"
```

Data Frame

```
ID      sex     age  BMI     physical_health_self  mental_
10001   Male    61   26.263  2                     4
10002   Male    69   24.077  5                     5
10005   Female  80       NA  4                     4
10006   Female  60       NA  3                     3
10010   Male    55       NA  4                     3
10011   Female  52   25.991  5                     4
10014   Male    57       NA  3                     3
10015   Female  53   32.121  3                     3
10017   Male    46       NA  3                     4
10018   Male    49   22.499  4                     4
10019   Male    51   29.987  4                     5
10020   Female  56       NA  3                     3
```



Import Dataset

Name: midus

Encoding: Automatic

Heading: ● Yes ○ No

Row names: Automatic

Separator: Comma

Decimal: Period

Quote: Double quote (")

Comment: None

na.strings: NA

□ Strings as factors

Input File

```
"ID","sex","age","BMI","physical_health_self","mental_health
10001,"Male",61,26.263,2,4,42,7.75,5.5,"No","No"
10002,"Male",69,24.077,5,5,34,8.25,6,"No","Yes"
10005,"Female",80,NA,4,4,49,9.333,4,"No","No"
10006,"Female",60,NA,3,3,NA,NA,NA,"No","Yes"
10010,"Male",55,NA,4,3,28,8.25,8,"No","Yes"
10011,"Female",52,25.991,5,4,41,7,5.5,"No","No"
10014,"Male",57,NA,3,3,NA,NA,NA,"No","No"
10015,"Female",53,32.121,3,3,31,7.375,6,"No","Yes"
10017,"Male",46,NA,3,4,NA,NA,NA,"No","No"
10018,"Male",49,22.499,4,4,41,8.5,6,"No","No"
10019,"Male",51,29.987,4,5,38,7.625,4.5,"No","No"
10020,"Female",56,NA,3,3,NA,NA,NA,"Yes","Yes"
```

Data Frame

```
ID      sex     age   BMI      physical_health_self  mental_h
10001   Male    61    26.263   2                     4
10002   Male    69    24.077   5                     5
10005   Female  80        NA   4                     4
10006   Female  60        NA   3                     3
10010   Male    55        NA   4                     3
10011   Female  52    25.991   5                     4
10014   Male    57        NA   3                     3
10015   Female  53    32.121   3                     3
10017   Male    46        NA   3                     4
10018   Male    49    22.499   4                     4
10019   Male    51    29.987   4                     5
10020   Female  56        NA   3                     3
```

Loading .csv files

