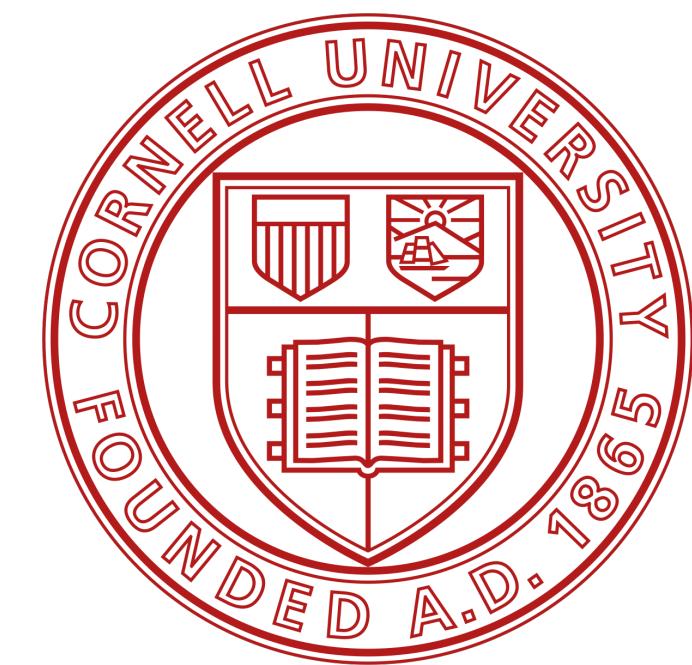


Introduction to R programming

STSCI 2120

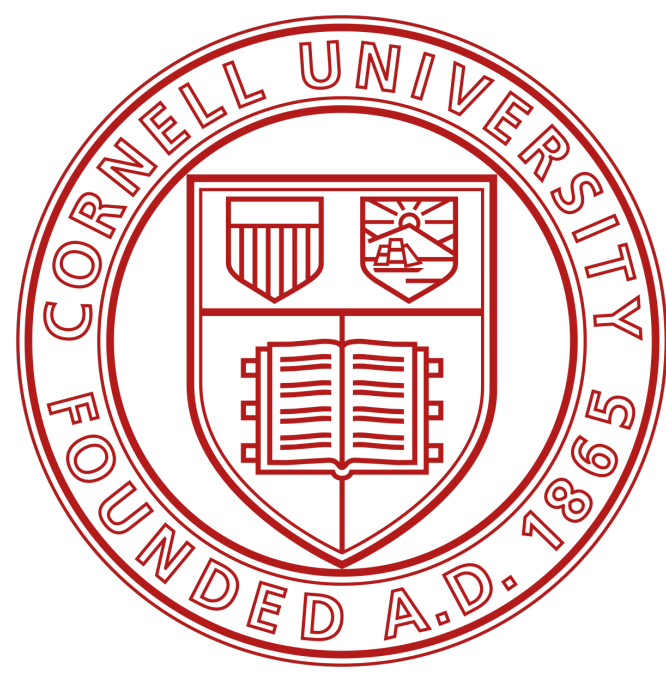
Nayel Bettache - Fall 2024

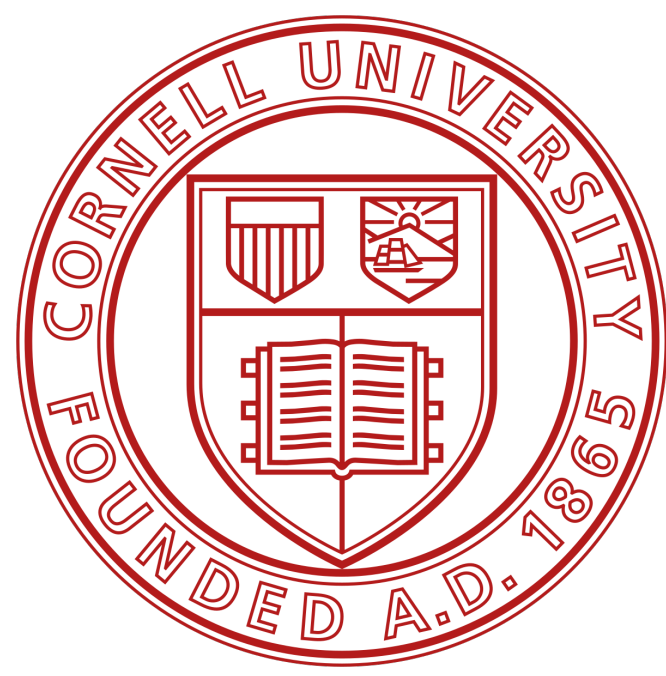


Personal Presentation

Nayel Bettache

Visiting Assistant Professor



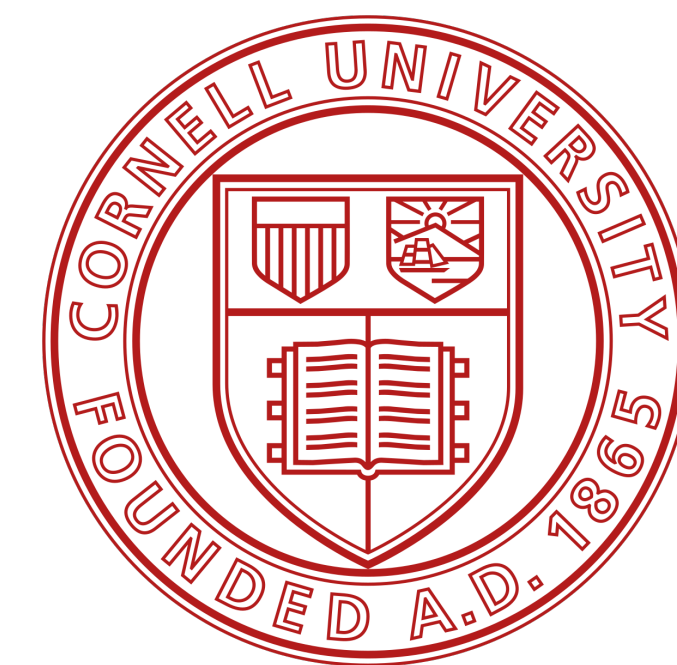


Nayel Bettache

Visiting Assistant Professor

- PhD in mathematical statistics from Institut Polytechnique de Paris



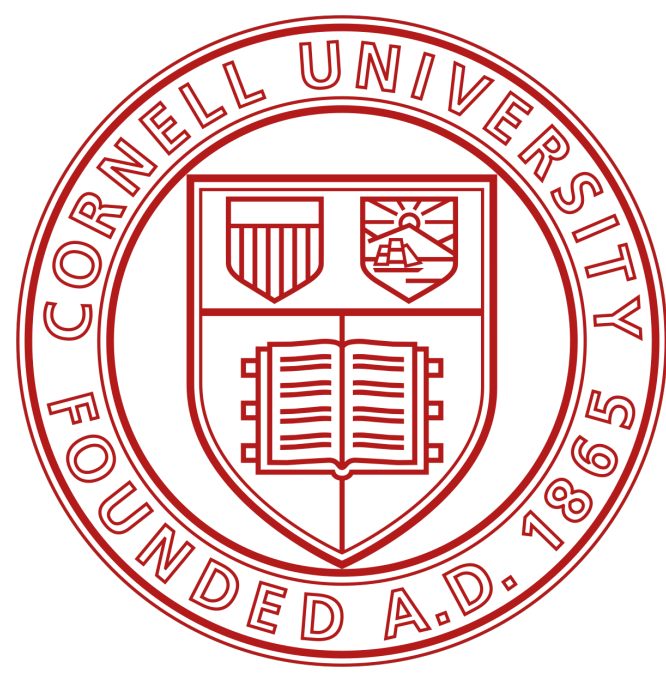


Nayel Bettache

Visiting Assistant Professor

- PhD in mathematical statistics from Institut Polytechnique de Paris
- MSc in DataScience from Ecole Polytechnique



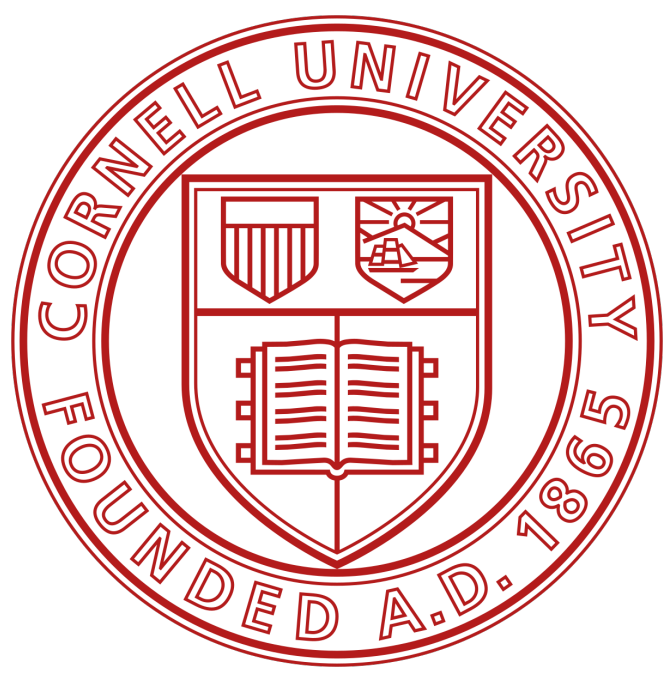


Nayel Bettache

Visiting Assistant Professor

- PhD in mathematical statistics from Institut Polytechnique de Paris
- MSc in DataScience from Ecole Polytechnique
- MSc in Statistics from ENSAE Paris

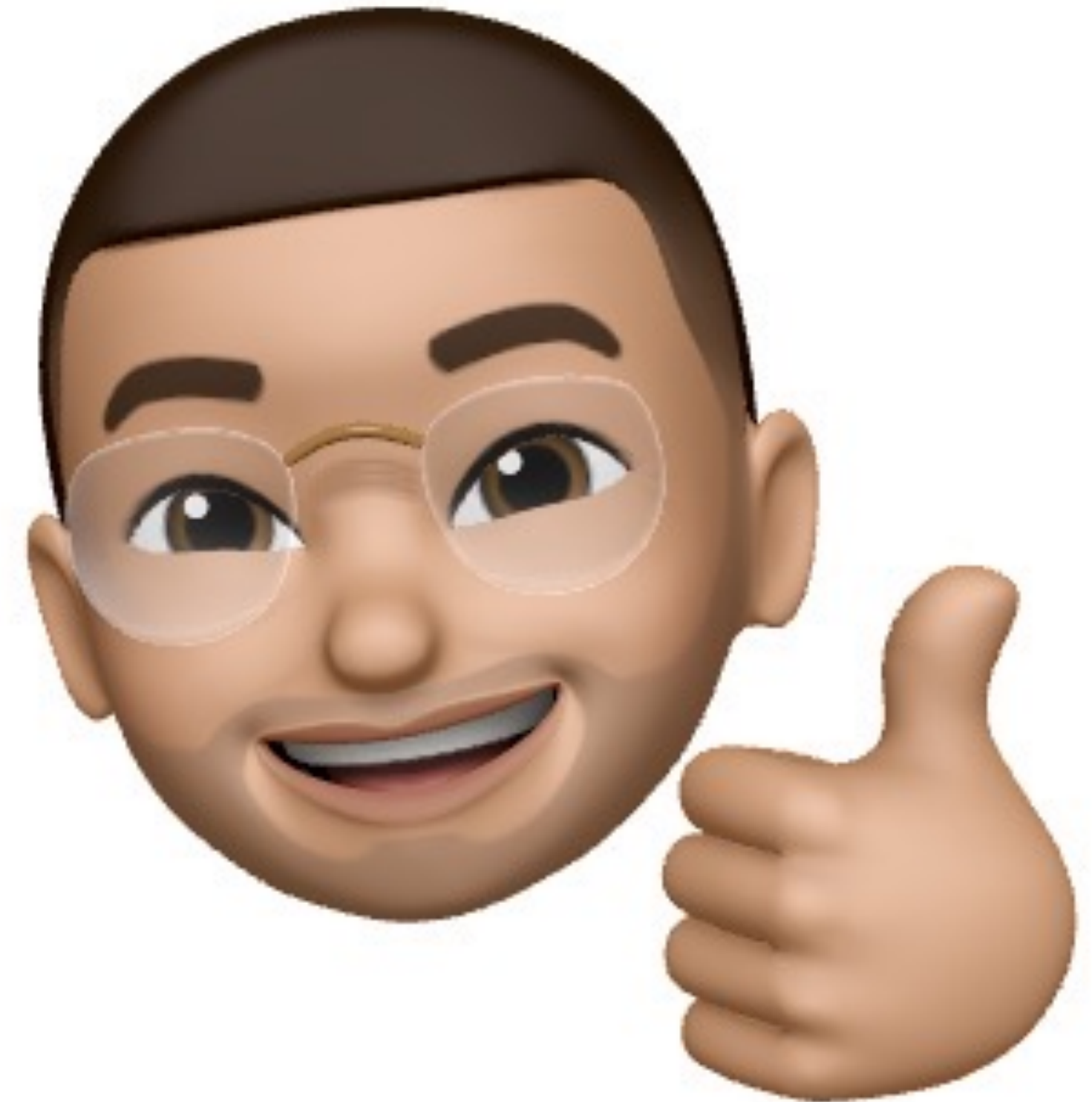
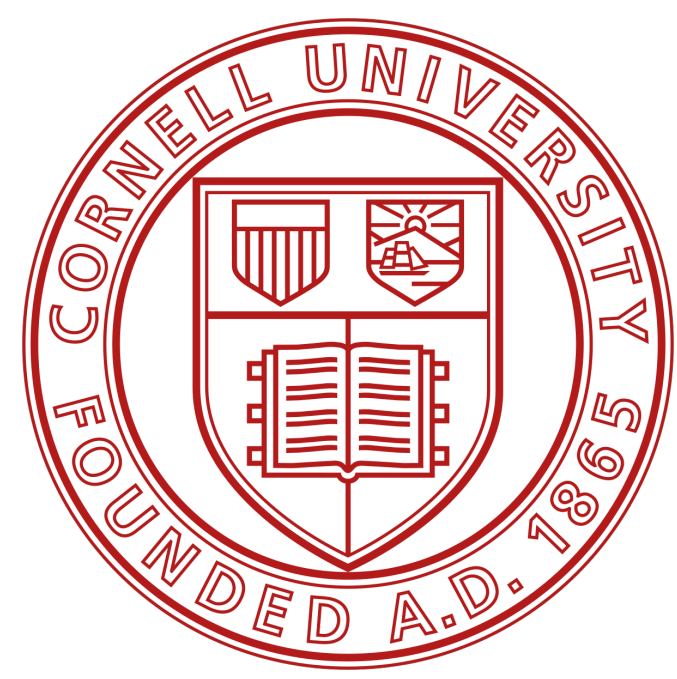




Overview of the course

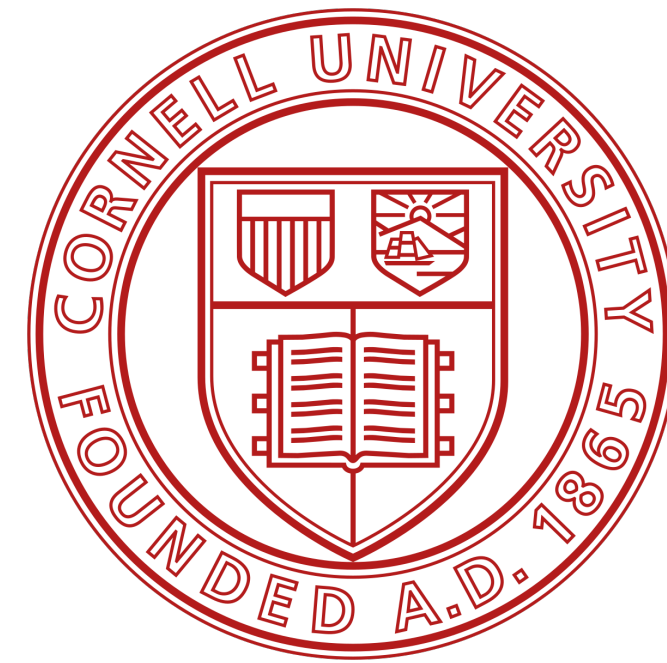
What will you learn ?

Objectives



What will you learn ?

Objectives

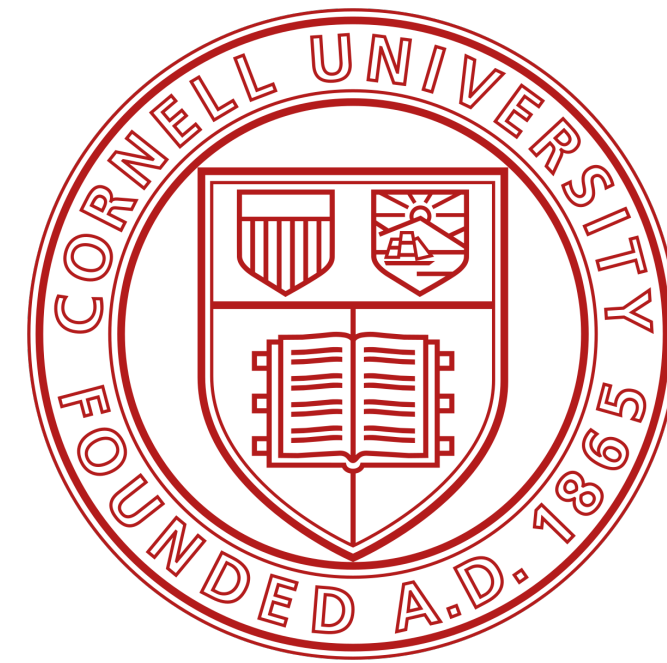


- Help you learn the most important tools in R



What will you learn ?

Objectives

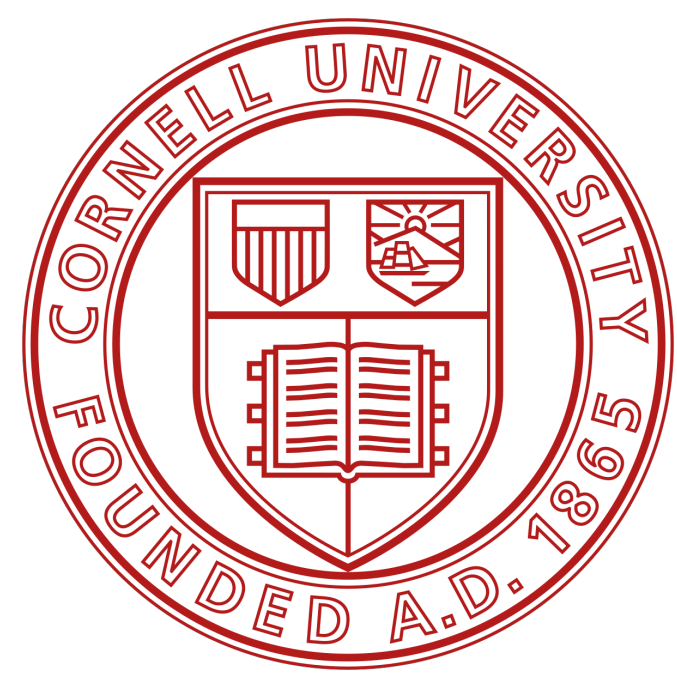


- Help you learn the most important tools in R
- Allow you to do data science efficiently and reproducibly



What will you learn ?

Objectives

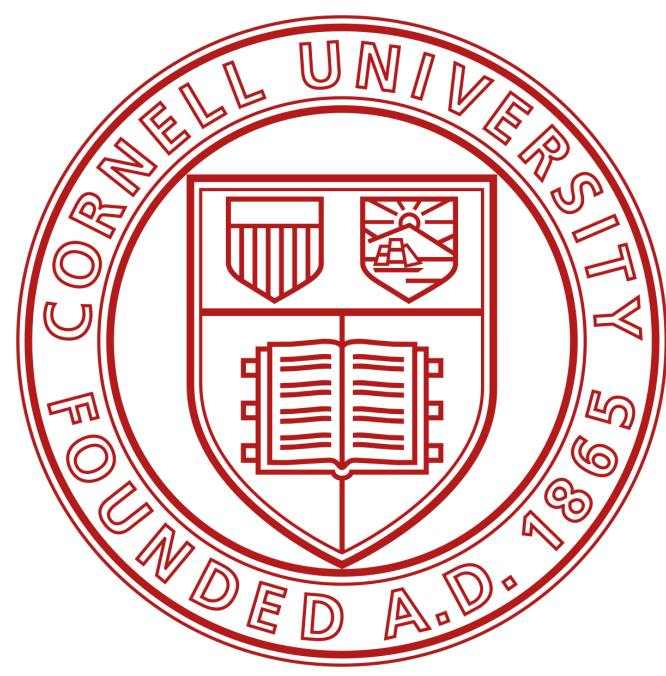


- Help you learn the most important tools in R
- Allow you to do data science efficiently and reproducibly
- Have some fun

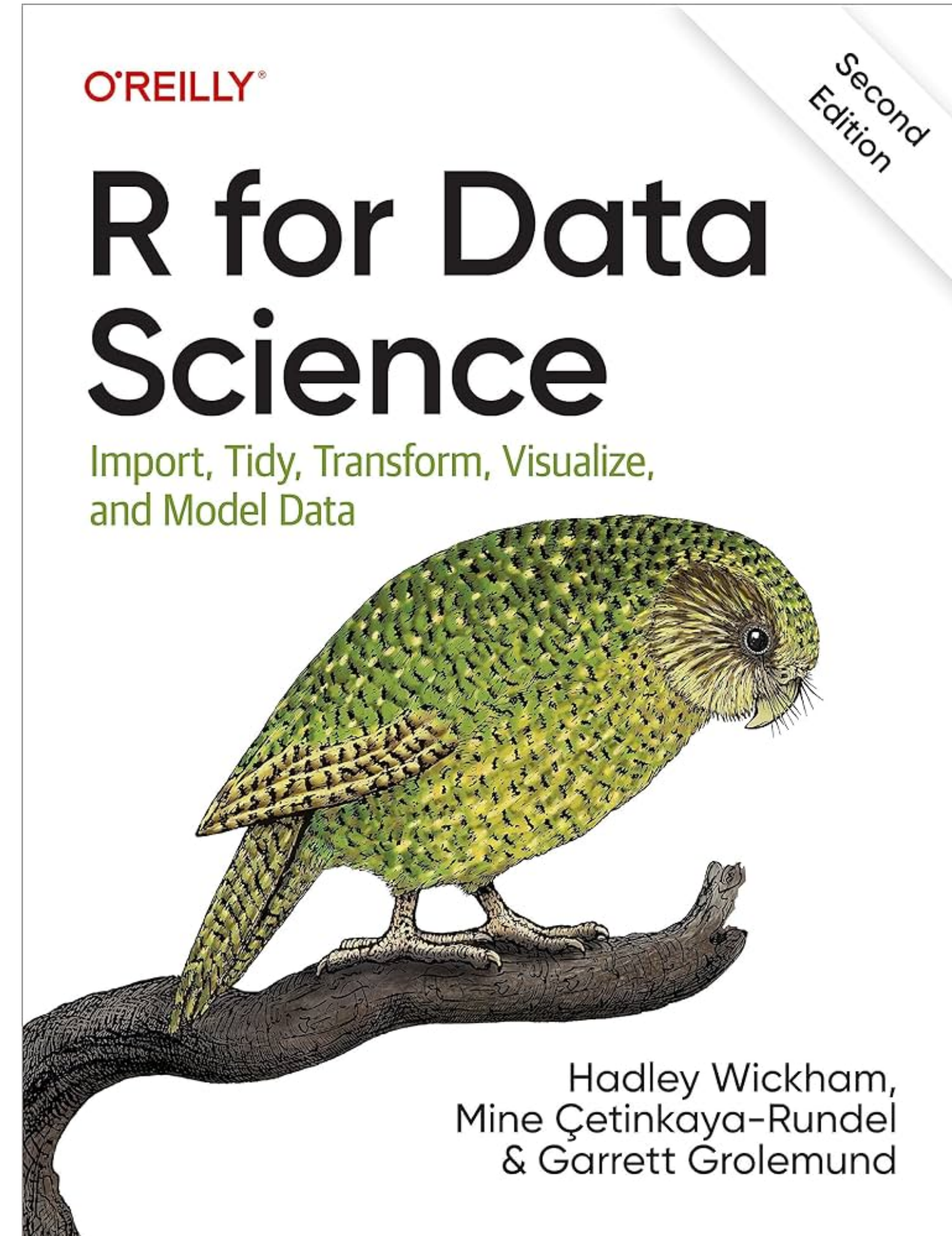


What will you learn ?

Ressources

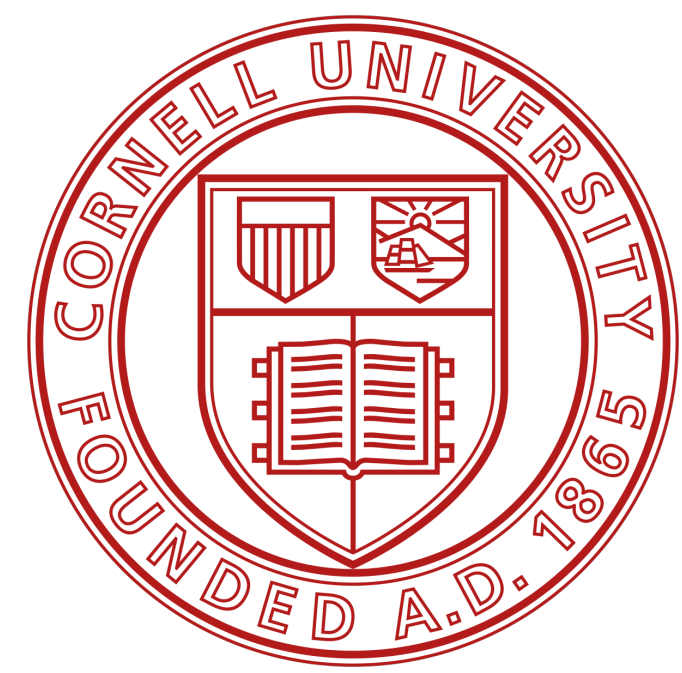


- R for Data Science

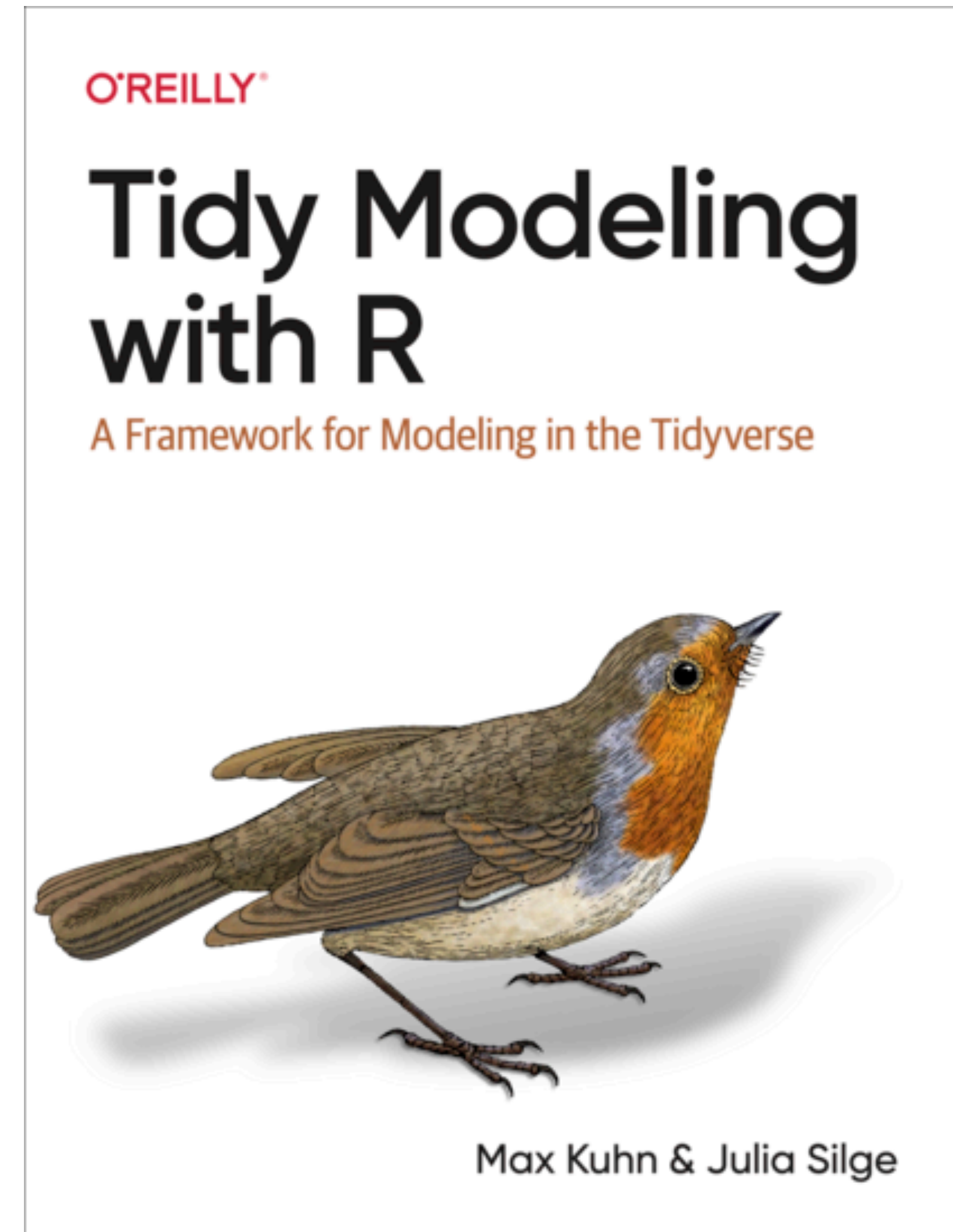


What will you learn ?

Ressources

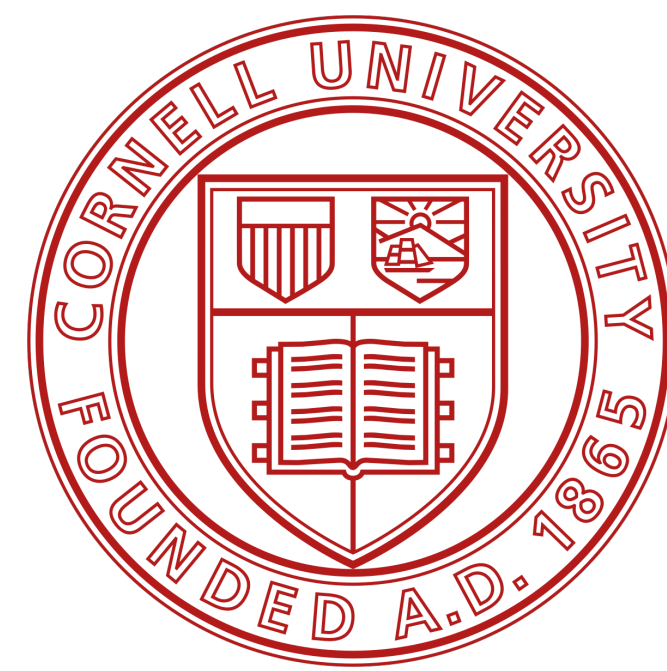


- R for Data Science
- Tidy modeling with R



What will you learn ?

Ressources



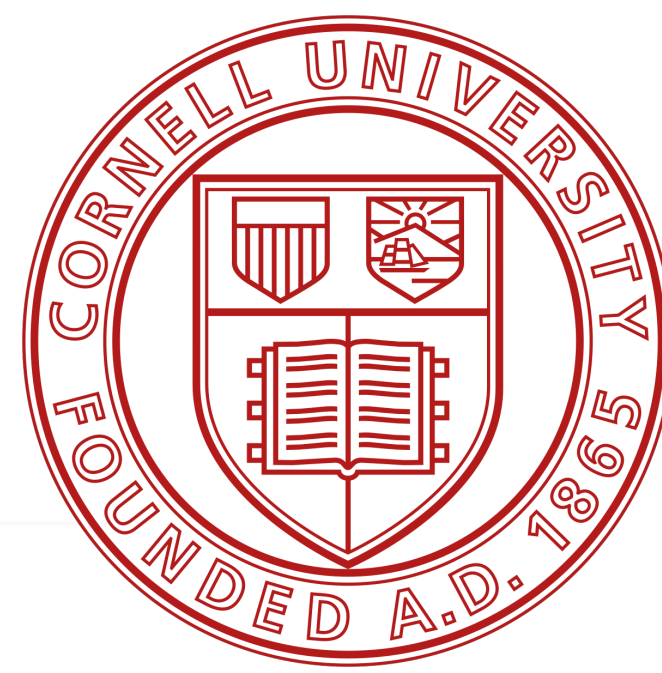
- R for Data Science
- Tidy modeling with R
- R programming tutorial



What will you learn ?

Ressources

- R for Data Science
- Tidy modeling with R
- R programming tutorial
- An introduction to R

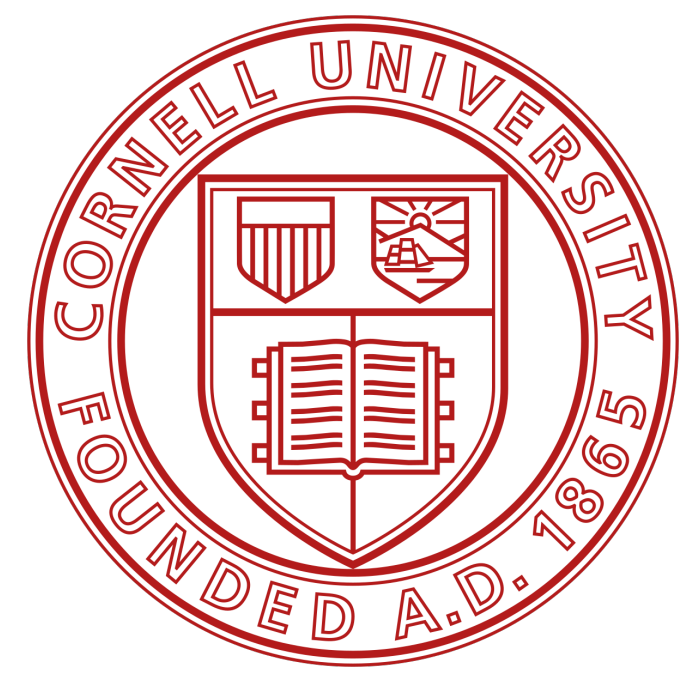


An Introduction to R

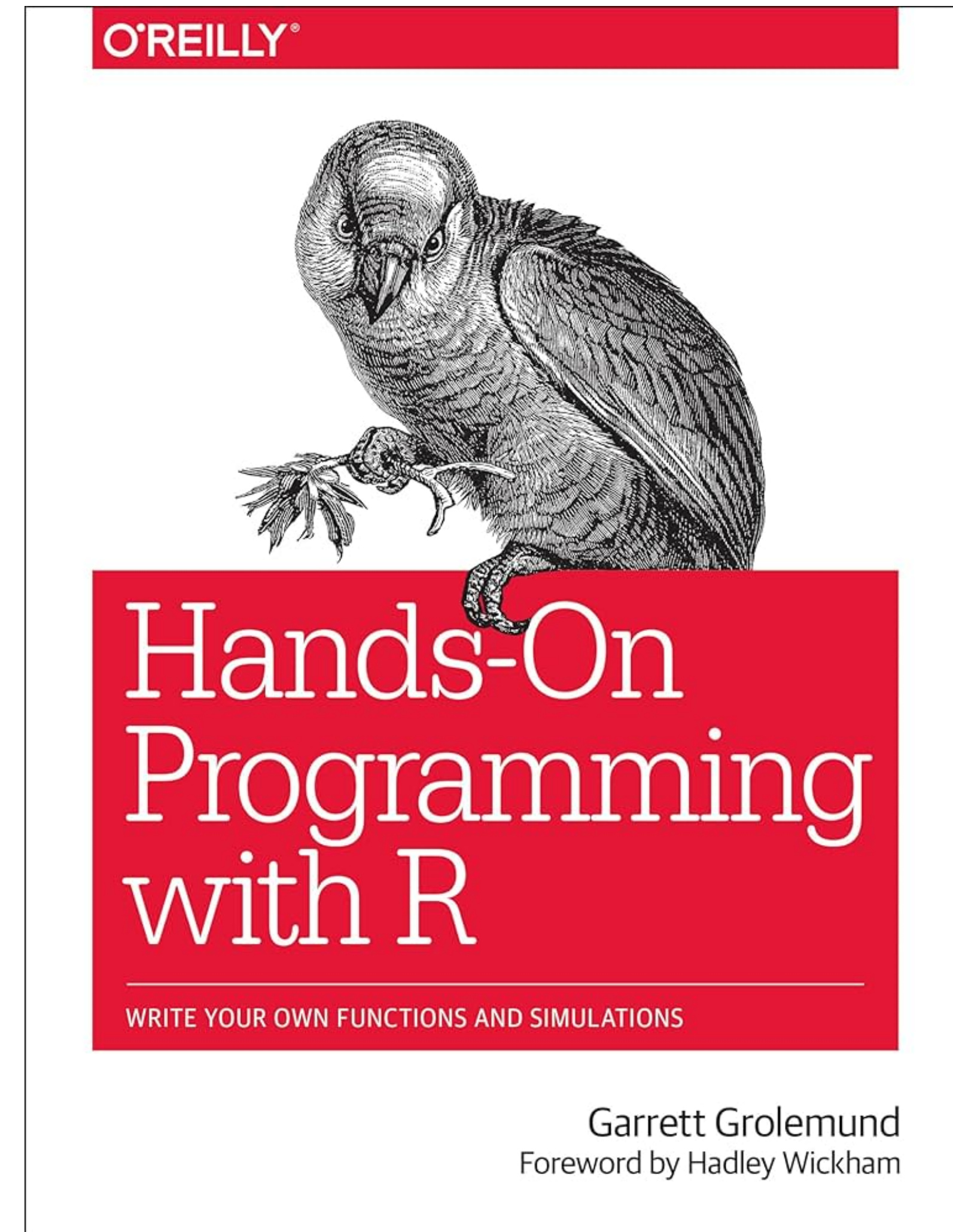
Notes on R: A Programming Environment for Data Analysis and Graphics
Version 4.4.1 (2024-06-14)

What will you learn ?

Ressources

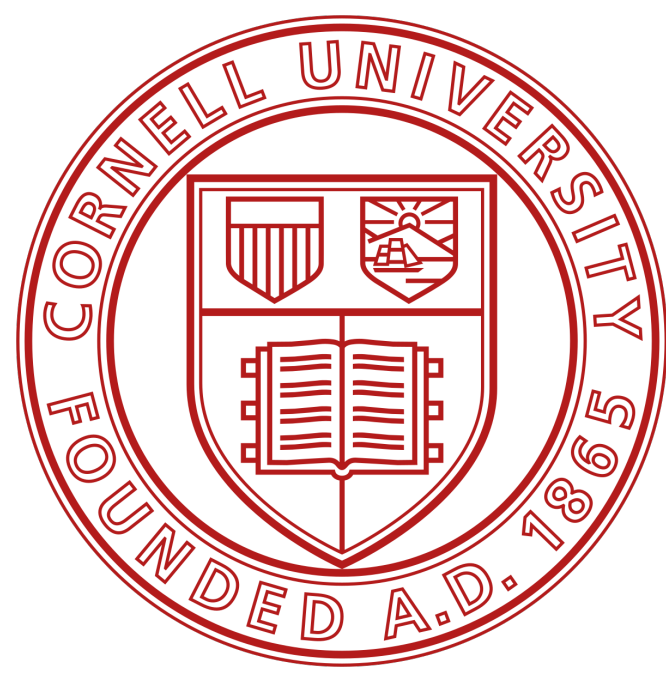


- R for Data Science
- Tidy modeling with R
- R programming tutorial
- An introduction to R
- Hands on Programming with R

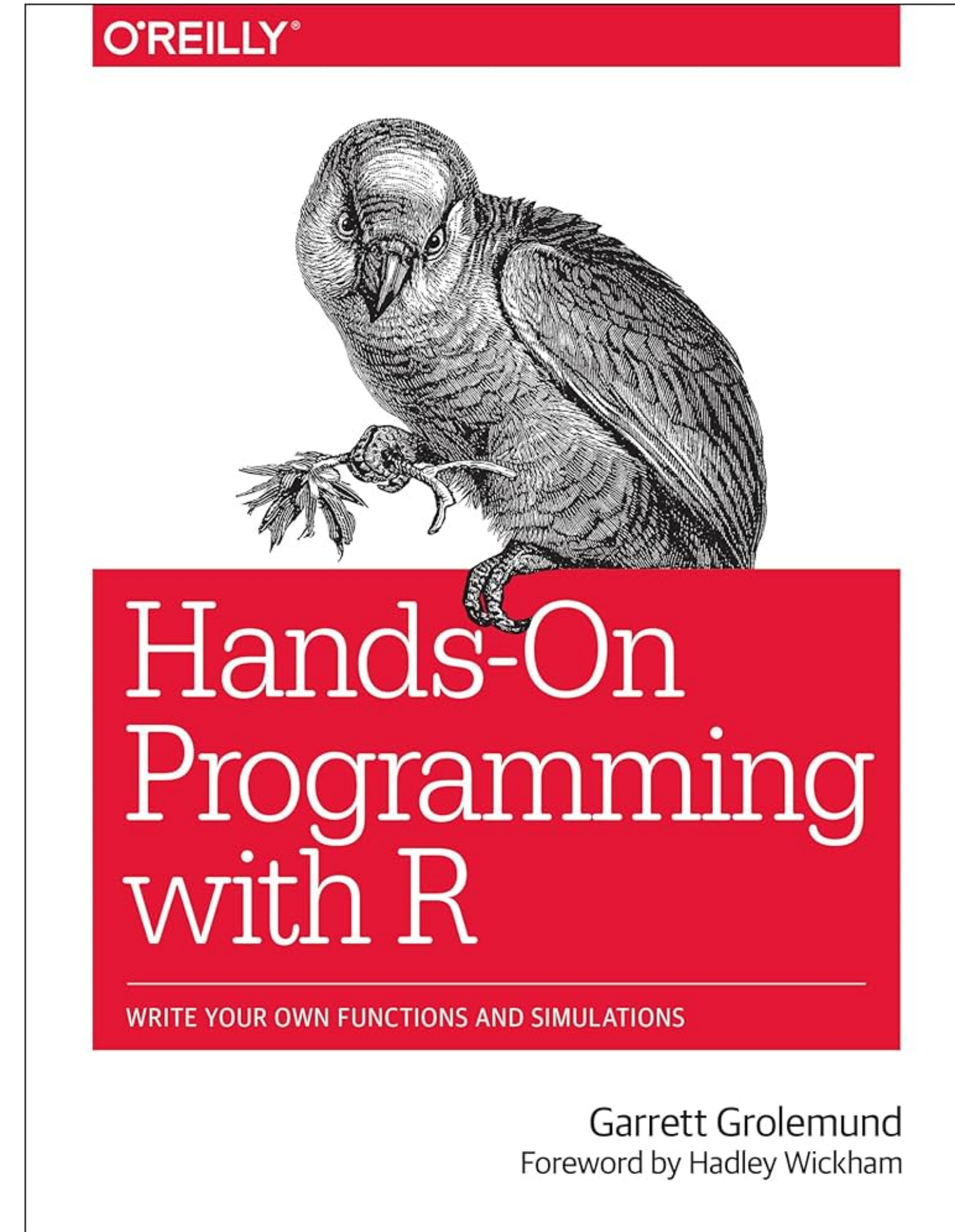


What will you learn ?

Main ressource

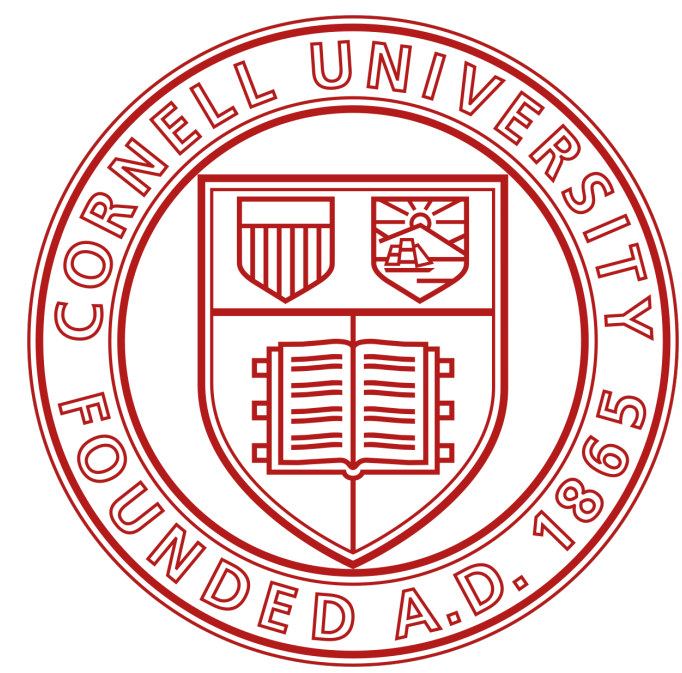


- Hands on Programming with R



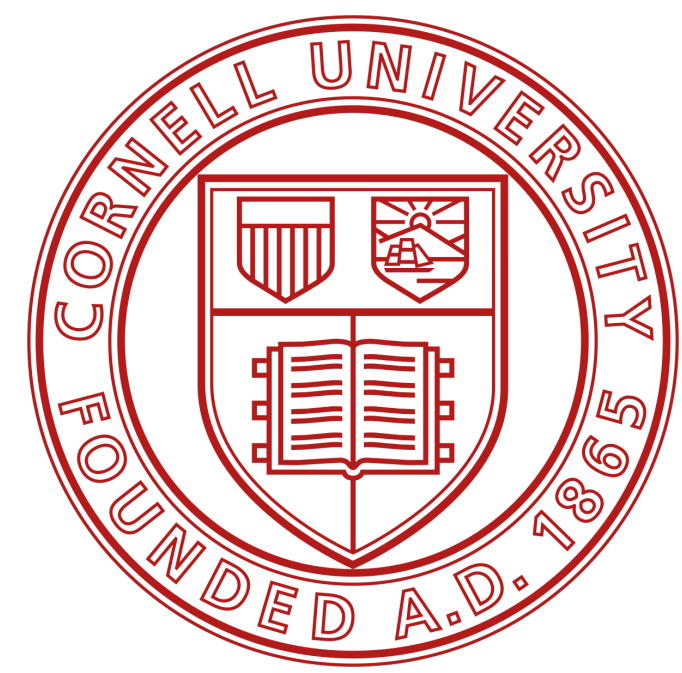
What will you learn ?

Details



What will you learn ?

Details

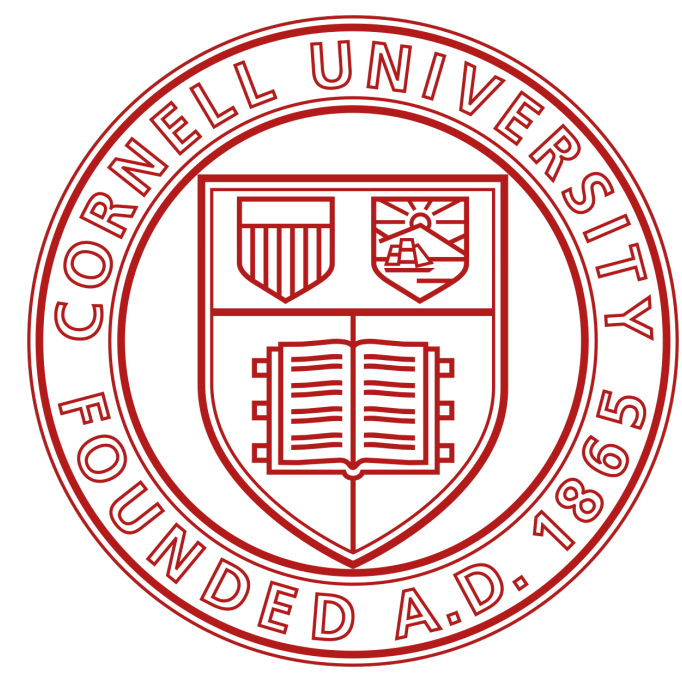


- <https://nayelbettache.github.io/STSCI2120.html>



What will you learn ?

Details

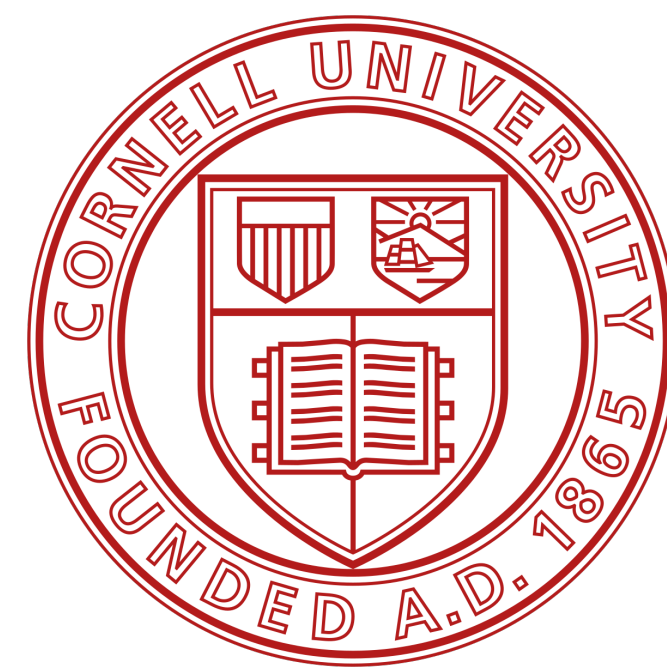


- <https://nayelbettache.github.io/STSCI2120.html>
- Info about the course, homework, syllabus.