```
midus <- read.csv("~/Desktop/rSkillLab/midus.csv")
```

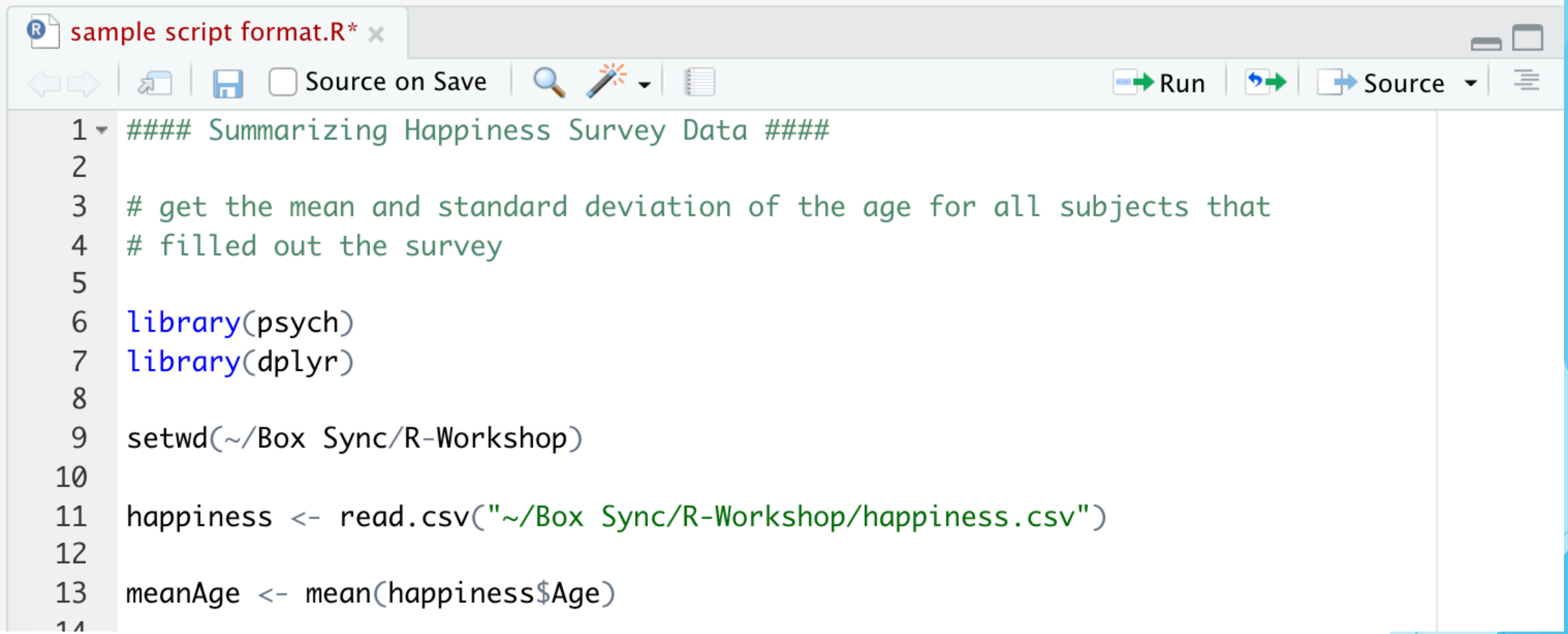
This will appear in your console

We strongly recommend copying and pasting this code into your script file!

Typical workflow in R

1. Open a script (new or existing).
2. Prepare to run analyses:
   - Set your working directory/open an RProject
   - Load your data
   - Load any packages you might want to use etc…
3. Write/run analyses.
4. Save!
   - Make sure that this includes the code to open your .csv from your Dropbox/Box/Github etc.
   - Again, note: R doesn't change the original data file!

Typical Format of .R File



```
#### Summarizing Happiness Survey Data ####

# get the mean and standard deviation of the age for all subjects that
# filled out the survey

library(psych)
library(dplyr)

setwd(~/Box Sync/R-Workshop)

happiness <- read.csv("~/Box Sync/R-Workshop/happiness.csv")

meanAge <- mean(happiness$Age)
```

Cleaning Global Environment

Sometimes your Global Environment can fill up with stuff that you don’t need. You can clean this!

One option: Delete EVERYTHING using the broom

Another option: Switch to GRID view, check boxes of individual objects you DON’T want, then press the broom button

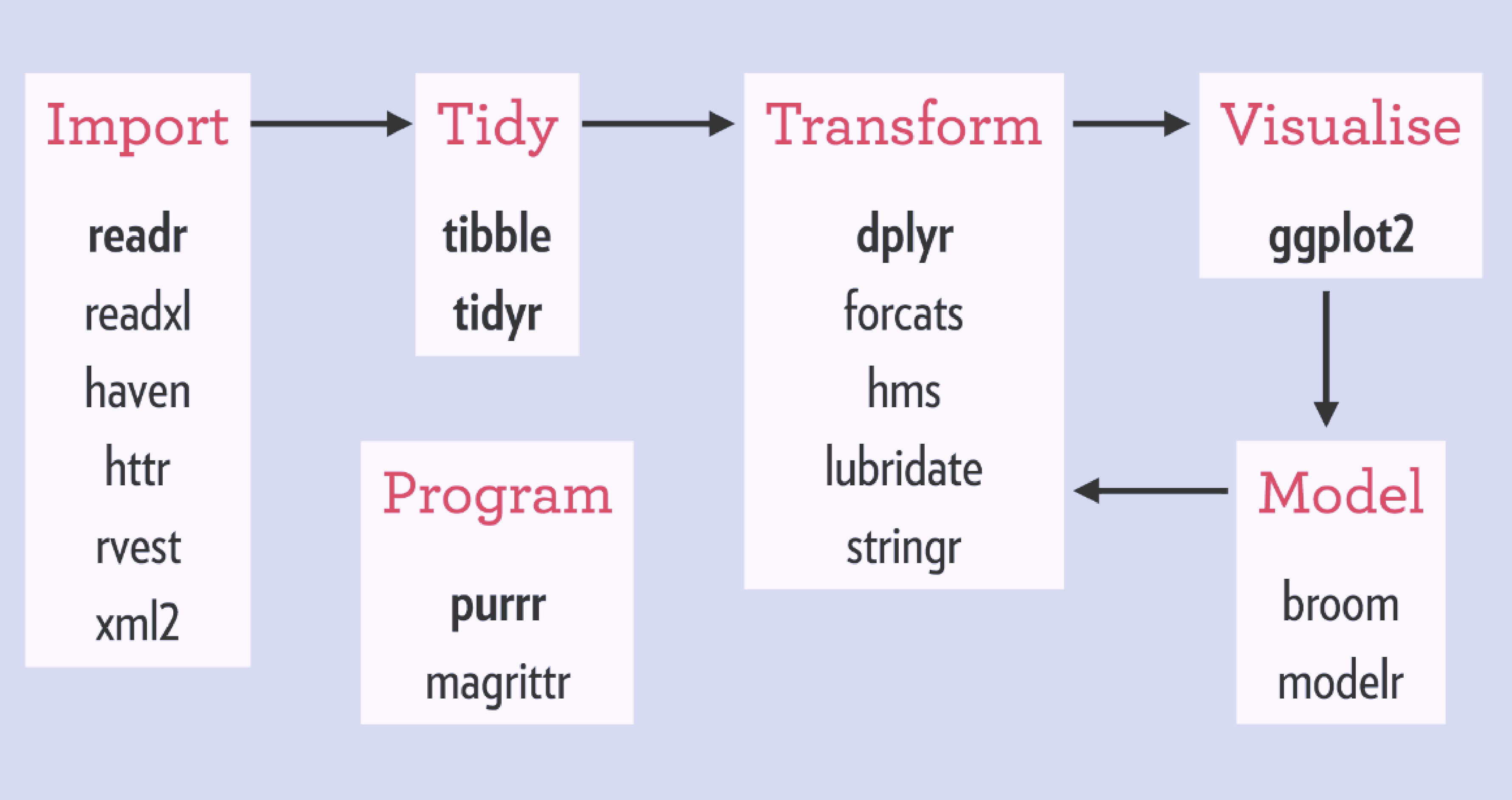
Agenda

- Understanding the basics of functions
- Get familiar with help documentation
- Working with packages
- R Scripts and loading files
- **Data Wrangling with `tidyverse`**

Data Prep for Analyses

- Open whatever script you have been using
- Add a comment that the next section will be with tidyverse
- Import the midus.csv file & load the tidyverse package
- Over-write the object so that there is no missing data (or no NA) values using the following code:
  -
    ```
    midus <- na.omit(midus)
    ```

"The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs."



Import

readr

readxl

haven

httr

rvest

xml2

Tidy

tibble

tidyr

Transform

dplyr

forcats

hms

lubridate

stringr

Visualise

ggplot2

Model

broom

modelr

Program

purrr

magrittr

tidyverse

- `library(tidyverse)`
  - Loads the tidyverse
- `tidyverse_packages()`
  - Lists all of the included packages
- When you load the tidyverse package, you’re actually loading a lot of packages at once!
- Today we mainly care about `dplyr`, `tidyr`, and Jenine will talk at length about `ggplot2`