What will you learn ?

Details

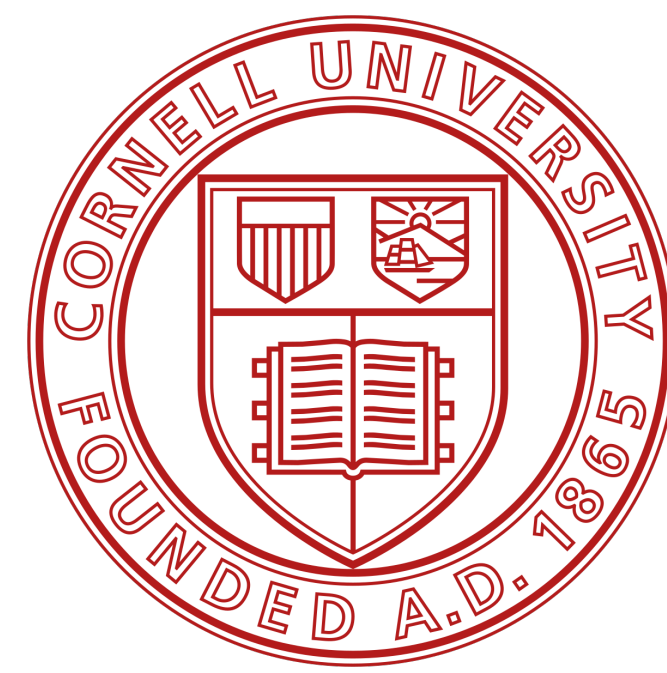


- <https://nayelbettache.github.io/STSCI2120.html>
- Info about the course, homework, syllabus.
- Lectures on Tuesdays and Thursdays, 10:10am-11:25am, Emerson Hall 135.



What will you learn ?

Details

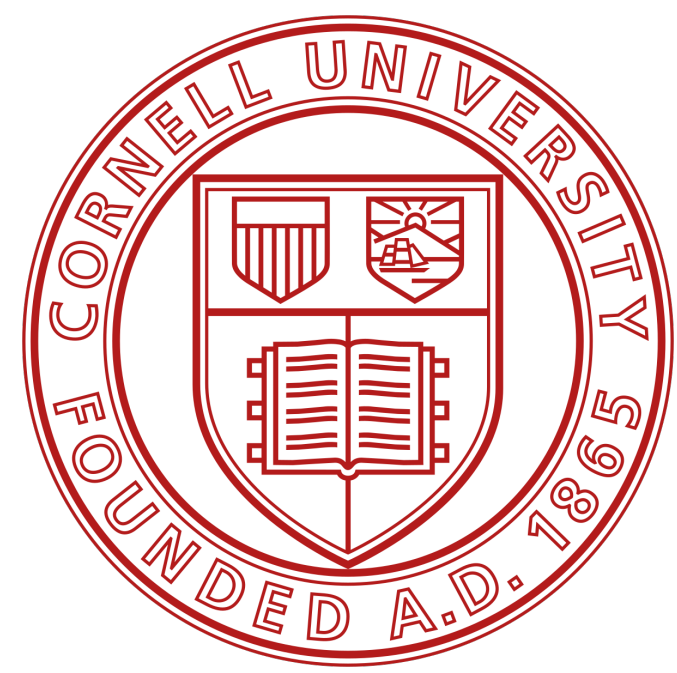


- <https://nayelbettache.github.io/STSCI2120.html>
- Info about the course, homework, syllabus.
- Lectures on Tuesdays and Thursdays, 10:10am-11:25am, Emerson Hall 135.
- Office hours: Thursdays 2:30 - 3:30



What won't you learn ?

Details

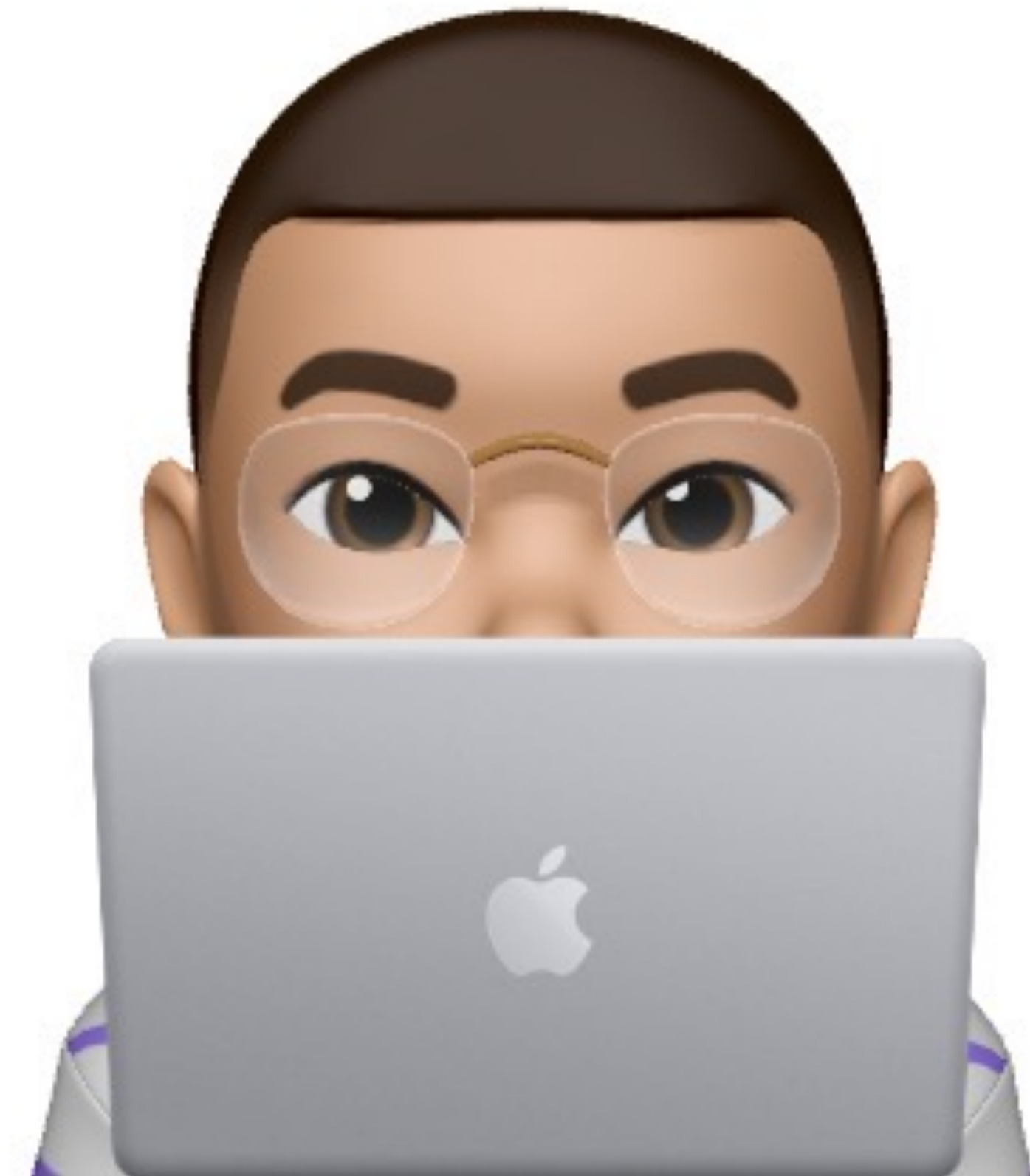
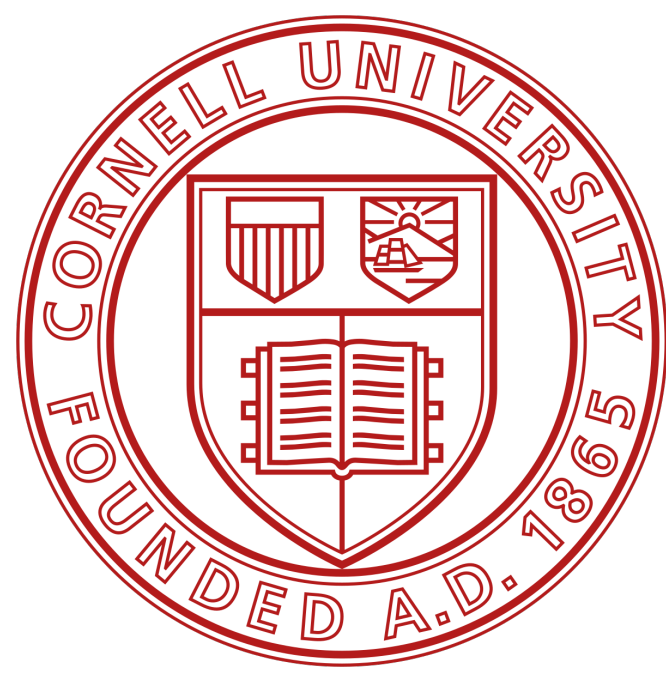


- Modeling
- Big data
- Python, Julia and others



7 weeks

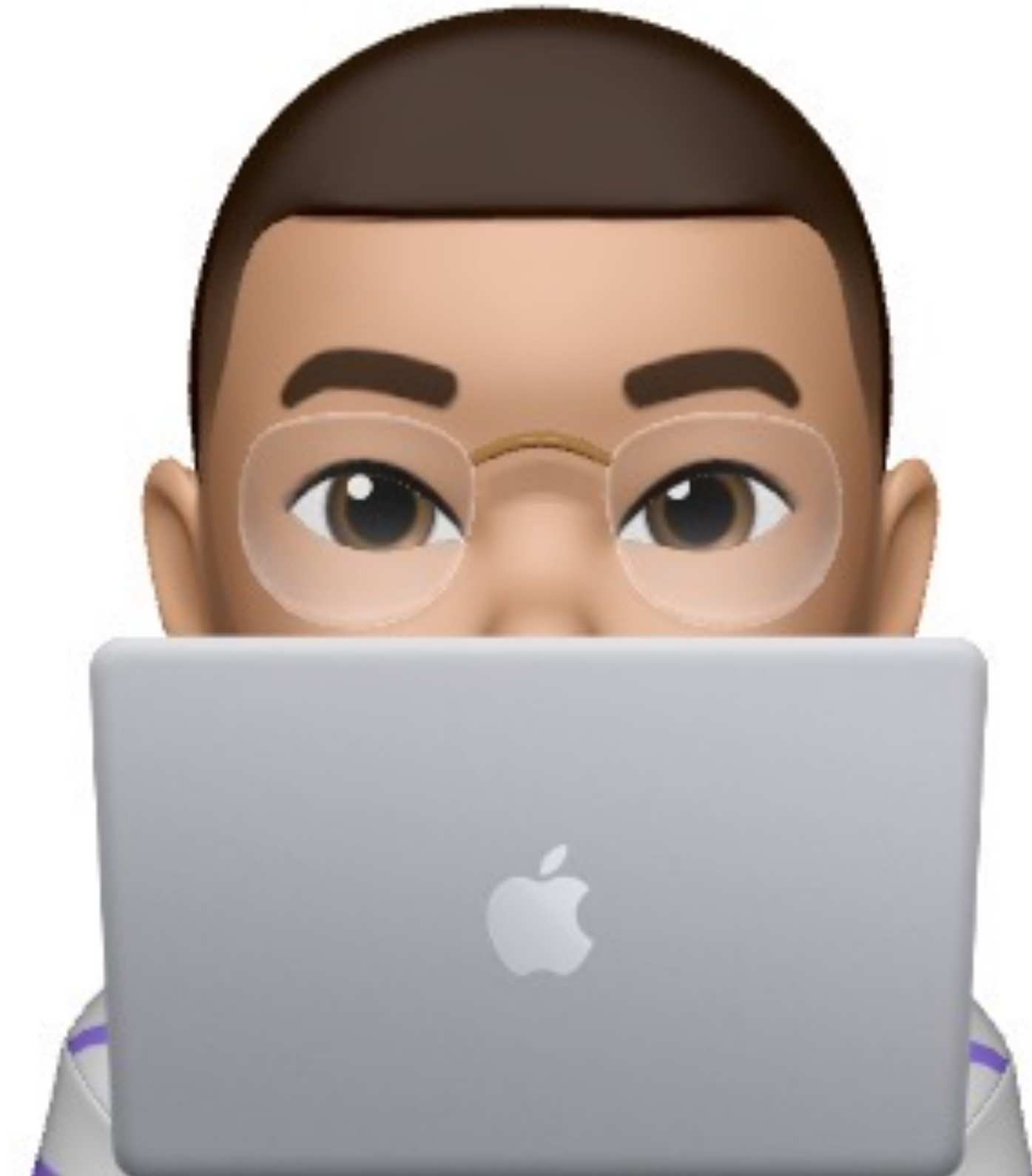
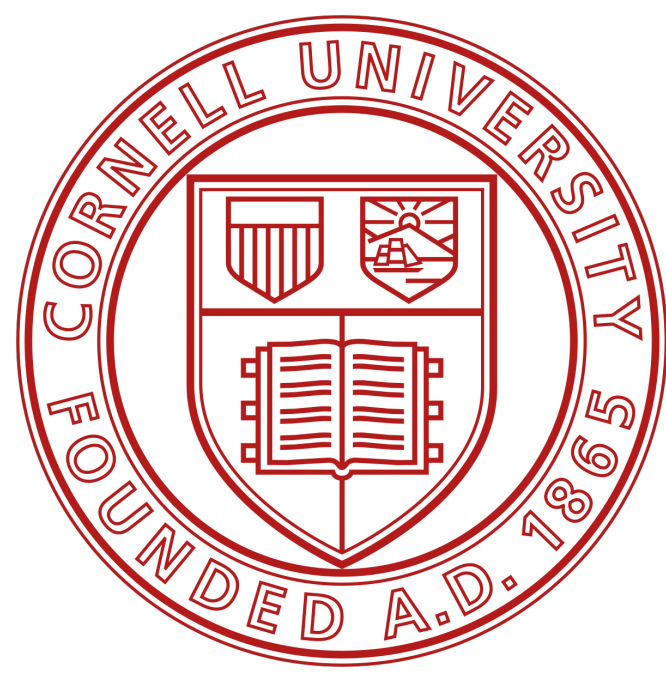
Overview



7 weeks

Overview

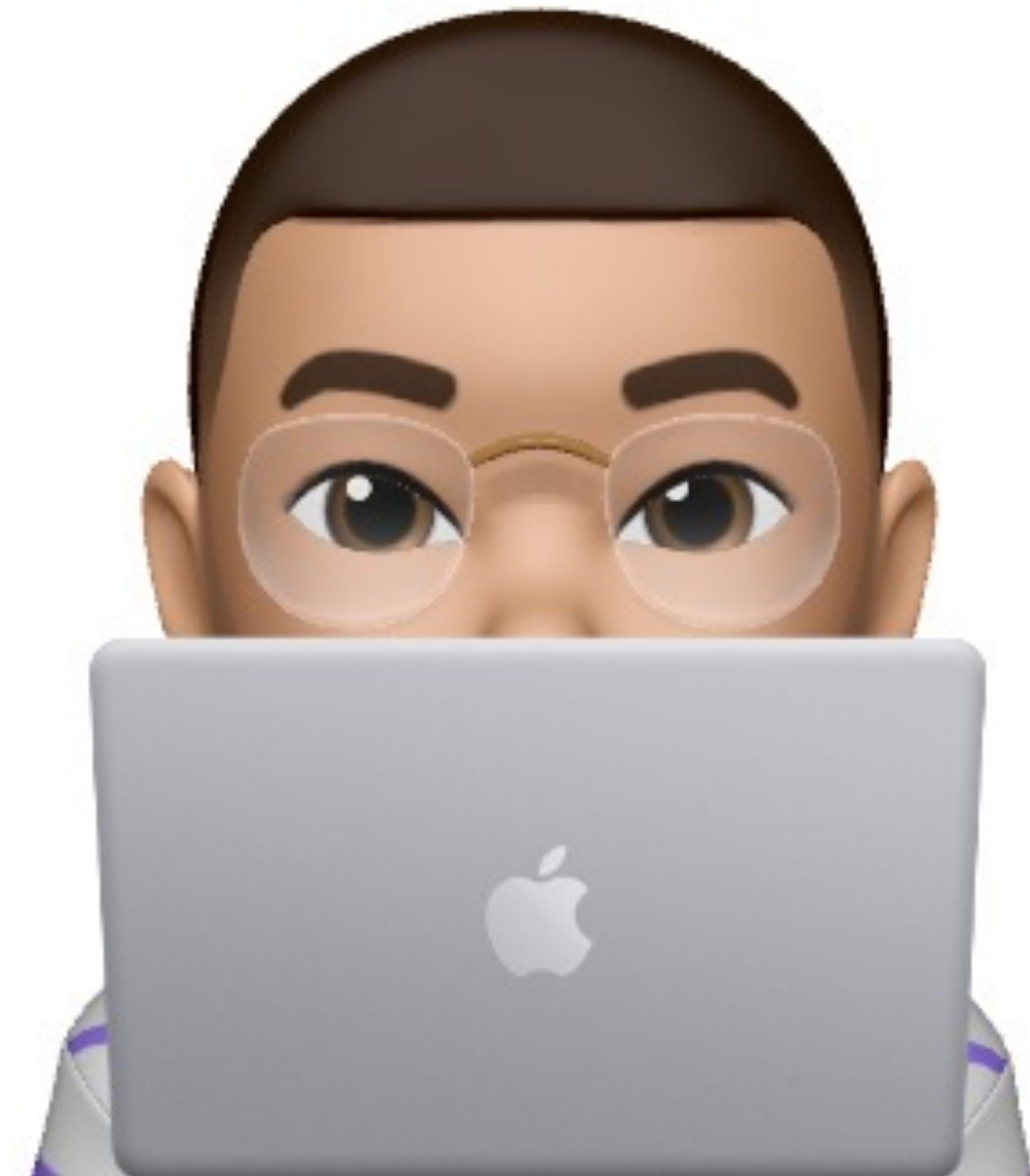
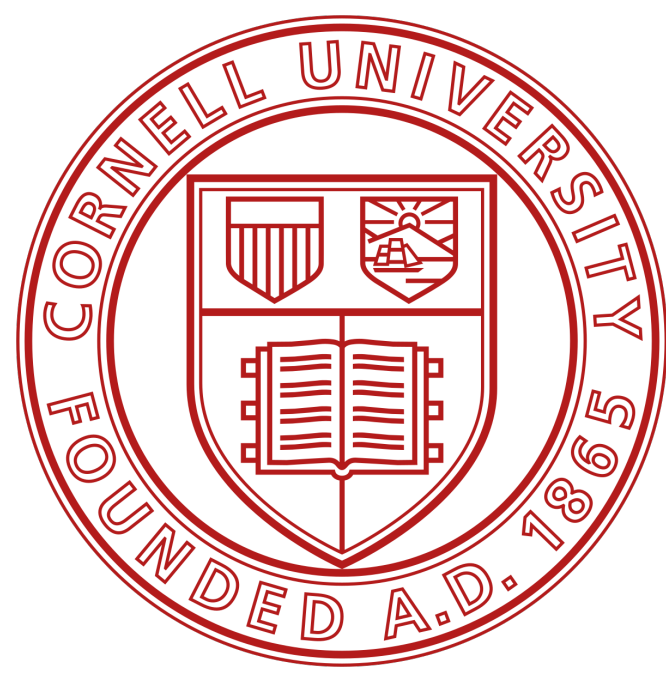
- W1: Introduction to R



7 weeks

Overview

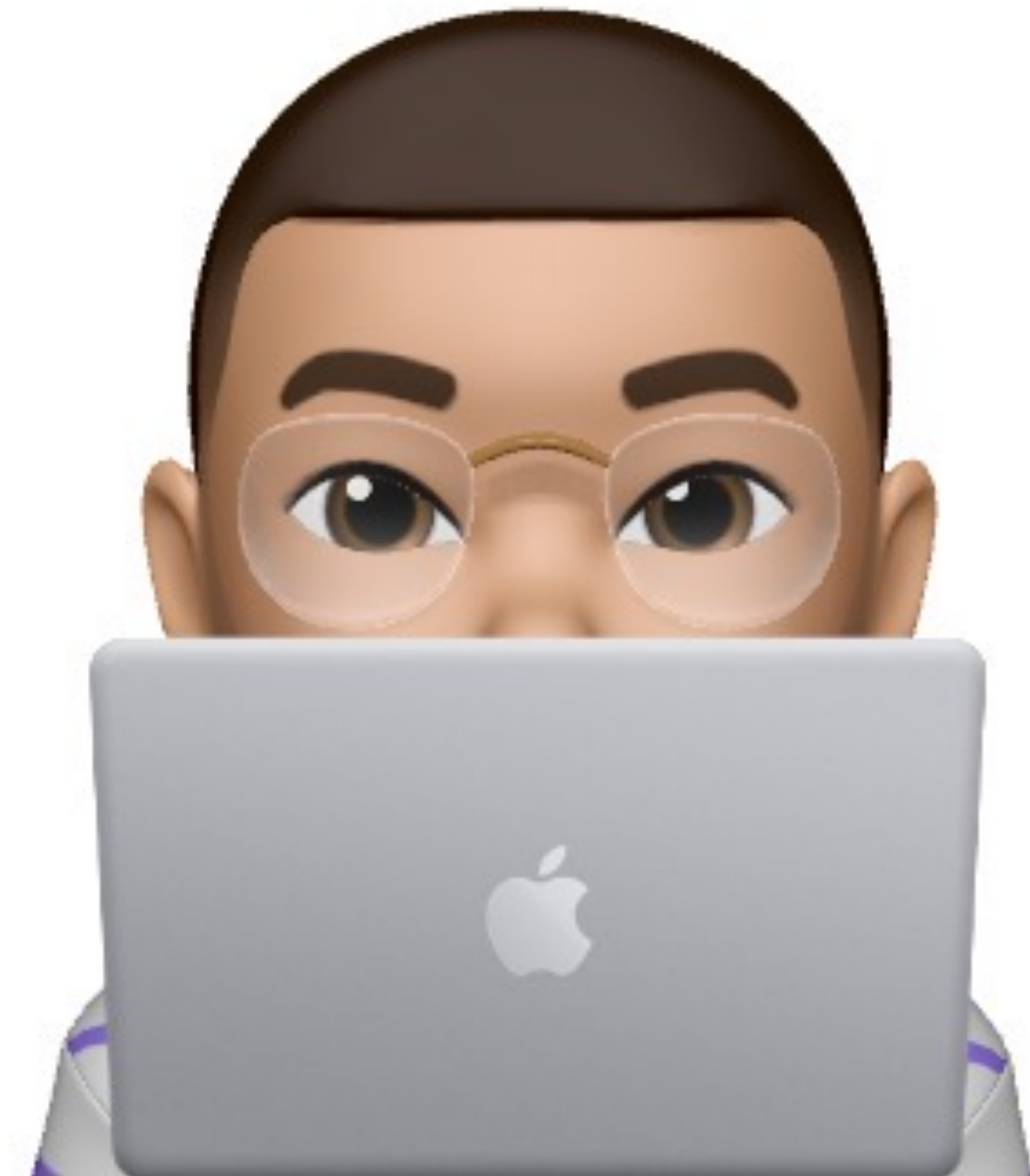
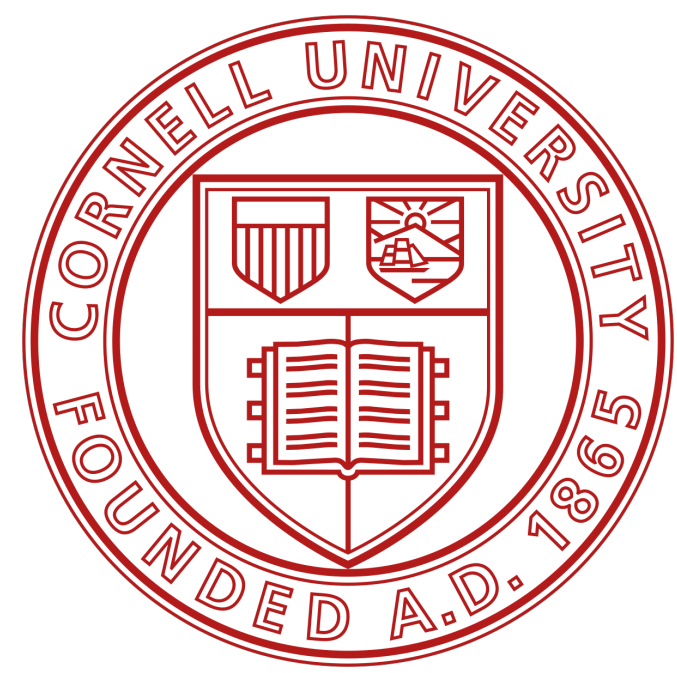
- W1: Introduction to R
- W2: R Objects and R Notation



7 weeks

Overview

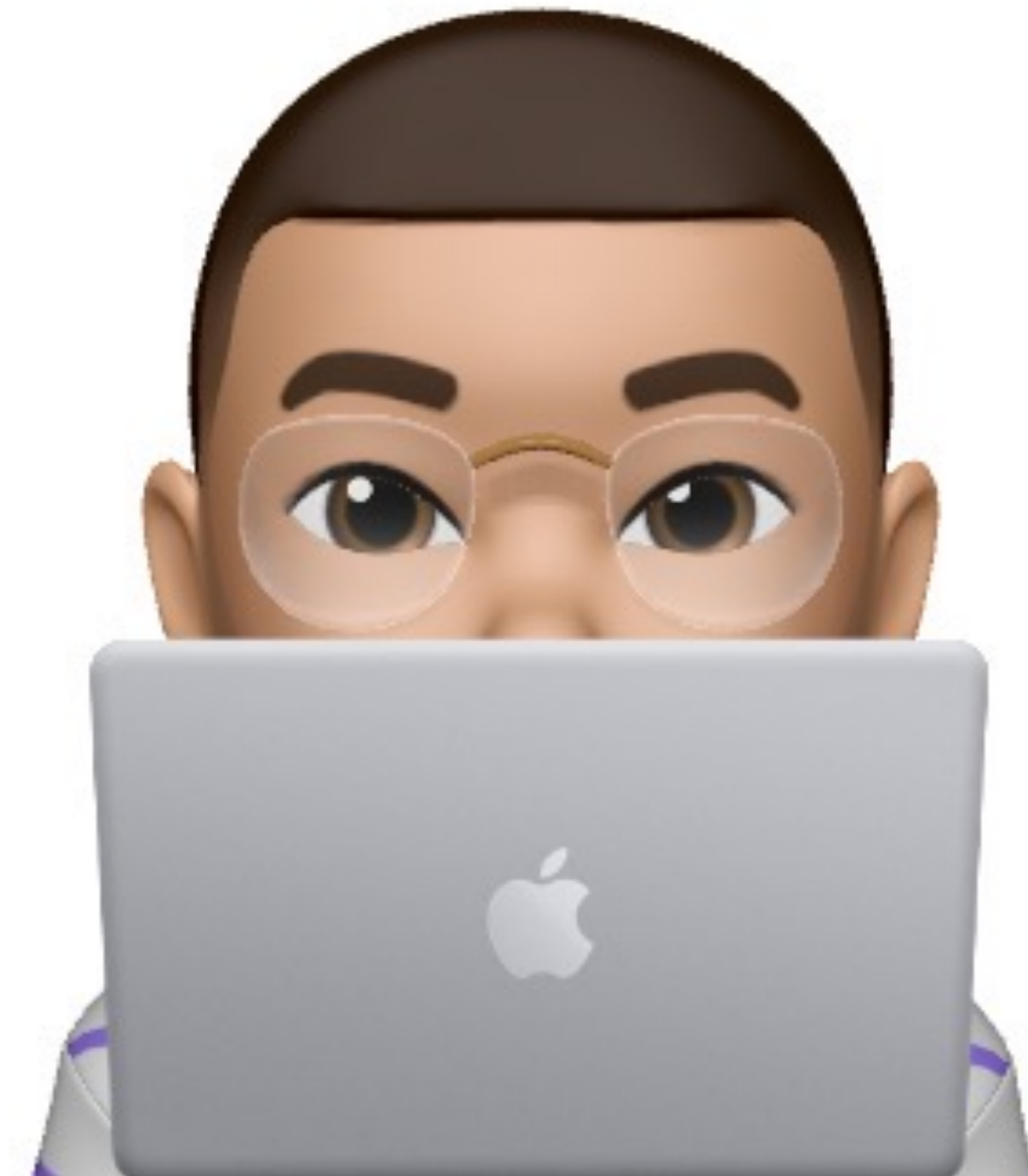
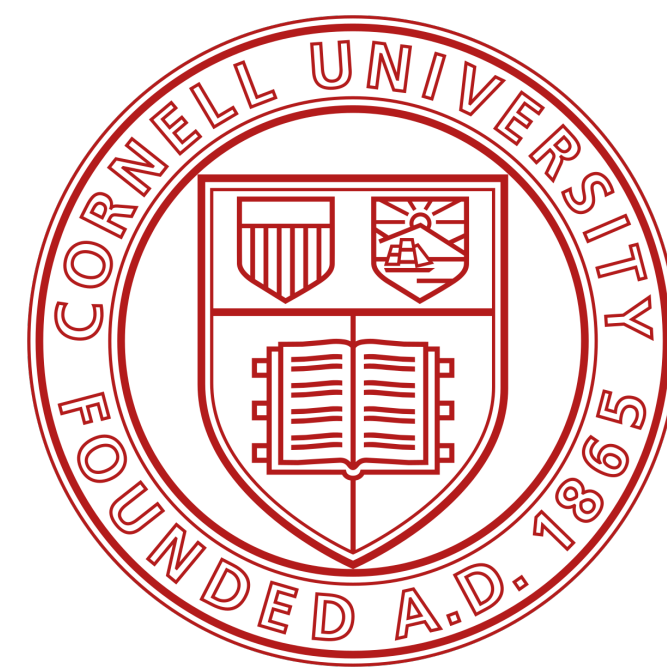
- W1: Introduction to R
- W2: R Objects and R Notation
- W3: Modifying values and Environments



7 weeks

Overview

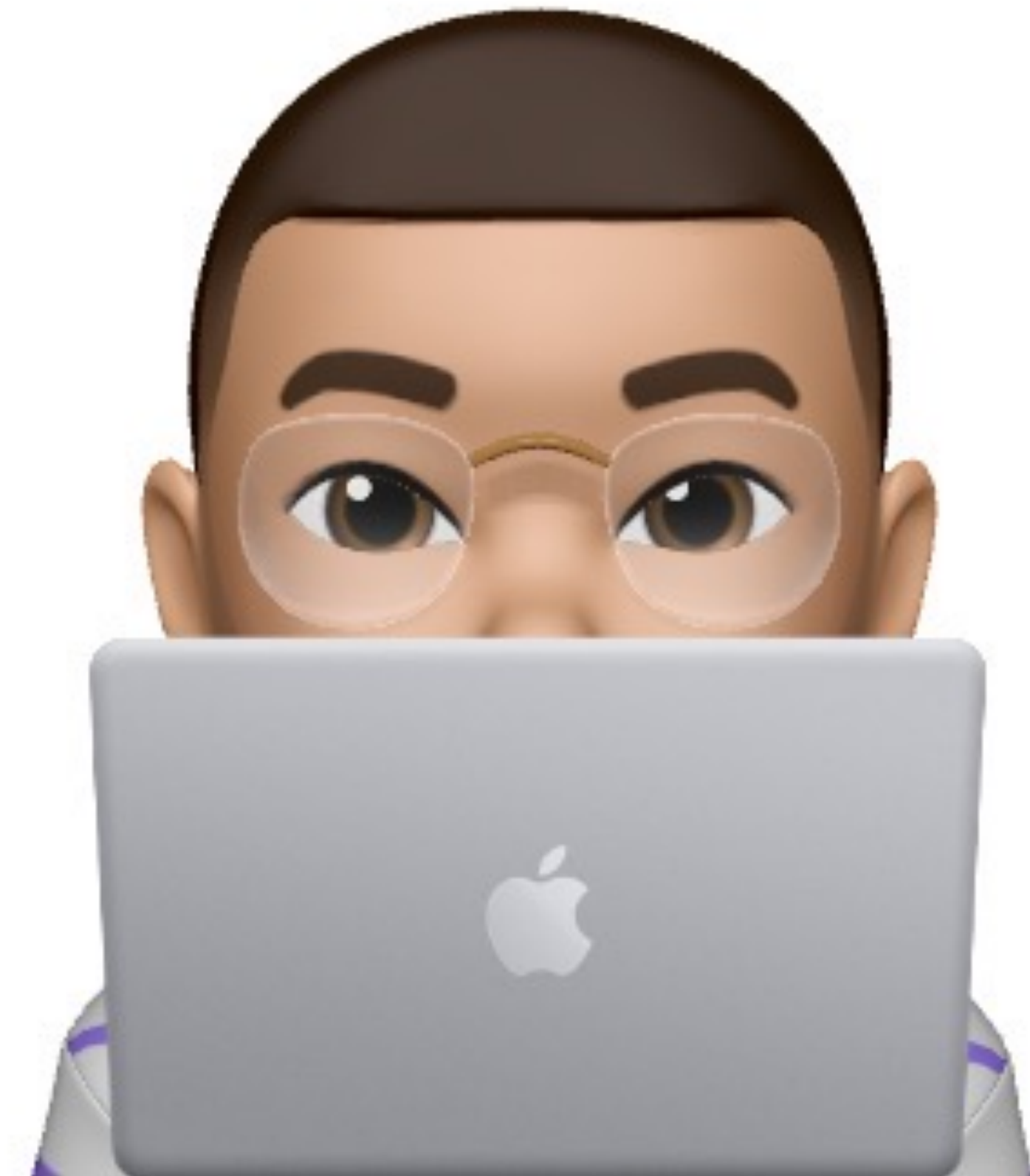
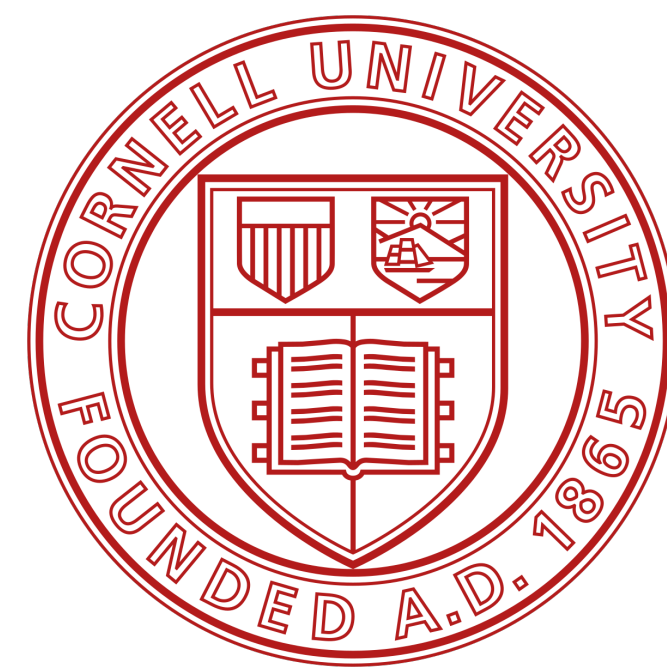
- W1: Introduction to R
- W2: R Objects and R Notation
- W3: Modifying values and Environments
- W4: Programs and S3



7 weeks

Overview

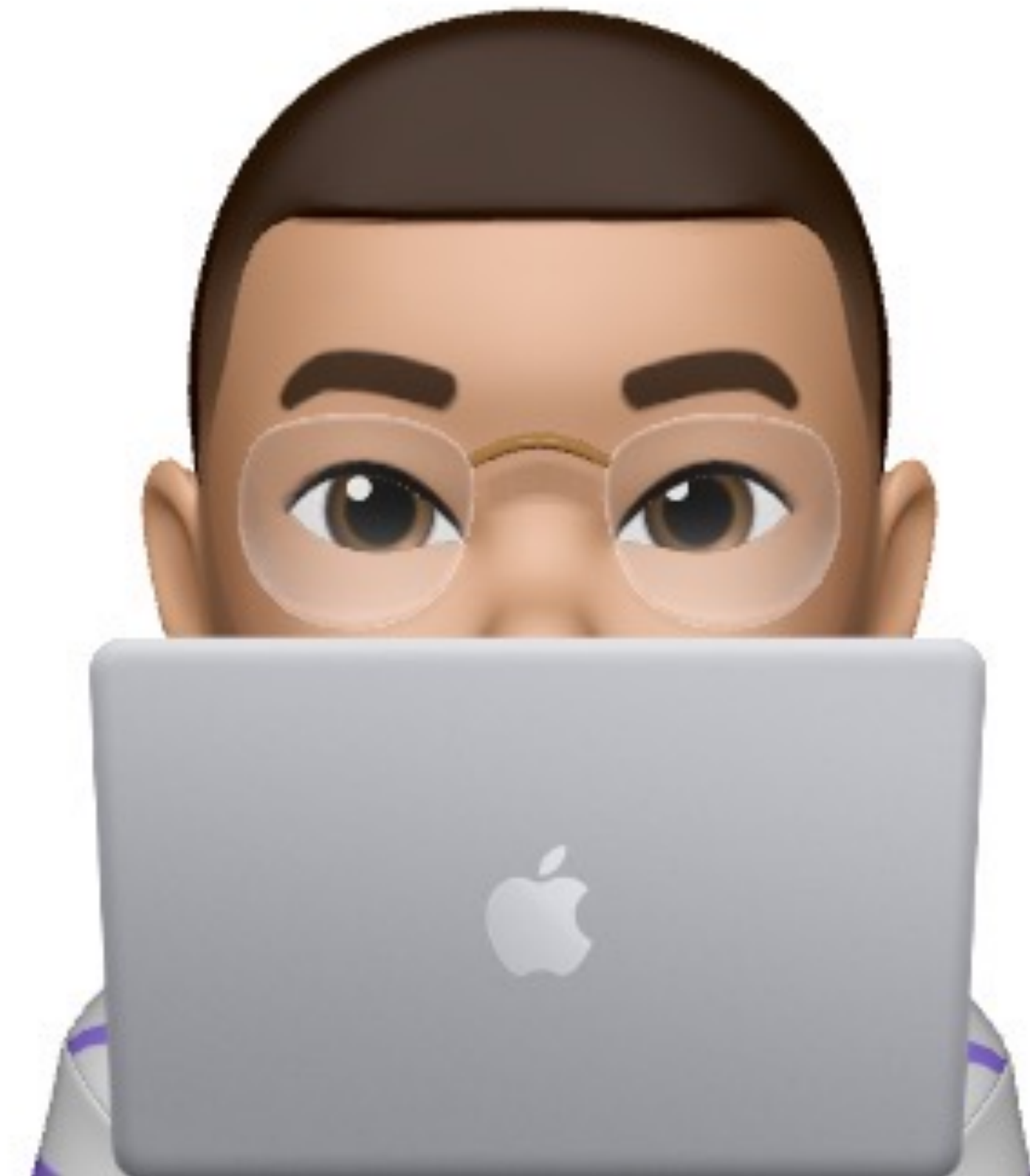
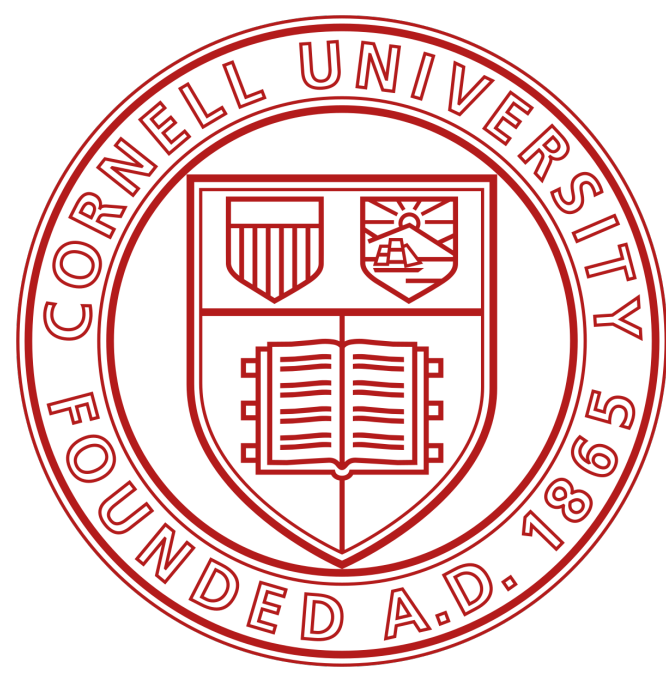
- W1: Introduction to R
- W2: R Objects and R Notation
- W3: Modifying values and Environments
- W4: Programs and S3
- W5: Programs and S3



7 weeks

Overview

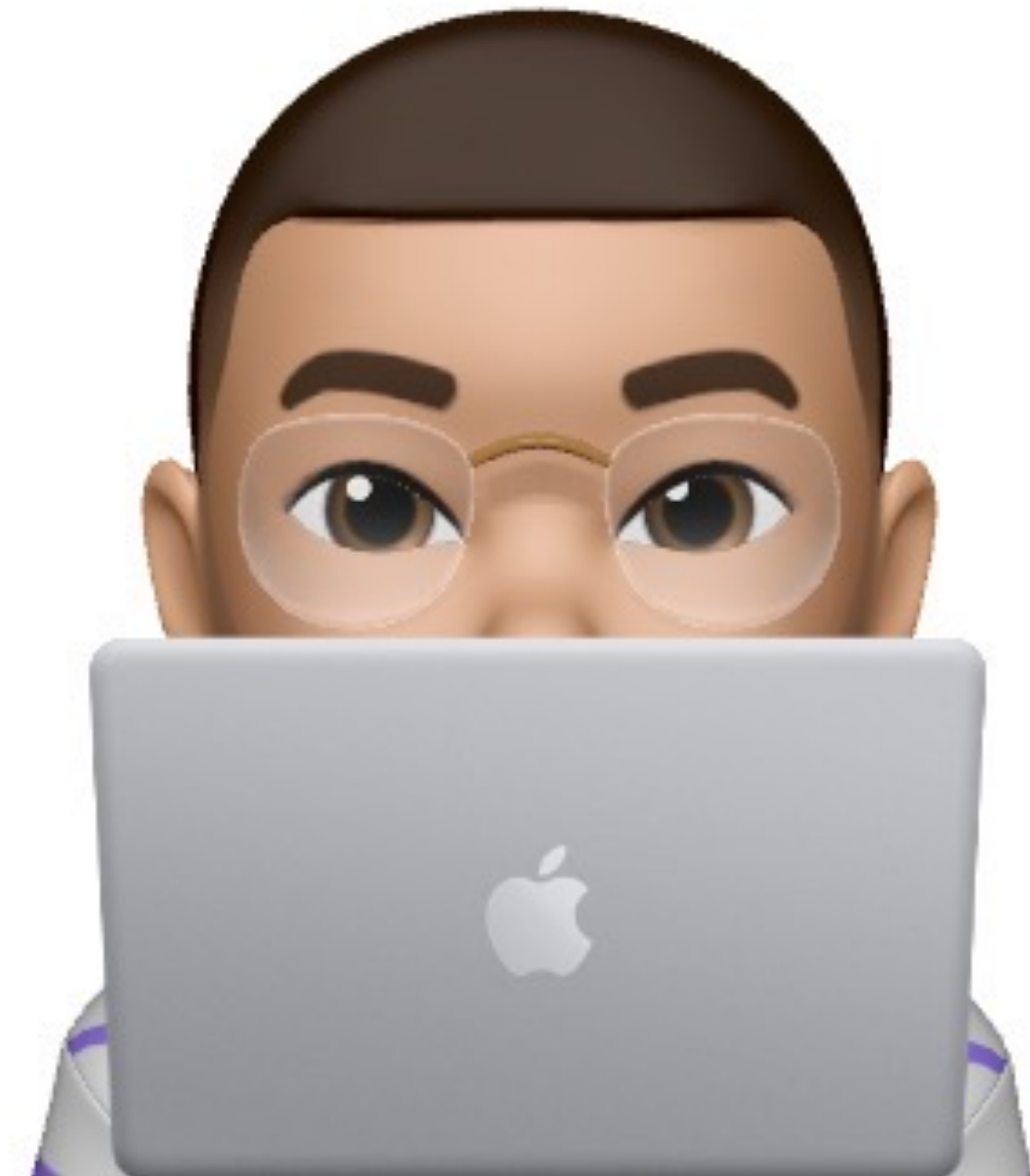
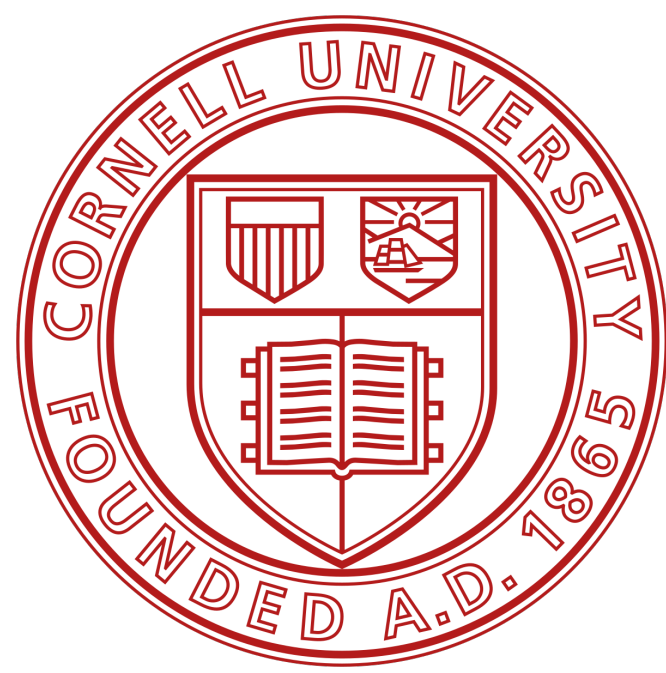
- W1: Introduction to R
- W2: R Objects and R Notation
- W3: Modifying values and Environments
- W4: Programs and S3
- W5: Programs and S3
- W6: Working with data

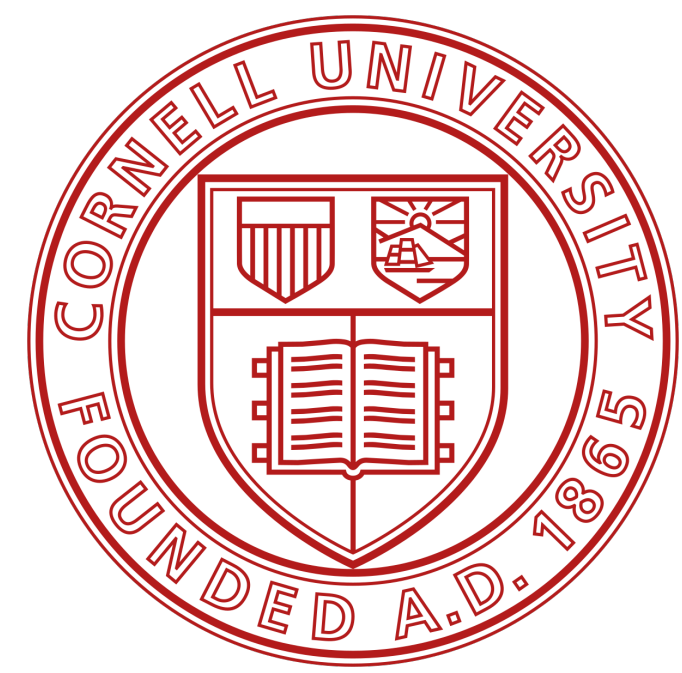


7 weeks

Overview

- W1: Introduction to R
- W2: R Objects and R Notation
- W3: Modifying values and Environments
- W4: Programs and S3
- W5: Programs and S3
- W6: Working with data
- W7: Final Project

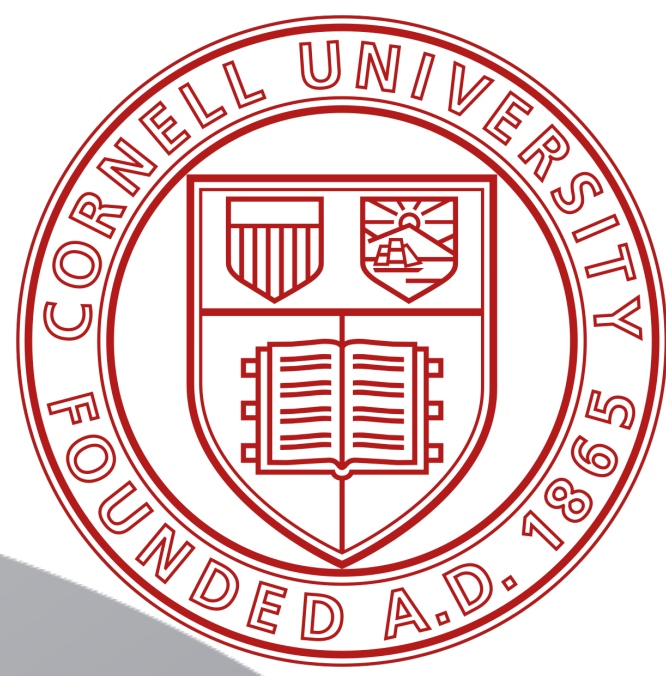


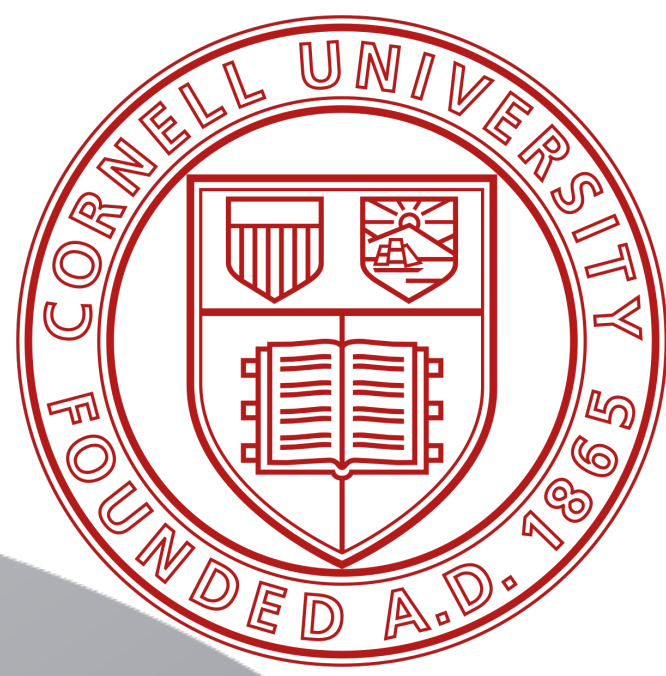


Introduction to R and RStudio

Install R

What is it ?