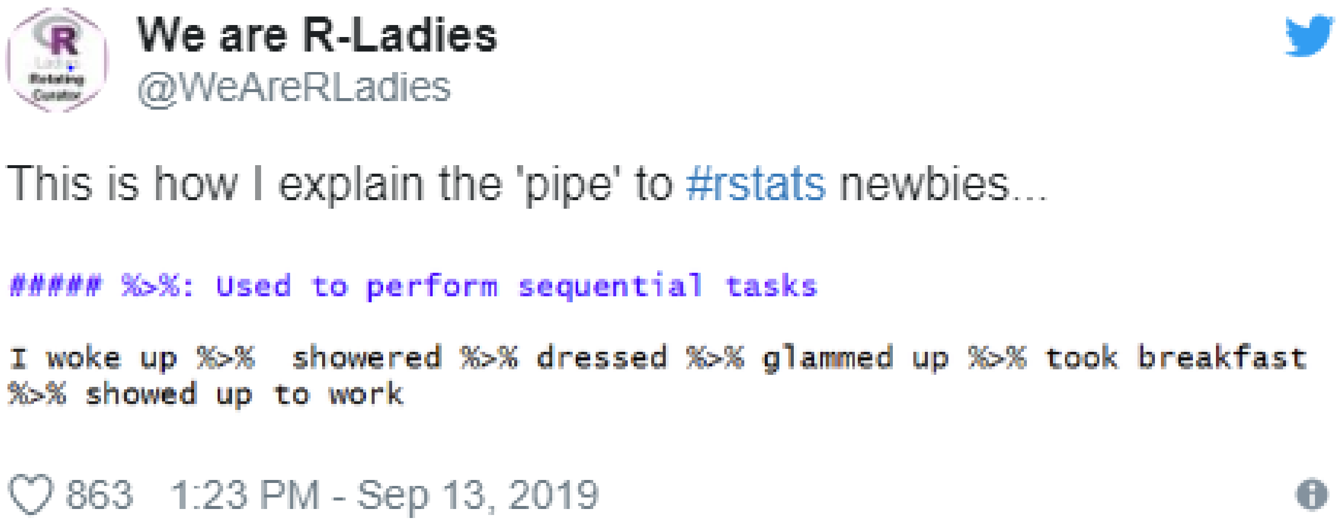
Piping

- All of the tidyverse packages use *piping* as a way to make your code easier to read.
- Format:

```
originalData %>%
  function1(someVariable) %>%
  function2(someVariable)
```

Piping

- Helps you write code that is easier to understand and read
- Used to perform sequential tasks
- Can be read as “and then”



We are R-Ladies
@WeAreRLadies

This is how I explain the 'pipe' to #rstats newbies...

```
##### %>%: Used to perform sequential tasks

I woke up %>%  showered %>% dressed %>% glammed up %>% took breakfast
%>% showed up to work
```

♡ 863   1:23 PM - Sep 13, 2019

- Tips:
  - Don’t use <- inside the piped operation (only at the beginning if you want to save output
  - Add %>% to the end of each line – this %>% is the actual “pipe”

dplyr Operations

All used for “transforming” your data.

`filter()` provides basic filtering capabilities

`select()` selecting variables

`mutate()` create new variables

`summarise()` summarize data by functions of choice

`group_by()` groups data by categorical levels

Function #1: Filter

- Alternative to the `subset()` function and/or indexing
- Still requires logical operators!

```
originalData %>%
  filter(variable == something)

Womean <- midus %>%
           filter(sex != “Male”)
```

should get 2061 obs

Function #1: Filter

```
originalData %>%
  filter(variable == something)

Womean <- midus %>%
             filter(sex != “Male”)
```

==	equality
!=	inequality
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

should get 2061 obs

Function #2: Select

- Alternative to indexing
- Can take in indexes, variable names or both

```
originalData %>%
  select(1, 3:4, VariableName)

midus %>%
  select(age, BMI, 9:11)
```

Function #3: Mutate

- `mutate()` is kind of tricky. On it’s own, it will simply add a new variable based on something.

What if we wanted to get the square root of the BMI variable?

```
midus %>%
  mutate(BMI_sqrt = sqrt(BMI))
```

Function #3: Mutate

- BUT, you can add different endings (suffixes) to it
  - `mutate_at()`
  - `mutate_all()`
  - `mutate_if()`

- I find `mutate_at()` to be most useful. It’s especially nice for making sure the variables you need to be factors are, indeed, factors!