Install R

What is it ?

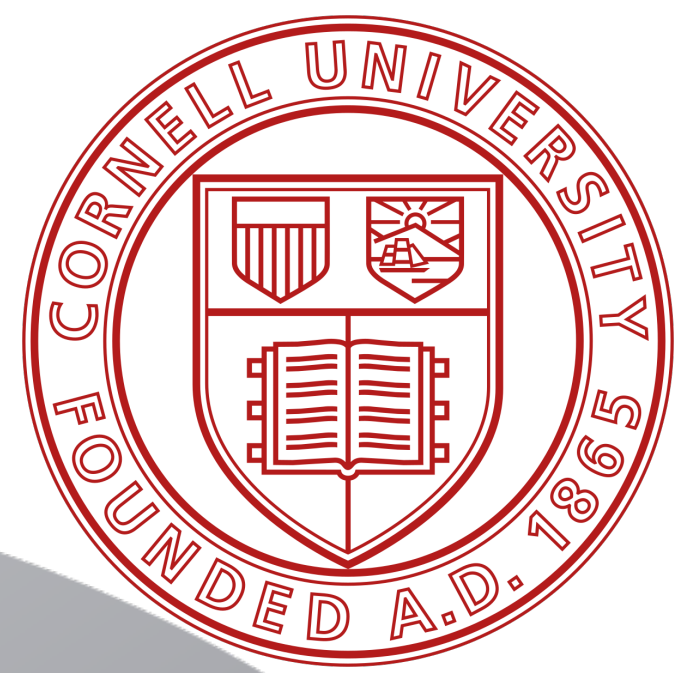
- R is a computer language

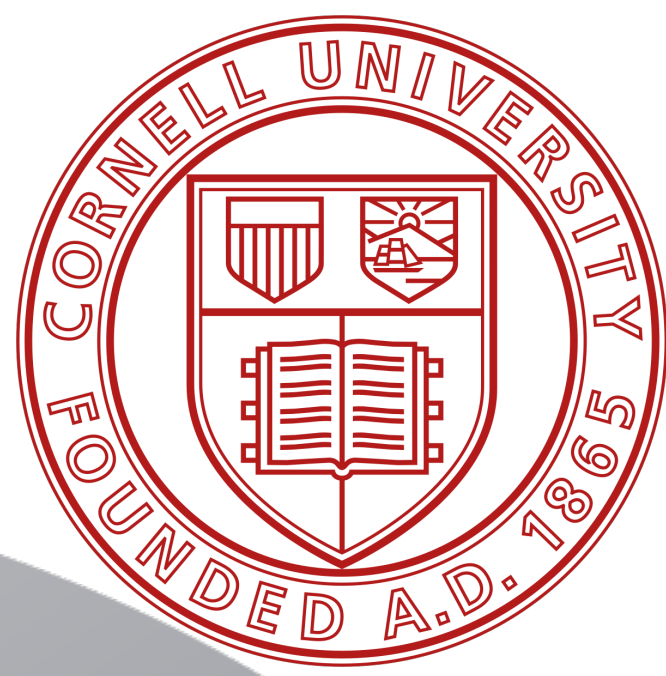


Install R

What is it ?

- R is a computer language
- It isn't a program you can open and start using



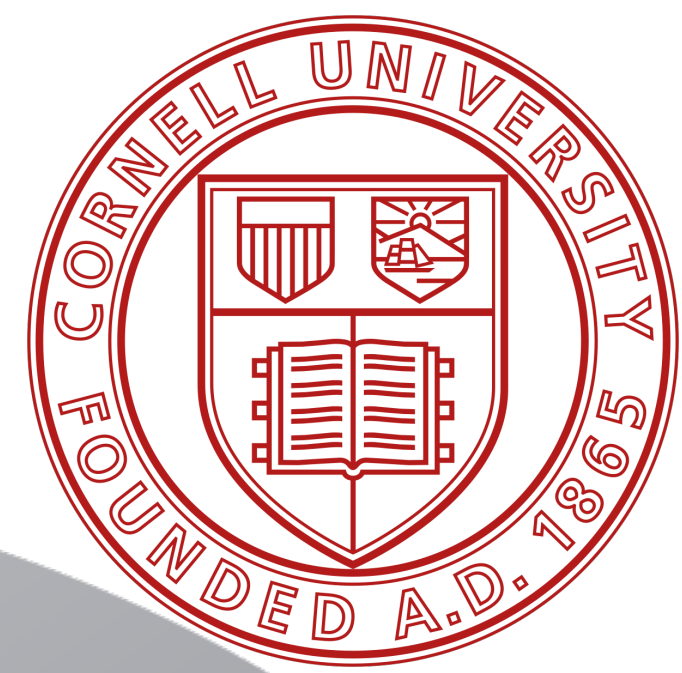


Install R

What is it ?

- R is a computer language
- It isn't a program you can open and start using
- R is maintained by an international team of developers





Install R

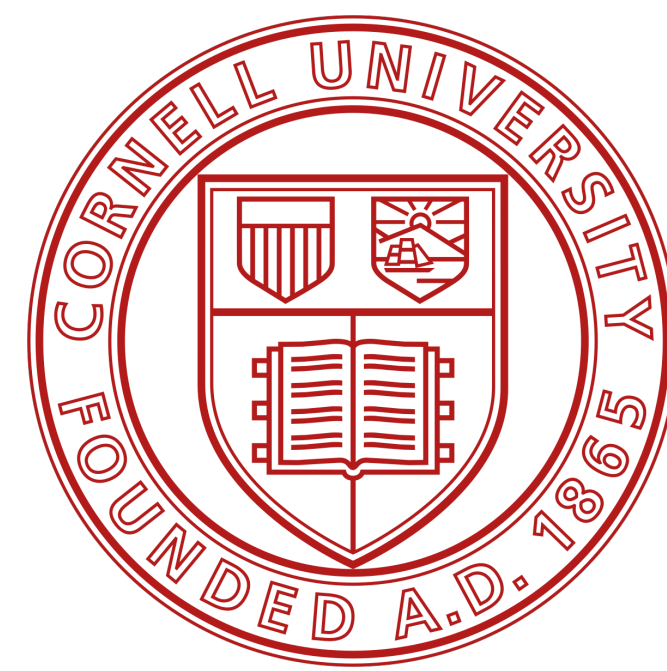
What is it ?

- R is a computer language
- It isn't a program you can open and start using
- R is maintained by an international team of developers
- Info: [The Comprehensive R Archive Network](#)



Install R

How to install it ?



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:

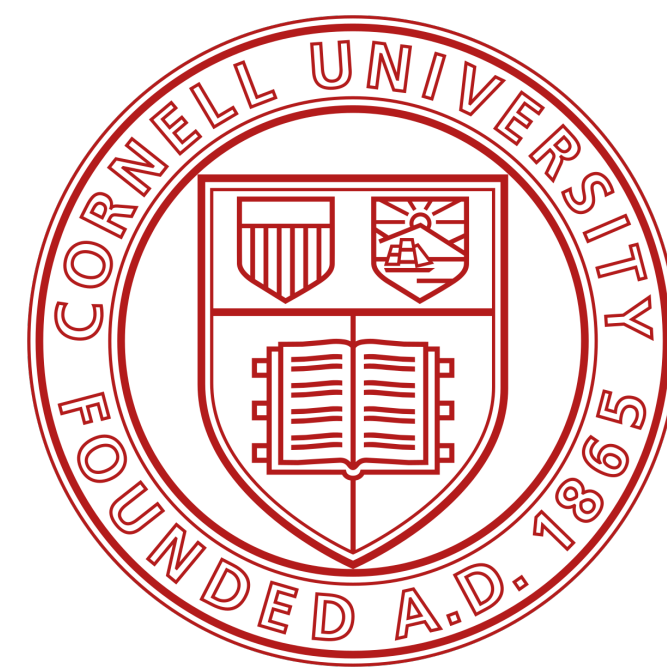
- [Download R for Linux](#) ([Debian](#), [Fedora/Redhat](#), [Ubuntu](#))
- [Download R for macOS](#)
- [Download R for Windows](#)

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

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2024-06-14, Race for Your Life) [R-4.4.1.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)



Install R

How to install it ?

- Go on: [The Comprehensive R Archive Network](#)

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:

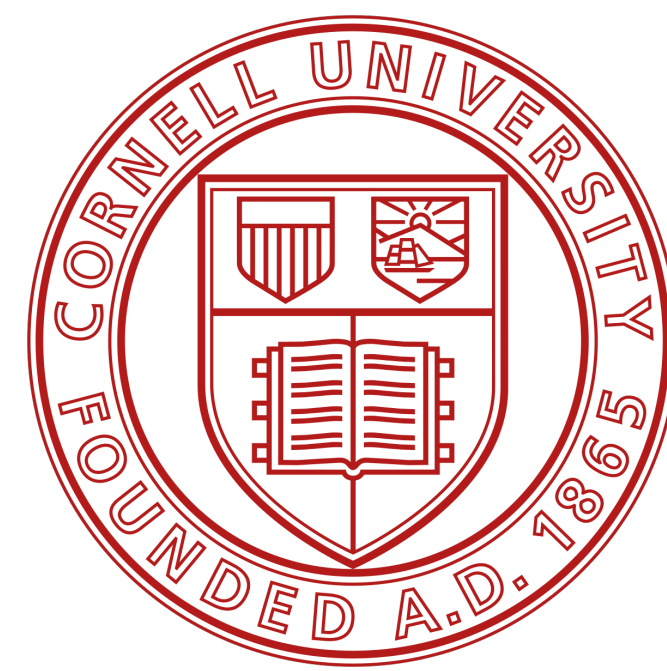
- [Download R for Linux](#) ([Debian](#), [Fedora/Redhat](#), [Ubuntu](#))
- [Download R for macOS](#)
- [Download R for Windows](#)

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

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2024-06-14, Race for Your Life) [R-4.4.1.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)



Install R

How to install it ?

- Go on: [The Comprehensive R Archive Network](#)
- Follow the link that describes your operating system: Windows, Mac, or Linux.

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](#) ([Debian](#), [Fedora/Redhat](#), [Ubuntu](#))
- [Download R for macOS](#)
- [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.

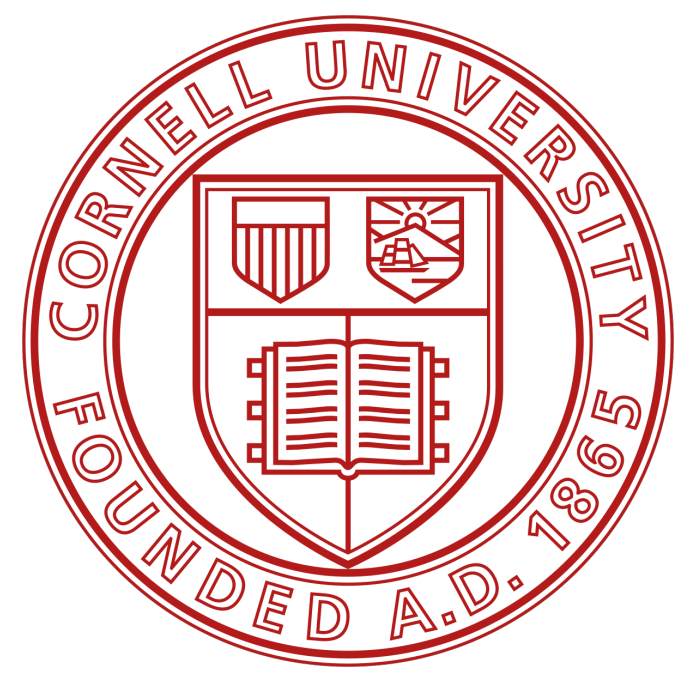
Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2024-06-14, Race for Your Life) [R-4.4.1.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Install RStudio

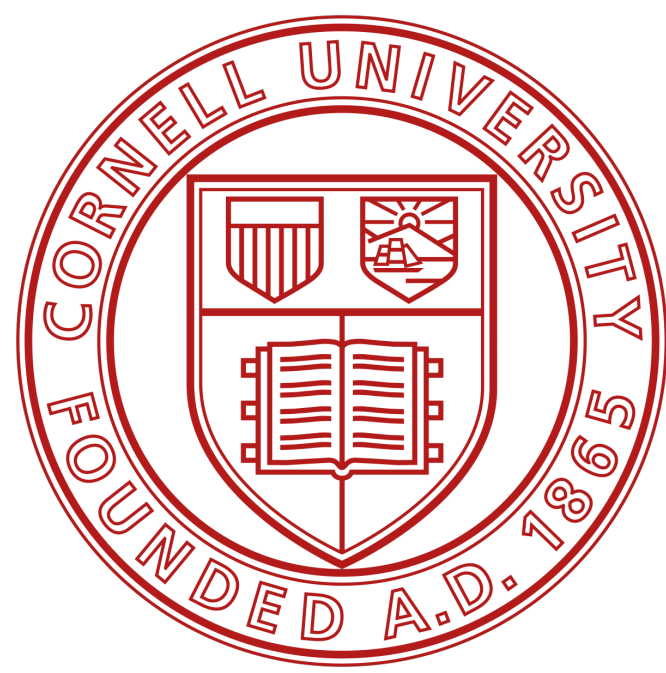
What is it ?



Install RStudio

What is it ?

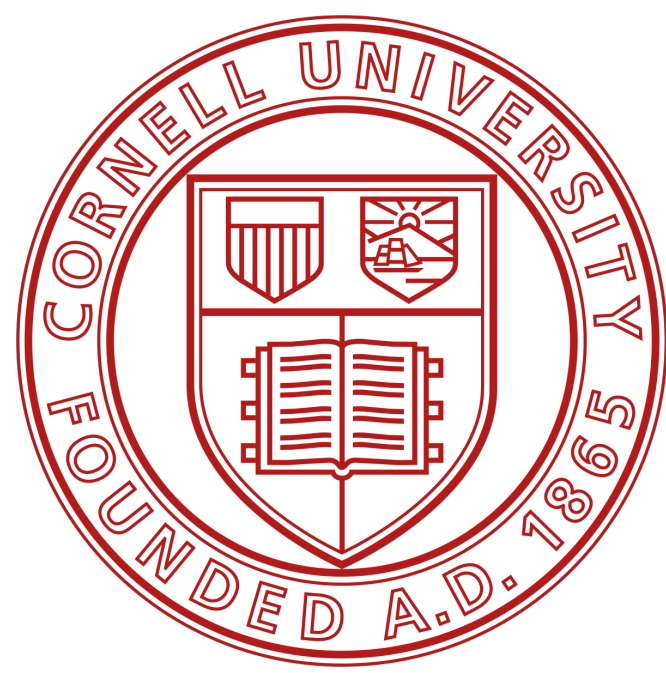
- RStudio is an application like Microsoft Word.



Install RStudio

What is it ?

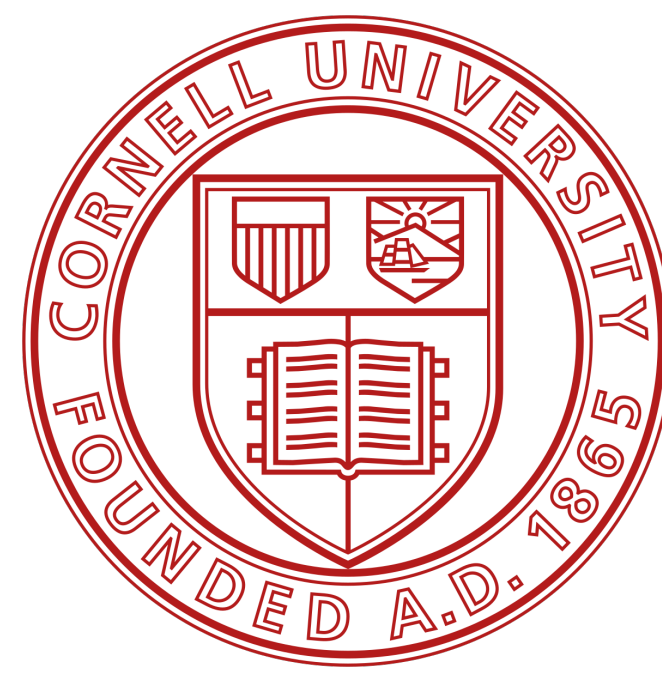
- RStudio is an application like Microsoft Word.
- Instead of helping you write in English, RStudio helps you write in R.



Install RStudio

What is it ?

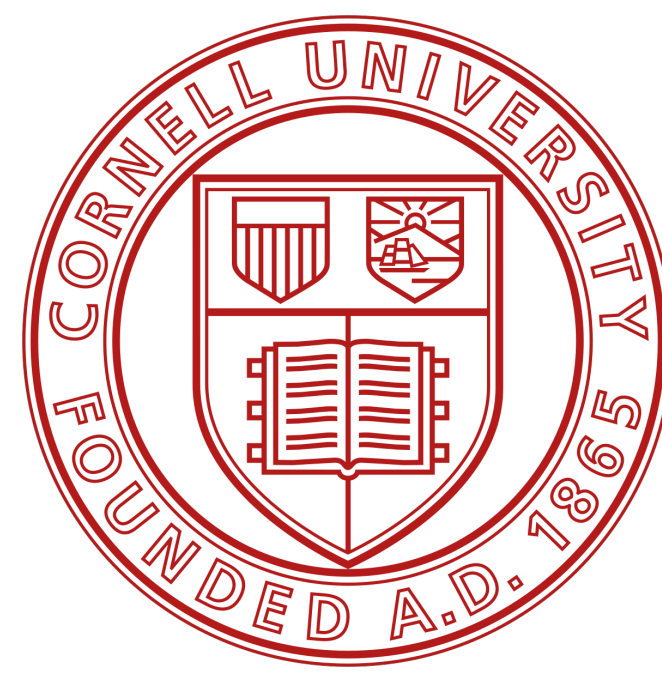
- RStudio is an application like Microsoft Word.
- Instead of helping you write in English, RStudio helps you write in R.
- The RStudio interface looks the same for Windows, Mac OS, and Linux.



Install RStudio

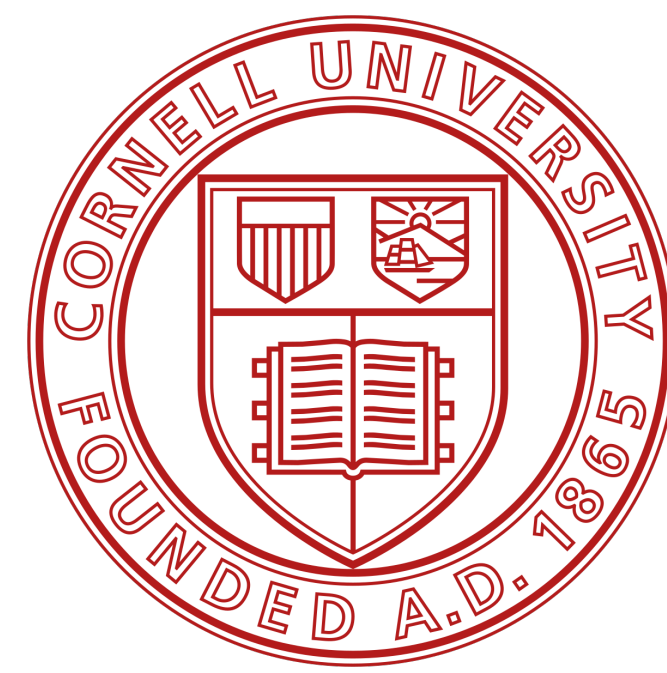
What is it ?

- RStudio is an application like Microsoft Word.
- Instead of helping you write in English, RStudio helps you write in R.
- The RStudio interface looks the same for Windows, Mac OS, and Linux.
- It is an IDE



Install RStudio

How to install it ?



The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for loading packages, creating a 'daily' dataset, and plotting a boxplot of flights per weekday.
- Environment:** Shows the 'daily' dataset with 365 observations and 3 variables.
- Console:** Displays the output of the R code, including a tibble of flight data and the execution of the plot command.
- Plots:** Shows a boxplot titled "Number of 2013 New York Flights Each Weekday" with the y-axis labeled "Flights" and the x-axis labeled "Weekday".

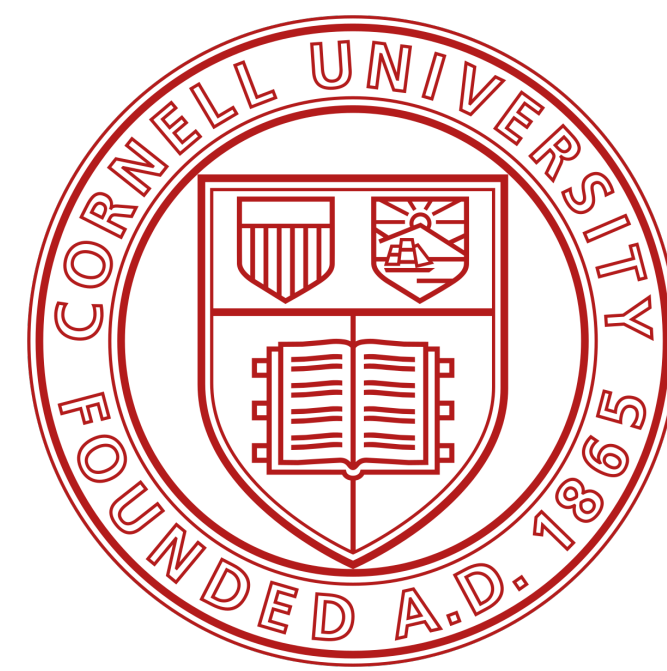
```
1 library(nycflights13) ## package containing flights dataset
2 library(lubridate)
3 library(dplyr)
4 library(ggplot2)
5
6 head(flights, n = 3)
7 daily <- flights %>%
8   mutate(date = make_date(year, month, day)) %>%
9   count(date) %>%
10  mutate(wday = wday(date, label = TRUE))
11 head(daily, n = 3)
12 ggplot(daily, aes(wday, n)) +
13   geom_boxplot(outlier.colour = "hotpink") +
14   labs(x = "Weekday", y = "Flights",
15        subtitle = "Number of 2013 New York Flights Each Weekday")
16
```

```
# A tibble: 3 x 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
<int> <int> <int> <int> <int> <dbl> <int> <int> <dbl> <chr>
1  2013     1     1     517         515           2     830         819           11 UA
2  2013     1     1     533         529           4     850         830           20 UA
3  2013     1     1     542         540           2     923         850           33 AA
# ... with 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
# distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
> daily <- flights %>%
+   mutate(date = make_date(year, month, day)) %>%
+   count(date) %>%
+   mutate(wday = wday(date, label = TRUE))
> head(daily, n = 3)
# A tibble: 3 x 3
  date           n wday
<date> <int> <ord>
1 2013-01-01  842 Tue
2 2013-01-02  943 Wed
3 2013-01-03  914 Thu
> ggplot(daily, aes(wday, n)) +
+   geom_boxplot(outlier.colour = "hotpink") +
+   labs(x = "Weekday", y = "Flights",
+        subtitle = "Number of 2013 New York Flights Each Weekday")
>
```

Weekday	Min	Q1	Median	Q3	Max
Sun	715	895	905	915	995
Mon	915	955	975	995	1000
Tue	765	945	965	985	1000
Wed	715	945	965	985	1000
Thu	735	945	965	985	1000
Fri	765	945	965	985	1000
Sat	685	695	745	765	865

Install RStudio

How to install it ?



- [Download RStudio for free.](#)

The screenshot displays the RStudio interface with the following components:

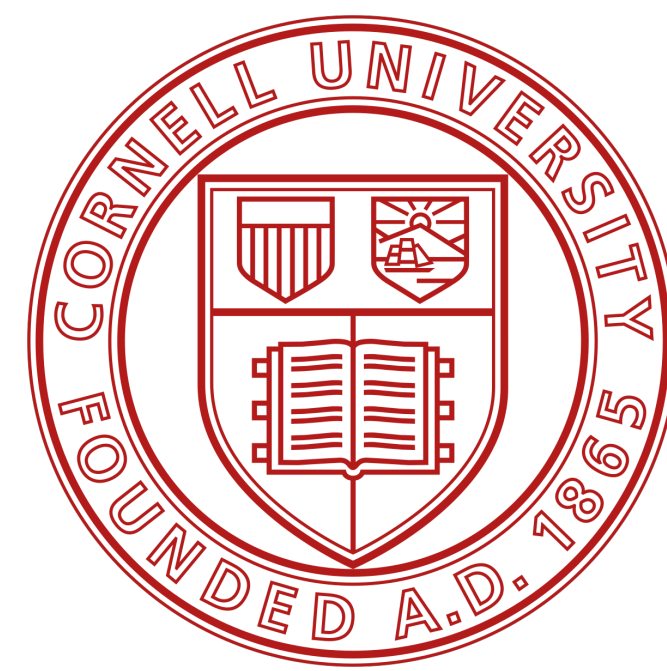
- Source Editor:** Contains R code for loading packages, creating a 'daily' dataset, and plotting the number of flights per weekday.
- Environment:** Shows the 'daily' dataset with 365 observations and 3 variables (date, n, wday).
- Console:** Displays the output of the R code, including a tibble of flight data and the execution of the plot command.
- Plots:** Shows a boxplot titled "Number of 2013 New York Flights Each Weekday" with the y-axis labeled "Flights" and the x-axis labeled "Weekday".

```
1 library(nycflights13) ## package containing flights dataset
2 library(lubridate)
3 library(dplyr)
4 library(ggplot2)
5
6 head(flights, n = 3)
7 daily <- flights %>%
8   mutate(date = make_date(year, month, day)) %>%
9   count(date) %>%
10  mutate(wday = wday(date, label = TRUE))
11 head(daily, n = 3)
12 ggplot(daily, aes(wday, n)) +
13   geom_boxplot(outlier.colour = "hotpink") +
14   labs(x = "Weekday", y = "Flights",
15        subtitle = "Number of 2013 New York Flights Each Weekday")
16
```

```
# A tibble: 3 x 19
  year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
<int> <int> <int> <int> <int> <dbl> <int> <int> <dbl> <dbl> <chr>
1 2013 1 1 517 515 2 830 819 11 UA
2 2013 1 1 533 529 4 850 830 20 UA
3 2013 1 1 542 540 2 923 850 33 AA
# ... with 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
# distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
> daily <- flights %>%
+   mutate(date = make_date(year, month, day)) %>%
+   count(date) %>%
+   mutate(wday = wday(date, label = TRUE))
> head(daily, n = 3)
# A tibble: 3 x 3
  date n wday
<date> <int> <ord>
1 2013-01-01 842 Tue
2 2013-01-02 943 Wed
3 2013-01-03 914 Thu
> ggplot(daily, aes(wday, n)) +
+   geom_boxplot(outlier.colour = "hotpink") +
+   labs(x = "Weekday", y = "Flights",
+        subtitle = "Number of 2013 New York Flights Each Weekday")
>
```

Install RStudio

How to install it ?



- [Download RStudio for free.](#)
- Use Studio on [Posit Cloud.](#)

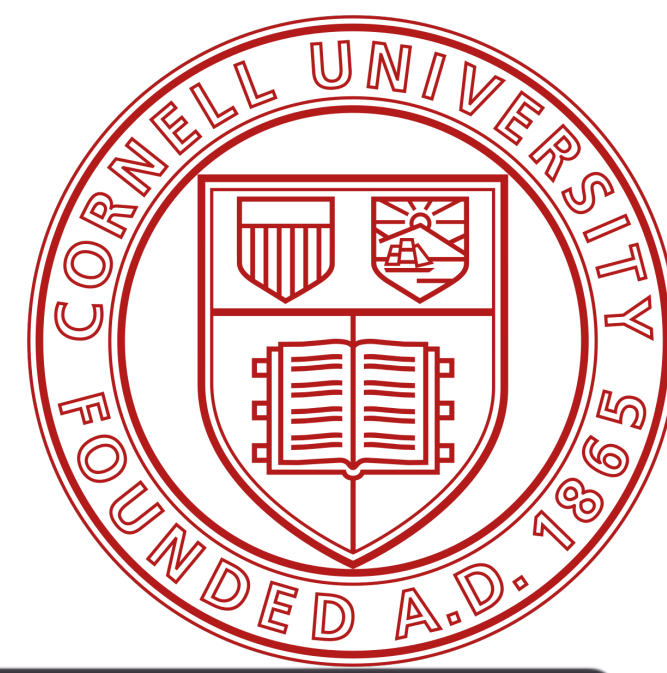
The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for loading the 'nycflights13' dataset, creating a 'daily' tibble, and plotting the number of flights per weekday.
- Environment:** Shows the 'daily' object with 365 observations and 3 variables.
- Console:** Displays the execution of the code, including the output of the 'head()' function and the execution of the 'ggplot()' function.
- Plots:** Shows a boxplot titled "Number of 2013 New York Flights Each Weekday" with the y-axis labeled "Flights" and the x-axis labeled "Weekday".

```
1 library(nycflights13) ## package containing flights dataset
2 library(lubridate)
3 library(dplyr)
4 library(ggplot2)
5
6 head(flights, n = 3)
7 daily <- flights %>%
8   mutate(date = make_date(year, month, day)) %>%
9   count(date) %>%
10  mutate(wday = wday(date, label = TRUE))
11 head(daily, n = 3)
12 ggplot(daily, aes(wday, n)) +
13   geom_boxplot(outlier.colour = "hotpink") +
14   labs(x = "Weekday", y = "Flights",
15        subtitle = "Number of 2013 New York Flights Each Weekday")
16
```

```
# A tibble: 3 x 19
  year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
<int> <int> <int> <int> <int> <dbl> <int> <int> <dbl> <chr>
1 2013 1 1 517 515 2 830 819 11 UA
2 2013 1 1 533 529 4 850 830 20 UA
3 2013 1 1 542 540 2 923 850 33 AA
# ... with 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
# distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
> daily <- flights %>%
+   mutate(date = make_date(year, month, day)) %>%
+   count(date) %>%
+   mutate(wday = wday(date, label = TRUE))
> head(daily, n = 3)
# A tibble: 3 x 3
  date n wday
<date> <int> <ord>
1 2013-01-01 842 Tue
2 2013-01-02 943 Wed
3 2013-01-03 914 Thu
> ggplot(daily, aes(wday, n)) +
+   geom_boxplot(outlier.colour = "hotpink") +
+   labs(x = "Weekday", y = "Flights",
+        subtitle = "Number of 2013 New York Flights Each Weekday")
>
```


Install RStudio Interface



The screenshot displays the RStudio interface with the following components:

- Console:** Shows the execution of R code:

```
> library(ggplot2)
> ggplot(mpg, aes(displ, hwy)) +
+   geom_point(aes(colour = class))
> |
```
- Environment:** Displays "Environment is empty".
- Plots:** Shows a scatter plot of highway mileage (hwy) versus engine displacement (displ) for different car classes. The plot is color-coded by class, with a legend on the right:

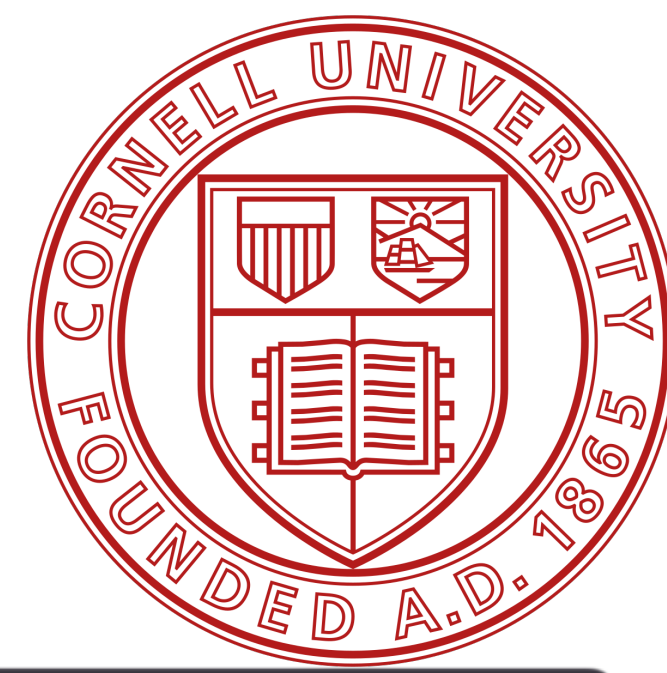
class	color
2seater	red
compact	orange
midsize	green
minivan	teal
pickup	blue
subcompact	purple
suv	pink

The plot shows a negative correlation between engine displacement and highway mileage, with different car classes clustered together based on their characteristics.

Console

Output

Install RStudio Interface



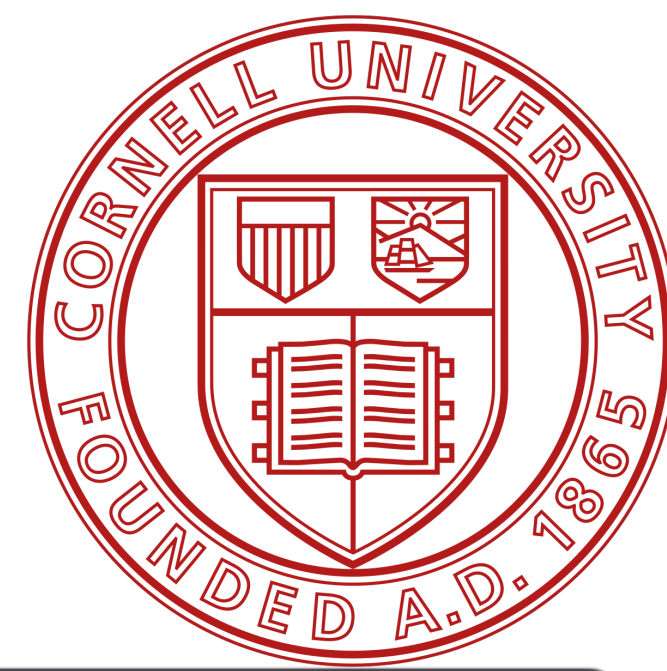
- Two key regions in the interface: the console pane and the output pane.

A screenshot of the RStudio interface. The window title is "rstudio-screenshots - whole-game-feedback - RStudio". The console pane on the left shows the following R code:

```
> library(ggplot2)
> ggplot(mpg, aes(displ, hwy)) +
+   geom_point(aes(colour = class))
> |
```

The output pane on the right shows a scatter plot of highway mileage (hwy) versus engine displacement (displ) for different car classes. The plot is titled "Environment is empty". The legend indicates the following classes: 2seater (red), compact (yellow), midsize (green), minivan (teal), pickup (cyan), subcompact (purple), and suv (pink). The x-axis (displ) ranges from 2 to 7, and the y-axis (hwy) ranges from 20 to 40. The plot is labeled "Console" at the bottom left and "Output" at the bottom right.

Install RStudio Interface



- Two key regions in the interface: the console pane and the output pane.
- You type the R code in the console pane and press enter to run it.

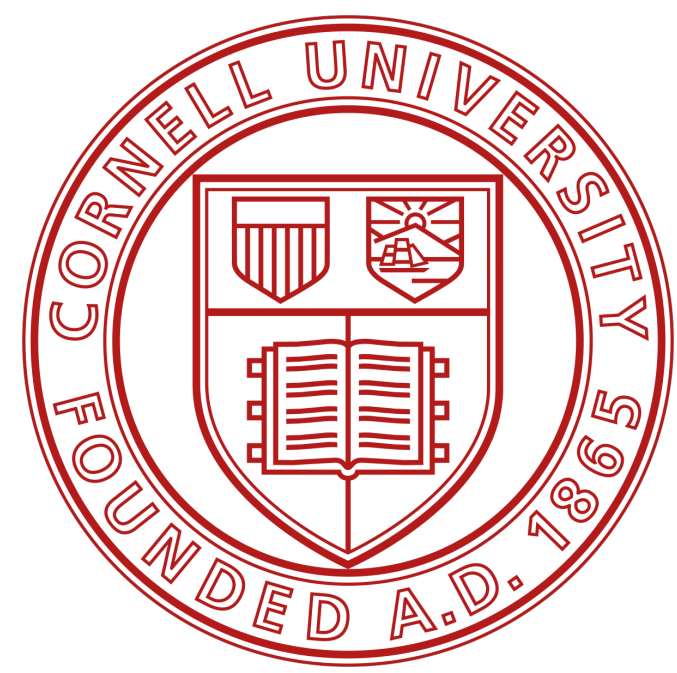
A screenshot of the RStudio interface. The window title is "rstudio-screenshots - whole-game-feedback - RStudio". The console pane on the left shows the following R code:

```
> library(ggplot2)
> ggplot(mpg, aes(displ, hwy)) +
+   geom_point(aes(colour = class))
> |
```

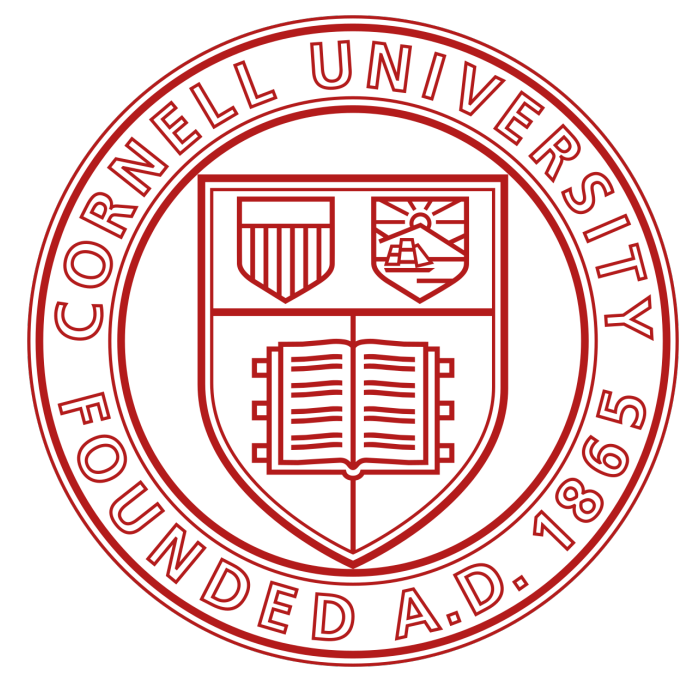
The output pane on the right shows a scatter plot of highway mileage (hwy) versus engine displacement (displ) for different car classes. The plot is titled "Environment is empty". The legend indicates the following classes: 2seater (red), compact (yellow), midsize (green), minivan (teal), pickup (cyan), subcompact (purple), and suv (pink). The x-axis (displ) ranges from 2 to 7, and the y-axis (hwy) ranges from 20 to 40. The plot shows a general negative correlation between displacement and highway mileage, with different car classes clustered together. The console pane is labeled "Console" and the output pane is labeled "Output".

R packages

What is it ?



```
install.packages("<package name>")
```

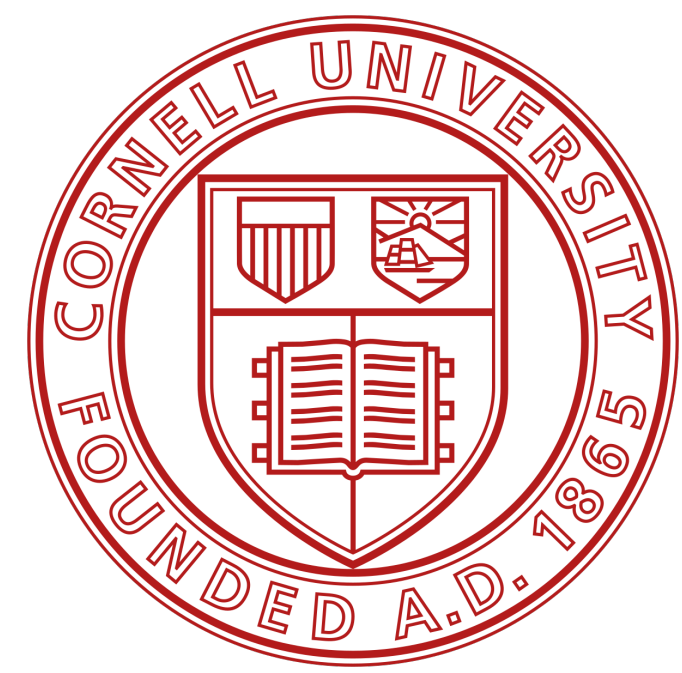


R packages

What is it ?

- An R package is a collection of functions, data, and documentation that extends the capabilities of base R.

```
install.packages("<package name>")
```

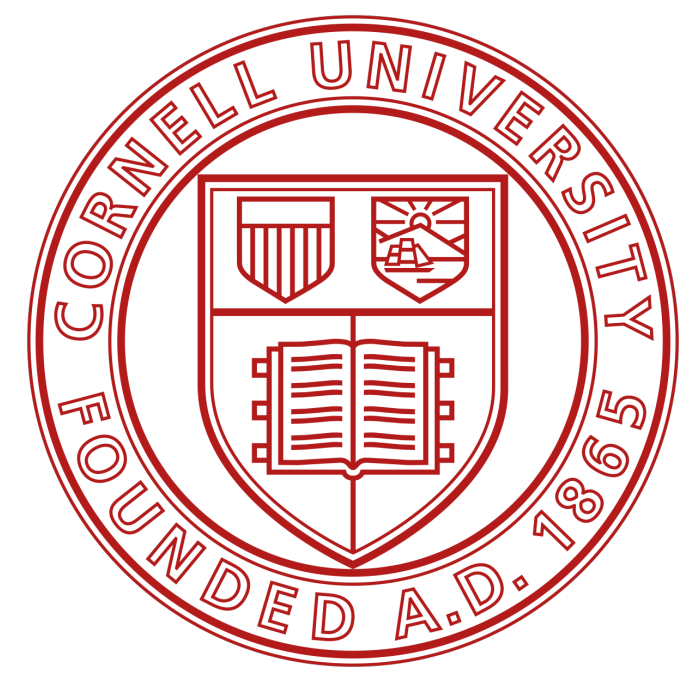


R packages

What is it ?

- An R package is a collection of functions, data, and documentation that extends the capabilities of base R.
- Using packages is key to the successful use of R.

```
install.packages("<package name>")
```



R packages

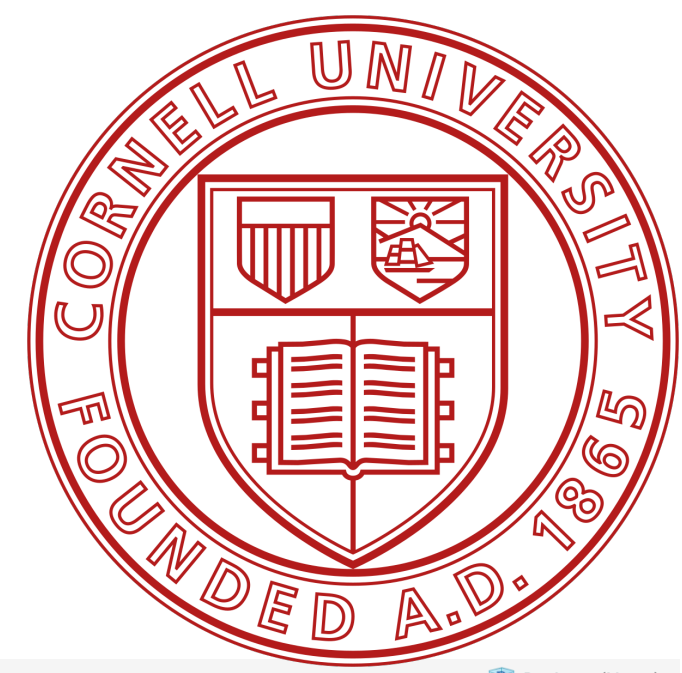
What is it ?

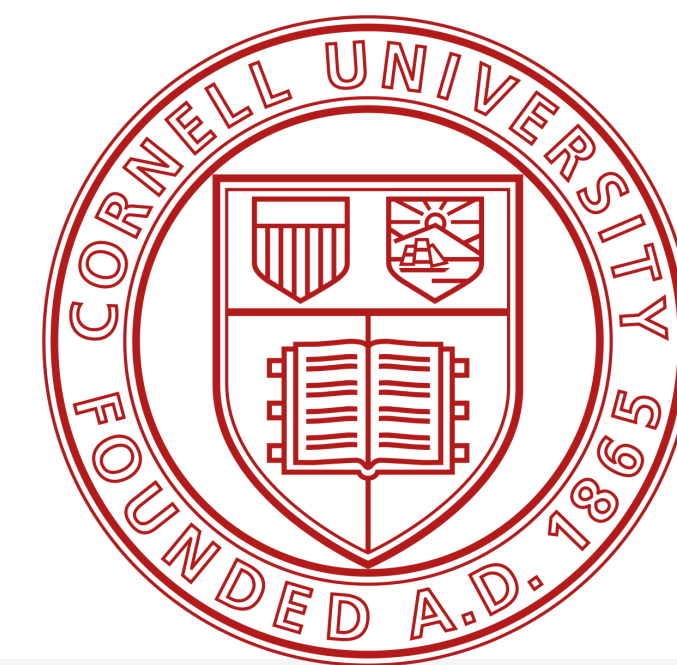
- An R package is a collection of functions, data, and documentation that extends the capabilities of base R.
- Using packages is key to the successful use of R.
- You can install the complete tidyverse package with a single line of code.

```
install.packages("<package name>")
```

R packages

Installing packages

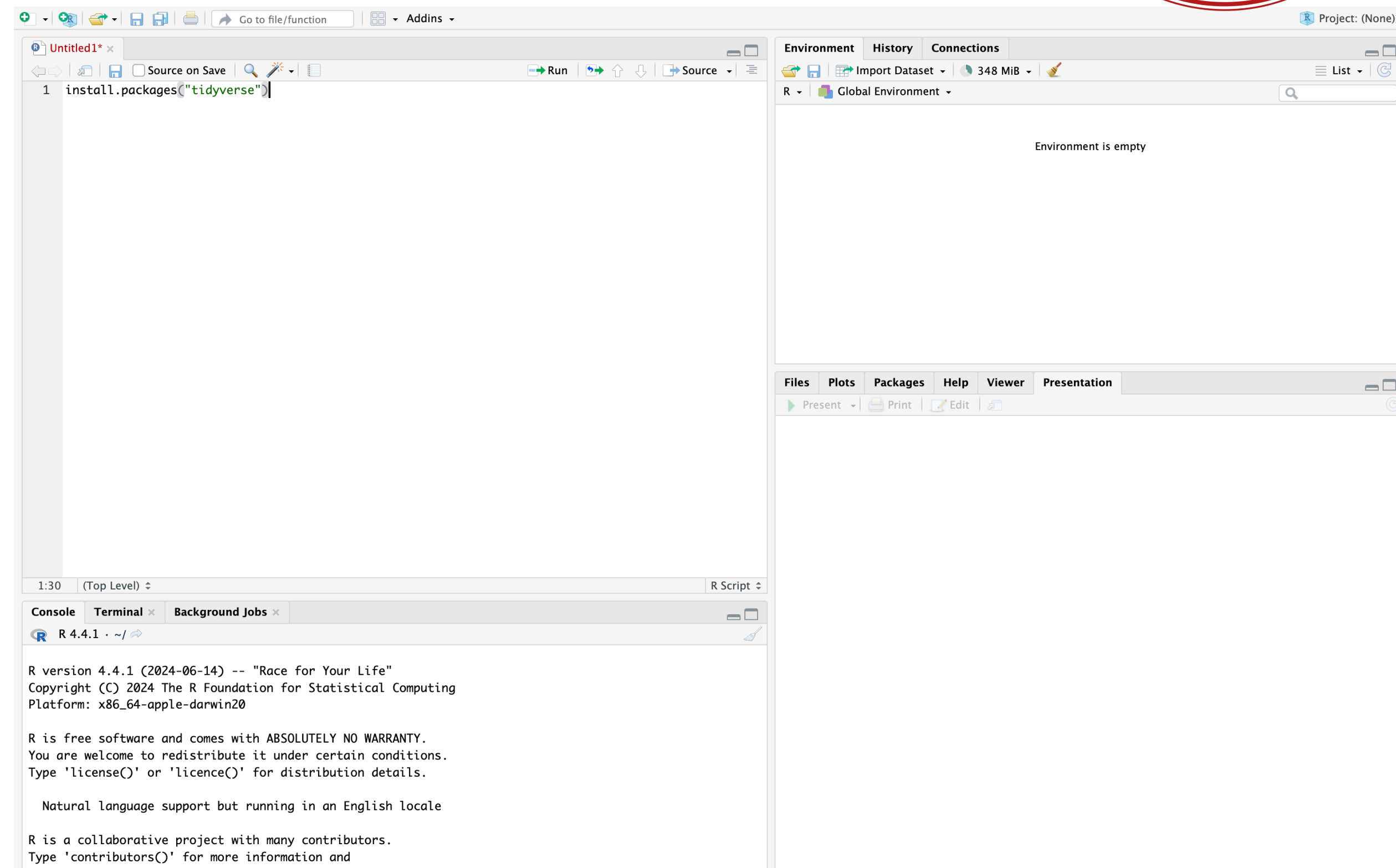
A screenshot of the RStudio interface. The main editor window shows a single line of code: `install.packages("tidyverse")`. The Environment pane on the right is empty, displaying "Environment is empty". The Console pane at the bottom shows the R version information: "R version 4.4.1 (2024-06-14) -- 'Race for Your Life'", copyright information, and a disclaimer: "R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details." The status bar at the bottom indicates "1:30 (Top Level) R Script".