Function #3: Mutate

- EXAMPLE:
  - What if we filtered by 2 different categorical variables?
  - If we filter, then we should have a different number of categories than when we began.
  - We can use `mutate_at` to make sure those particular variables get re-converted into a factor, so that we have the correct number of levels

Exercise

- Using tidyverse code do the following in a single code "chunk"
  - Get rid out outliers whose BMI is more than 2 standard deviations away from the mean of BMI
  - Keep only the following variables: ID, sex, age, physical_health_self, mental_health_self, and self_esteem
  - Now add a column that is the z-score of self_esteem. Do this "by hand":

$$\frac{x - \mu}{\sigma}$$

Function #4: Summarize

- Great for *_shock_*…summarizing your data
- Useful if you want to make a bar plot of means, and if you want error bars for standard deviation.

```
originalData %>%
  summarize(meanVar = mean(var))
```

You can go crazy with this!

```
midus %>%
   summarize(meanAge = mean(age),
             sdAge = sd(age),
             meanBMI = mean(BMI),
             sdBMI = sd(BMI))
```

Function #5: Group_By

- What if we wanted to get the means and standard deviations, but you wanted it *per level* of a factor?

```
originalData %>%
  group_by(groupingVariable) %>%
  summarize(means, sds)
```

- **YOU TRY!**

  What does your result look like?????

dplyr Operations

All used for “transforming” your data.

`filter()` provides basic filtering capabilities

`select()` selecting variables

`mutate()` create new variables

`summarise()` summarize data by functions of choice

`group_by()` groups data by categorical levels

*** Lots of others, but you’ll need to explore those on your own! Some others:*

- *`recode`*
- *`arrange`*
- *`n_distinct`*
- *Joining multiple data.frames together*

tidyr Operations

All used for “tidying” – getting your data in the format you want.

Technically, this is a separate package from dplyr. But it uses the same syntax (piping) and does very similar things.