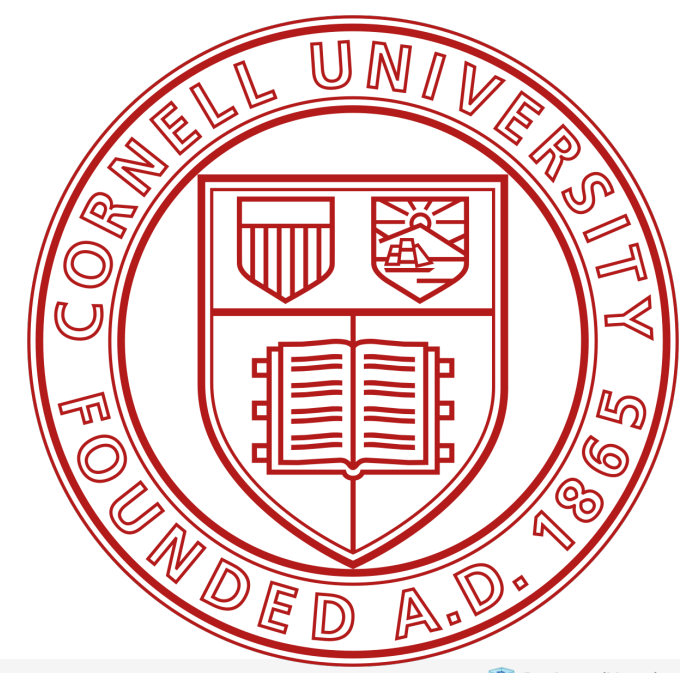


R packages

Installing packages

- Open a new R script.

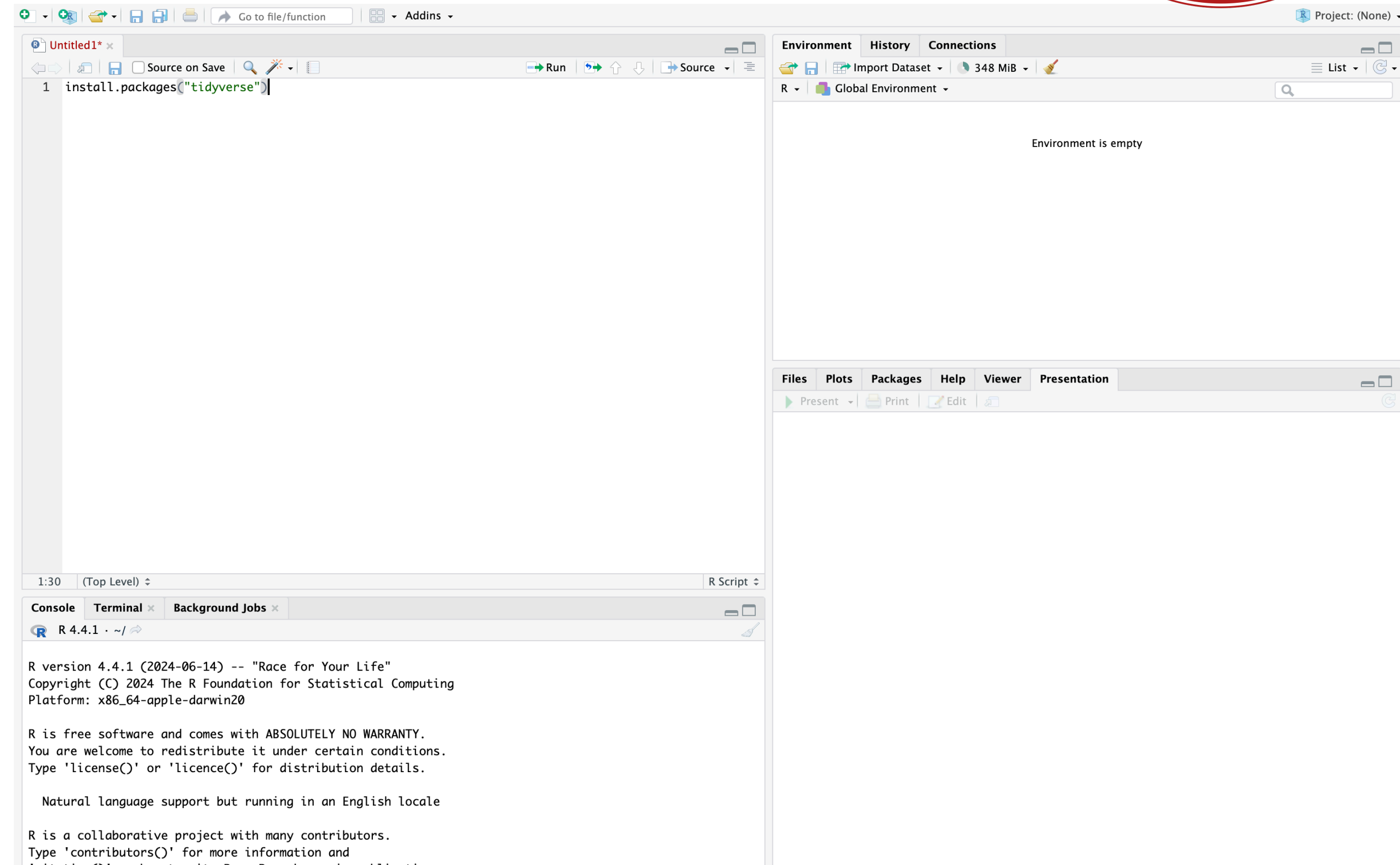


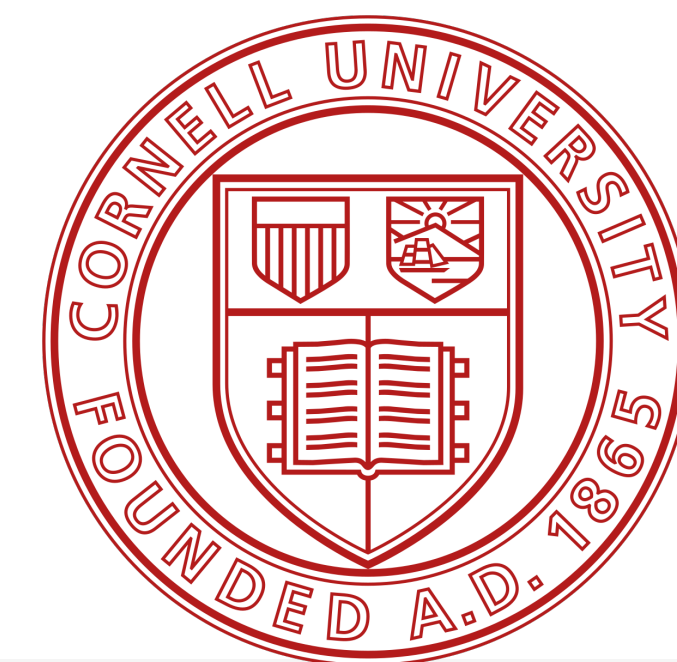


R packages

Installing packages

- Open a new R script.
- Type the `install.packages("tidyverse")` line of code.





R packages

Installing packages

- Open a new R script.
- Type the `install.packages("tidyverse")` line of code.
- Run it.

A screenshot of the RStudio interface. The main editor window shows a single line of code: `install.packages("tidyverse")`. The Environment pane on the right is empty, displaying "Environment is empty". The Console pane at the bottom shows the R version information: "R version 4.4.1 (2024-06-14) -- 'Race for Your Life'", copyright information, and the license details. The status bar at the bottom indicates "R 4.4.1 · ~/".

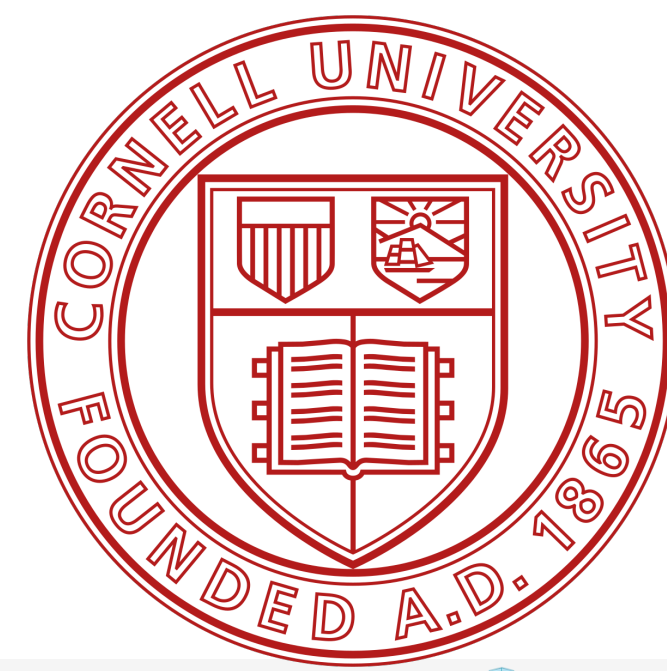
```
1 install.packages("tidyverse")
```

R version 4.4.1 (2024-06-14) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin20

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.



R packages

Installing packages

- Open a new R script.
- Type the `install.packages("tidyverse")` line of code.
- Run it.
- R will download the packages from CRAN and install them on your computer.

A screenshot of the RStudio interface. The main editor window shows a single line of code: `install.packages("tidyverse")`. The Environment pane on the right shows "Environment is empty". The Console pane at the bottom displays the R startup message for version 4.4.1 (2024-06-14) on a Darwin platform. The interface includes a menu bar at the top with options like "File", "Edit", "Session", and "Help". The status bar at the bottom shows the current time as 1:30 and the project name as "Project: (None)".

```
1 install.packages("tidyverse")
```

R version 4.4.1 (2024-06-14) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin20

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Untitled1* x

Source on Save | Run | Source

```
1 install.packages("tidyverse")
2
```

2:1 (Top Level) R Script

Console | Terminal x | Background Jobs x

```
R 4.4.1 · ~/
Content type 'application/x-gzip' length 527797 bytes (515 KB)
=====
downloaded 515 KB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-x86_64/contrib/4.4/tidyverse_2.0.0.tgz'
Content type 'application/x-gzip' length 428830 bytes (418 KB)
=====
downloaded 418 KB

The downloaded binary packages are in
  /var/folders/2r/9nmfdty958xcz1d179s1fxgw0000gn/T//RtmpKNckL4/downloaded_packages
>
```

Environment | History | Connections

Import Dataset | 369 MiB | List

R | Global Environment

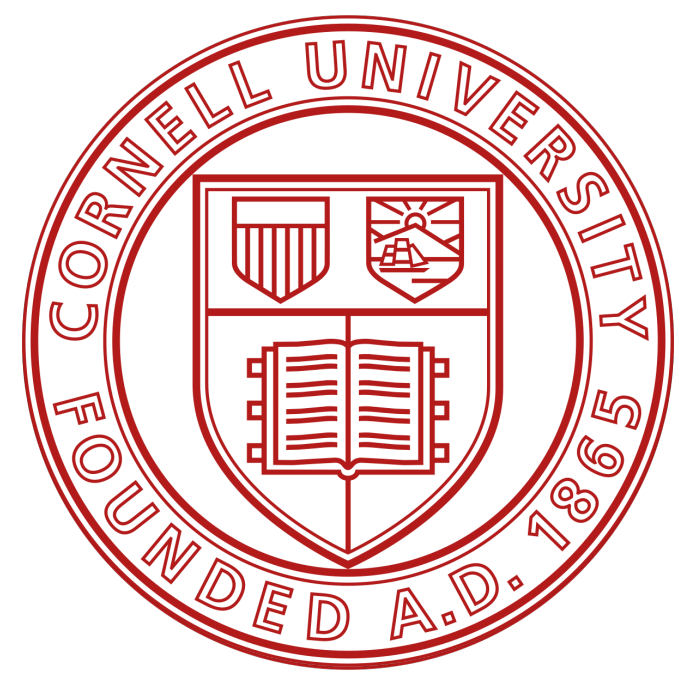
Environment is empty

Files | Plots | Packages | Help | Viewer | Presentation

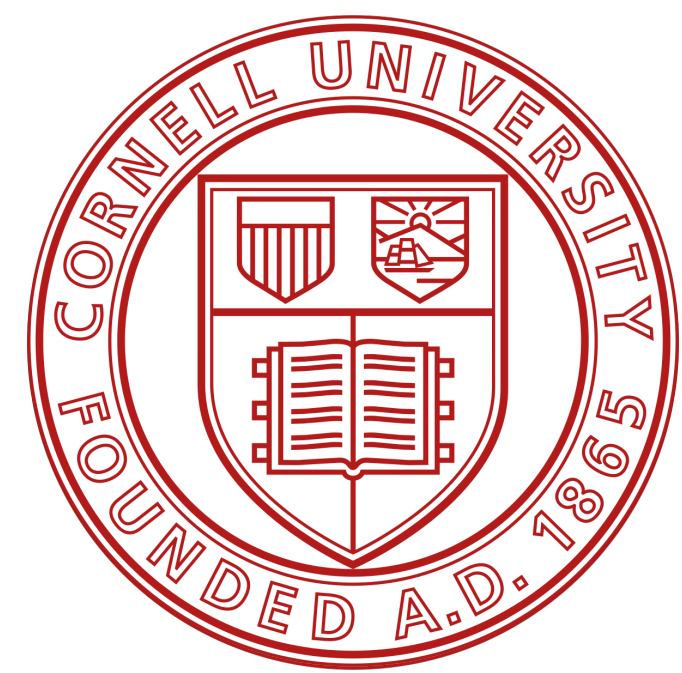
Present | Print | Edit

R packages

Loading packages



```
library(<package name>)
```

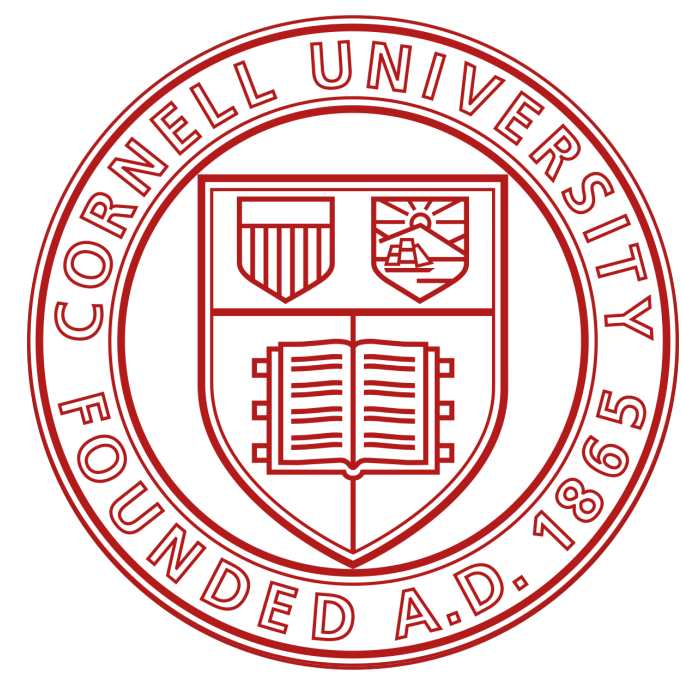


R packages

Loading packages

- Installing a package doesn't immediately place its functions at your fingertips.

```
library(<package name>)
```

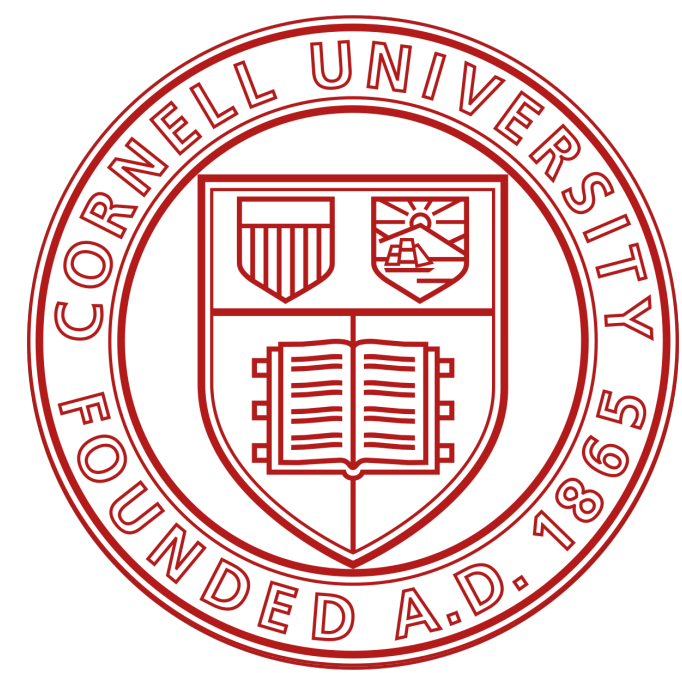


R packages

Loading packages

- Installing a package doesn't immediately place its functions at your fingertips.
- It just places them on your computer.

```
library(<package name>)
```

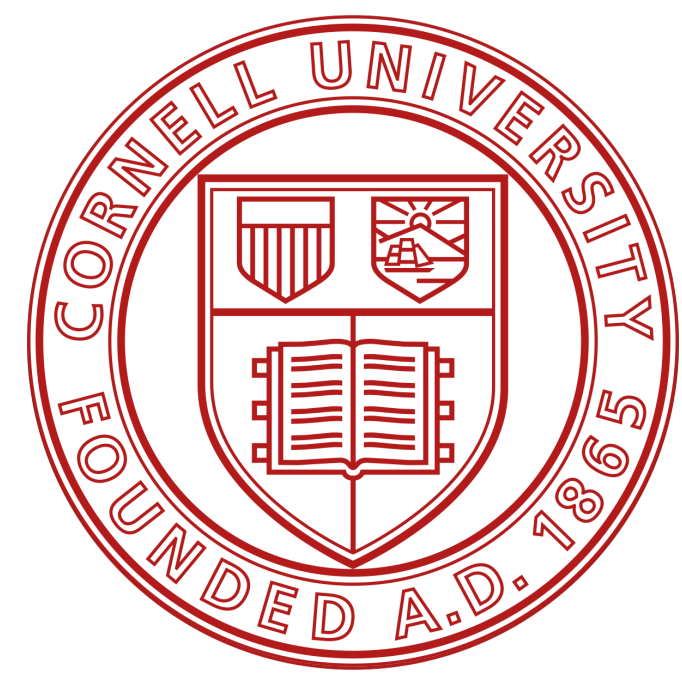



R packages

Loading packages

- Installing a package doesn't immediately place its functions at your fingertips.
- It just places them on your computer.
- To use an R package, you next have to load it in your R session with the command `library`.

```
library(<package name>)
```



R packages

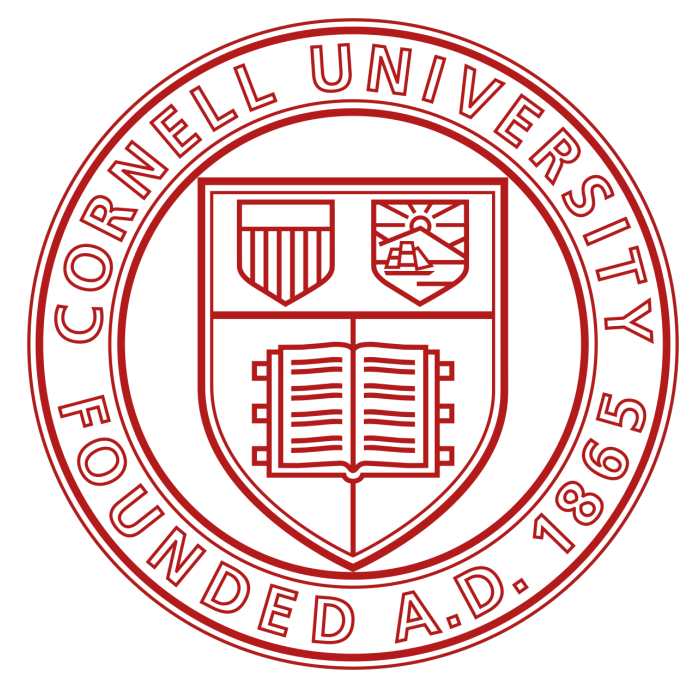
Loading packages

- Installing a package doesn't immediately place its functions at your fingertips.
- It just places them on your computer.
- To use an R package, you next have to load it in your R session with the command `library`.
- Notice that the quotation marks have disappeared.

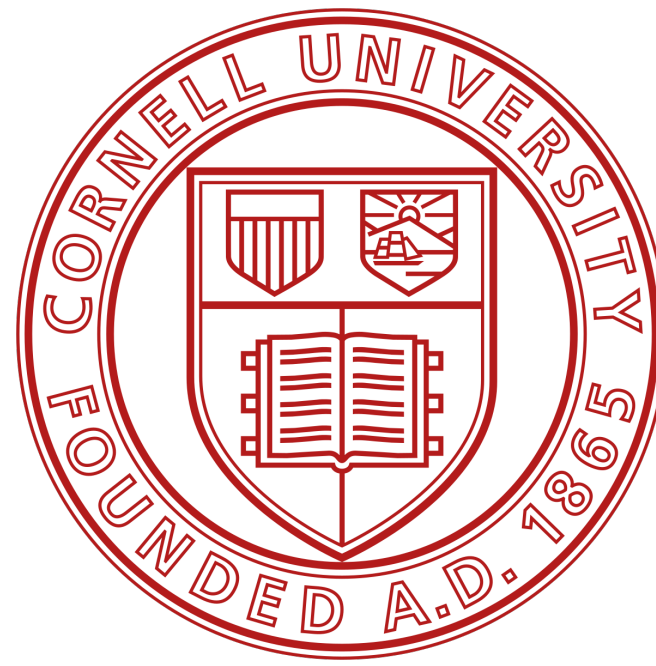
```
library(<package name>)
```

Updating R

R language



UPDATE...

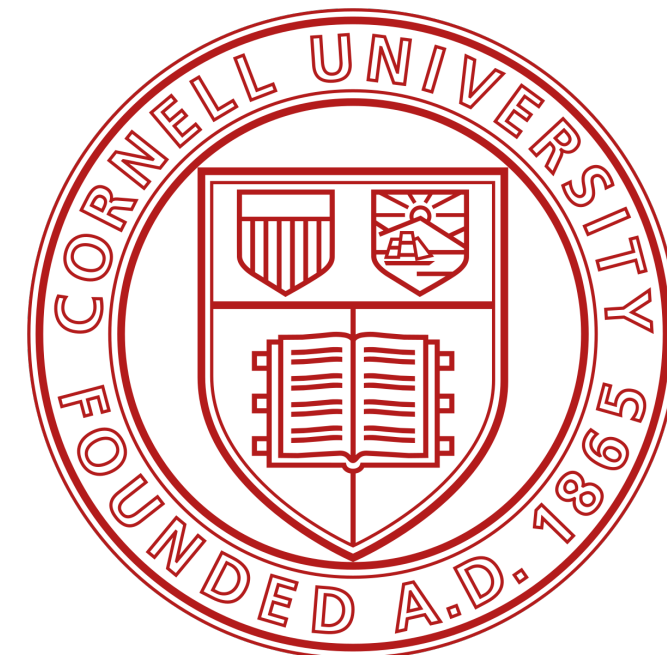


Updating R

R language

- New versions of R are released several times a year.





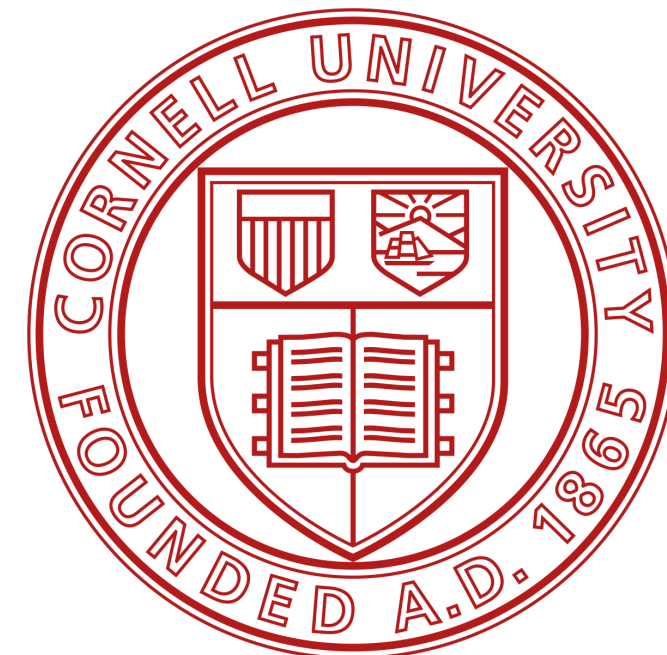
Updating R

R language

- New versions of R are released several times a year.
- The easiest way to stay current with R is to periodically check [the CRAN website](#).



UPDATE...

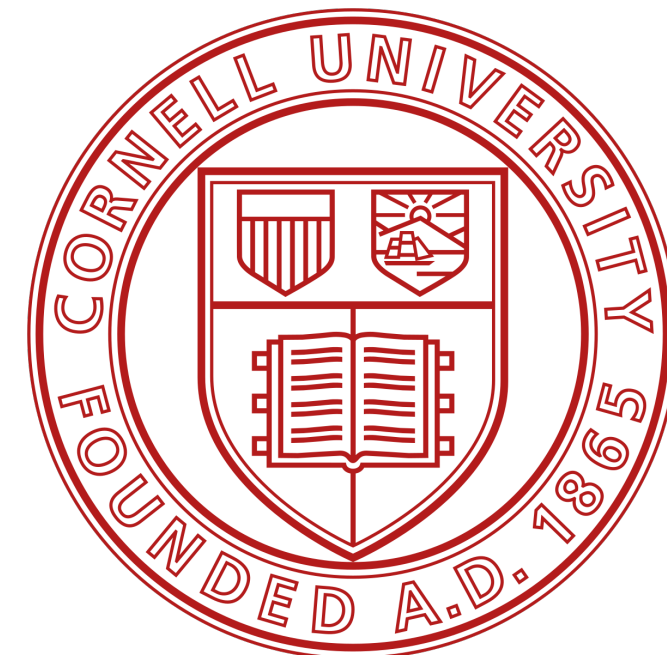


Updating R

R language

- New versions of R are released several times a year.
- The easiest way to stay current with R is to periodically check [the CRAN website](#).
- The process is the same as when you first installed R.





Updating R

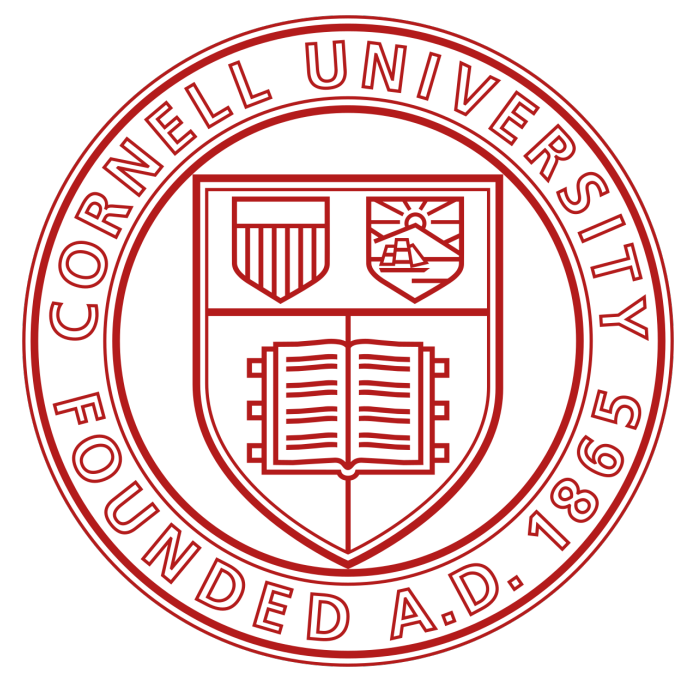
R language

- New versions of R are released several times a year.
- The easiest way to stay current with R is to periodically check [the CRAN website](#).
- The process is the same as when you first installed R.
- Updating to the current version of R is a good place to start if you ever encounter a bug that you can't explain.



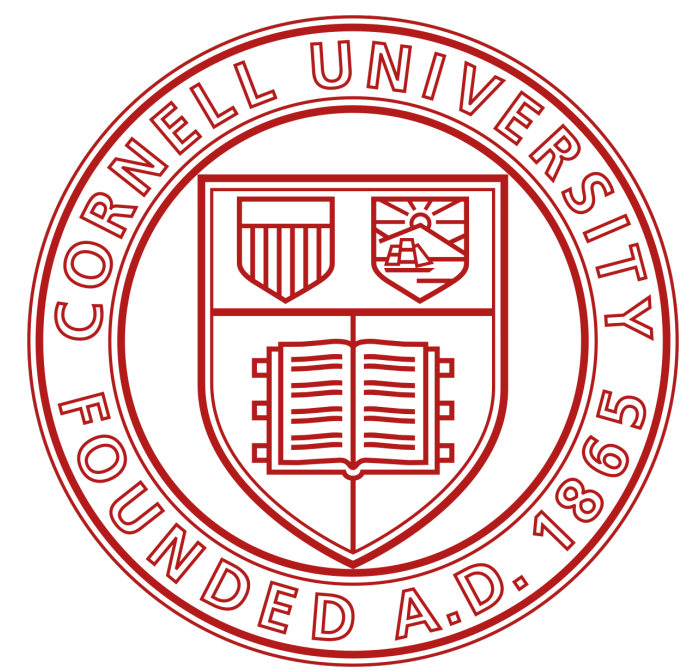
UPDATE...

Updating R R packages



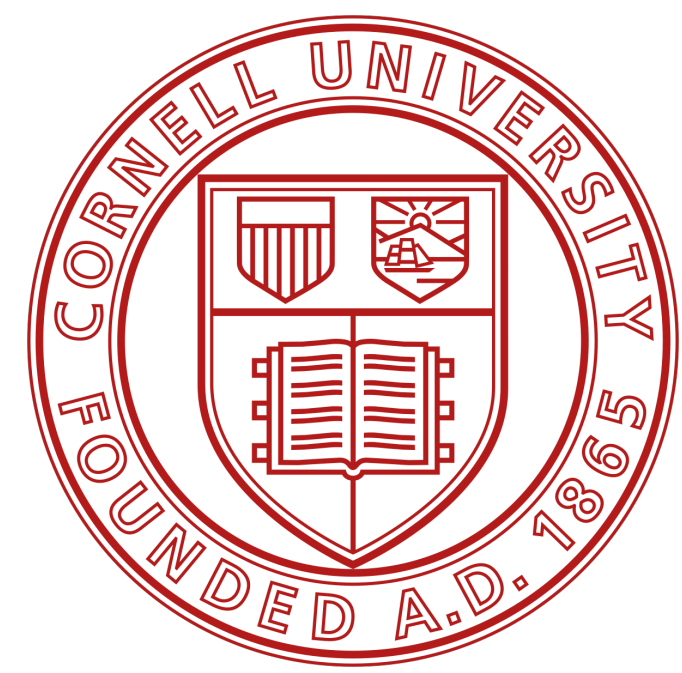
```
update.packages(c("ggplot2", "reshape2", "dplyr"))
```


Updating R R packages



- Package authors occasionally release new versions of their packages to add functions, fix bugs, or improve performance.

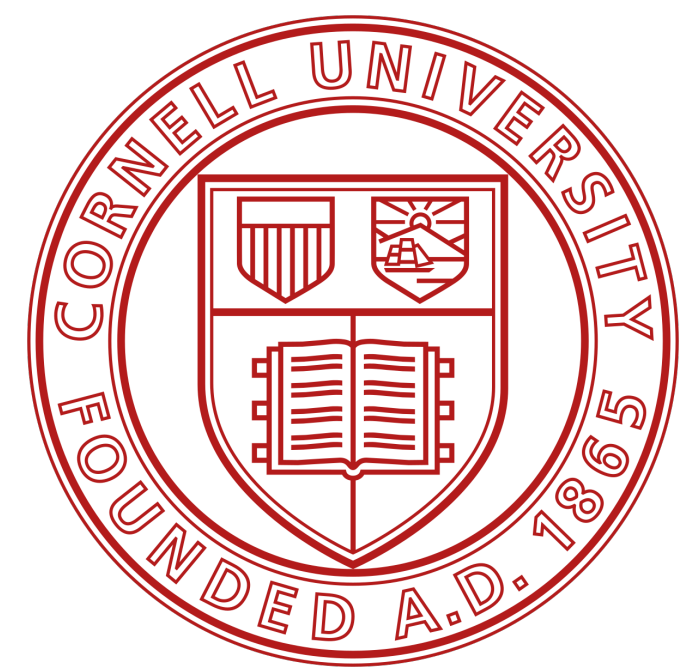
```
update.packages(c("ggplot2", "reshape2", "dplyr"))
```



Updating R R packages

- Package authors occasionally release new versions of their packages to add functions, fix bugs, or improve performance.
- The `update.packages` command checks whether you have the most current version of a package and installs the most current version if you do not.

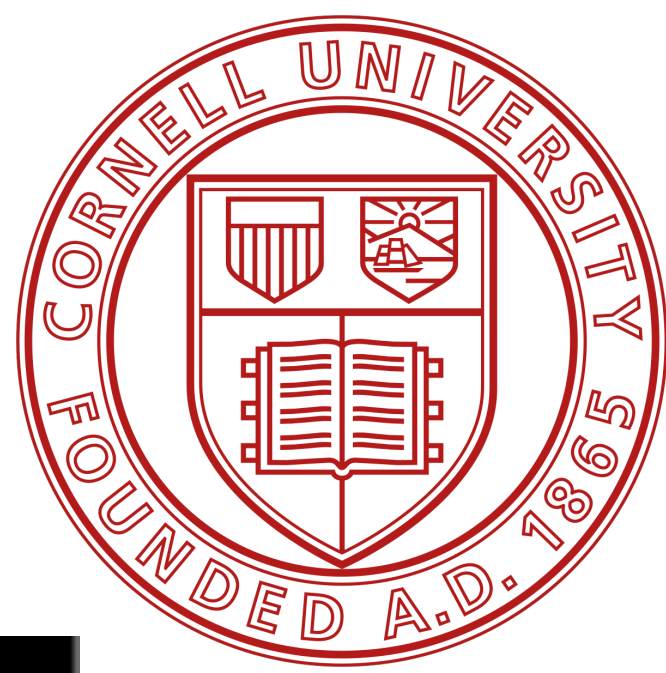
```
update.packages(c("ggplot2", "reshape2", "dplyr"))
```



Updating R R packages

- Package authors occasionally release new versions of their packages to add functions, fix bugs, or improve performance.
- The `update.packages` command checks whether you have the most current version of a package and installs the most current version if you do not.
- You should start a new R session after updating packages.

```
update.packages(c("ggplot2", "reshape2", "dplyr"))
```

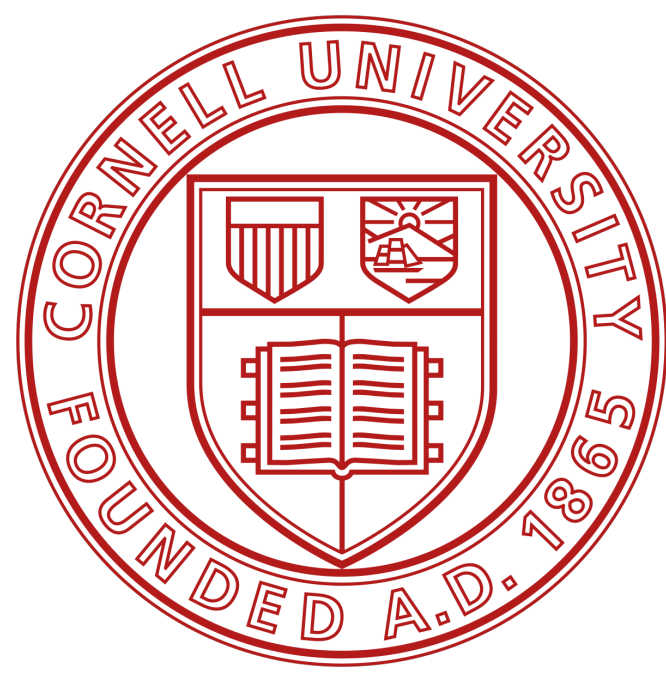


User guide

Comprehensive overview

- If you'd like a comprehensive overview of all of RStudio's features: <https://docs.posit.co/ide/use>.

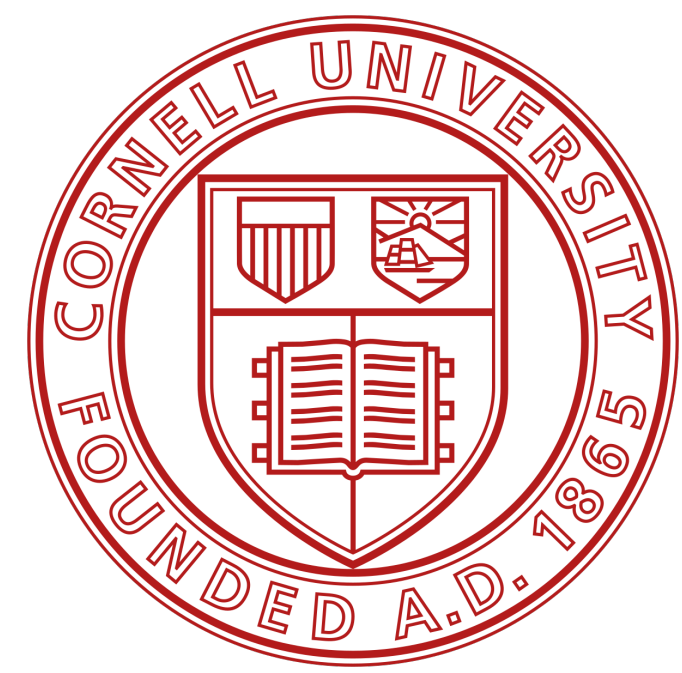




Running R code

Basic command

```
Console Terminal x  
R R 4.4.1 · ~/ ↵  
> 1+1  
[1] 2  
>
```



Running R code

Basic command

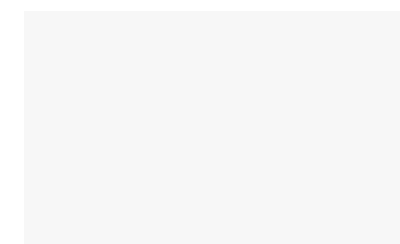
- Open RStudio.

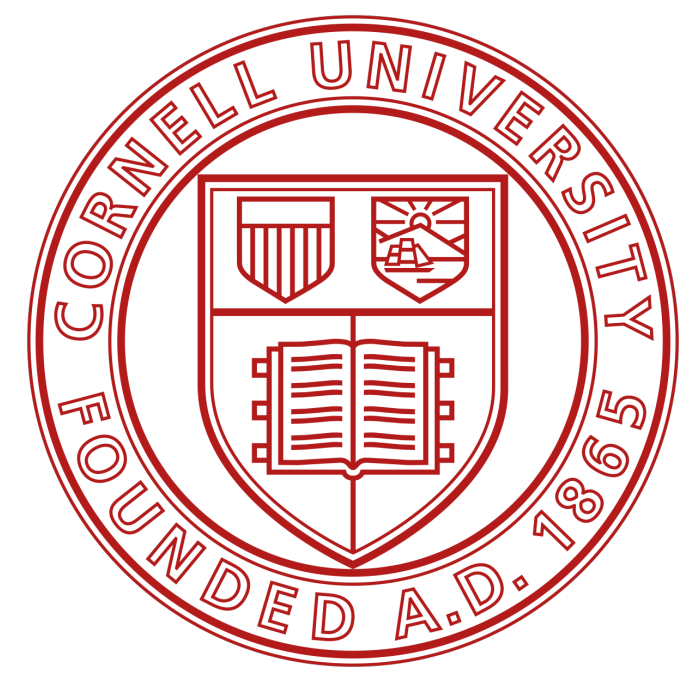


```
> 1+1
```

```
[1] 2
```

```
>
```





Running R code

Basic command

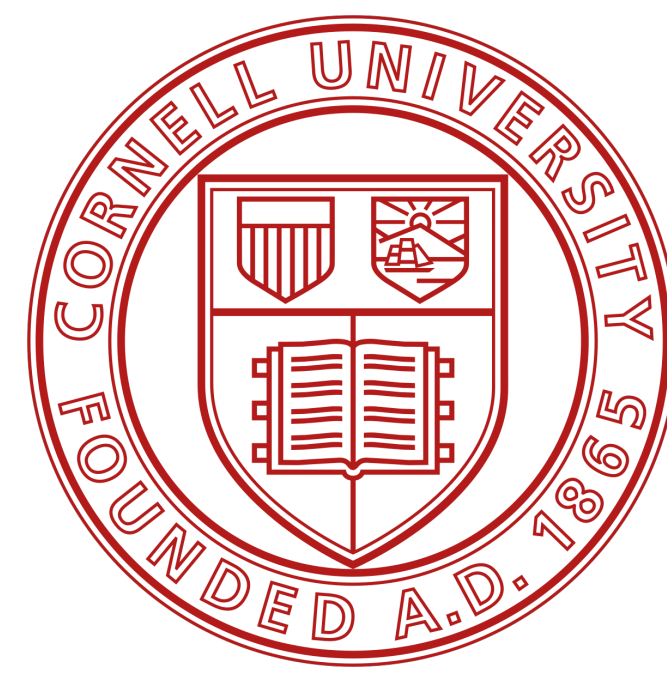
- Open RStudio.
- Type R code into the bottom line of the RStudio console pane.



```
> 1+1
```

```
[1] 2
```

```
>
```



Running R code

Basic command

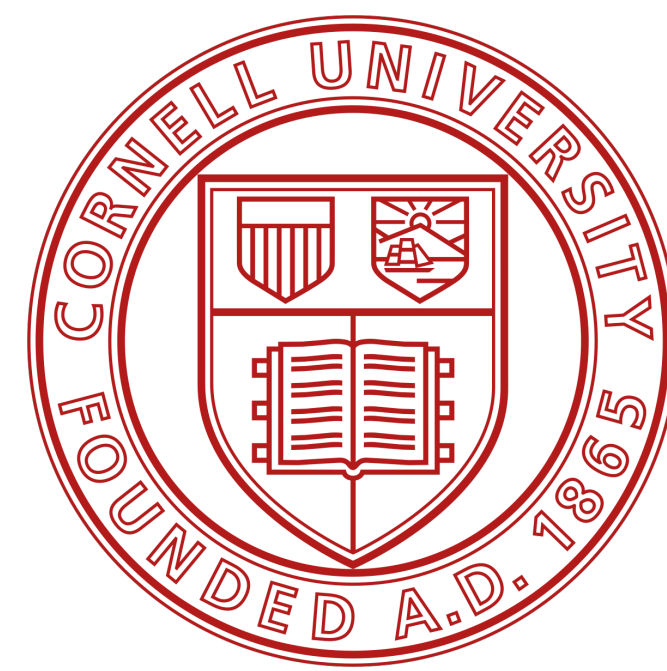
- Open RStudio.
- Type R code into the bottom line of the RStudio console pane.
- Click Enter.



```
> 1+1
```

```
[1] 2
```

```
>
```

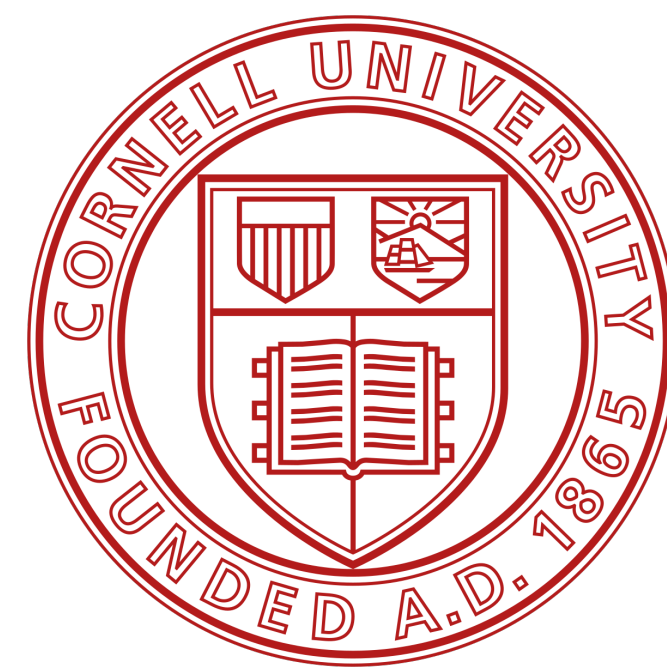
Running R code

Basic command

- Open RStudio.
- Type R code into the bottom line of the RStudio console pane.
- Click Enter.
- The code you type is called a *command*.

A screenshot of the RStudio console pane. The title bar shows "Console" and "Terminal" with a close button. The console displays the R logo, the version "R 4.4.1", and the prompt "~/" with a cursor. The command "> 1+1" is entered, and the output "[1] 2" is displayed. A new prompt ">" is visible on the next line.

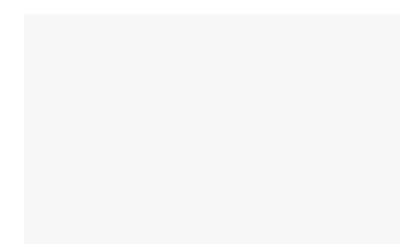
```
R 4.4.1 · ~/ ↵  
> 1+1  
[1] 2  
>
```



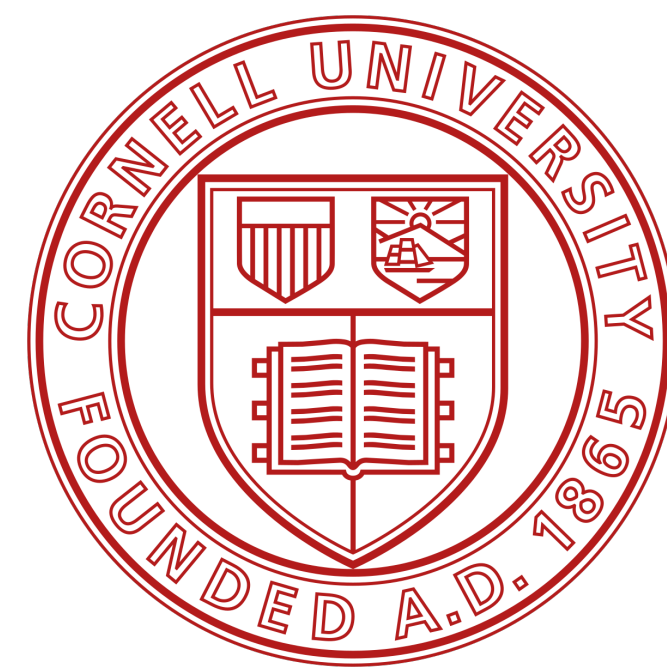
Running R code

Basic command

- Open RStudio.
- Type R code into the bottom line of the RStudio console pane.
- Click Enter.
- The code you type is called a *command*.
- The line you type it into is called the *command line*.

A screenshot of the RStudio interface showing the console and terminal tabs. The console tab is active, displaying the R logo, the version "R 4.4.1", and the prompt ">". The terminal tab is also visible, showing the prompt ">".

```
R 4.4.1 · ~/ ↵  
> 1+1  
[1] 2  
>
```



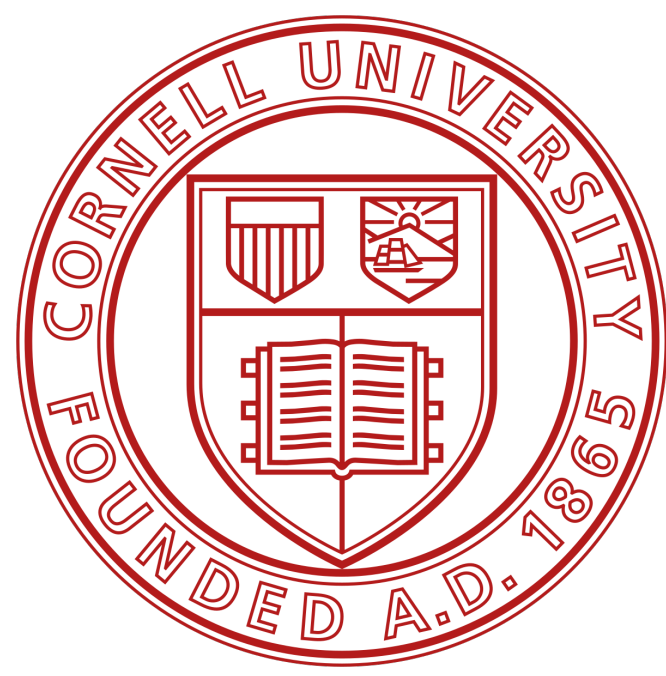
Running R code

Basic command

- Open RStudio.
- Type R code into the bottom line of the RStudio console pane.
- Click Enter.
- The code you type is called a *command*.
- The line you type it into is called the *command line*.
- For example, if you type `1 + 1` and hit Enter, RStudio will display:

A screenshot of the RStudio console pane. The title bar shows "Console" and "Terminal" with a close button. The console displays the R logo, the version "R 4.4.1", and the prompt "R 4.4.1 · ~/". Below this, the command "> 1+1" is entered. The output "[1] 2" is displayed. A new prompt ">" is shown on the next line, indicating the command has been executed.

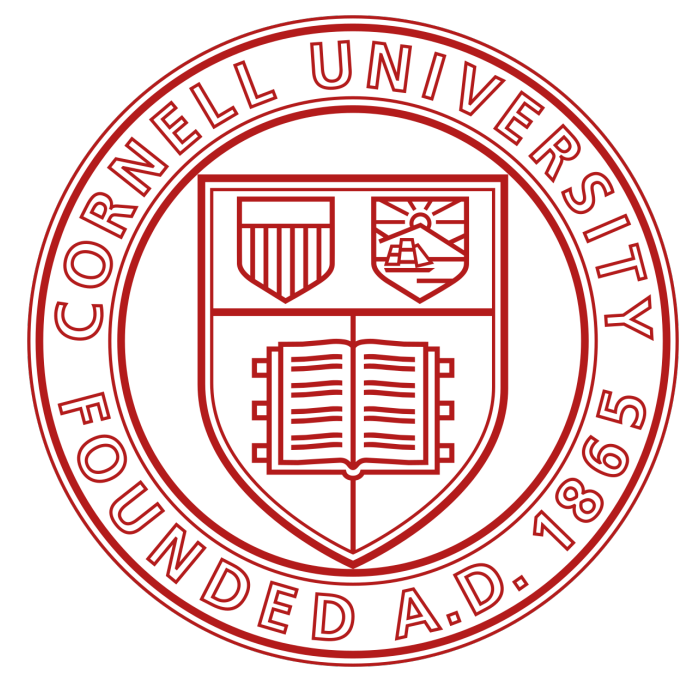
```
R 4.4.1 · ~/  
> 1+1  
[1] 2  
>
```



Basics

Sequence of numbers

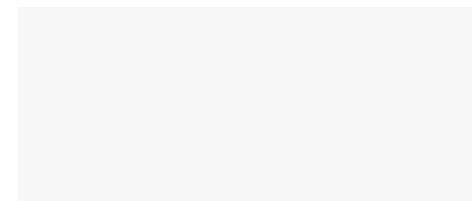
```
Console Terminal x
R 4.4.1 · ~/ ↗
> 100:130
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 [24] 123 124 125 126 127 128 129 130
> |
```



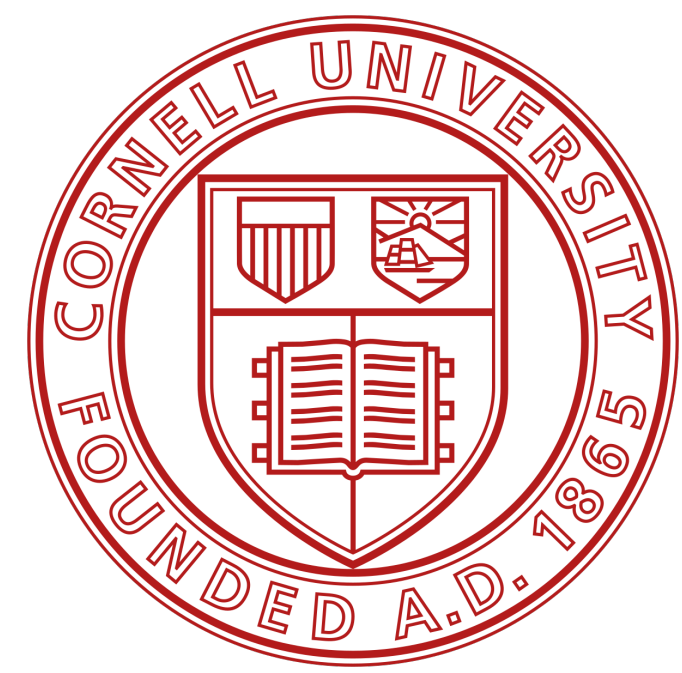
Basics

Sequence of numbers

- You'll notice that a `[1]` appears next to your result.



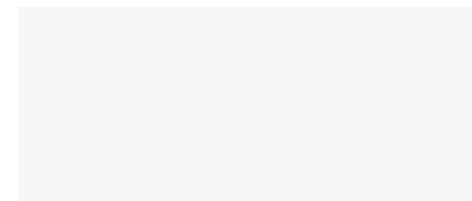
```
Console Terminal x
R 4.4.1 · ~/ ↵
> 100:130
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 [24] 123 124 125 126 127 128 129 130
> |
```



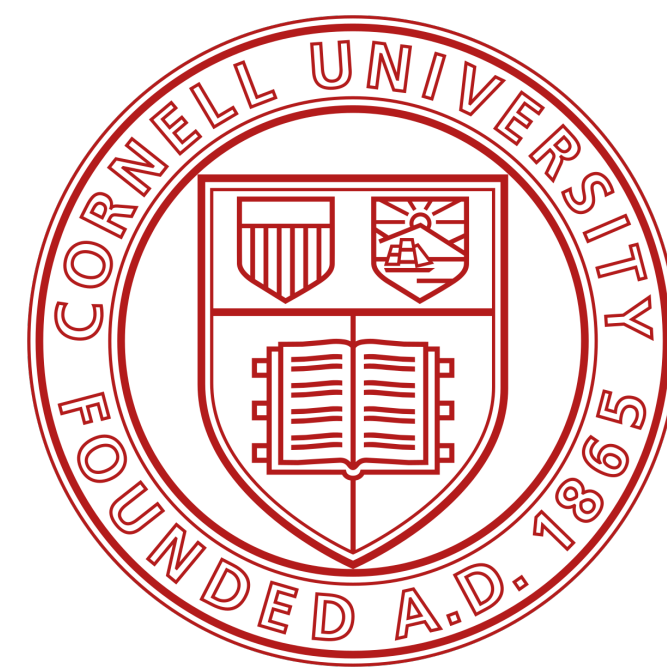
Basics

Sequence of numbers

- You'll notice that a `[1]` appears next to your result.
- R is just letting you know that this line begins with the first value in your result.



```
Console Terminal x
R 4.4.1 · ~/
> 100:130
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 [24] 123 124 125 126 127 128 129 130
> |
```

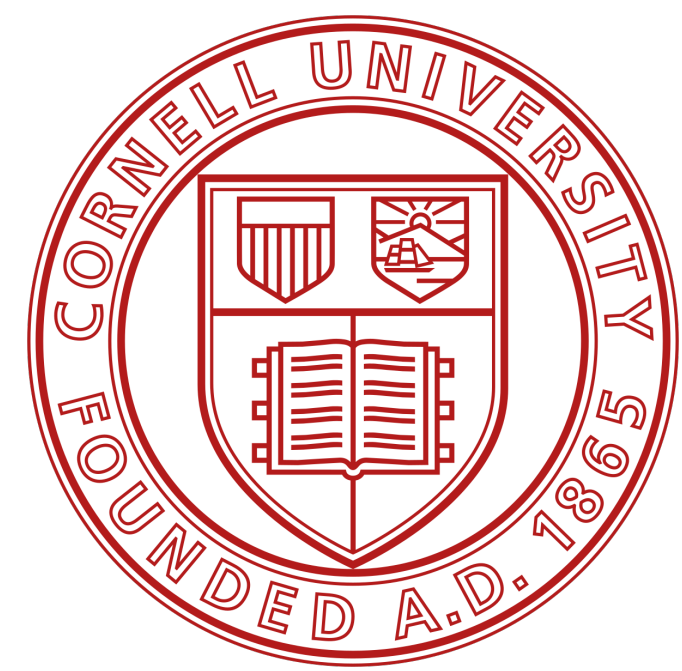


Basics

Sequence of numbers

- You'll notice that a `[1]` appears next to your result.
- R is just letting you know that this line begins with the first value in your result.
- Some commands return more than one value, and their results may fill up multiple lines.

```
Console Terminal x
R 4.4.1 · ~/
> 100:130
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 [24] 123 124 125 126 127 128 129 130
> |
```

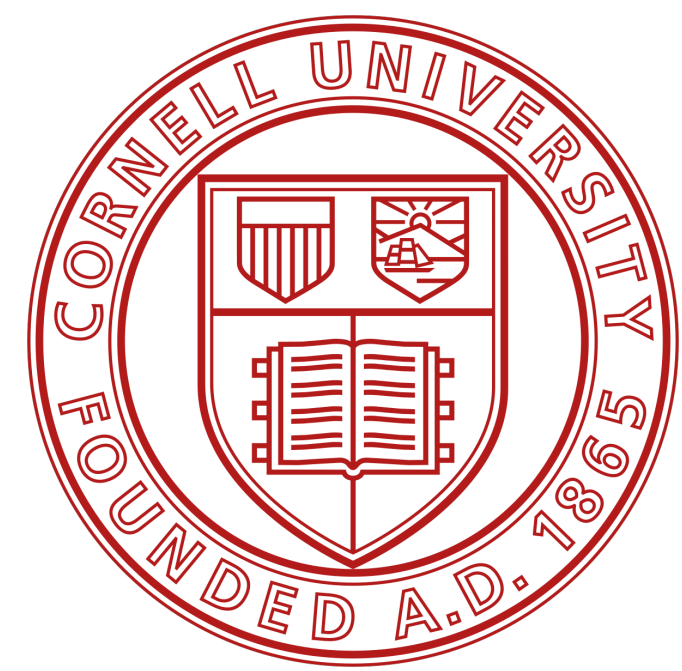


Basics

Sequence of numbers

- You'll notice that a `[1]` appears next to your result.
- R is just letting you know that this line begins with the first value in your result.
- Some commands return more than one value, and their results may fill up multiple lines.
- For example, the command `100:130` returns 31 values; it creates a sequence of integers from 100 to 130.

```
Console Terminal x
R 4.4.1 · ~/
> 100:130
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 [24] 123 124 125 126 127 128 129 130
> |
```

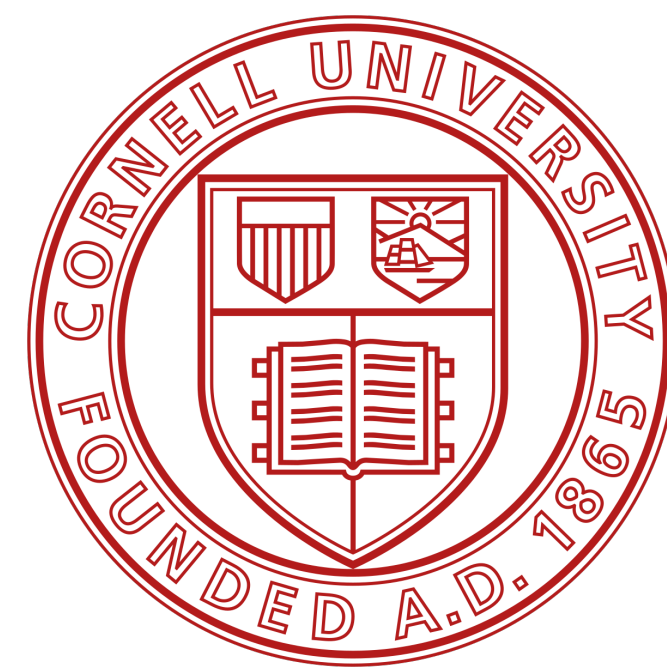



Basics

Sequence of numbers

- You'll notice that a `[1]` appears next to your result.
- R is just letting you know that this line begins with the first value in your result.
- Some commands return more than one value, and their results may fill up multiple lines.
- For example, the command `100:130` returns 31 values; it creates a sequence of integers from 100 to 130.
- Notice that new bracketed numbers appear at the start of the second line of output.

```
Console Terminal x
R 4.4.1 · ~/
> 100:130
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 [24] 123 124 125 126 127 128 129 130
> |
```

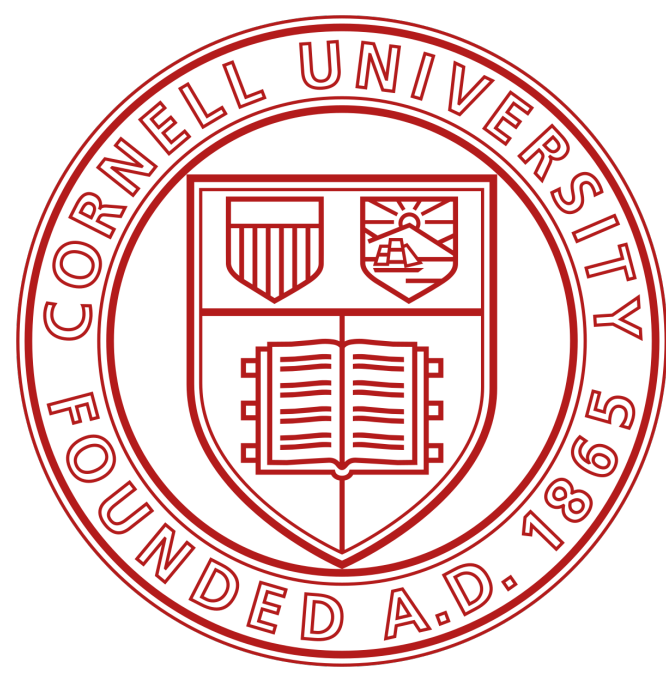


Basics

Sequence of numbers

- You'll notice that a `[1]` appears next to your result.
- R is just letting you know that this line begins with the first value in your result.
- Some commands return more than one value, and their results may fill up multiple lines.
- For example, the command `100:130` returns 31 values; it creates a sequence of integers from 100 to 130.
- Notice that new bracketed numbers appear at the start of the second line of output.
- It means that the second line begins with the 24th value in the result.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 100:130
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 [24] 123 124 125 126 127 128 129 130
> |
```



Basics

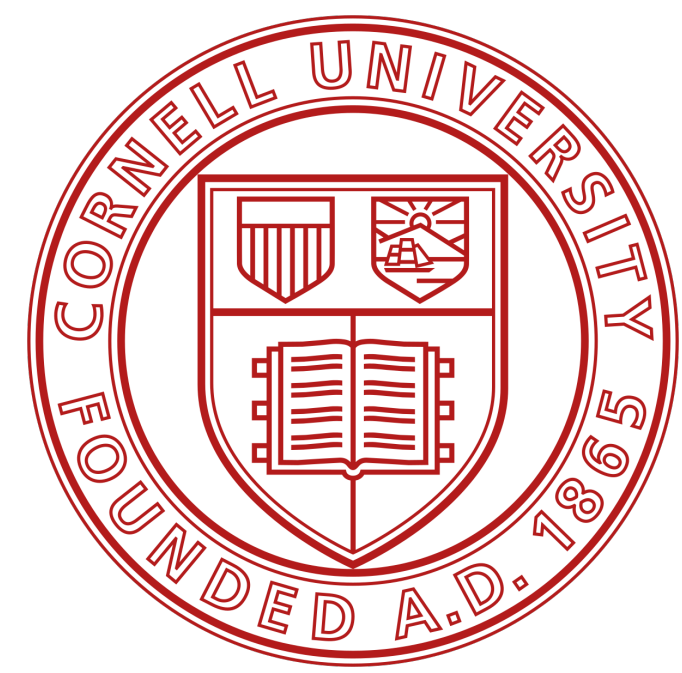
Incomplete command

```
Console Terminal x  
R R 4.4.1 · ~/ ↵
```

```
> 5 -  
+
```

```
R R 4.4.1 · ~/ ↵
```

```
> 5 -  
+ 1  
[1] 4
```



Basics

Incomplete command

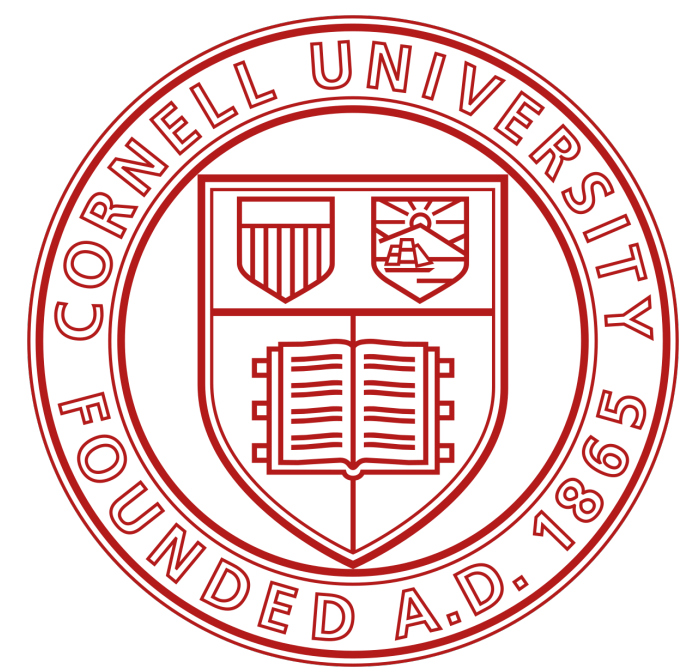
- If you type an incomplete command and press Enter, R will display a `+` prompt.

```
Console Terminal x
R 4.4.1 · ~/ ↵
```

```
> 5 -
+ 
```

```
R 4.4.1 · ~/ ↵
```

```
> 5 -
+ 1
[1] 4
```



Basics

Incomplete command

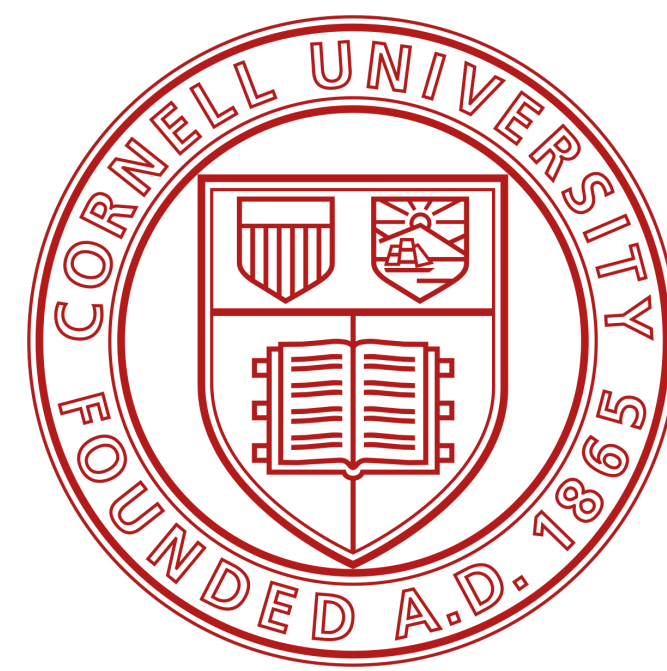
- If you type an incomplete command and press Enter, R will display a `+` prompt.
- It means R is waiting for you to type the rest of your command.

```
Console Terminal x
R 4.4.1 · ~/ ↵
```

```
> 5 -
+
```

```
R 4.4.1 · ~/ ↵
```

```
> 5 -
+ 1
[1] 4
```



Basics

Incomplete command

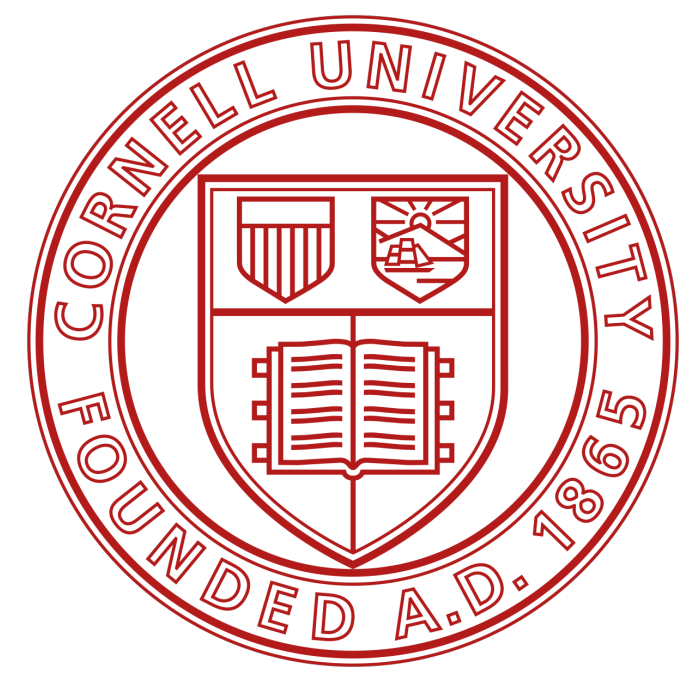
- If you type an incomplete command and press Enter, R will display a `+` prompt.
- It means R is waiting for you to type the rest of your command.
- Either finish the command or hit Escape to start over.

```
Console Terminal x
R 4.4.1 · ~/ ↵
```

```
> 5 -
+
```

```
R 4.4.1 · ~/ ↵
```

```
> 5 -
+ 1
[1] 4
```

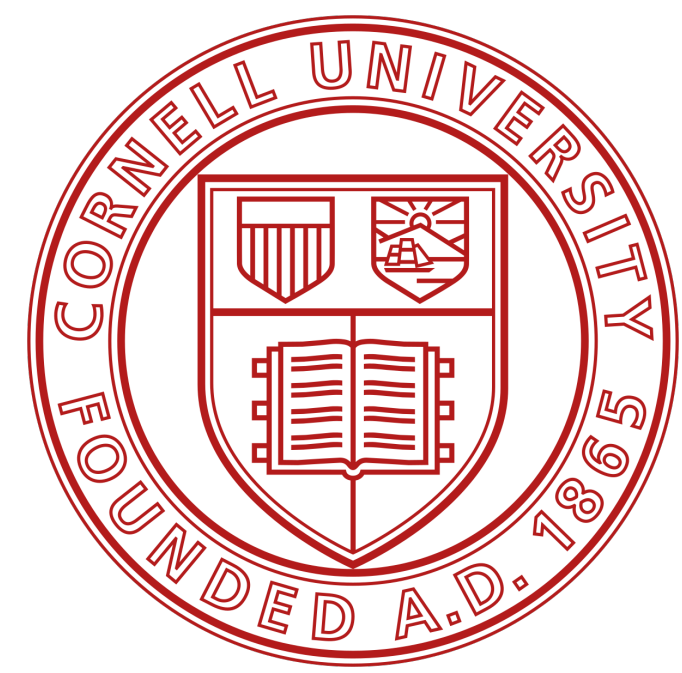


Basics

Error message

- If you type a command that R doesn't recognize, R will return an error message.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 3%5
Error: unexpected input in "3%5"
> |
```



Basics

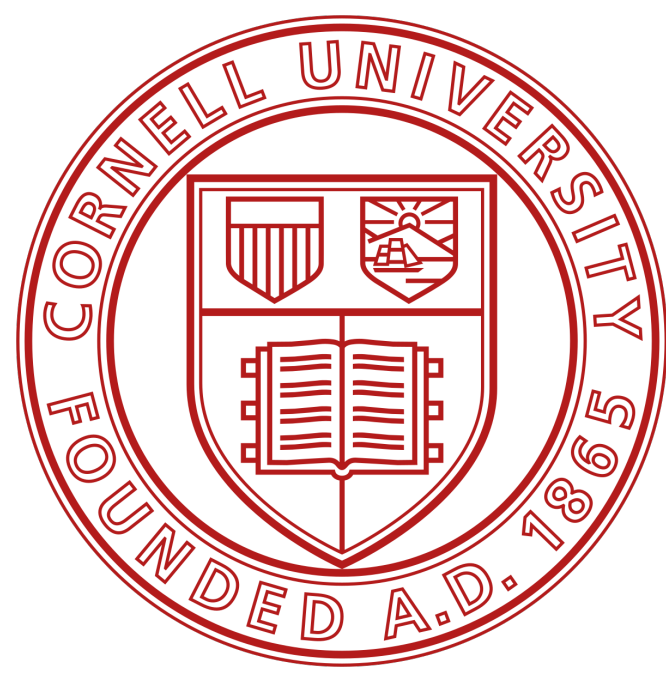
Calculator

- You can easily do anything in R that you would do with a calculator.

```
Console Terminal x
R 4.4.1 · ~/ ↗
> 2*3
[1] 6
> 4-1
[1] 3
> 6/(4-1)
[1] 2
>
```


Basics

Comments



Console

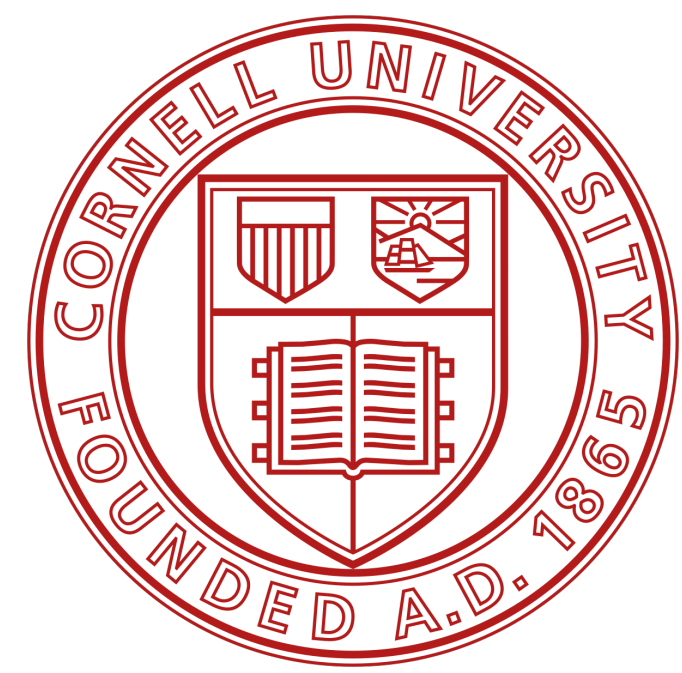
Terminal ×

 R 4.4.1 · ~/ 

```
> 1+1 # outputs the sum of one and one
```

```
[1] 2
```

```
>
```

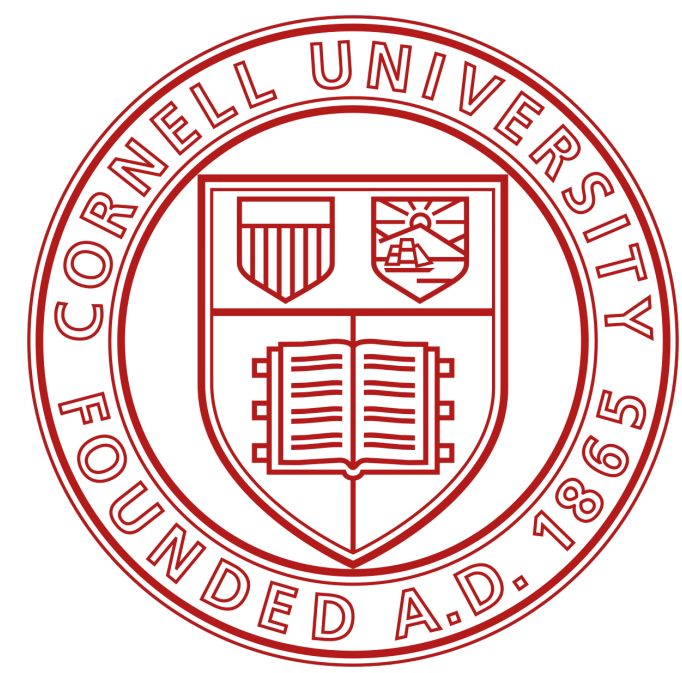


Basics

Comments

- R treats the hashtag character, #, in a special way.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1+1 # outputs the sum of one and one
[1] 2
>
```

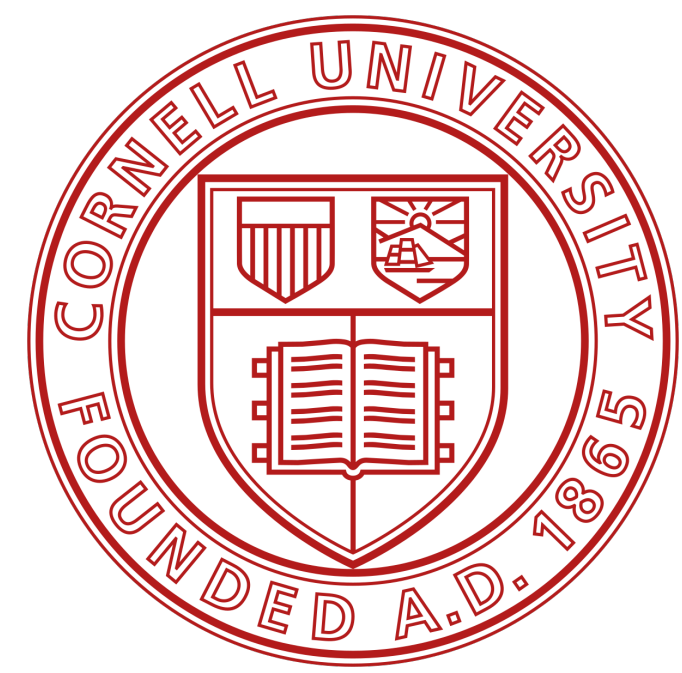


Basics

Comments

- R treats the hashtag character, #, in a special way.
- R will not run anything that follows a hashtag on a line.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1+1 # outputs the sum of one and one
[1] 2
>
```

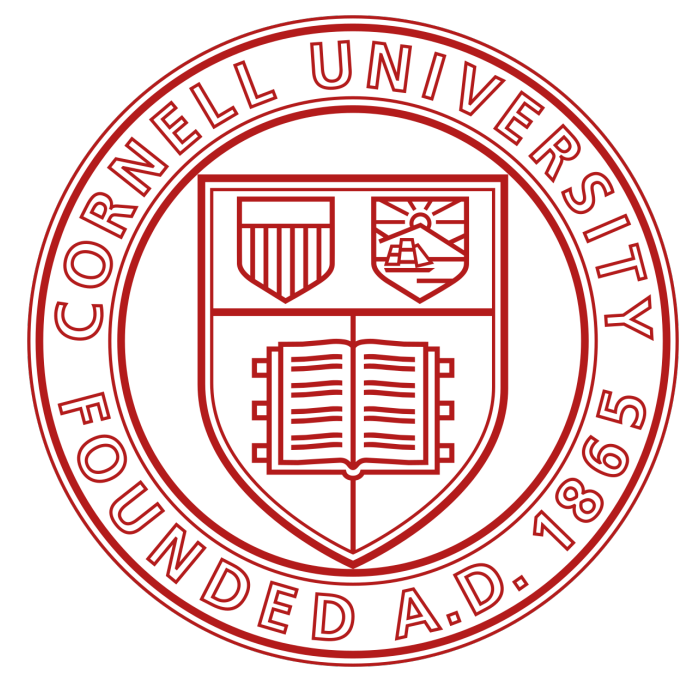


Basics

Comments

- R treats the hashtag character, #, in a special way.
- R will not run anything that follows a hashtag on a line.
- This makes hashtags very useful for adding comments and annotations to your code.

```
Console Terminal x
R 4.4.1 · ~/ ↗
> 1+1 # outputs the sum of one and one
[1] 2
>
```



Basics

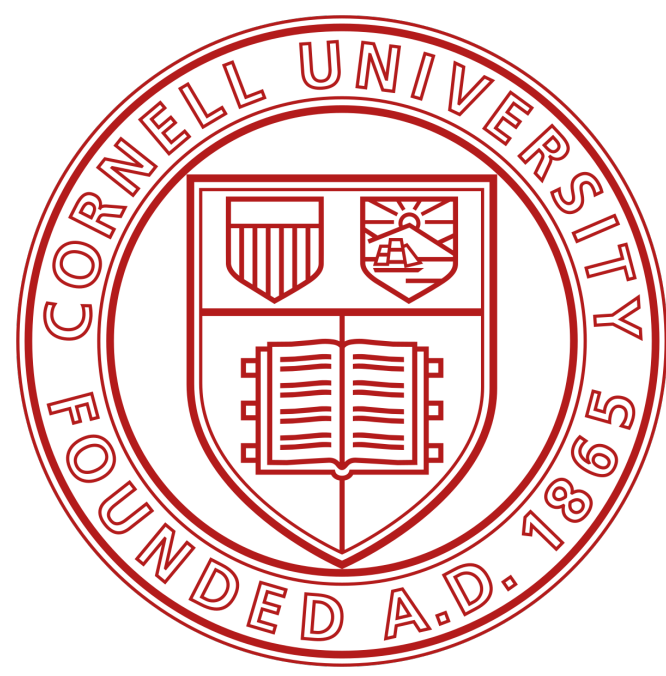
Comments

- R treats the hashtag character, #, in a special way.
- R will not run anything that follows a hashtag on a line.
- This makes hashtags very useful for adding comments and annotations to your code.
- Humans will be able to read the comments, but your computer will pass over them.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1+1 # outputs the sum of one and one
[1] 2
>
```

Basics

Console vs R. file



```
First_lesson.R x
```

← → | ↩ | 💾 Source | 🔍

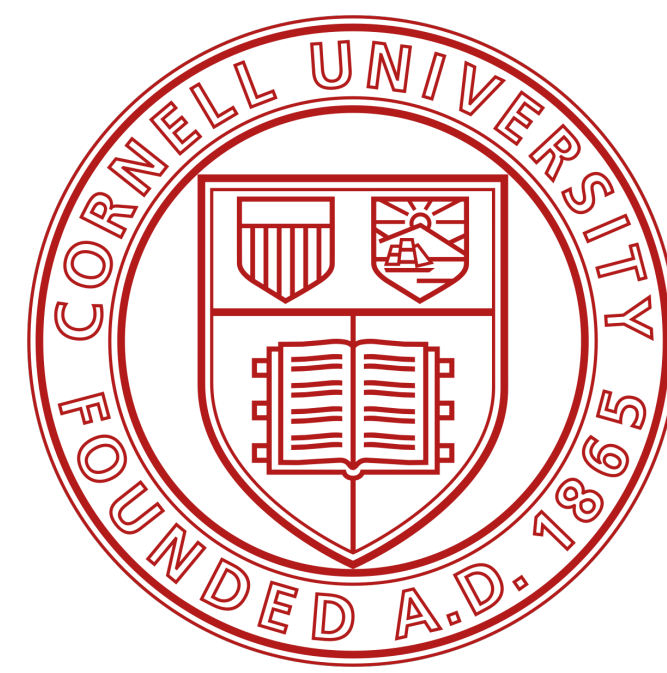
```
1 1+1
2 2*2
3 3/5
4 |
```

4:1 (Top Level) ⌵ R Script ⌵

Console Terminal x

R 4.4.1 · ~/ ↩

>

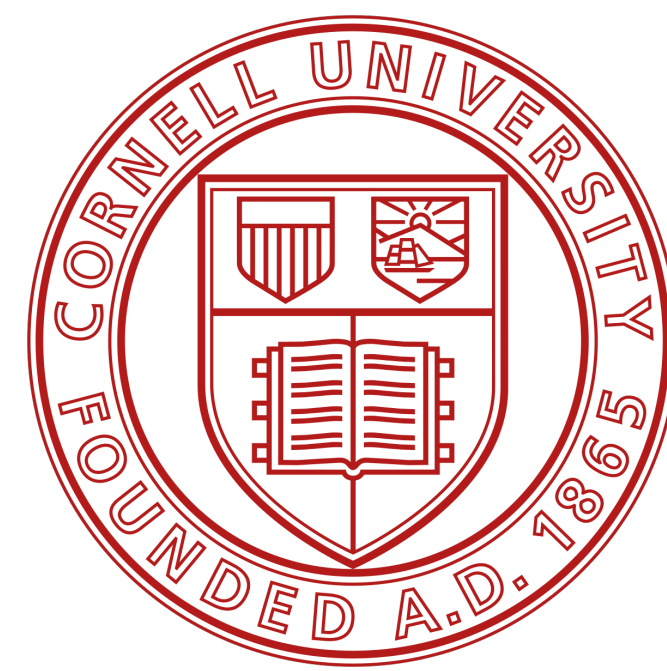


Basics

Console vs R. file

- In RStudio, code can be executed either by writing it directly in the console or by creating and running scripts from a .R file.

The screenshot shows the RStudio interface. The top pane is the script editor, titled "First_lesson.R". It contains four lines of R code: `1 1+1`, `2 2*2`, `3 3/5`, and `4` followed by a vertical cursor. The bottom pane is the console, titled "Terminal", showing the R prompt `>` and the R version `R 4.4.1 · ~/`.



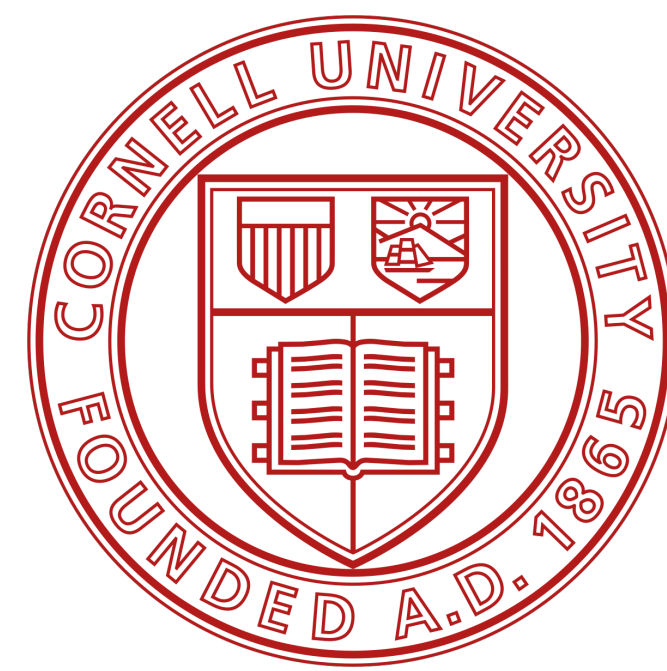
Basics

Console vs R. file

- In RStudio, code can be executed either by writing it directly in the console or by creating and running scripts from a .R file.
- The console in RStudio is an interactive environment where users can write and execute commands line-by-line.

A screenshot of the RStudio interface. The top pane shows a script editor with a file named "First_lesson.R". The script contains four lines of code: "1 1+1", "2 2*2", "3 3/5", and "4 |". The bottom pane shows the console, which is currently empty and has a blue prompt character ">". The console title is "Terminal" and it shows "R 4.4.1 · ~/".

```
First_lesson.R x
Source
1 1+1
2 2*2
3 3/5
4 |
4:1 (Top Level) R Script
Terminal x
R 4.4.1 · ~/
>
```

Basics

Console vs R. file

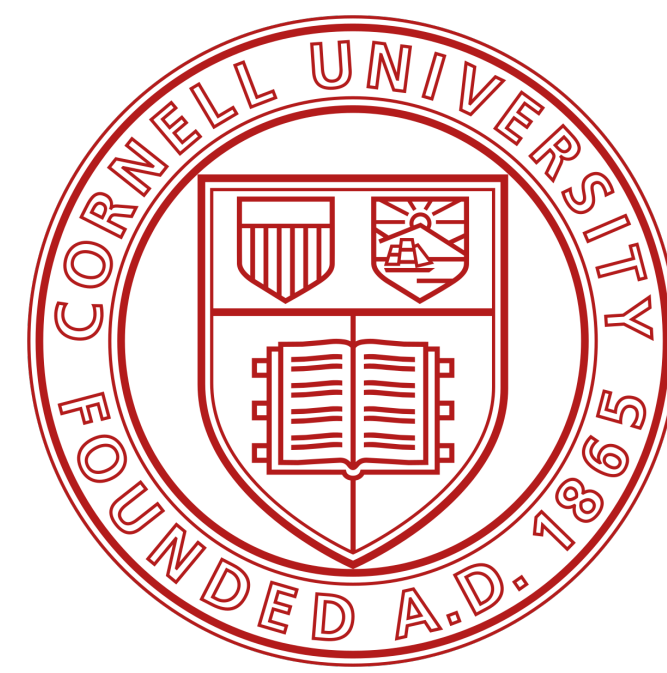
- In RStudio, code can be executed either by writing it directly in the console or by creating and running scripts from a .R file.
- The console in RStudio is an interactive environment where users can write and execute commands line-by-line.
- A .R file, on the other hand, is a script file where users can write, edit, and save multiple lines of code.

A screenshot of the RStudio interface. The top pane shows a script editor for a file named "First_lesson.R". The script contains four lines of code: "1 1+1", "2 2*2", "3 3/5", and "4 |". The bottom pane shows the console, which is currently empty and displays the R version "R 4.4.1" and the current directory "~/" followed by a prompt character ">".

```
First_lesson.R x
Source
1 1+1
2 2*2
3 3/5
4 |
4:1 (Top Level) R Script
Console Terminal x
R 4.4.1 · ~/
>
```

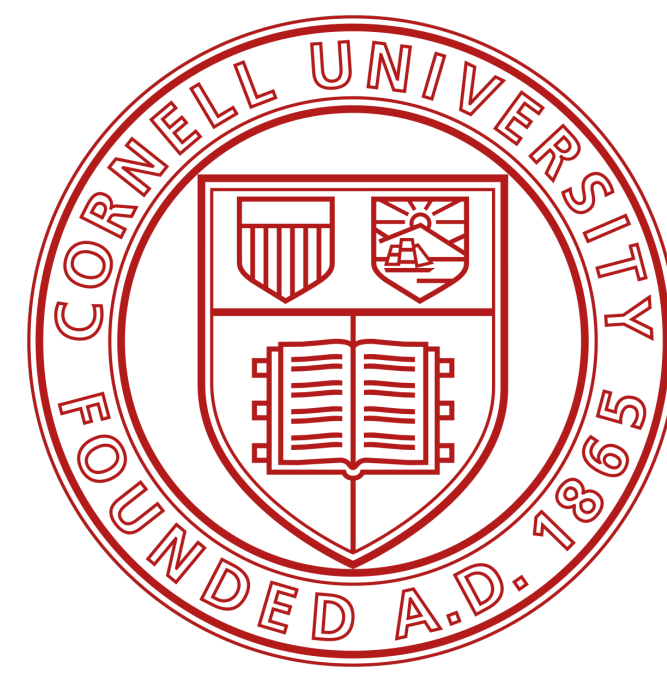
Basics

Run vs Source

A screenshot of an R script editor window. The window has two tabs: "First_lesson.R" and "Exercises_W1.R". The toolbar includes icons for navigation, a "Source on Save" checkbox, a search icon, a "Run" button (a green arrow), and a "Source" button (a blue arrow). The code area contains four lines of R code:

```
1 1+1
2 2*2
3 3/5
4 |
```

The status bar at the bottom shows "4:1" and "(Top Level)".



Basics

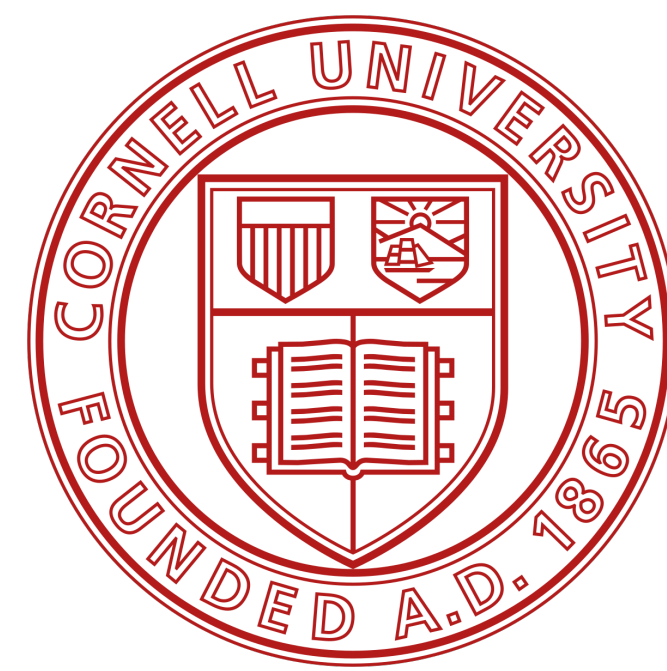
Run vs Source

- The 'Source' function is used to execute all the code within an entire script file. When you source a file, R reads and executes every command in the script from start to finish.

A screenshot of the R Studio interface. The top toolbar shows the 'Source' button (a blue arrow pointing right) and the 'Run' button (a green arrow pointing right). Below the toolbar, the script editor shows four lines of code:

```
1 1+1
2 2*2
3 3/5
4 |
```

The status bar at the bottom indicates the current position is 4:1 at the top level of the script.



Basics

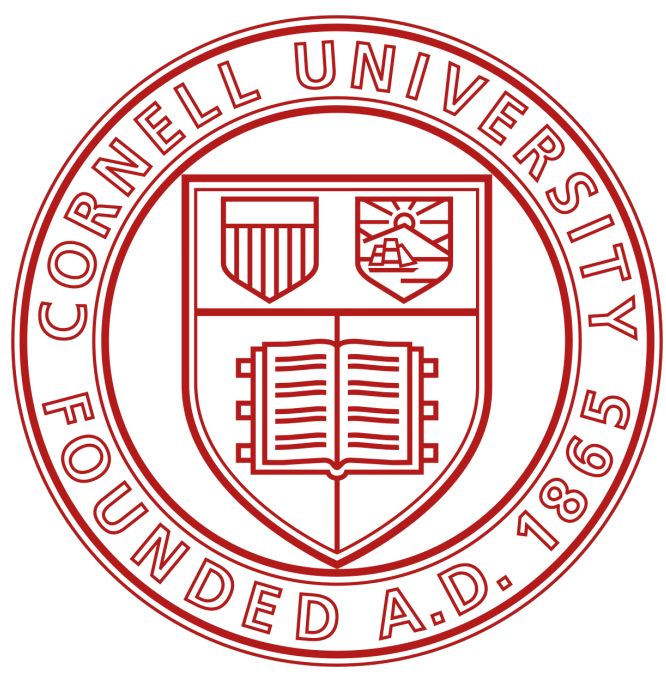
Run vs Source

- The 'Source' function is used to execute all the code within an entire script file. When you source a file, R reads and executes every command in the script from start to finish.
- The 'Run' command is used to execute selected lines of code from a script file. In RStudio, users can highlight specific portions of the script and use the 'Run' button to execute only those highlighted lines.

A screenshot of the RStudio interface. The top window shows two tabs: "First_lesson.R" and "Exercises_W1.R". Below the tabs is a toolbar with various icons, including a "Source on Save" checkbox, a search icon, a "Run" button (a green arrow), and a "Source" button (a blue arrow). The main editor area contains four lines of code:

```
1 1+1
2 2*2
3 3/5
4 |
```

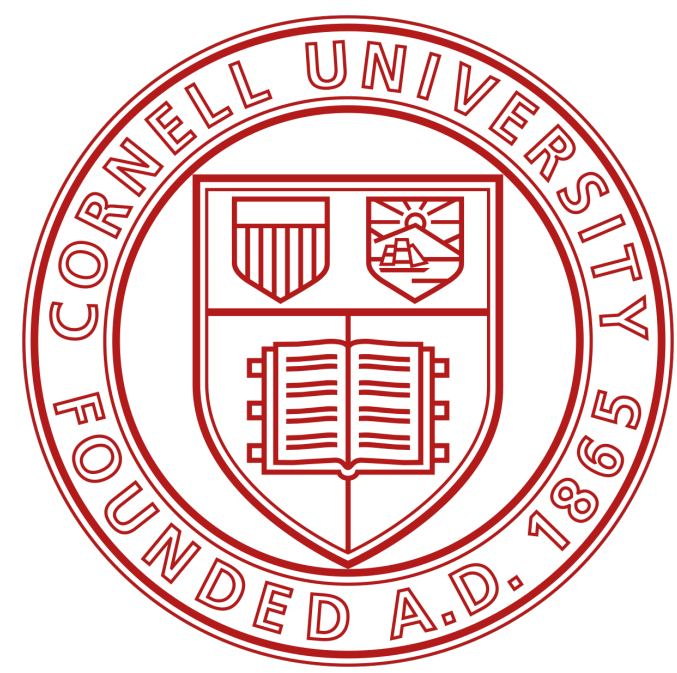
The first three lines are highlighted in blue. At the bottom of the editor, the status bar shows "4:1" and "(Top Level)".



Playing with dices

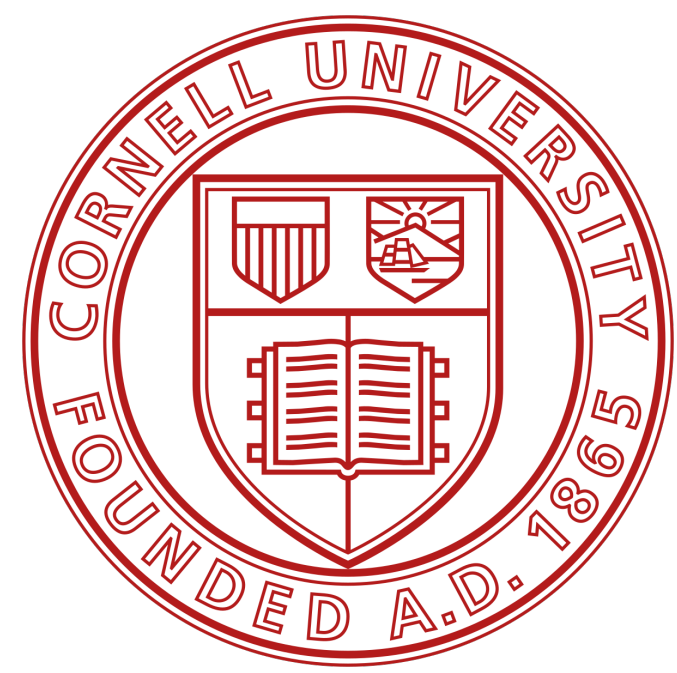
Basics

Objects



Basics

Objects

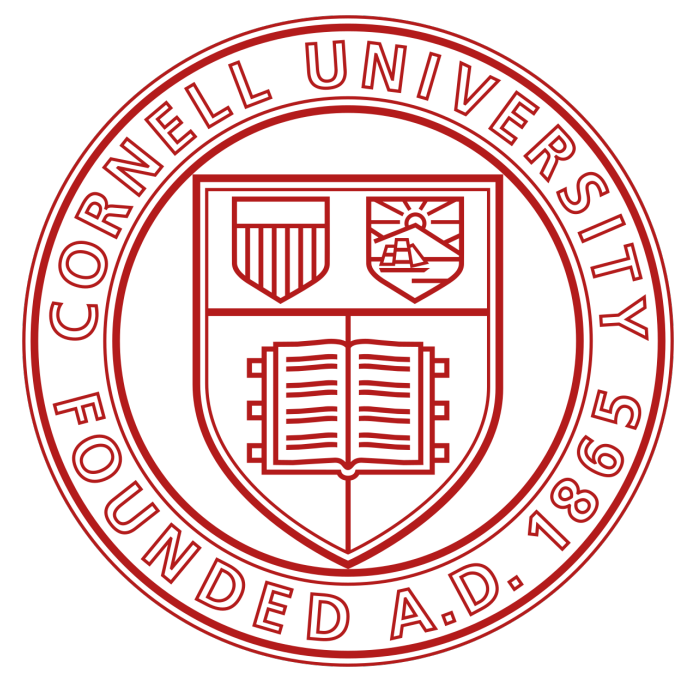


- Let's create a virtual die



Basics

Objects

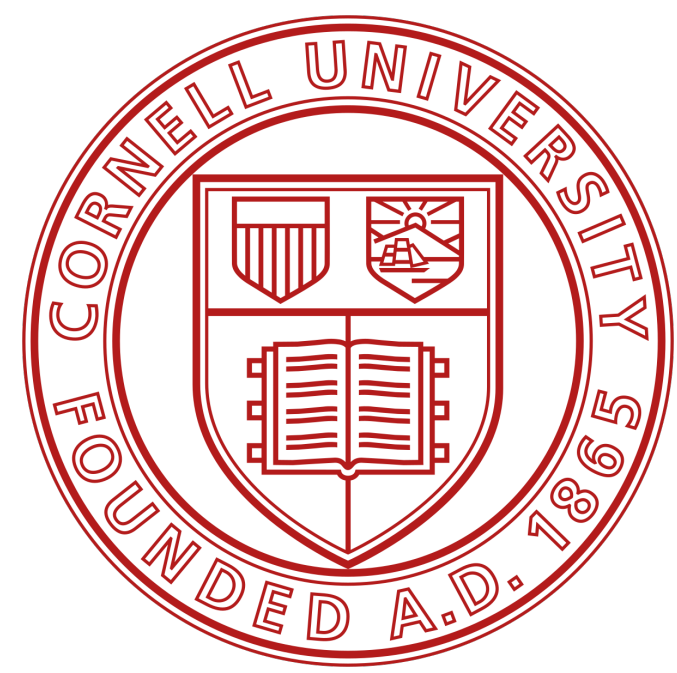


- Let's create a virtual die
- The `:` operator returns its results as a vector, a one-dimensional set of numbers.

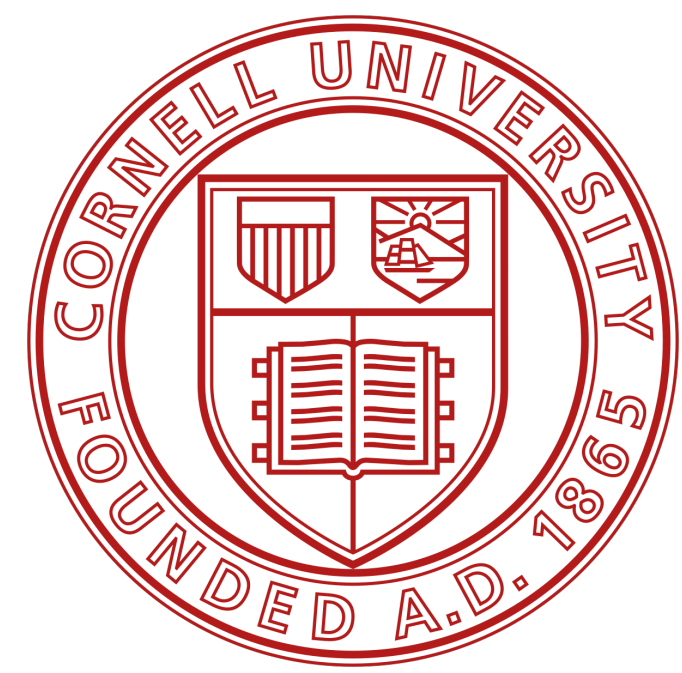


Basics

Vector



```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1:6
[1] 1 2 3 4 5 6
> |
```

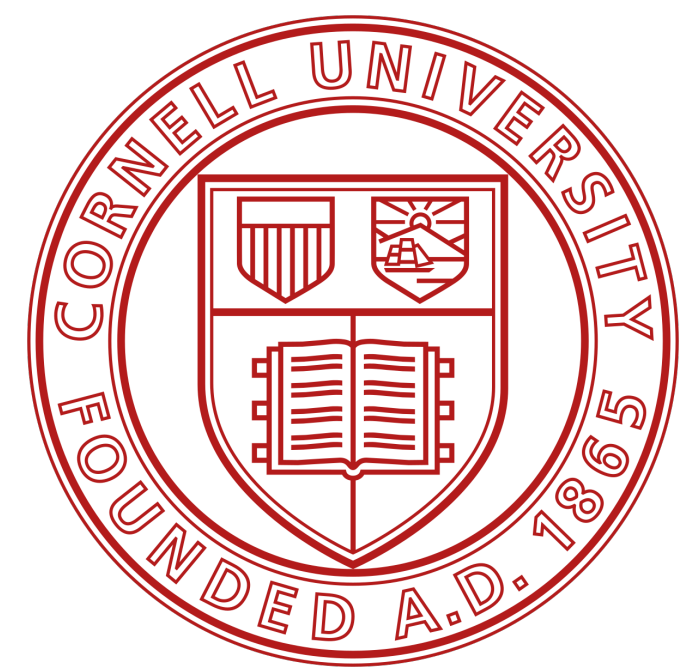


Basics

Vector

- Running `1:6` generated a vector of numbers for you to see, but it didn't save that vector anywhere in your computer's memory.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1:6
[1] 1 2 3 4 5 6
> |
```

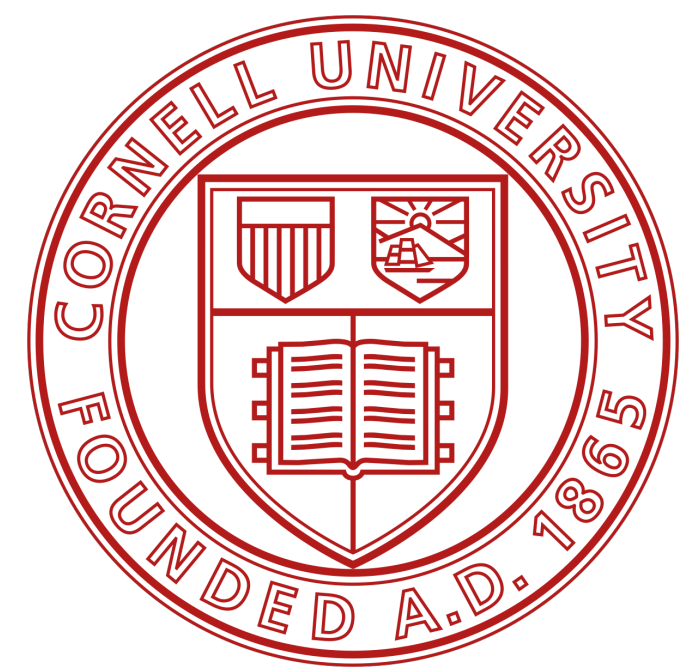


Basics

Vector

- Running `1:6` generated a vector of numbers for you to see, but it didn't save that vector anywhere in your computer's memory.
- If you want to use those numbers again, you'll have to ask your computer to save them somewhere.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1:6
[1] 1 2 3 4 5 6
> |
```



Basics

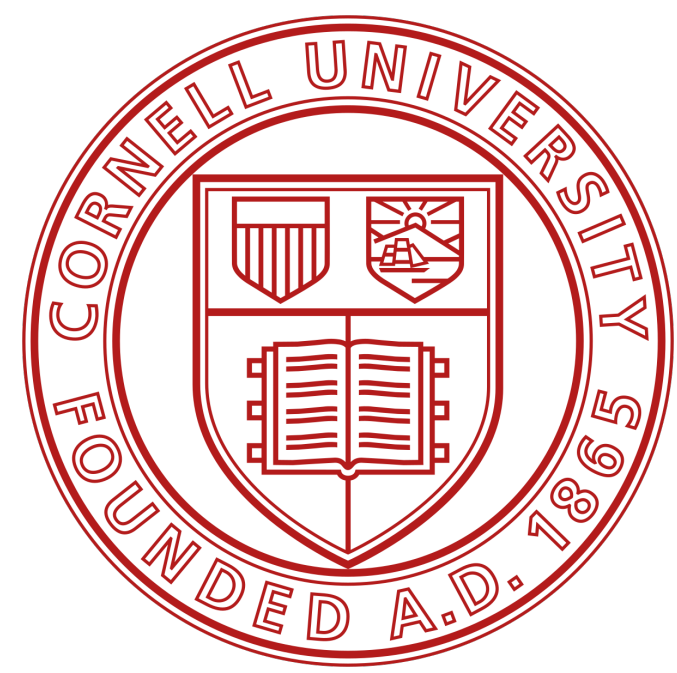
Vector

- Running `1:6` generated a vector of numbers for you to see, but it didn't save that vector anywhere in your computer's memory.
- If you want to use those numbers again, you'll have to ask your computer to save them somewhere.
- You can do that by creating an *R object*.

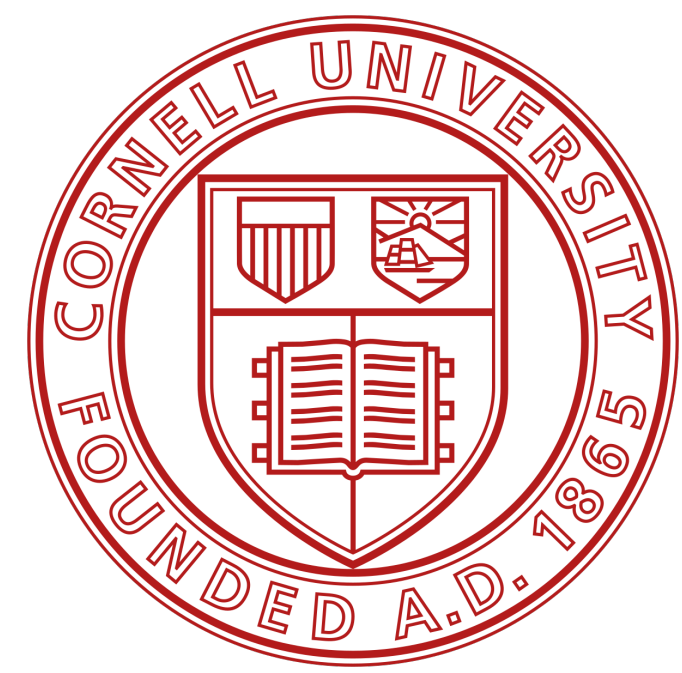
```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1:6
[1] 1 2 3 4 5 6
> |
```

Basics

Storing data



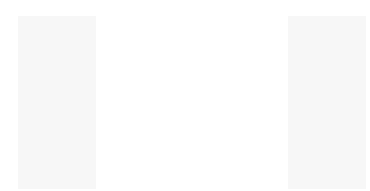
```
Console Terminal x
R 4.4.1 · ~/ ↵
> a <- 1
> a
[1] 1
> a +2
[1] 3
> |
```



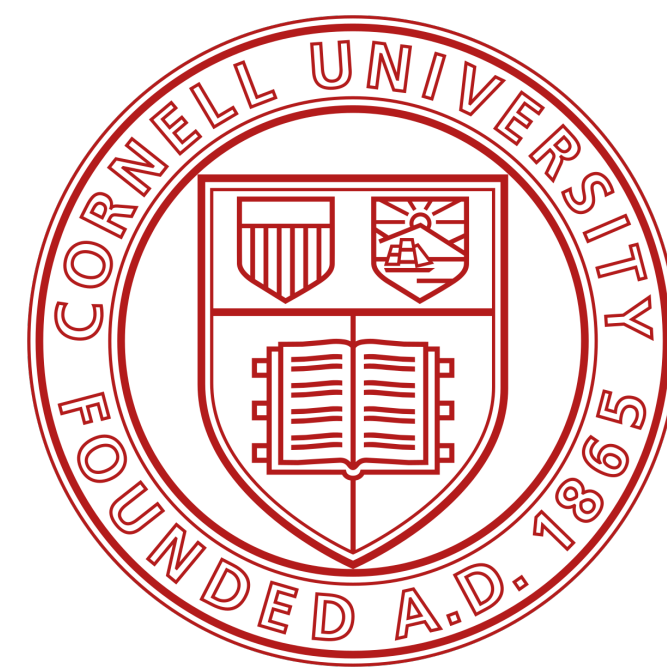
Basics

Storing data

- R lets you save data by storing it inside an R object.



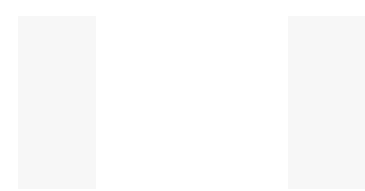
```
Console Terminal x
R 4.4.1 · ~/ ↵
> a <- 1
> a
[1] 1
> a +2
[1] 3
> |
```



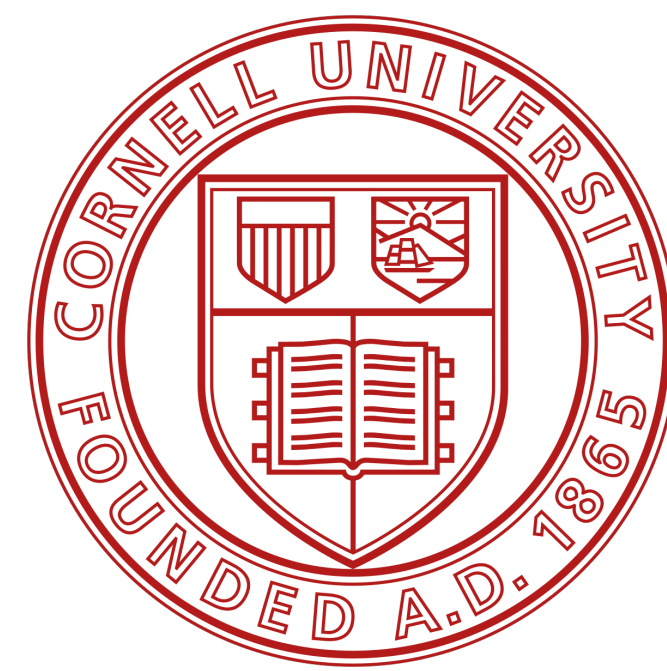
Basics

Storing data

- R lets you save data by storing it inside an R object.
- What is an object? Just a name that you can use to call up stored data.



```
Console Terminal x
R 4.4.1 · ~/ ↵
> a <- 1
> a
[1] 1
> a +2
[1] 3
> |
```

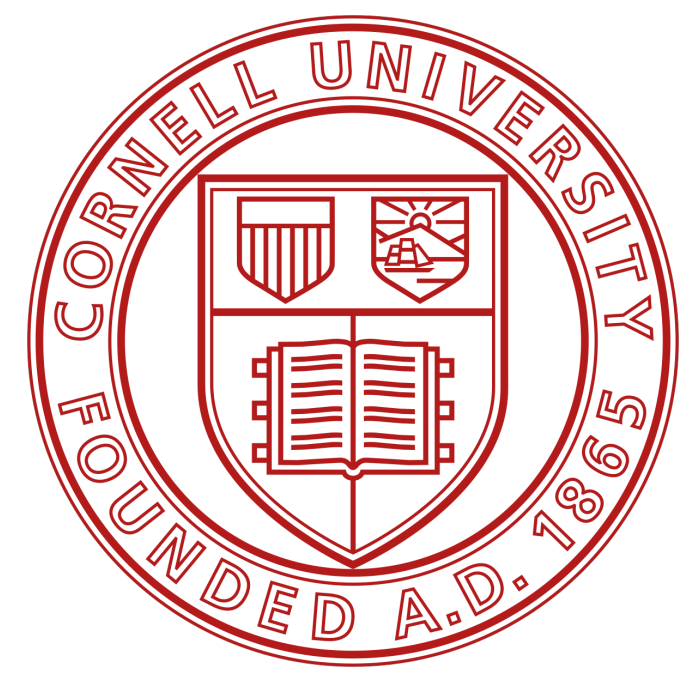


Basics

Storing data

- R lets you save data by storing it inside an R object.
- What is an object? Just a name that you can use to call up stored data.
- For example, you can save data into an object like `a` or `b`. Wherever R encounters the object, it will replace it with the data saved inside.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> a <- 1
> a
[1] 1
> a + 2
[1] 3
> |
```

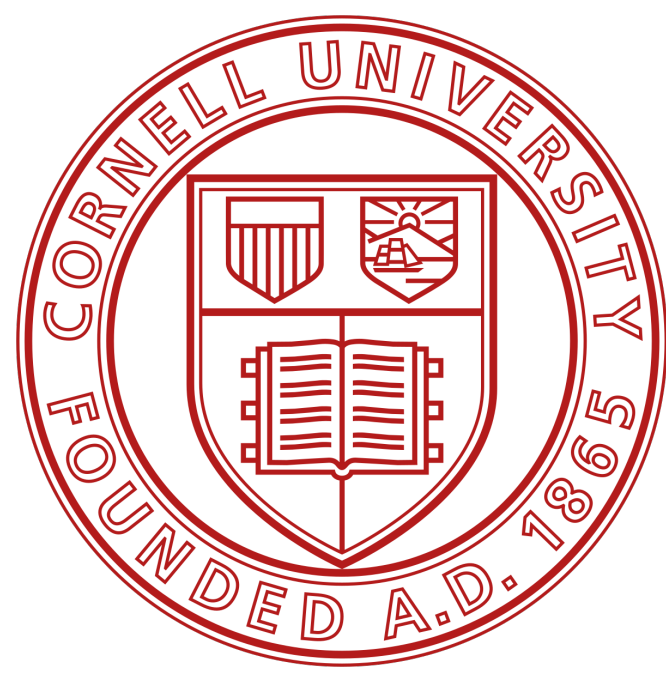



Running R code

Create an object

- Create an object named `die` that contains the numbers one through six.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> die <- 1:6
> die
[1] 1 2 3 4 5 6
> |
```



Running R code

Environment pane

Environment History Connections

Import Dataset 252 MiB

R Global Environment

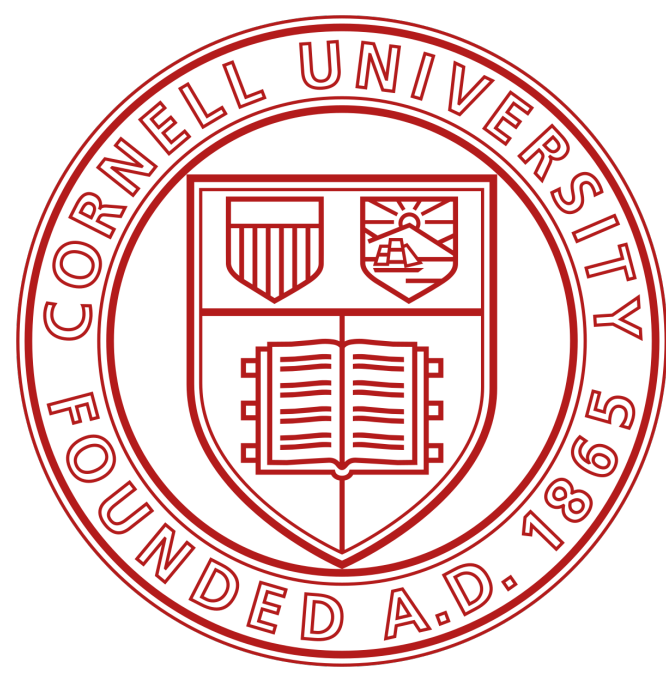
Values

die	int [1:6] 1 2 3 4 5 6
-----	-----------------------

Console Terminal

R 4.4.1 · ~/

```
> ls()
[1] "die"          "my_number"
>
```



Running R code

Environment pane

- When you create an object, the object will appear in the environment pane of RStudio.

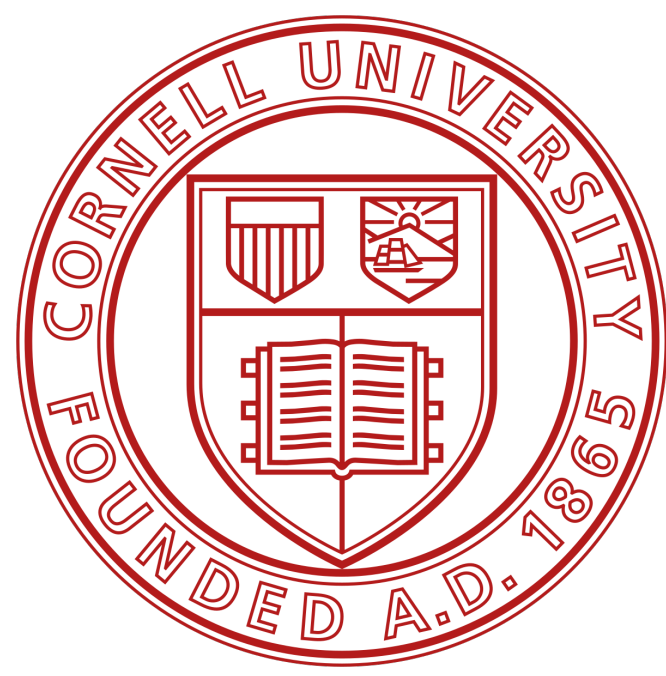
A screenshot of the RStudio Environment pane. The pane has three tabs: "Environment", "History", and "Connections". Below the tabs, there are icons for file operations (folder, save, import) and a memory usage indicator showing "252 MiB". The main area of the pane is titled "Values" and contains a table with one row and two columns. The first column is labeled "die" and the second column is labeled "int [1:6]" and contains the values "1 2 3 4 5 6".

die	int [1:6]
	1 2 3 4 5 6

A screenshot of the RStudio Console and Terminal panes. The "Console" tab is active, showing the R prompt and the output of the "ls()" command. The "Terminal" tab is also visible but inactive.

```
R 4.4.1 · ~/
```

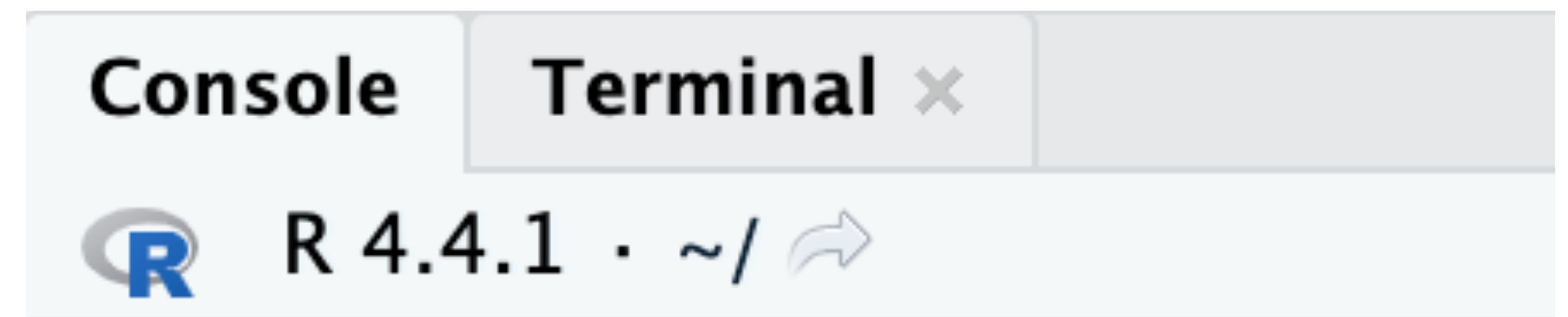
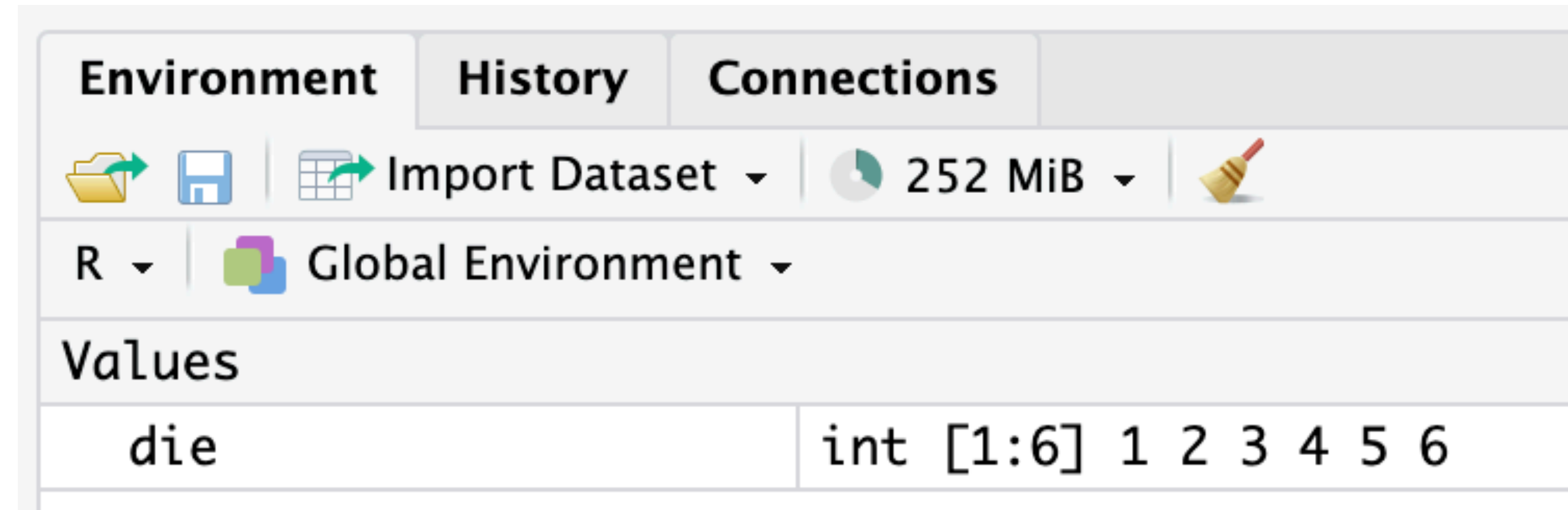
```
> ls()
[1] "die"          "my_number"
>
```



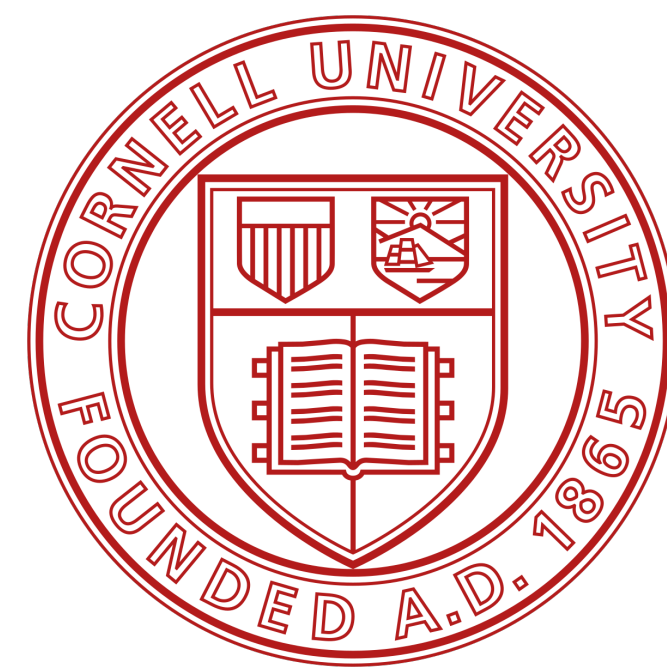
Running R code

Environment pane

- When you create an object, the object will appear in the environment pane of RStudio.
- This pane will show you all of the objects you've created since opening RStudio.



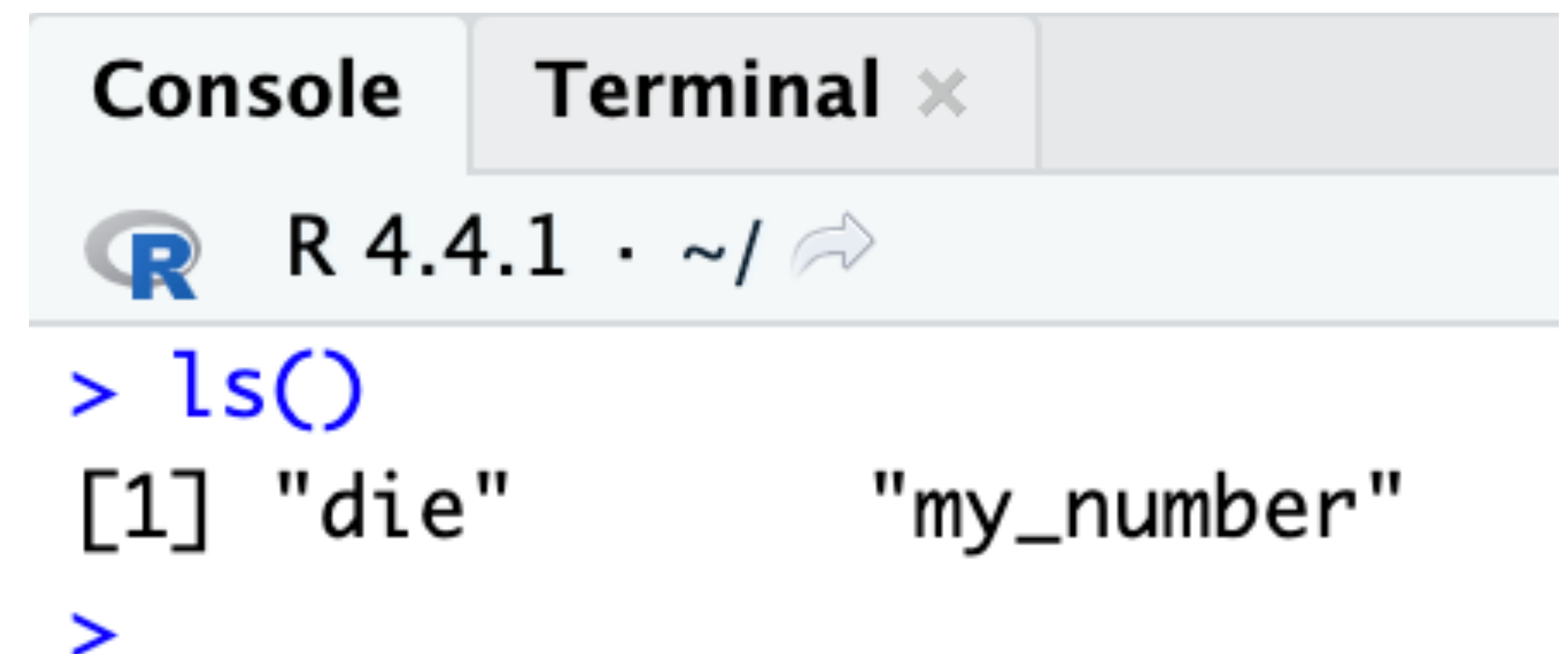
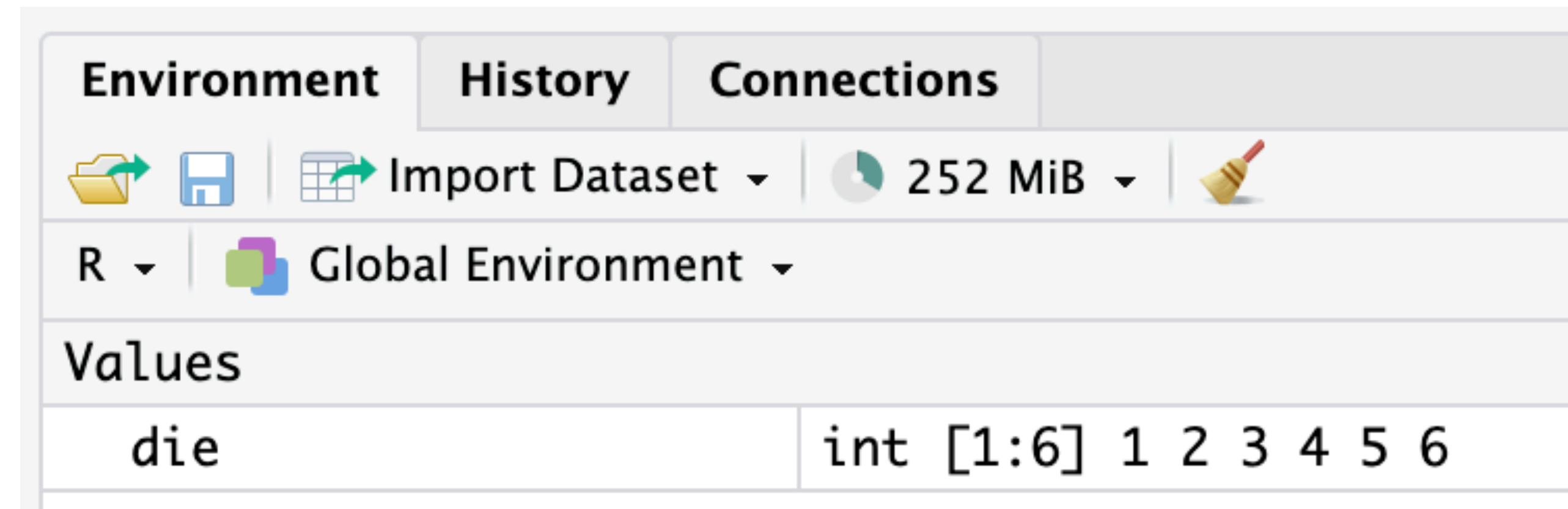
```
> ls()
[1] "die"      "my_number"
>
```

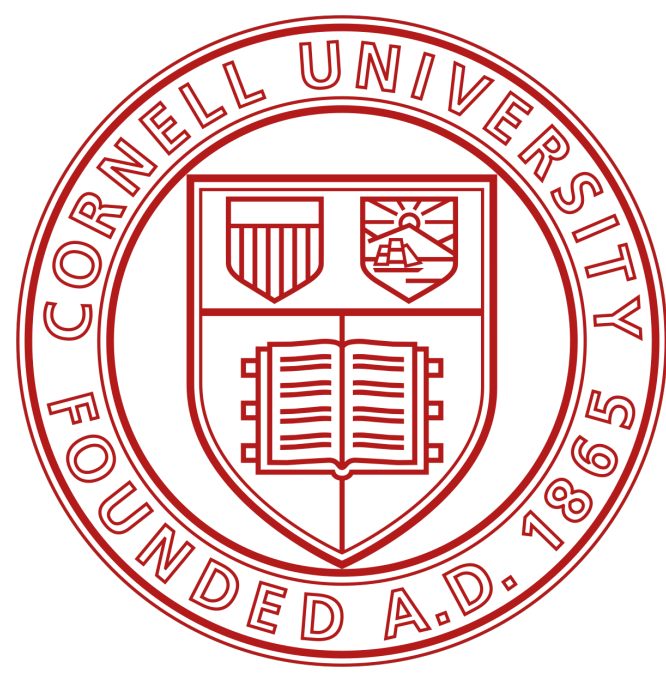


Running R code

Environment pane

- When you create an object, the object will appear in the environment pane of RStudio.
- This pane will show you all of the objects you've created since opening RStudio.
- You can see which object names you have already used with the function `ls`

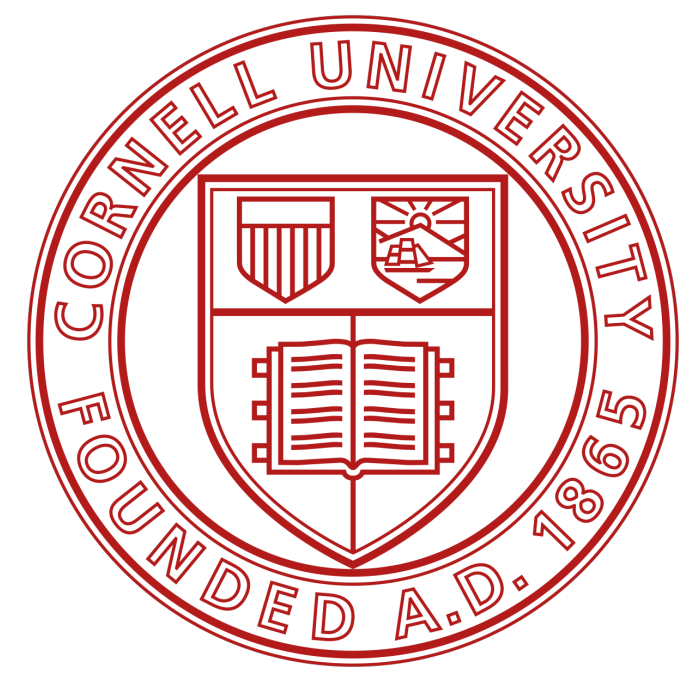




Running R code

Name an object

Good names	Names that cause errors
a	1trial
b	\$
FOO	^mean
my_var	2nd
.day	!bad



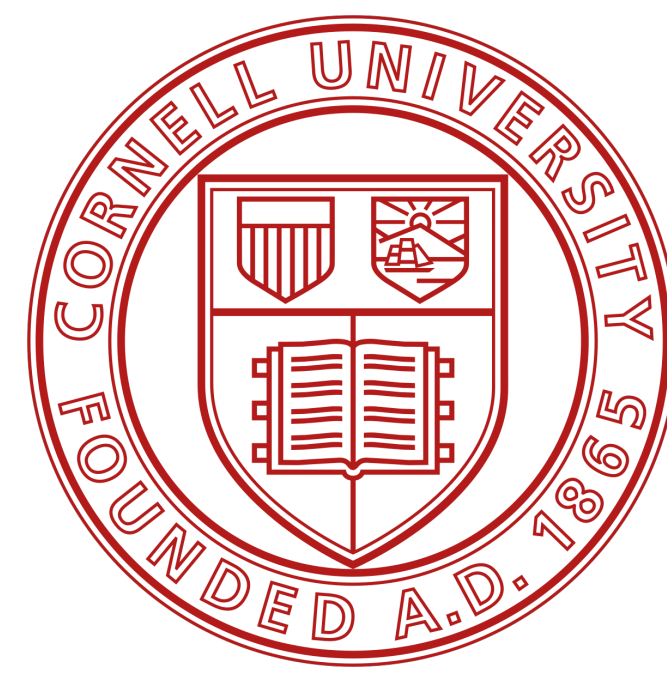
Running R code

Name an object

- You can name an object in R almost anything you want, but there are a few rules.



Good names	Names that cause errors
a	1trial
b	\$
FOO	^mean
my_var	2nd
.day	!bad



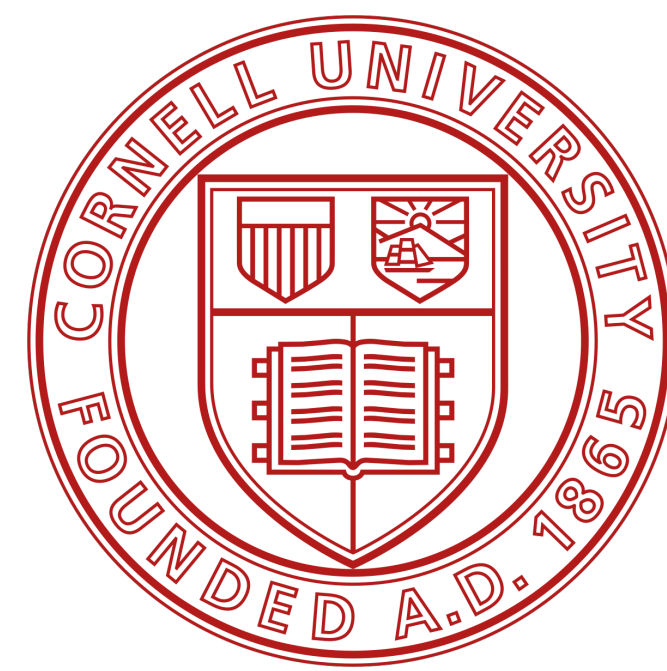
Running R code

Name an object

- You can name an object in R almost anything you want, but there are a few rules.
- First, a name cannot start with a number.



Good names	Names that cause errors
a	1trial
b	\$
FOO	^mean
my_var	2nd
.day	!bad

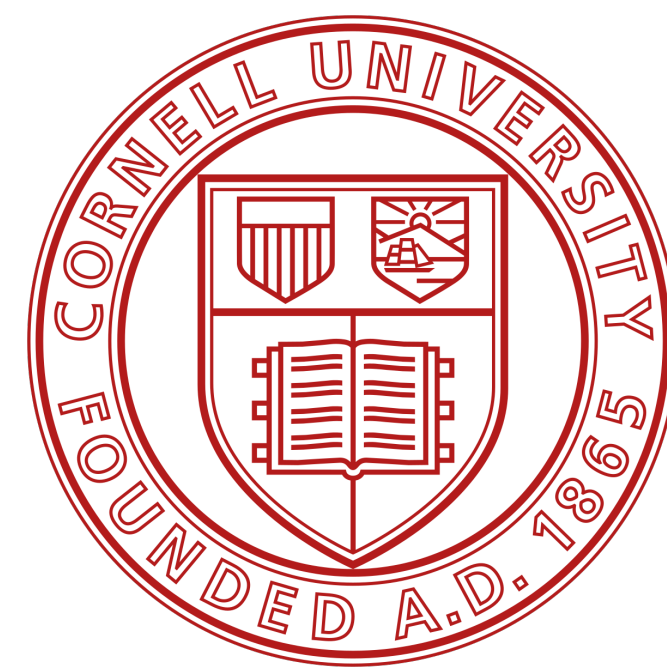


Running R code

Name an object

- You can name an object in R almost anything you want, but there are a few rules.
- First, a name cannot start with a number.
- Second, a name cannot use some special symbols, like `^`, `!`, `$`, `@`, `+`, `-`, `/`, or `*`

Good names	Names that cause errors
a	1trial
b	\$
FOO	^mean
my_var	2nd
.day	!bad

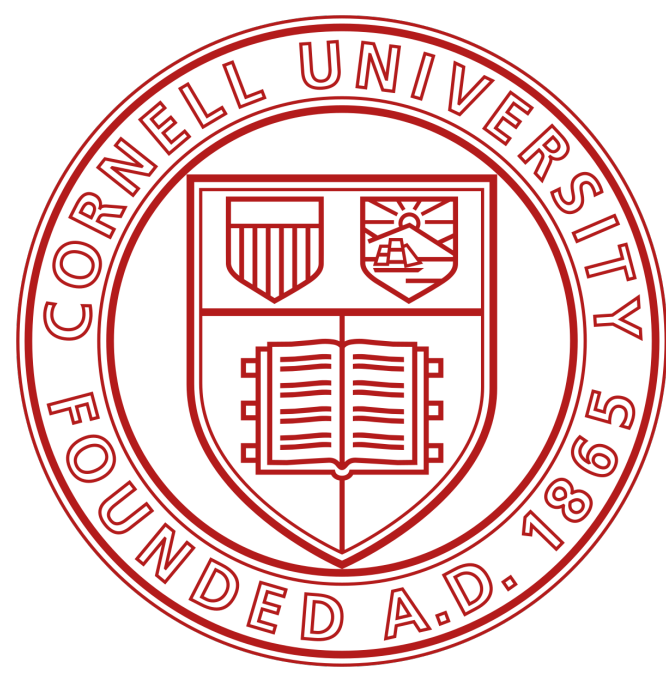


Running R code

Name an object

- You can name an object in R almost anything you want, but there are a few rules.
- First, a name cannot start with a number.
- Second, a name cannot use some special symbols, like `^`, `!`, `$`, `@`, `+`, `-`, `/`, or `*`
- R is case-sensitive, so `name` and `Name` will refer to different objects

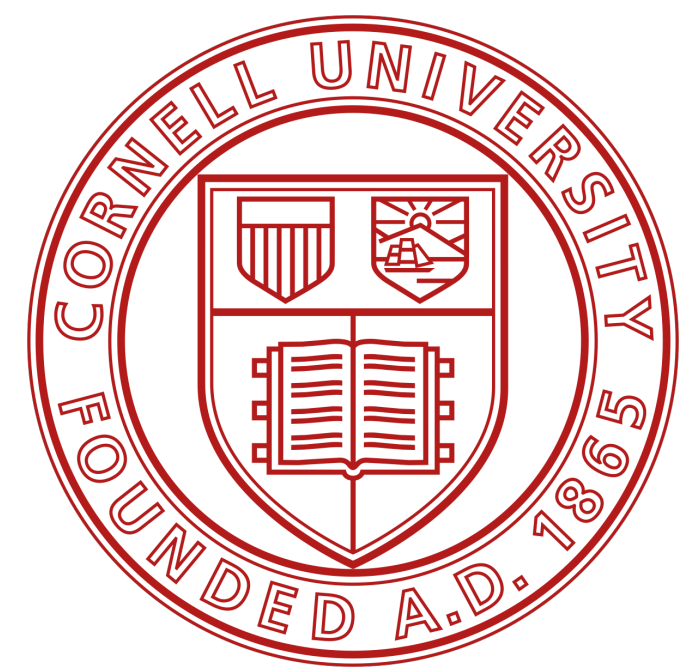
Good names	Names that cause errors
a	1trial
b	\$
FOO	^mean
my_var	2nd
.day	!bad



Running R code

Name overwriting

```
Console Terminal x
R 4.4.1 · ~/ ↵
> my_number <- 1
> my_number
[1] 1
> my_number <- 999
> my_number
[1] 999
> |
```

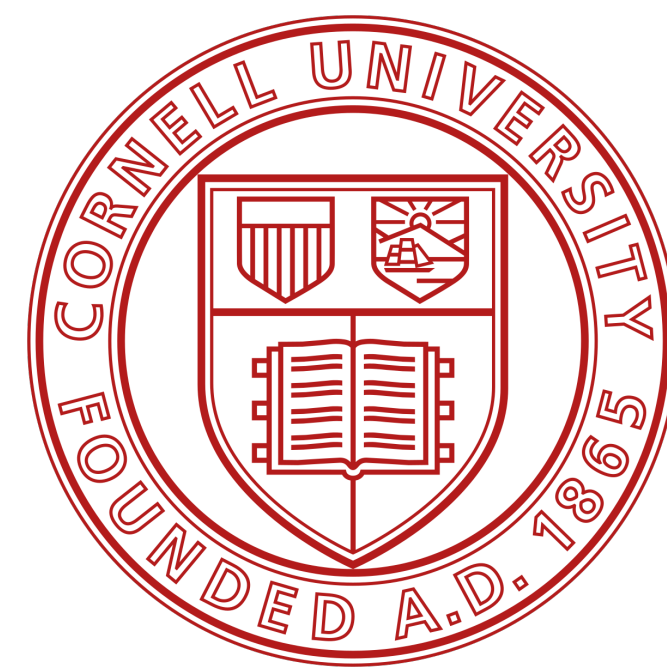


Running R code

Name overwriting

- R will overwrite any previous information stored in an object without asking you for permission.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> my_number <- 1
> my_number
[1] 1
> my_number <- 999
> my_number
[1] 999
> |
```

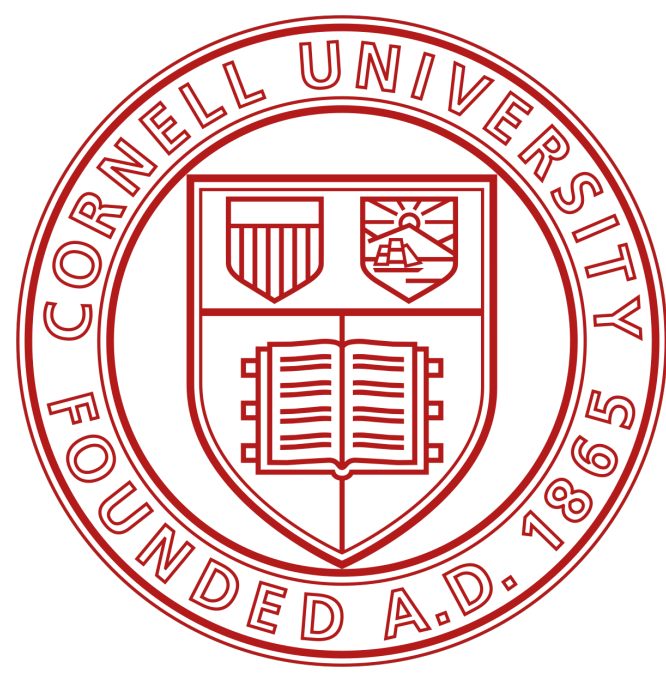


Running R code

Name overwriting

- R will overwrite any previous information stored in an object without asking you for permission.
- It is a good idea to *not* use names that are already taken.