Some tidyr functions I find particularly useful

`pivot_wider()` long to wide format

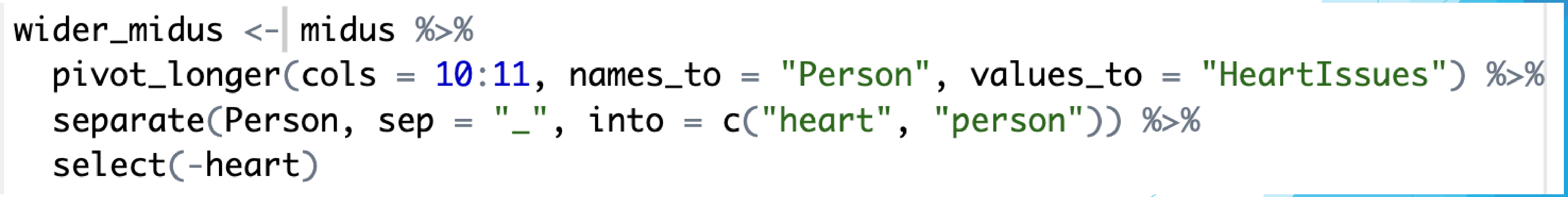
`pivot_longer()` wide to long format

`separate()` splits a single column into multiple columns

`unite()` combines multiple columns into a single column

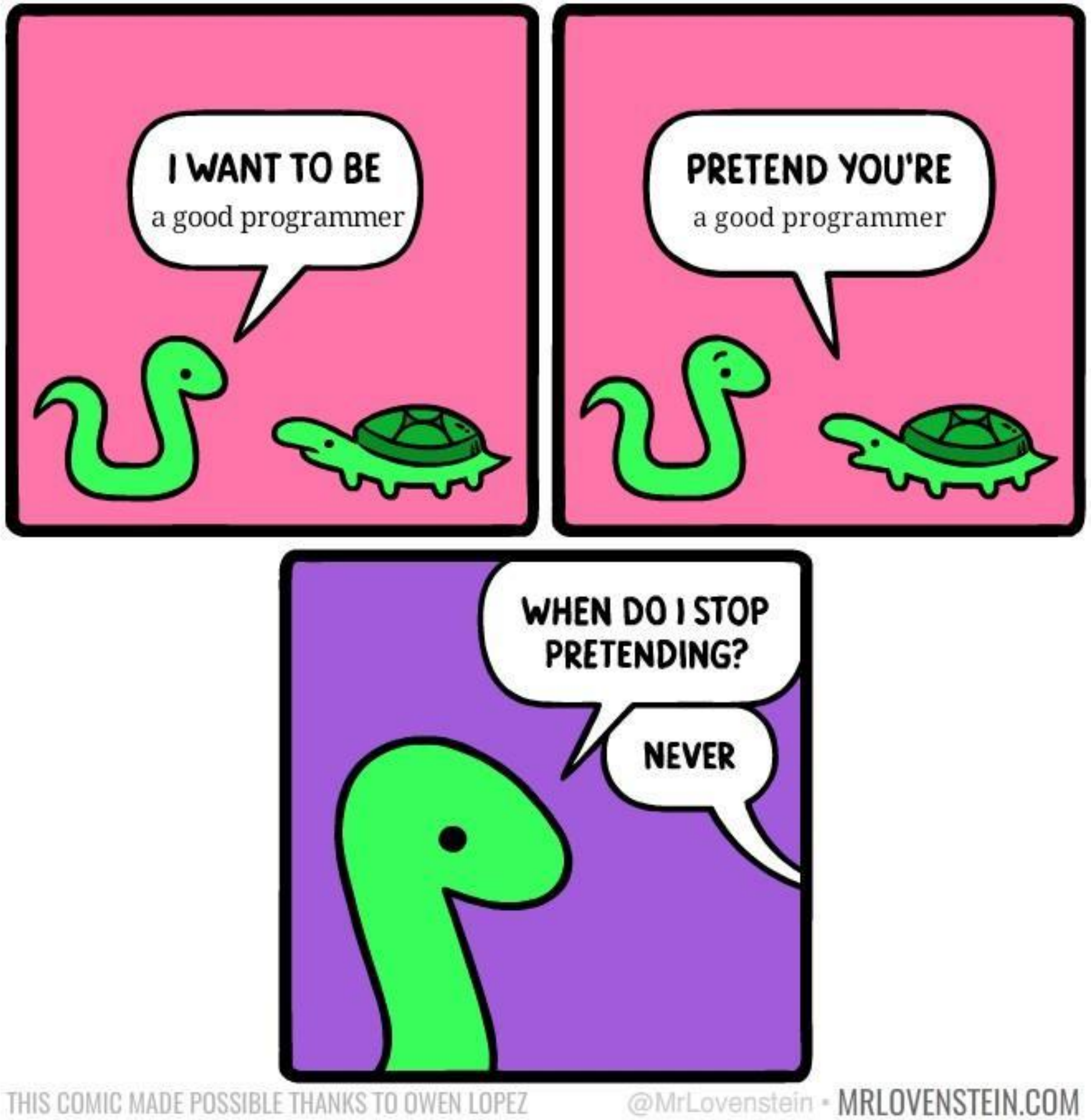
Let’s walk through an example together

- First we pivot to make it into the LONG format
- Then I separate 1 column into 2 columns
- Then I remove an unused column



```r
wider_midus <- midus %>%
  pivot_longer(cols = 10:11, names_to = "Person", values_to = "HeartIssues") %>%
  separate(Person, sep = "_", into = c("heart", "person")) %>%
  select(-heart)
```

Final Thoughts


I WANT TO BE a good programmer
PRETEND YOU'RE a good programmer
WHEN DO I STOP PRETENDING?
NEVER
THIS COMIC MADE POSSIBLE THANKS TO OWEN LOPEZ
@MrLovenstein • MRLOVENSTEIN.COM

Final Thoughts

- Practice, practice, practice
- If you want to throw your computer out a window, you’re doing it right!
- Google is your friend
- Thoughts on Chat—GPT
- Resources:
  - R-Ladies STL
  - Cheatsheets
  - R4DS
  - Reddit & Twitter
- R is particularly good at data wrangling, statistics, and plotting. Use wisely!