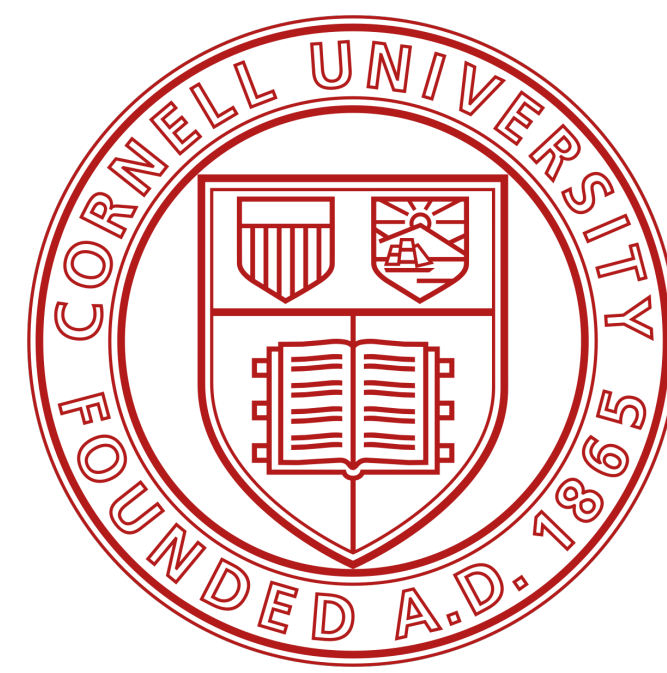
```
Console Terminal x
R 4.4.1 · ~/ ↵
> my_number <- 1
> my_number
[1] 1
> my_number <- 999
> my_number
[1] 999
> |
```



Running R code

Element-wise execution

```
Console Terminal x  
R 4.4.1 · ~/ ↵  
> die  
[1] 1 2 3 4 5 6  
> die - 1  
[1] 0 1 2 3 4 5  
> die / 2  
[1] 0.5 1.0 1.5 2.0 2.5 3.0  
> die * die  
[1] 1 4 9 16 25 36  
>
```

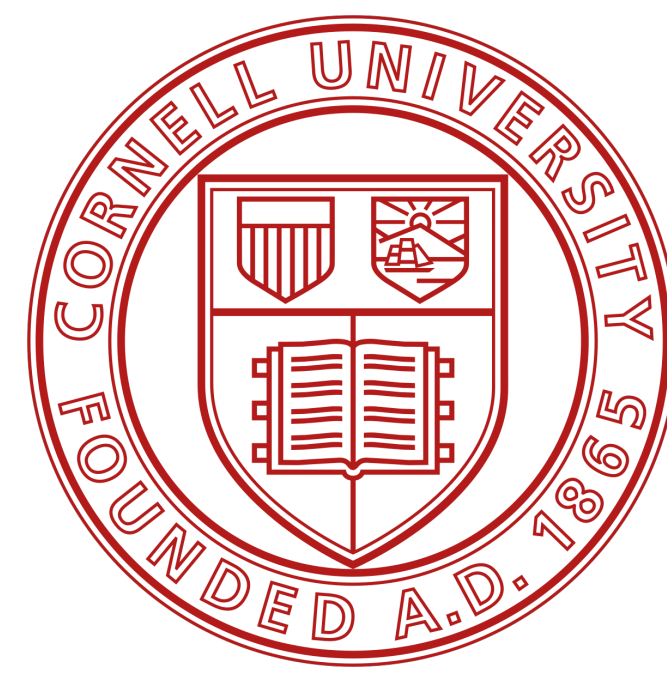


Running R code

Element-wise execution

- You now have a virtual die that is stored in your computer's memory.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> die
[1] 1 2 3 4 5 6
> die - 1
[1] 0 1 2 3 4 5
> die / 2
[1] 0.5 1.0 1.5 2.0 2.5 3.0
> die * die
[1] 1 4 9 16 25 36
>
```

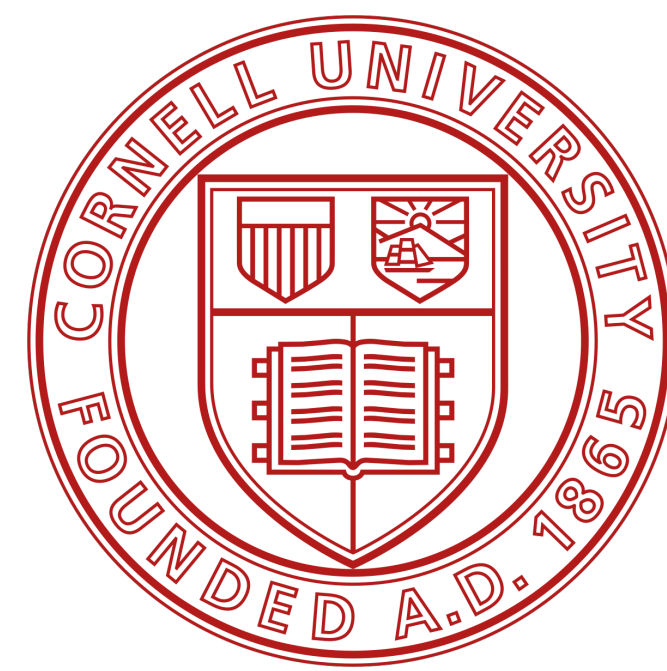


Running R code

Element-wise execution

- You now have a virtual die that is stored in your computer's memory.
- You can do all sorts of math with the die.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> die
[1] 1 2 3 4 5 6
> die - 1
[1] 0 1 2 3 4 5
> die / 2
[1] 0.5 1.0 1.5 2.0 2.5 3.0
> die * die
[1] 1 4 9 16 25 36
>
```

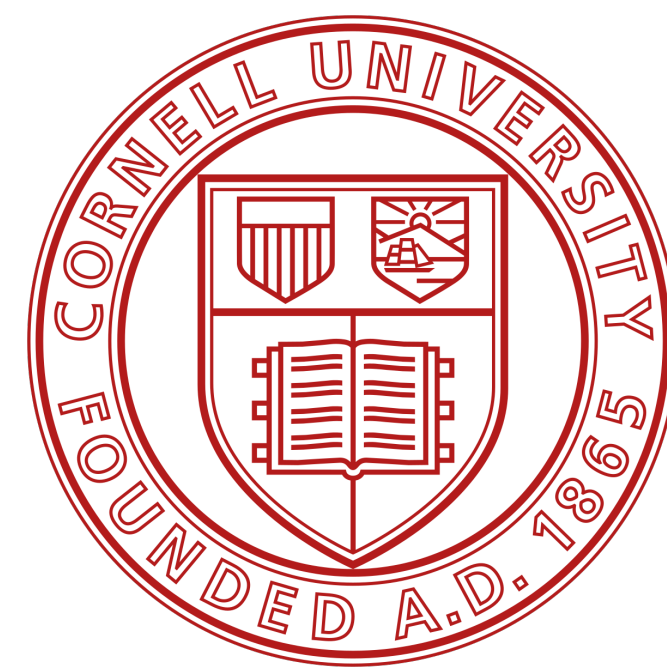



Running R code

Element-wise execution

- You now have a virtual die that is stored in your computer's memory.
- You can do all sorts of math with the die.
- R does not always follow the rules of matrix multiplication.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> die
[1] 1 2 3 4 5 6
> die - 1
[1] 0 1 2 3 4 5
> die / 2
[1] 0.5 1.0 1.5 2.0 2.5 3.0
> die * die
[1] 1 4 9 16 25 36
>
```

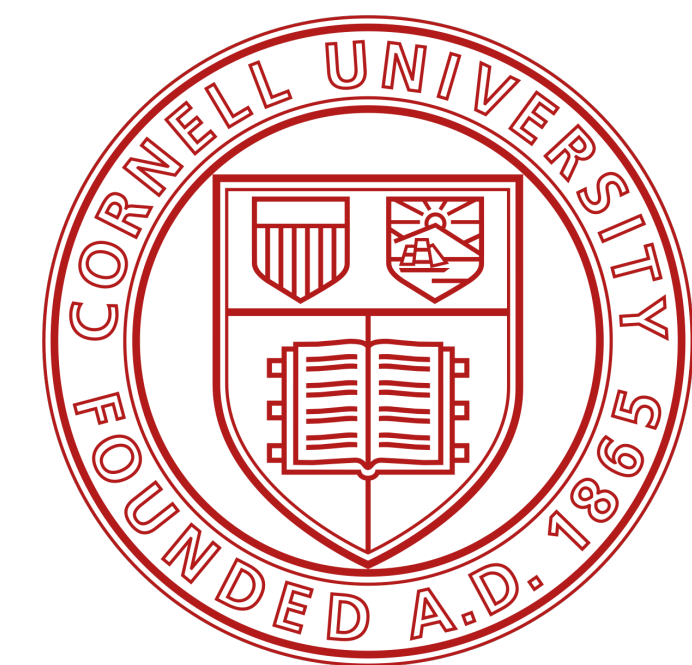


Running R code

Element-wise execution

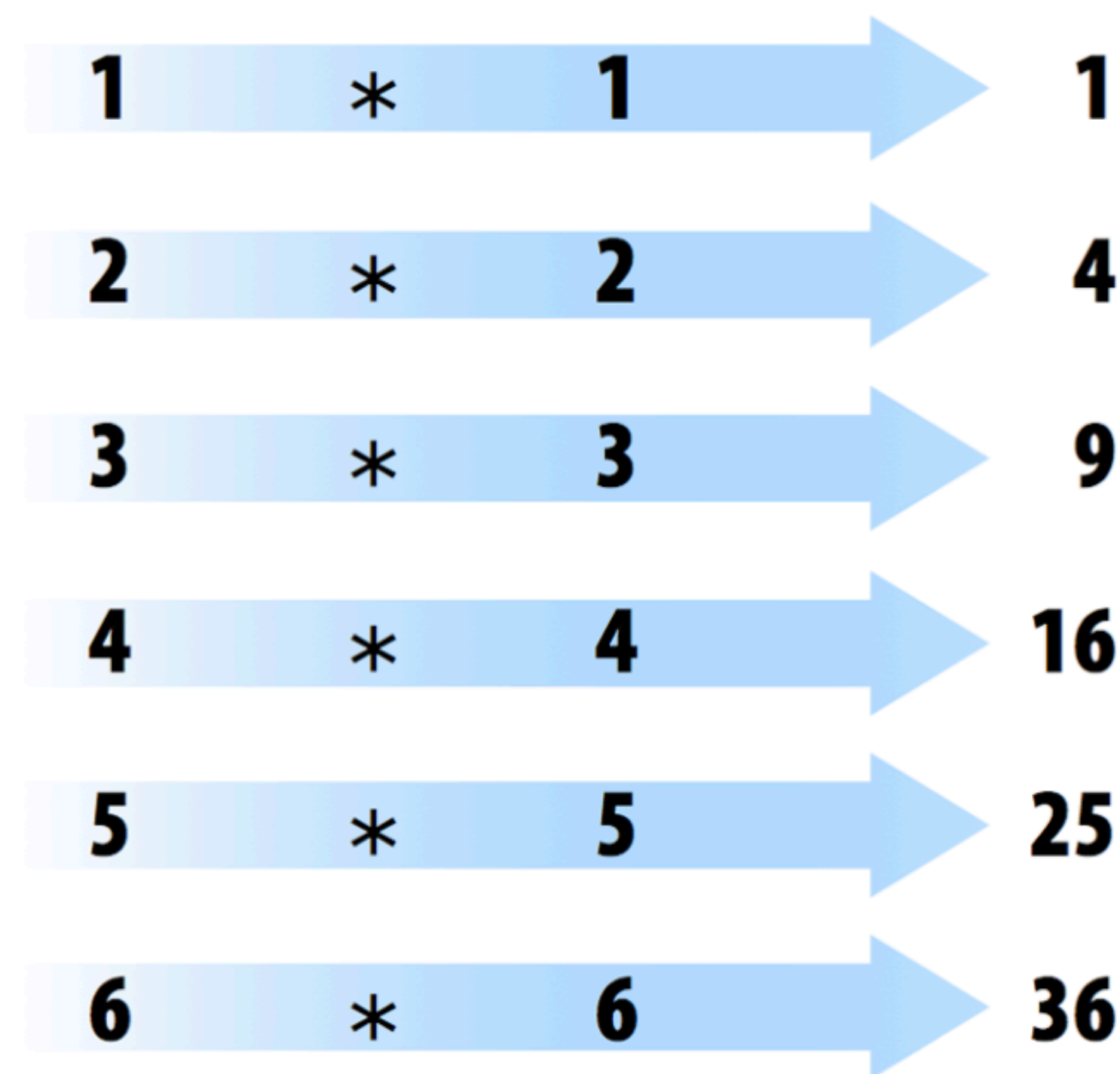
- You now have a virtual die that is stored in your computer's memory.
- You can do all sorts of math with the die.
- R does not always follow the rules of matrix multiplication.
- R uses element-wise execution.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> die
[1] 1 2 3 4 5 6
> die - 1
[1] 0 1 2 3 4 5
> die / 2
[1] 0.5 1.0 1.5 2.0 2.5 3.0
> die * die
[1] 1 4 9 16 25 36
>
```

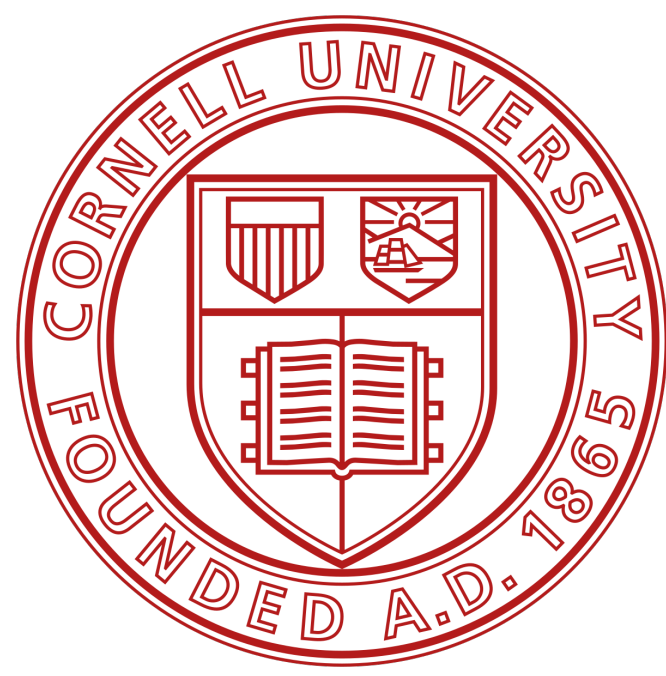


Running R code

Element-wise execution



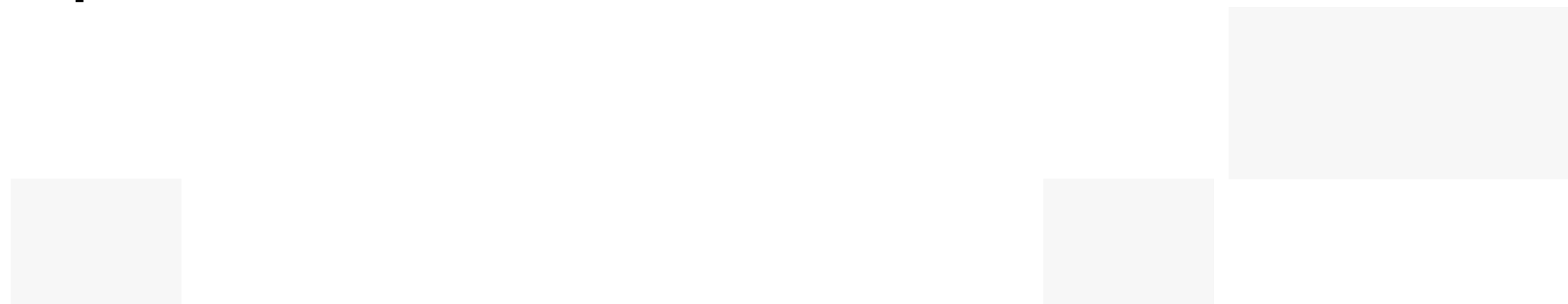
die * die =

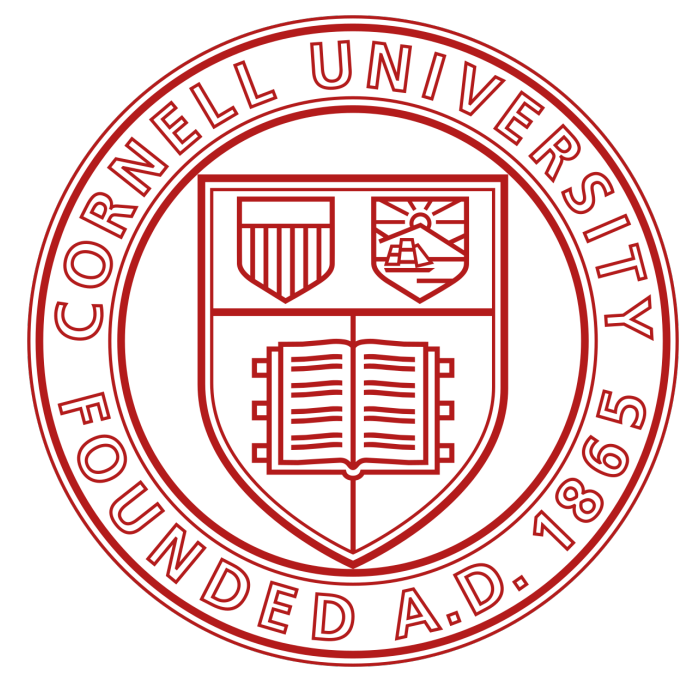


Running R code

Element-wise execution

- When you use two or more vectors in an operation, R will line up the vectors and perform a sequence of individual operations.

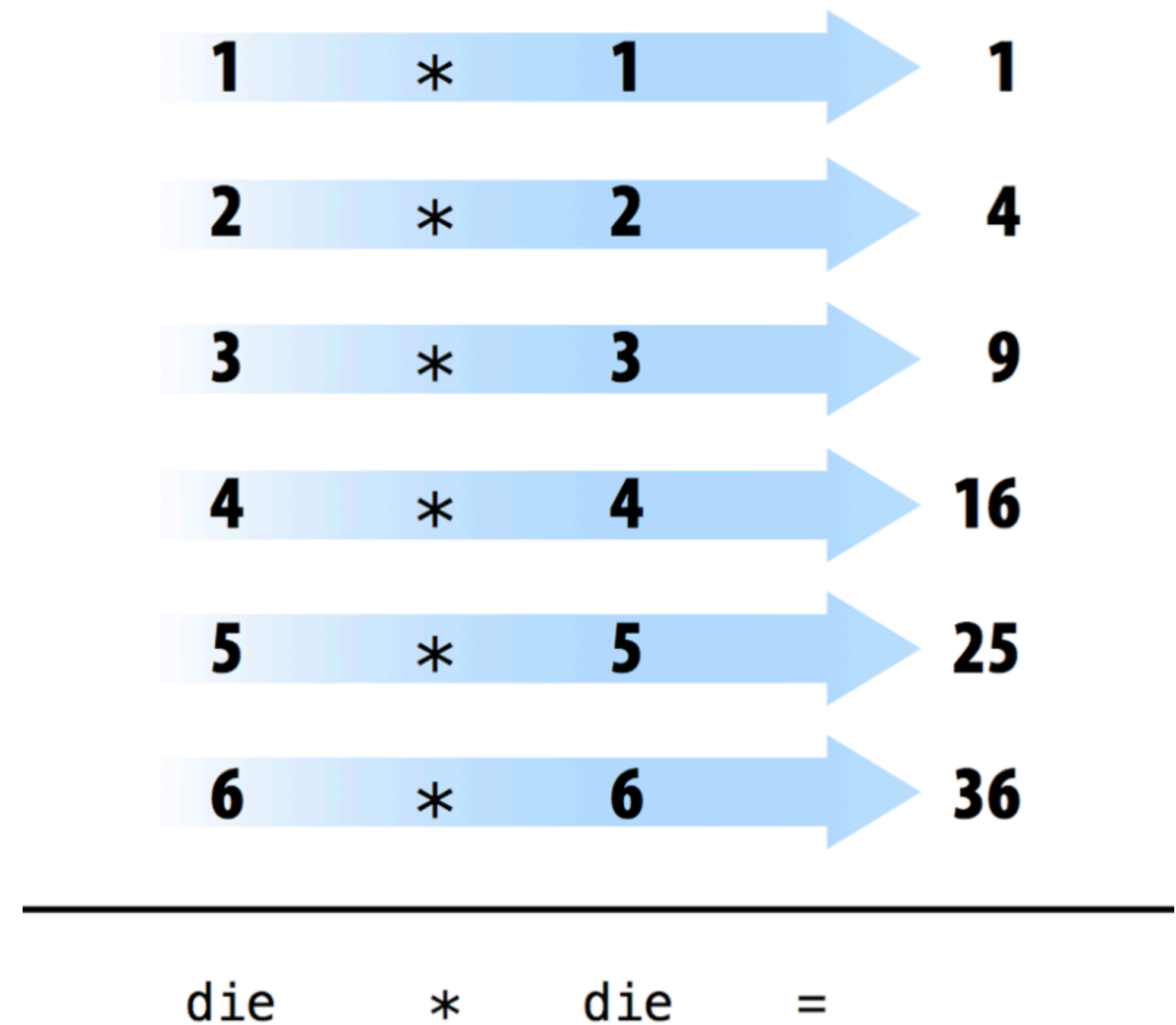


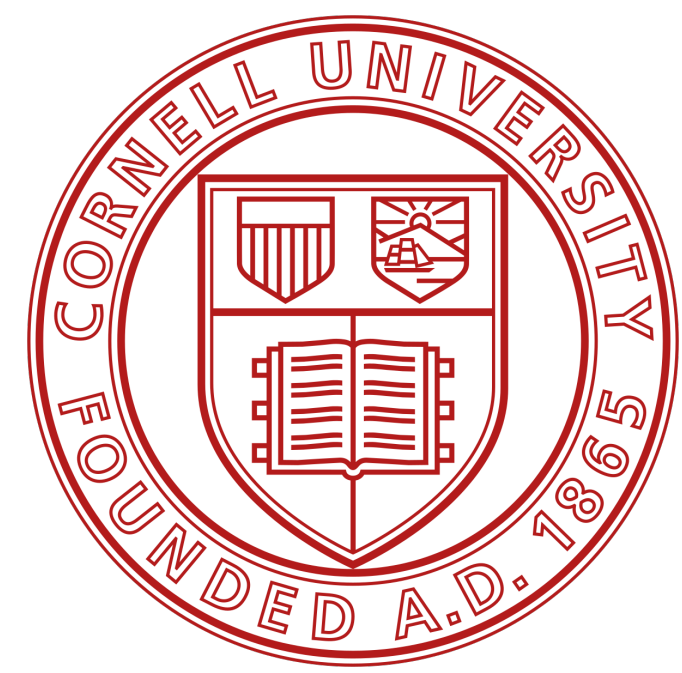


Running R code

Element-wise execution

- When you use two or more vectors in an operation, R will line up the vectors and perform a sequence of individual operations.
- For example, when you run `die * die`, R lines up the two `die` vectors and then multiplies the first element of vector 1 by the first element of vector 2.



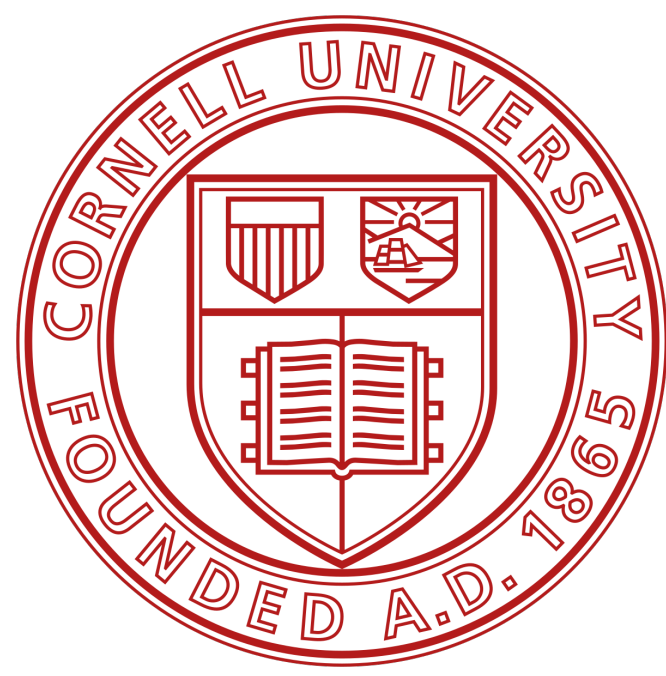


Running R code

Element-wise execution

- When you use two or more vectors in an operation, R will line up the vectors and perform a sequence of individual operations.
- For example, when you run `die * die`, R lines up the two `die` vectors and then multiplies the first element of vector 1 by the first element of vector 2.
- R then multiplies the second element of vector 1 by the second element of vector 2, and so on, until every element has been multiplied.

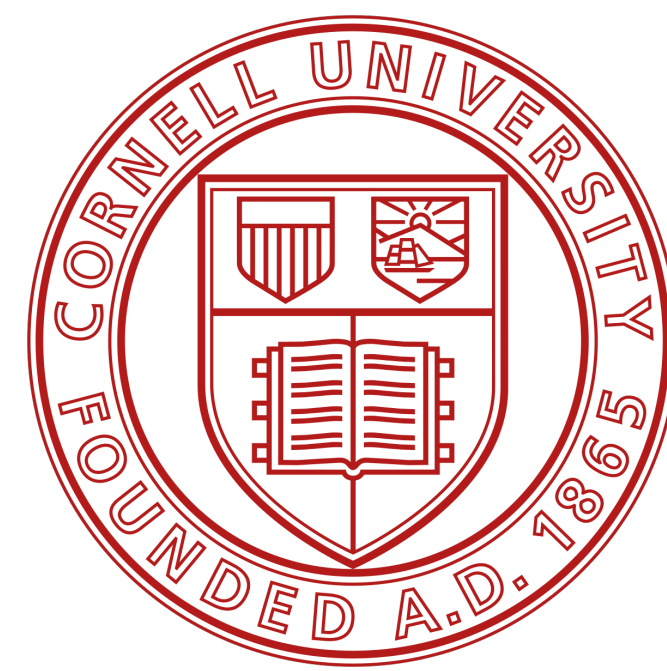




Running R code

Vector recycling

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1:2
[1] 1 2
> 1:4
[1] 1 2 3 4
> die
[1] 1 2 3 4 5 6
> die + 1:2
[1] 2 4 4 6 6 8
> die + 1:4
[1] 2 4 6 8 6 8
Warning message:
In die + 1:4 :
  longer object length is not a multiple of shorter object length
```

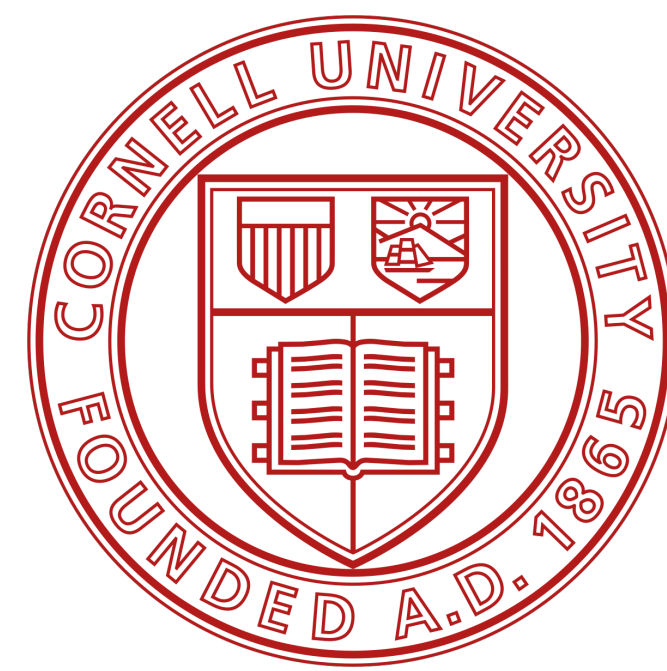


Running R code

Vector recycling

- If you give R two vectors of unequal lengths, R will repeat the shorter vector until it is as long as the longer vector, and then do the math (*vector recycling*)

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1:2
[1] 1 2
> 1:4
[1] 1 2 3 4
> die
[1] 1 2 3 4 5 6
> die + 1:2
[1] 2 4 4 6 6 8
> die + 1:4
[1] 2 4 6 8 6 8
Warning message:
In die + 1:4 :
  longer object length is not a multiple of shorter object length
```

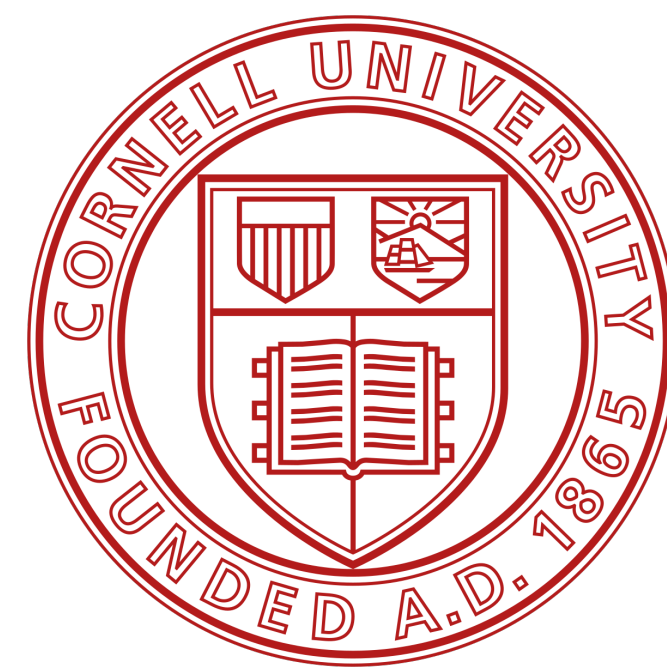



Running R code

Vector recycling

- If you give R two vectors of unequal lengths, R will repeat the shorter vector until it is as long as the longer vector, and then do the math (*vector recycling*)
- This isn't a permanent change, the shorter vector will be its original size after R does the math.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1:2
[1] 1 2
> 1:4
[1] 1 2 3 4
> die
[1] 1 2 3 4 5 6
> die + 1:2
[1] 2 4 4 6 6 8
> die + 1:4
[1] 2 4 6 8 6 8
Warning message:
In die + 1:4 :
  longer object length is not a multiple of shorter object length
```



Running R code

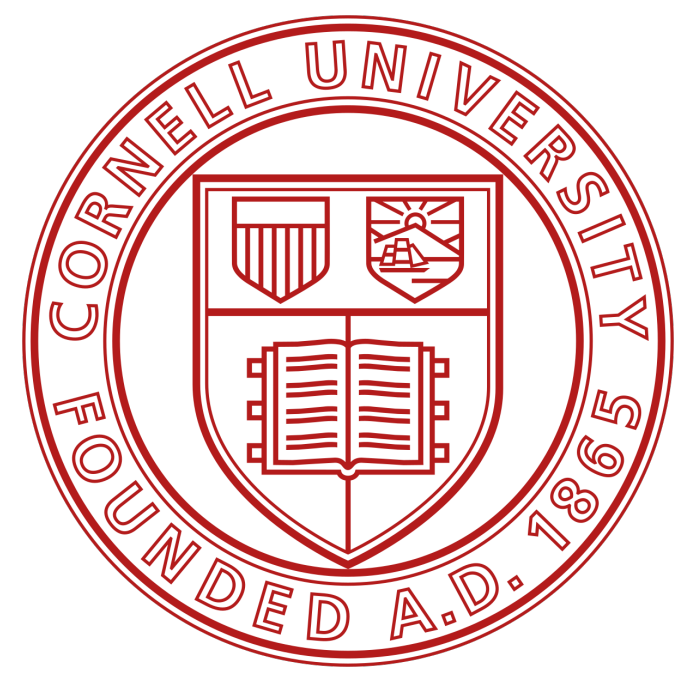
Vector recycling

- If you give R two vectors of unequal lengths, R will repeat the shorter vector until it is as long as the longer vector, and then do the math (*vector recycling*)
- This isn't a permanent change, the shorter vector will be its original size after R does the math.
- If the length of the short vector does not divide evenly into the length of the long vector, R will return a warning message.

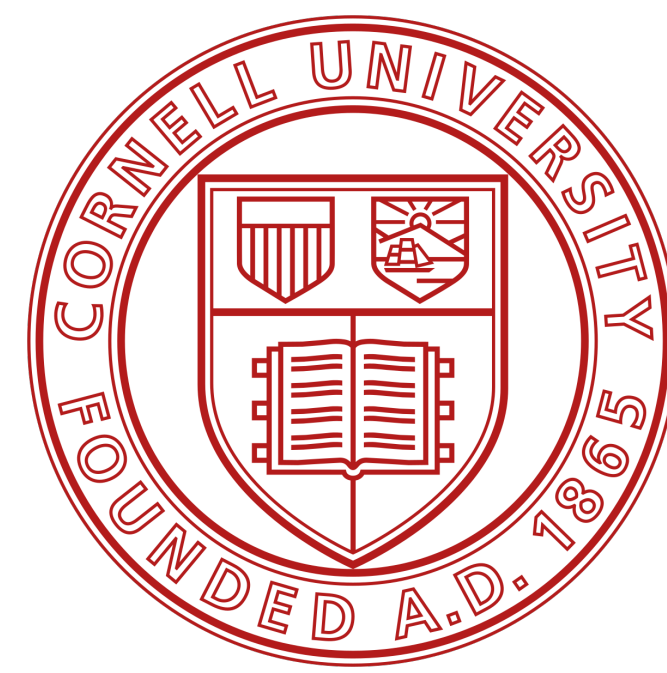
```
Console Terminal x
R 4.4.1 · ~/ ↵
> 1:2
[1] 1 2
> 1:4
[1] 1 2 3 4
> die
[1] 1 2 3 4 5 6
> die + 1:2
[1] 2 4 4 6 6 8
> die + 1:4
[1] 2 4 6 8 6 8
Warning message:
In die + 1:4 :
  longer object length is not a multiple of shorter object length
```

Running R code

Linear algebra operations



```
Console Terminal x
R 4.4.1 · ~/ ↵
> die %% die
      [,1]
[1,]  91
> die %% die
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    2    4    6    8   10   12
[3,]    3    6    9   12   15   18
[4,]    4    8   12   16   20   24
[5,]    5   10   15   20   25   30
[6,]    6   12   18   24   30   36
```

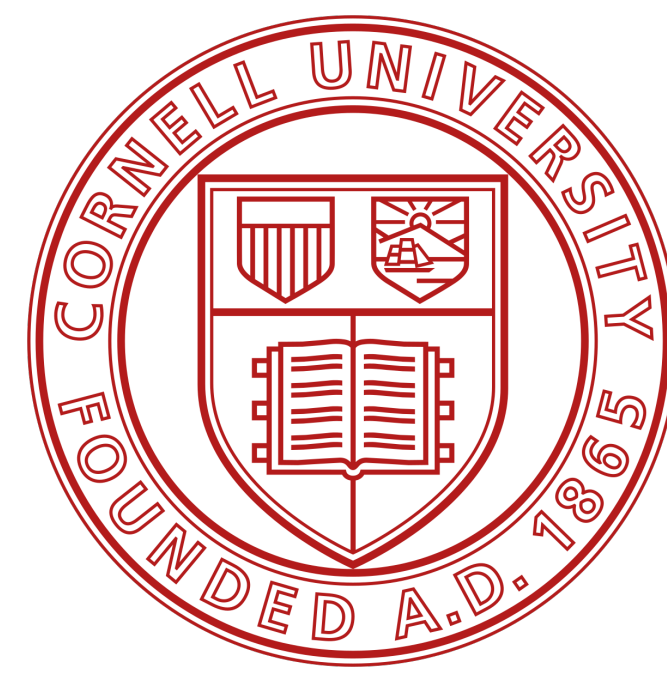


Running R code

Linear algebra operations

- Working with data sets, element-wise operations ensure that values from one observation are only paired with values from the same observation.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> die %*% die
      [,1]
[1,]   91
> die %o% die
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    2    4    6    8   10   12
[3,]    3    6    9   12   15   18
[4,]    4    8   12   16   20   24
[5,]    5   10   15   20   25   30
[6,]    6   12   18   24   30   36
```

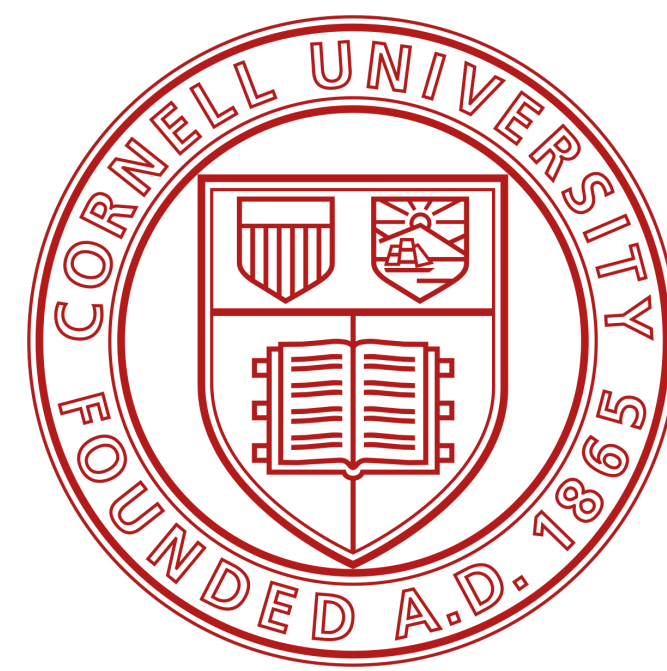


Running R code

Linear algebra operations

- Working with data sets, element-wise operations ensure that values from one observation are only paired with values from the same observation.
- Don't think that R has given up on traditional matrix multiplication.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> die %*% die
      [,1]
[1,]  91
> die %%% die
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  1    2    3    4    5    6
[2,]  2    4    6    8   10   12
[3,]  3    6    9   12   15   18
[4,]  4    8   12   16   20   24
[5,]  5   10   15   20   25   30
[6,]  6   12   18   24   30   36
```

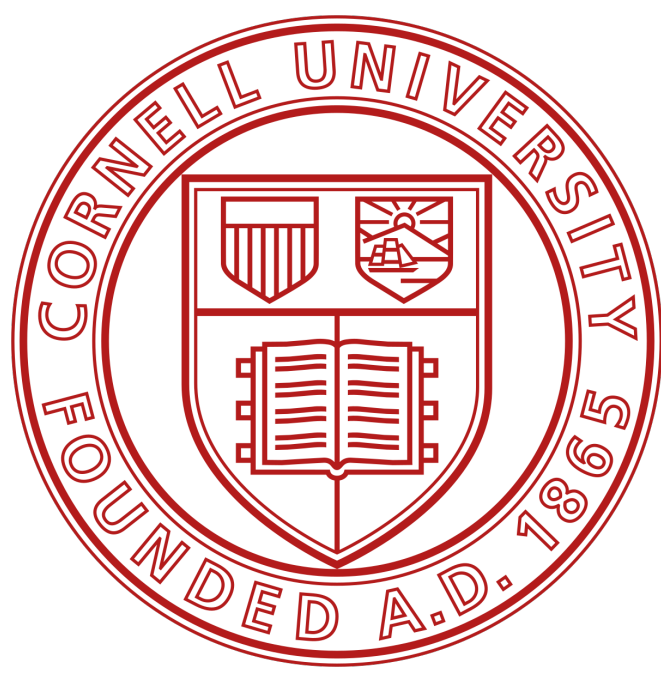


Running R code

Linear algebra operations

- Working with data sets, element-wise operations ensure that values from one observation are only paired with values from the same observation.
- Don't think that R has given up on traditional matrix multiplication.
- You just have to ask for it when you want it. You can do inner multiplication with the `%*%` operator and outer multiplication with the `%o%` operator.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> die %*% die
      [,1]
[1,]   91
> die %o% die
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    2    4    6    8   10   12
[3,]    3    6    9   12   15   18
[4,]    4    8   12   16   20   24
[5,]    5   10   15   20   25   30
[6,]    6   12   18   24   30   36
```

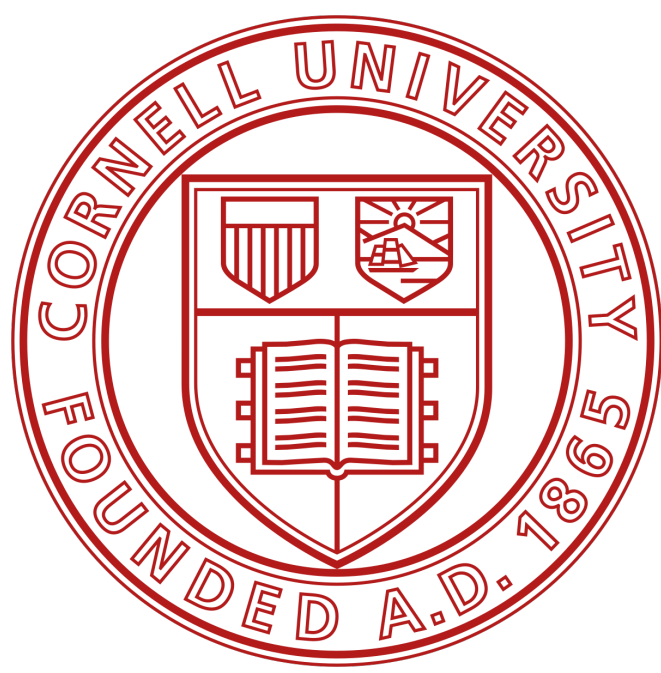


Running R code

Vector inner product

$$\left\langle \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\rangle = x^T y = \sum_{i=1}^n x_i y_i = x_1 y_1 + \cdots + x_n y_n$$

$$\langle \cdot, \cdot \rangle$$



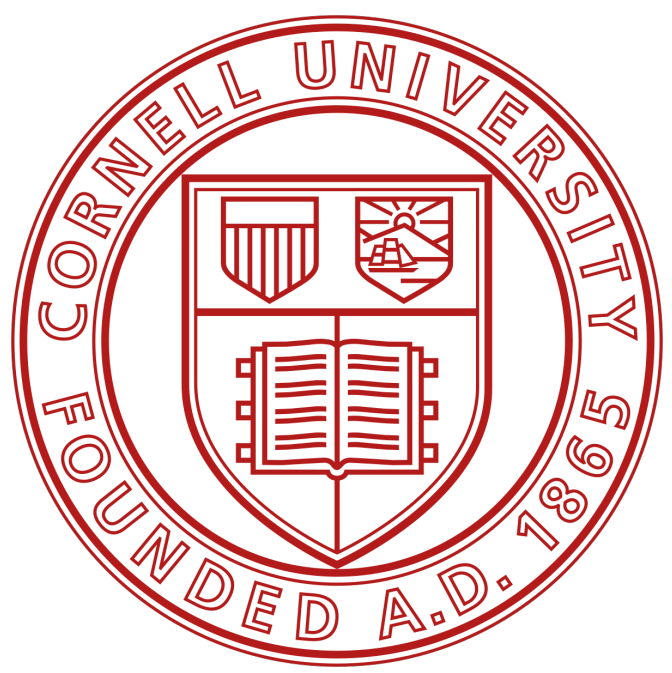
Running R code

Vector inner product

- A vector inner product is a binary operation that takes two vectors and returns a scalar.

$$\langle \cdot, \cdot \rangle$$

$$\left\langle \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\rangle = x^T y = \sum_{i=1}^n x_i y_i = x_1 y_1 + \cdots + x_n y_n$$

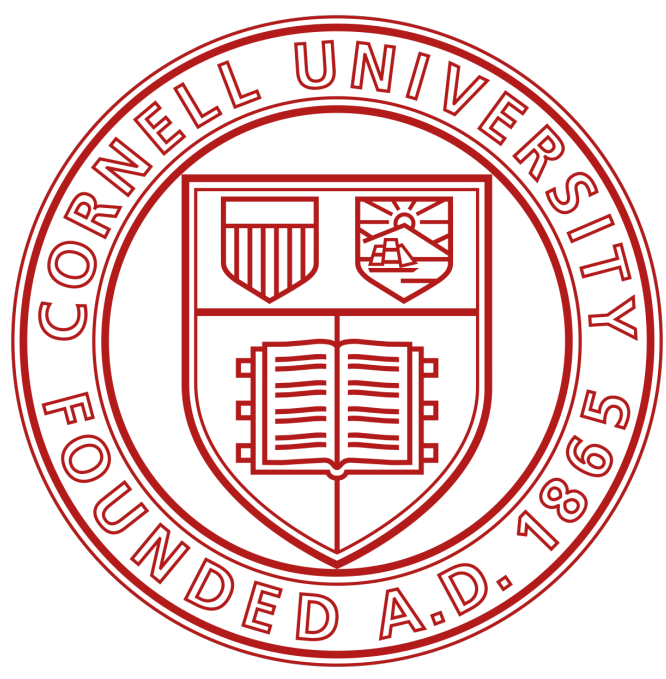


Running R code

Vector inner product

- A vector inner product is a binary operation that takes two vectors and returns a scalar.
- It is often denoted $\langle \cdot, \cdot \rangle$

$$\left\langle \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\rangle = x^T y = \sum_{i=1}^n x_i y_i = x_1 y_1 + \cdots + x_n y_n$$



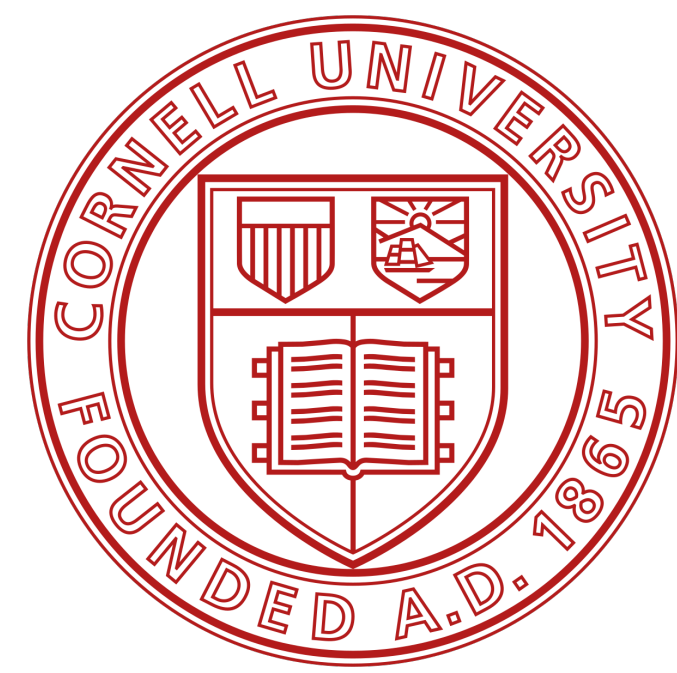
Running R code

Vector outer product

Given two vectors of size $m \times 1$ and $n \times 1$ respectively

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{A} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}$$



Running R code

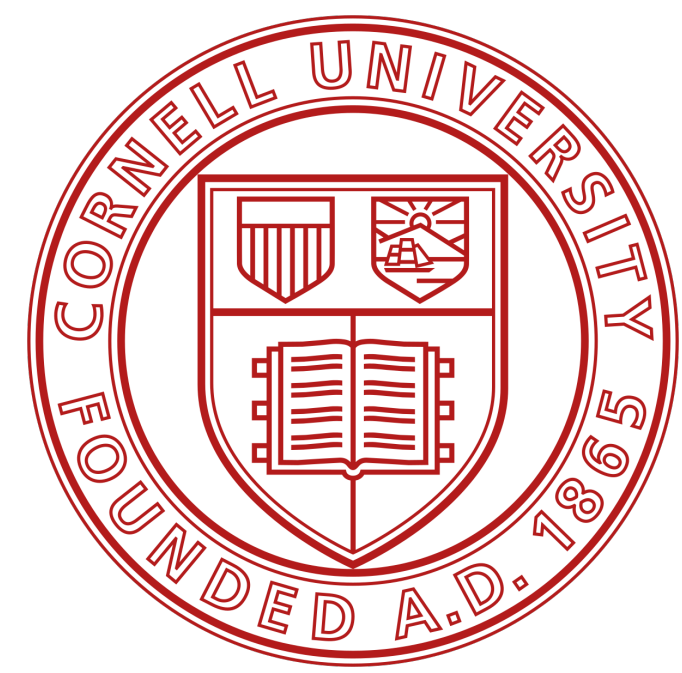
Vector outer product

- The outer product of two coordinate vectors is the matrix whose entries are all products of an element in the first vector with an element in the second vector.

Given two vectors of size $m \times 1$ and $n \times 1$ respectively

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{A} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}$$



Running R code

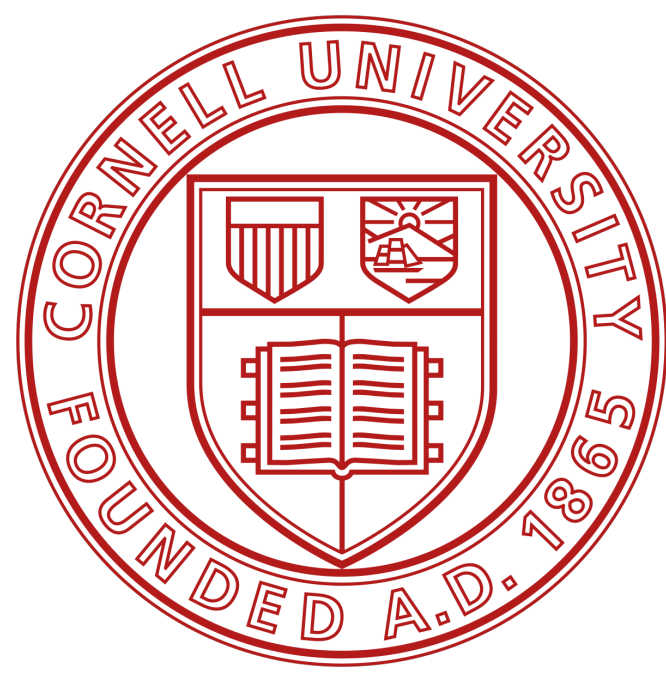
Vector outer product

- The outer product of two coordinate vectors is the matrix whose entries are all products of an element in the first vector with an element in the second vector.
- If the two coordinate vectors have dimensions n and m , then their outer product is an $n \times m$ matrix.

Given two vectors of size $m \times 1$ and $n \times 1$ respectively

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{A} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}$$

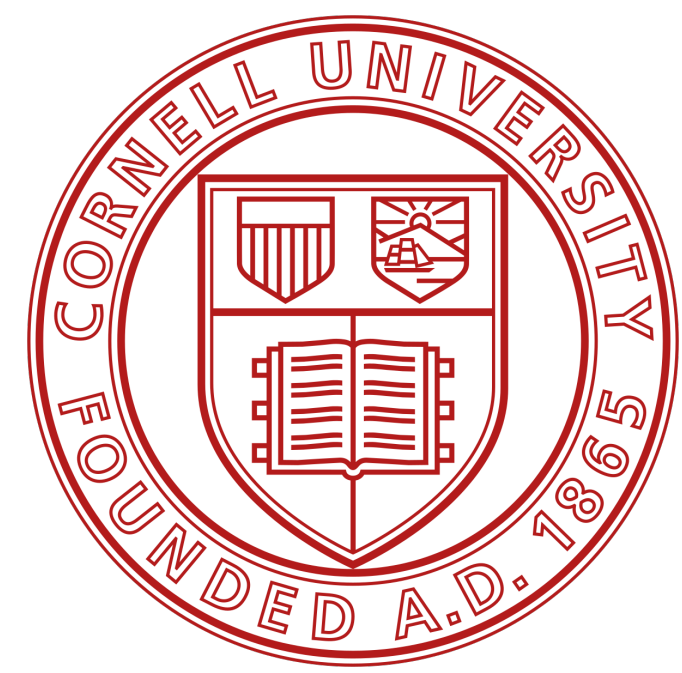


Running R code

Functions

```
Console Terminal x  
R 4.4.1 · ~/ ↗
```

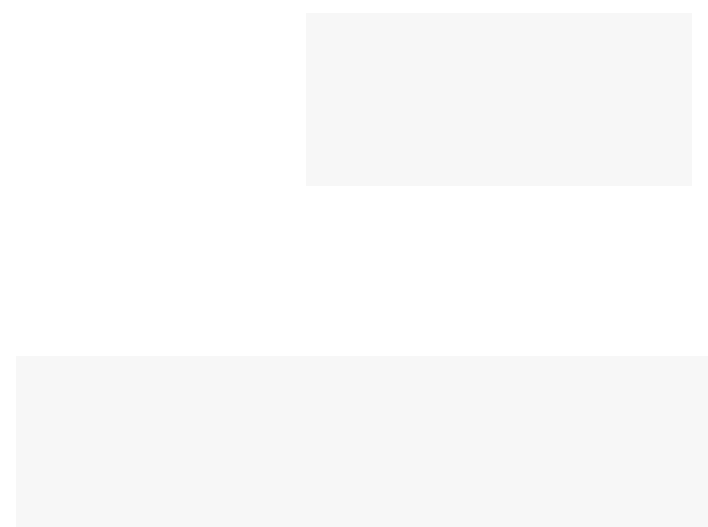
```
> round(3.1415)  
[1] 3  
> factorial(3)  
[1] 6  
> |
```



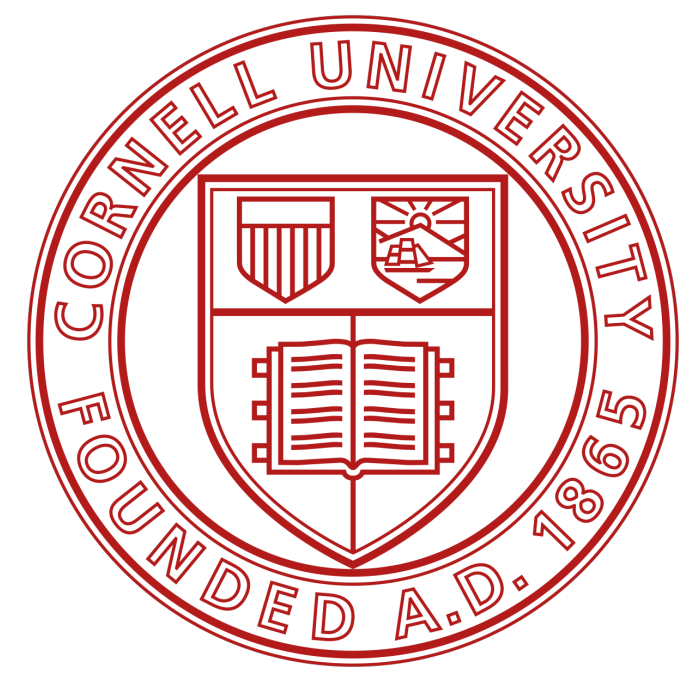
Running R code

Functions

- R comes with many functions that you can use to do sophisticated tasks like random sampling.



```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415)
[1] 3
> factorial(3)
[1] 6
> |
```

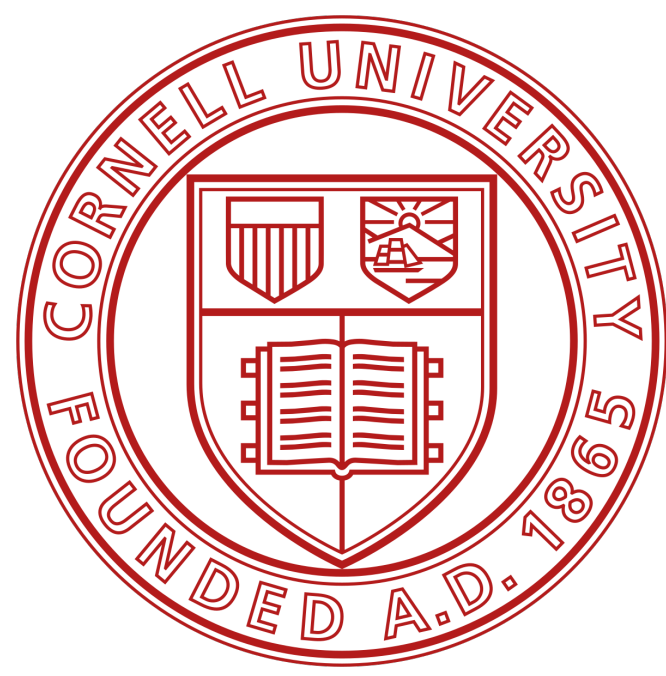


Running R code

Functions

- R comes with many functions that you can use to do sophisticated tasks like random sampling.
- For example, you can round a number with the `round` function, or calculate its factorial with the `factorial` function.

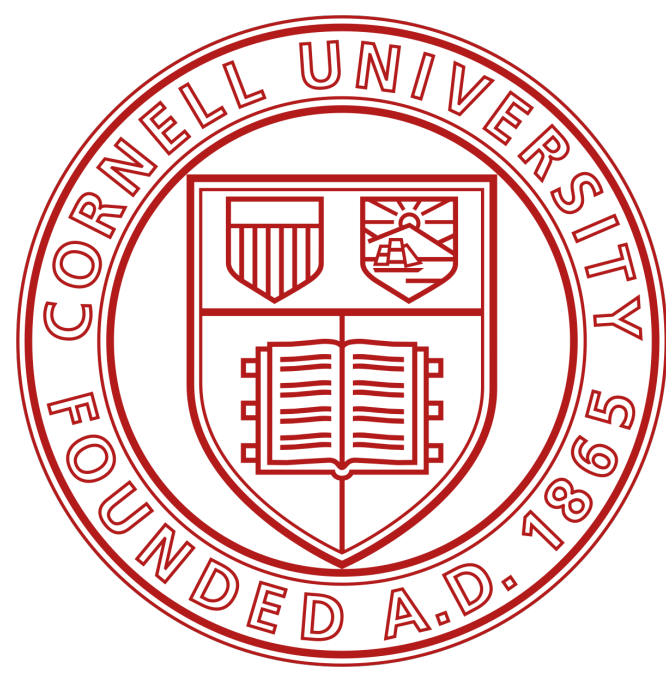
```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415)
[1] 3
> factorial(3)
[1] 6
> |
```



Running R code

Functions

```
Console Terminal x
R 4.4.1 · ~/ ↵
> mean(1:6)
[1] 3.5
> mean(die)
[1] 3.5
> round(mean(die))
[1] 4
> |
```

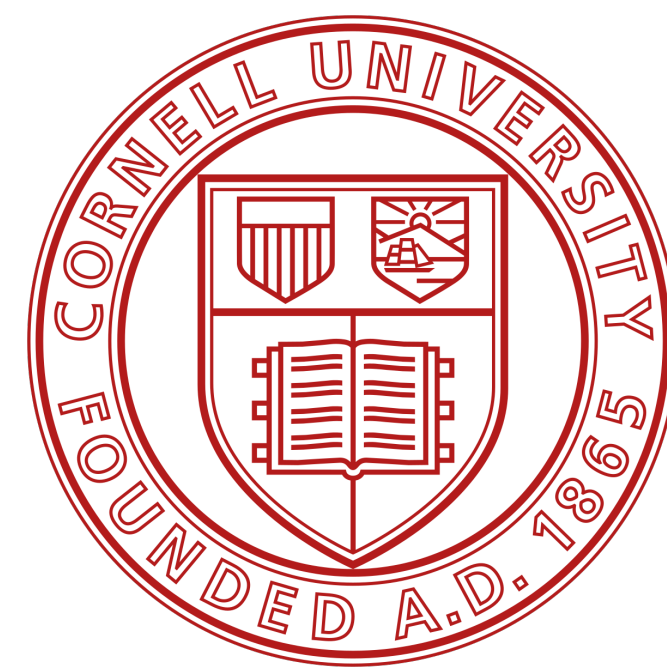



Running R code

Functions

- The data that you pass into the function is called the function's *argument*.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> mean(1:6)
[1] 3.5
> mean(die)
[1] 3.5
> round(mean(die))
[1] 4
> |
```

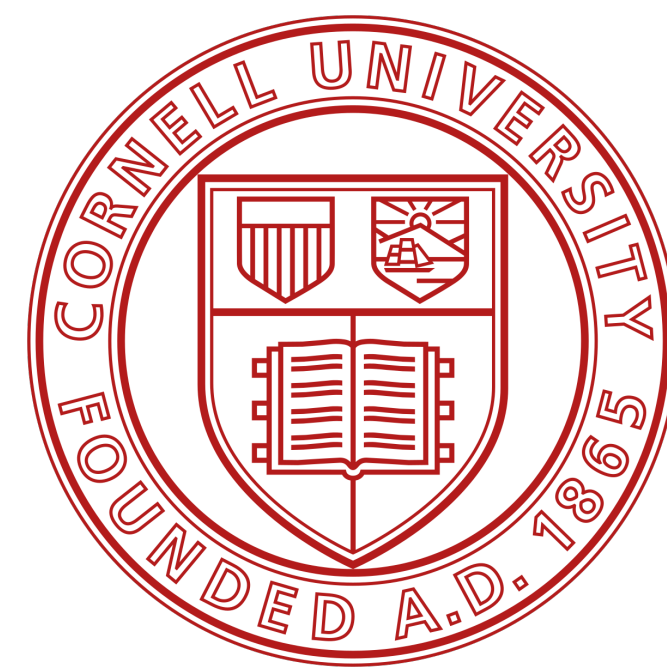


Running R code

Functions

- The data that you pass into the function is called the function's *argument*.
- The argument can be raw data, an R object, or even the results of another R function.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> mean(1:6)
[1] 3.5
> mean(die)
[1] 3.5
> round(mean(die))
[1] 4
> |
```

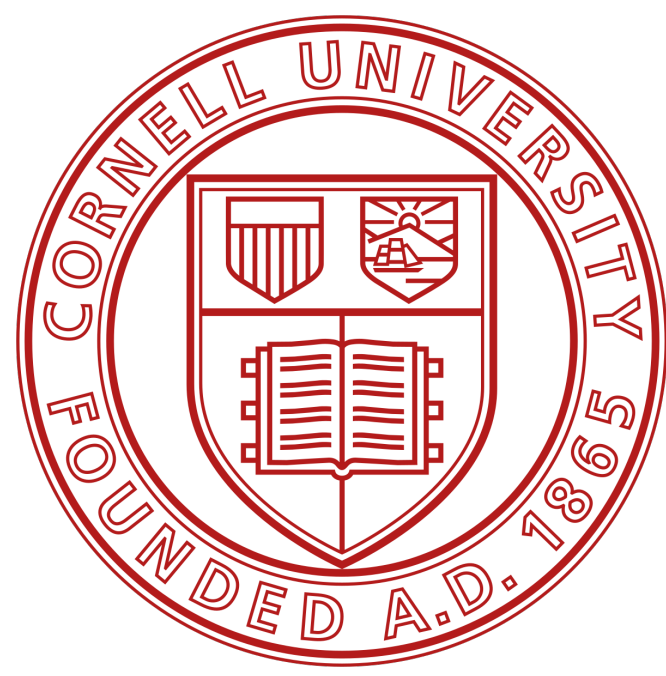


Running R code

Functions

- The data that you pass into the function is called the function's *argument*.
- The argument can be raw data, an R object, or even the results of another R function.
- In this last case, R will work from the innermost function to the outermost

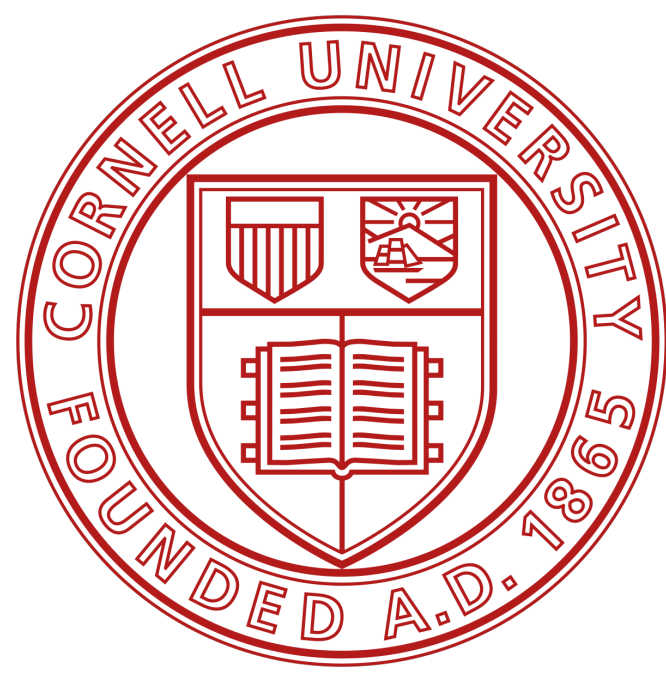
```
Console Terminal x
R 4.4.1 · ~/ ↵
> mean(1:6)
[1] 3.5
> mean(die)
[1] 3.5
> round(mean(die))
[1] 4
> |
```



Running R code

Functions

```
Console Terminal x
R 4.4.1 · ~/ ↵
> mean(1:6)
[1] 3.5
> mean(die)
[1] 3.5
> round(mean(die))
[1] 4
> |
```

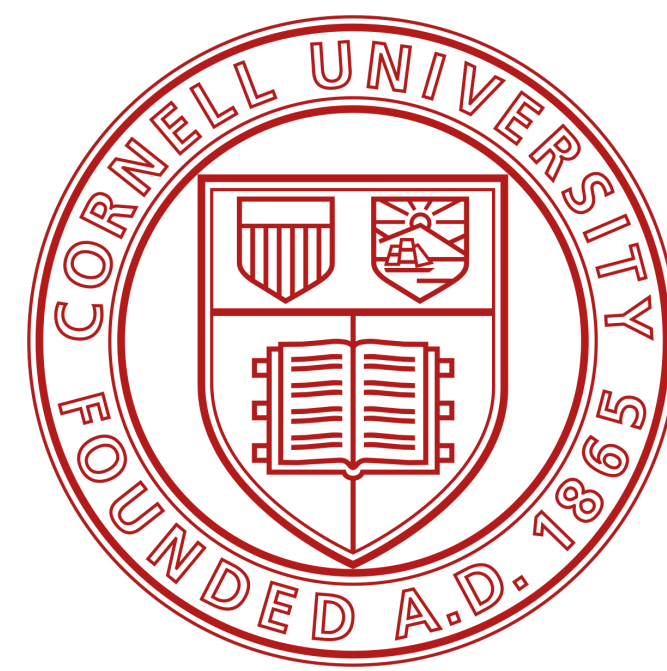


Running R code

Functions

- The data that you pass into the function is called the function's *argument*.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> mean(1:6)
[1] 3.5
> mean(die)
[1] 3.5
> round(mean(die))
[1] 4
> |
```

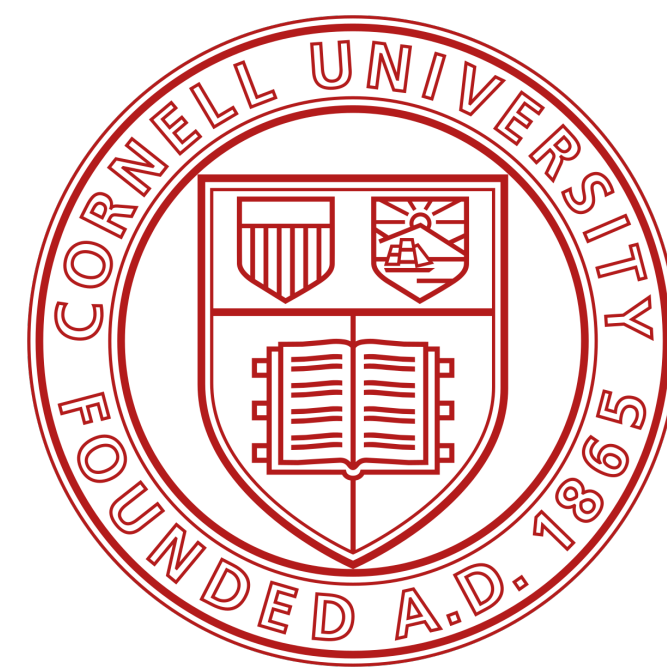


Running R code

Functions

- The data that you pass into the function is called the function's *argument*.
- The argument can be raw data, an R object, or even the results of another R function.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> mean(1:6)
[1] 3.5
> mean(die)
[1] 3.5
> round(mean(die))
[1] 4
> |
```



Running R code

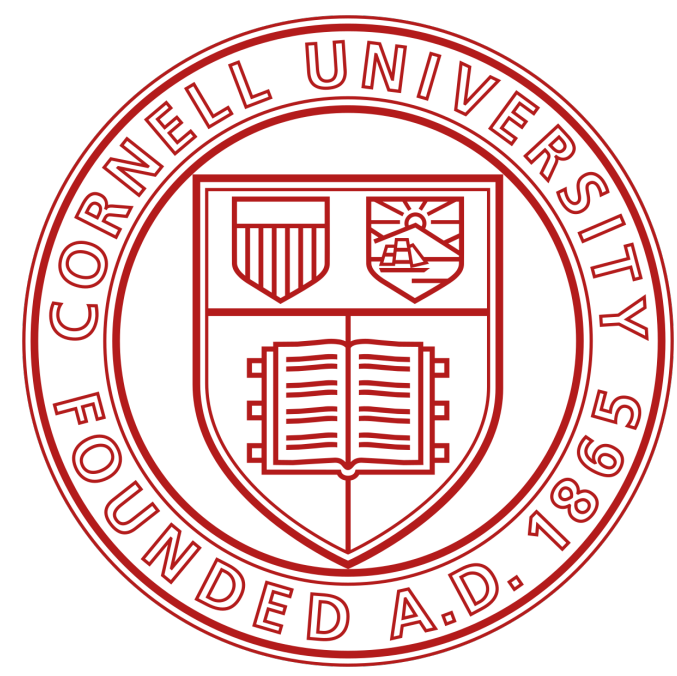
Functions

- The data that you pass into the function is called the function's *argument*.
- The argument can be raw data, an R object, or even the results of another R function.
- In this last case, R will work from the innermost function to the outermost

```
Console Terminal x
R 4.4.1 · ~/ ↵
> mean(1:6)
[1] 3.5
> mean(die)
[1] 3.5
> round(mean(die))
[1] 4
> |
```

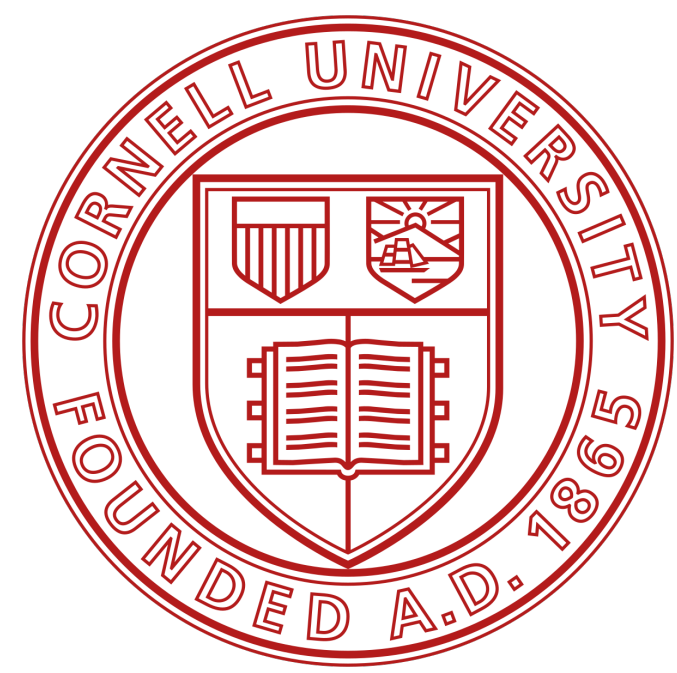
Basics

Roll a dice

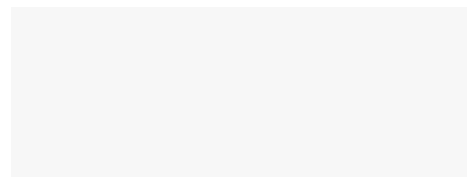


Basics

Roll a dice

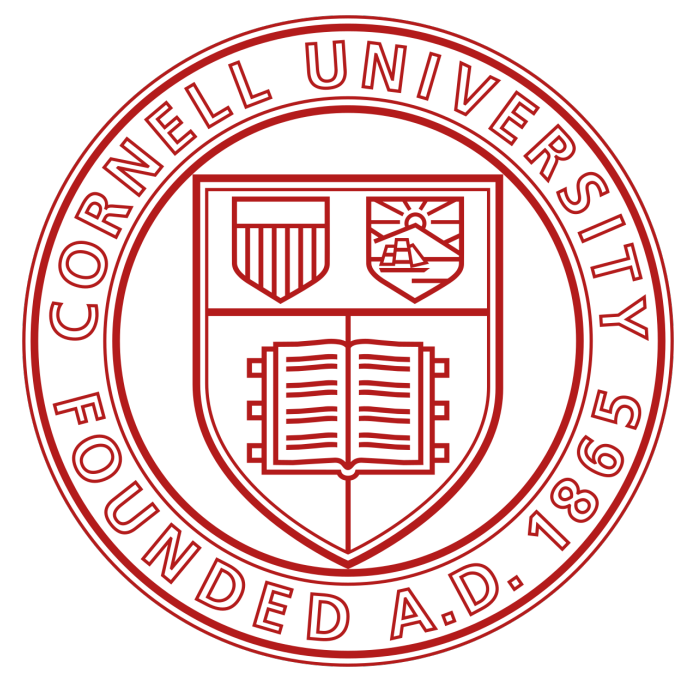


- Let's roll the virtual die



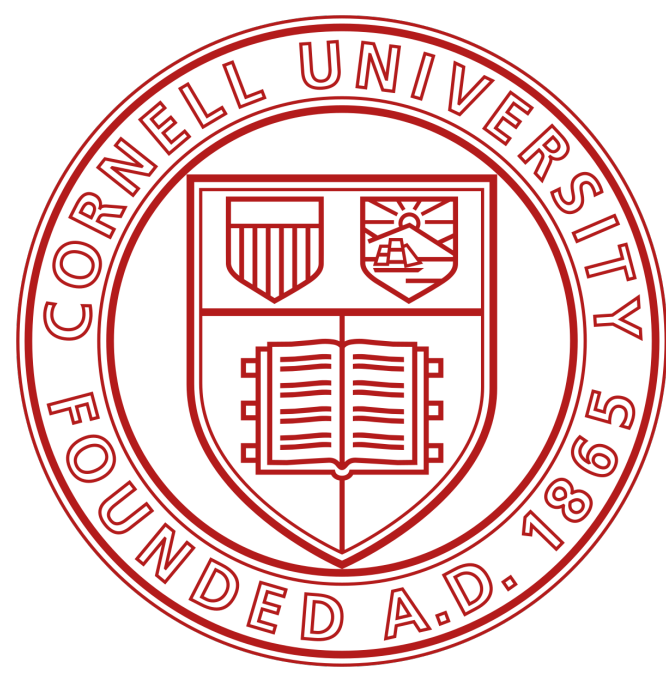
Basics

Roll a dice



- Let's roll the virtual die
- You can simulate a roll of the die with R's `sample` function.

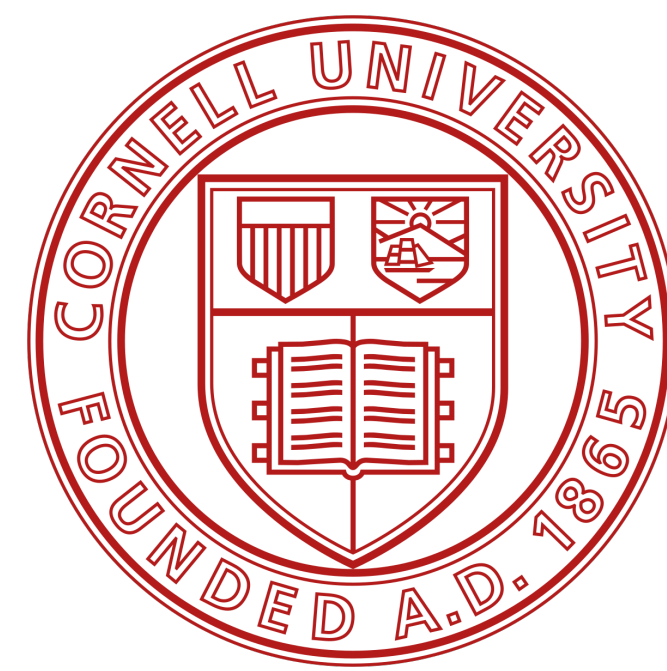




Basics

sample function

```
Console Terminal x  
R 4.4.1 · ~/ ↵  
> sample(x = 1:4, size = 2)  
[1] 3 2  
> sample(x = die, size = 1)  
[1] 2  
> sample(x = die, size = 1)  
[1] 3  
> sample(x = die, size = 1)  
[1] 2  
> sample(x = die, size = 1)  
[1] 1  
> sample(x = die, size = 1)  
[1] 5  
> |
```

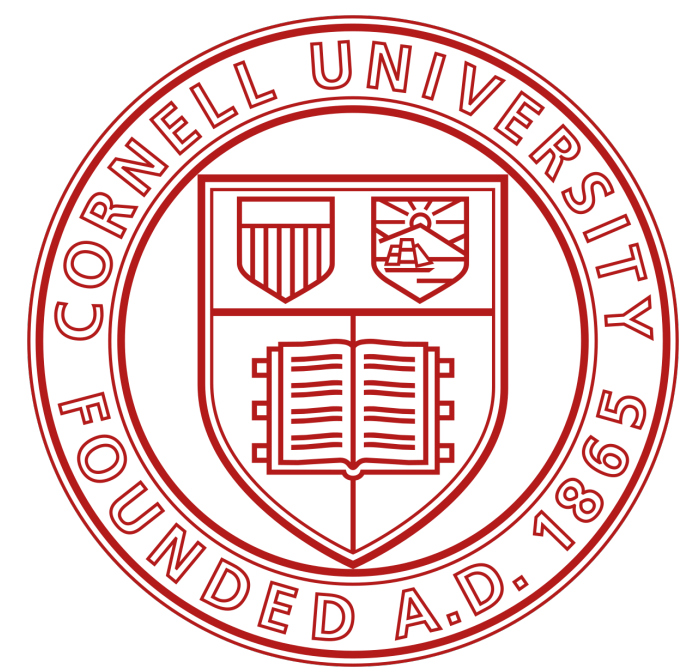


Basics

sample function

- `sample` takes *two* arguments: a vector named `x` and a number named `size`.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> sample(x = 1:4, size = 2)
[1] 3 2
> sample(x = die, size = 1)
[1] 2
> sample(x = die, size = 1)
[1] 3
> sample(x = die, size = 1)
[1] 2
> sample(x = die, size = 1)
[1] 1
> sample(x = die, size = 1)
[1] 5
> |
```



Basics

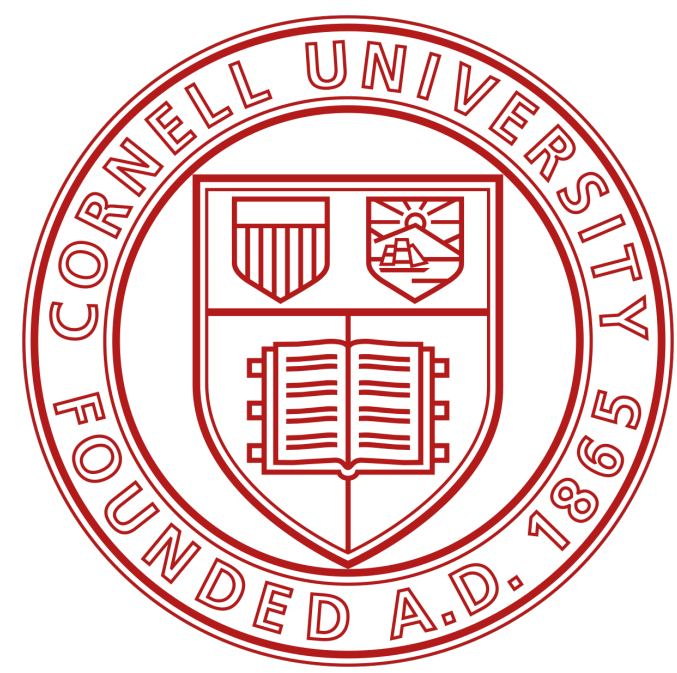
sample function

- `sample` takes *two* arguments: a vector named `x` and a number named `size`.
- `sample` will return `size` elements from the vector

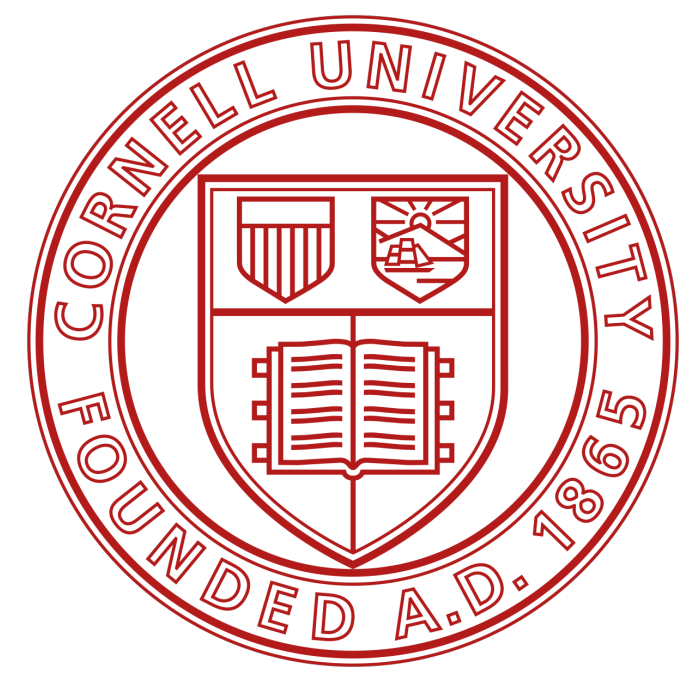
```
Console Terminal x
R 4.4.1 · ~/
> sample(x = 1:4, size = 2)
[1] 3 2
> sample(x = die, size = 1)
[1] 2
> sample(x = die, size = 1)
[1] 3
> sample(x = die, size = 1)
[1] 2
> sample(x = die, size = 1)
[1] 1
> sample(x = die, size = 1)
[1] 5
> |
```

Basics

Arguments



```
Console Terminal x  
R 4.4.1 · ~/ ↵  
> sample(die, size = 1)  
[1] 4  
> sample(die, size = 1)  
[1] 3  
> |
```

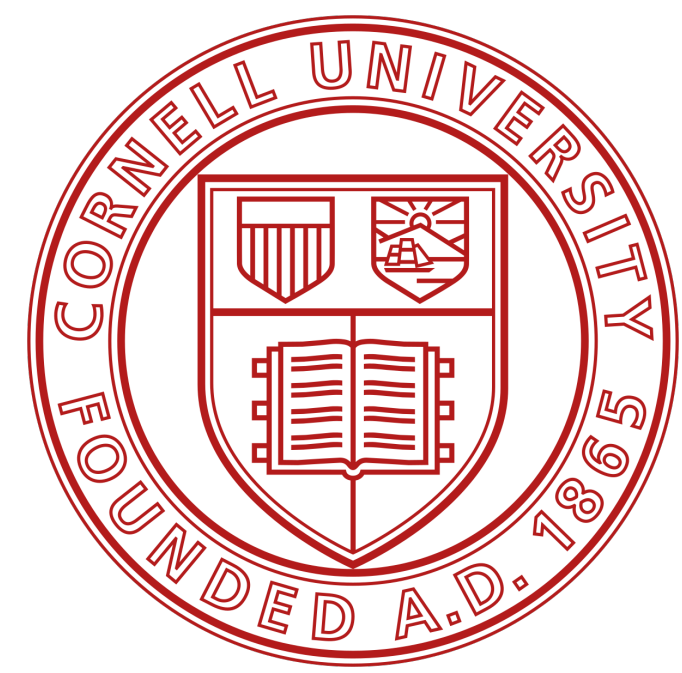


Basics

Arguments

- Every argument in every R function has a name.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> sample(die, size = 1)
[1] 4
> sample(die, size = 1)
[1] 3
> |
```

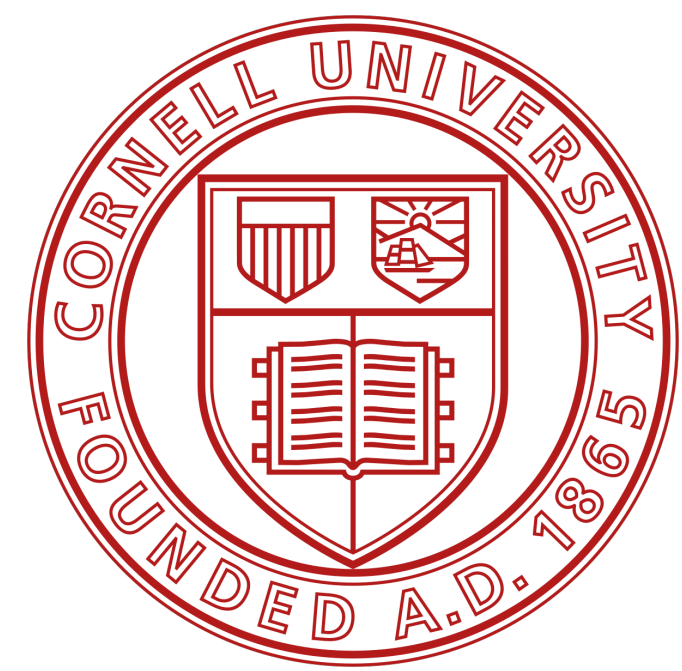


Basics

Arguments

- Every argument in every R function has a name.
- You can specify which data should be assigned to which argument by setting a name equal to data.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> sample(die, size = 1)
[1] 4
> sample(die, size = 1)
[1] 3
> |
```

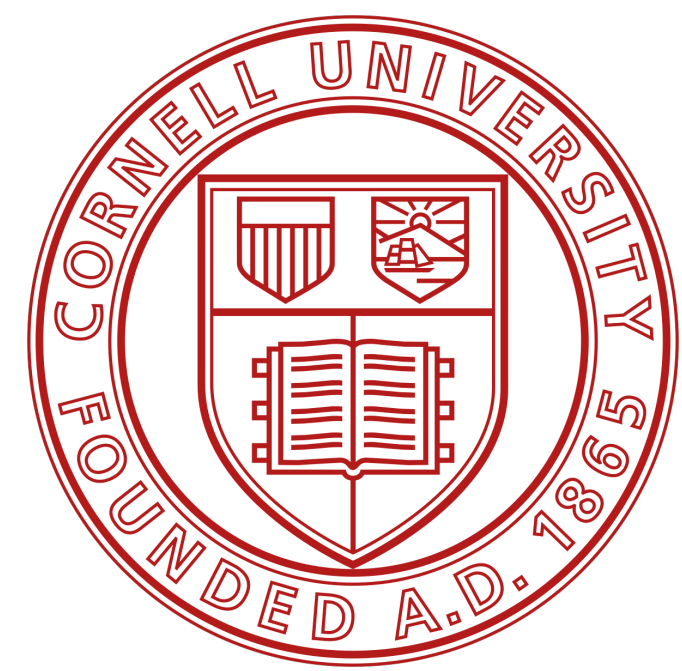



Basics

Arguments

- Every argument in every R function has a name.
- You can specify which data should be assigned to which argument by setting a name equal to data.
- Names help you avoid passing the wrong data to the wrong argument.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> sample(die, size = 1)
[1] 4
> sample(die, size = 1)
[1] 3
> |
```

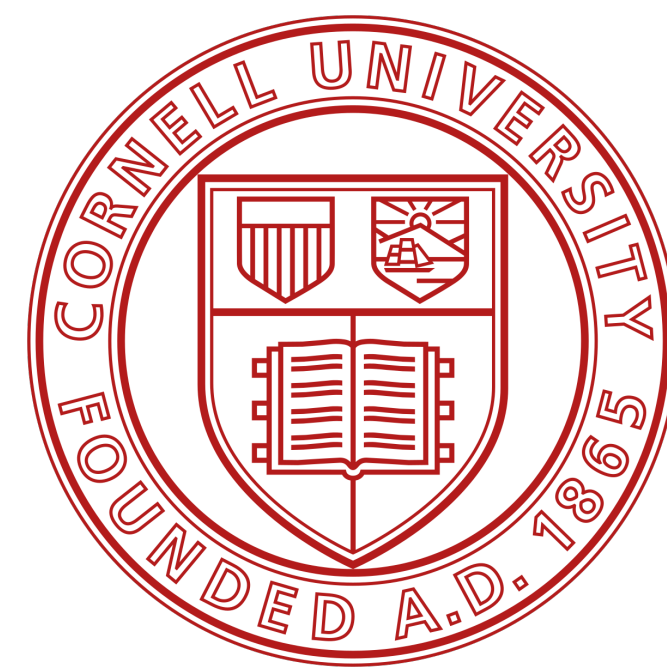


Basics

Arguments

- Every argument in every R function has a name.
- You can specify which data should be assigned to which argument by setting a name equal to data.
- Names help you avoid passing the wrong data to the wrong argument.
- Using names is optional in R.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> sample(die, size = 1)
[1] 4
> sample(die, size = 1)
[1] 3
> |
```



Basics

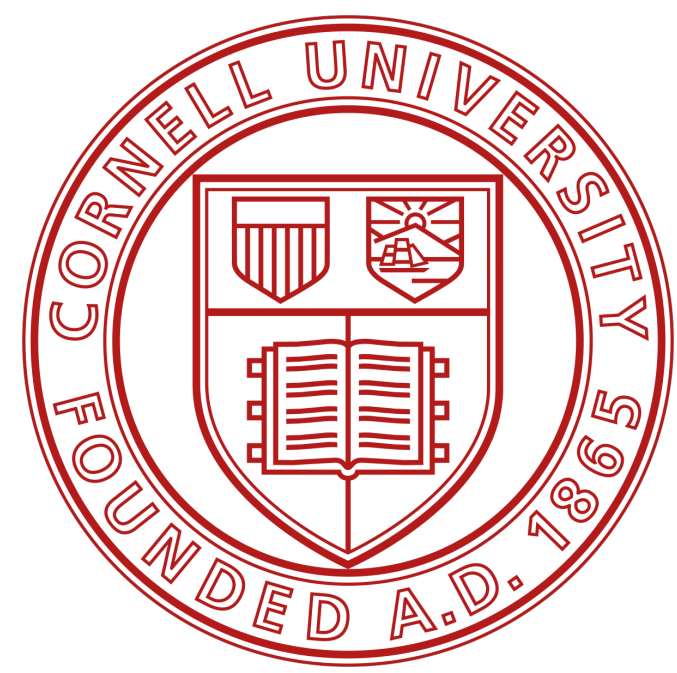
Arguments

- Every argument in every R function has a name.
- You can specify which data should be assigned to which argument by setting a name equal to data.
- Names help you avoid passing the wrong data to the wrong argument.
- Using names is optional in R.
- R users do not often use the name of the first argument in a function

```
Console Terminal x
R 4.4.1 · ~/ ↵
> sample(die, size = 1)
[1] 4
> sample(die, size = 1)
[1] 3
> |
```

Basics

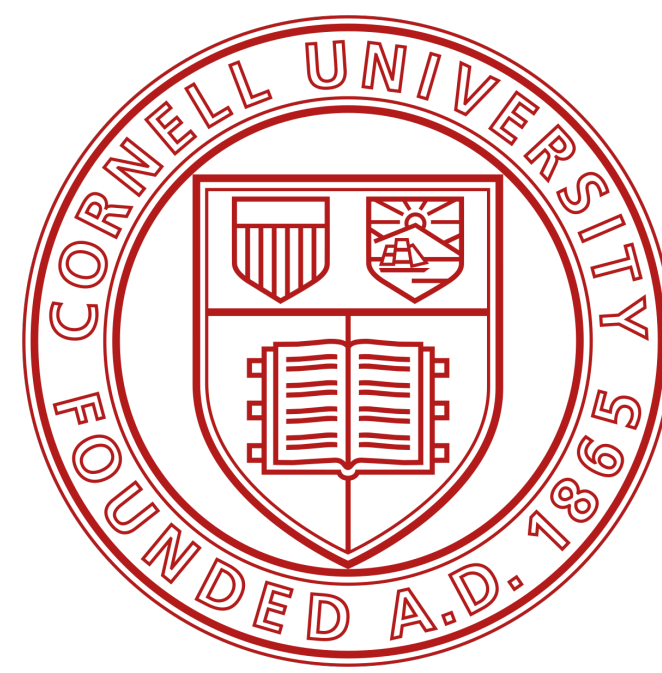
Arguments



```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415, corners = 2)
Error in round(3.1415, corners = 2) : unused argument
(corners = 2)
> |
```

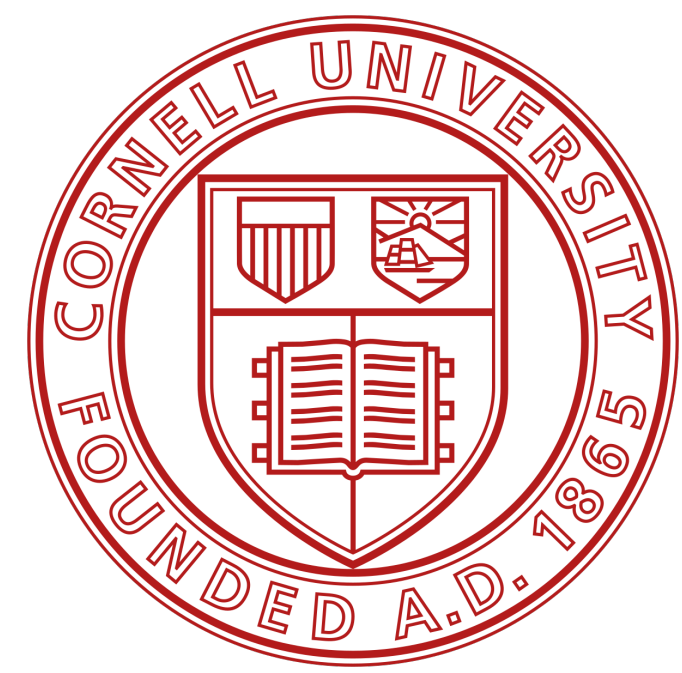
Basics

Arguments



- How do you know which argument names to use?

```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415, corners = 2)
Error in round(3.1415, corners = 2) : unused argument
(corners = 2)
> |
```



Basics

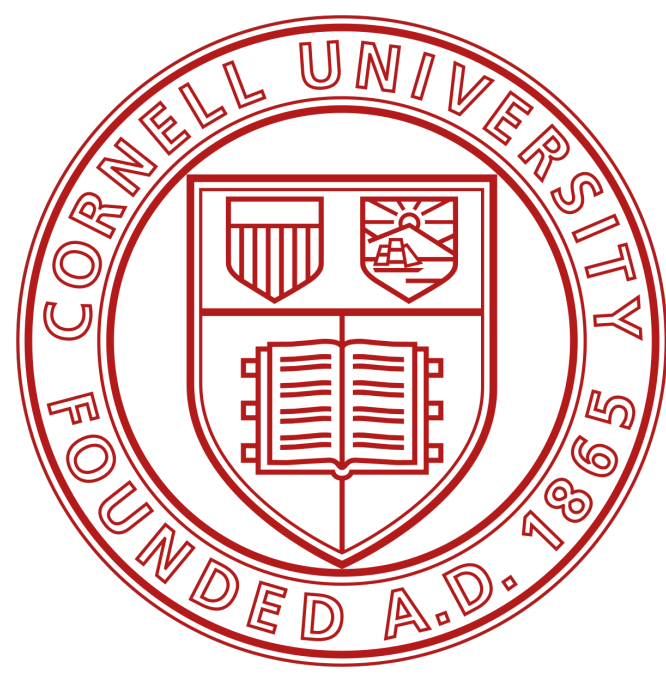
Arguments

- How do you know which argument names to use?
- If you try to use a name that a function does not expect, you will likely get an error.

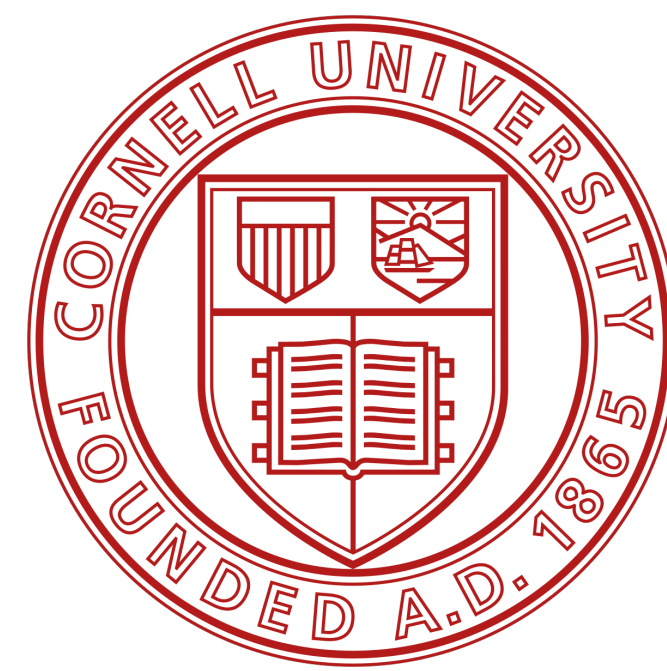
```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415, corners = 2)
Error in round(3.1415, corners = 2) : unused argument
(corners = 2)
> |
```

Basics

Arguments



```
Console Terminal x  
R 4.4.1 · ~/ ↵  
> args(round)  
function (x, digits = 0, ...)  
NULL  
└
```

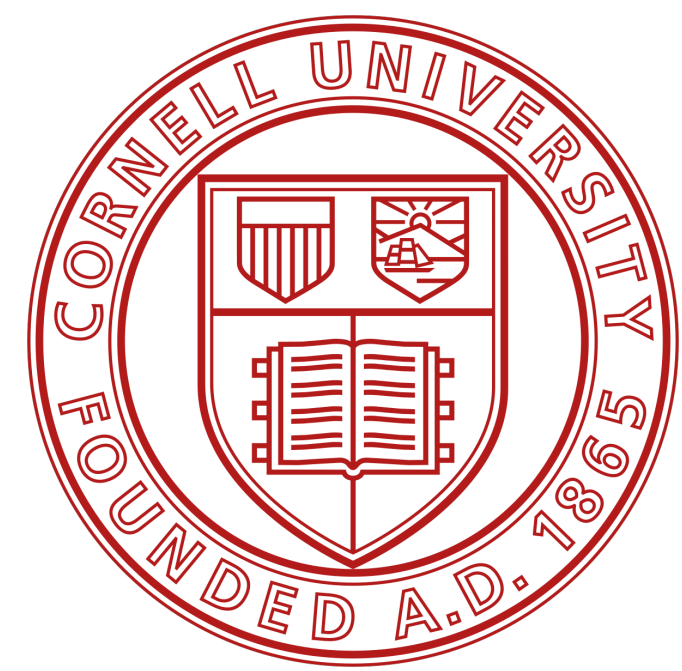


Basics

Arguments

- If you're not sure which names to use with a function, you can look up the function's arguments with `args`.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> args(round)
function (x, digits = 0, ...)
NULL
└
```

Basics

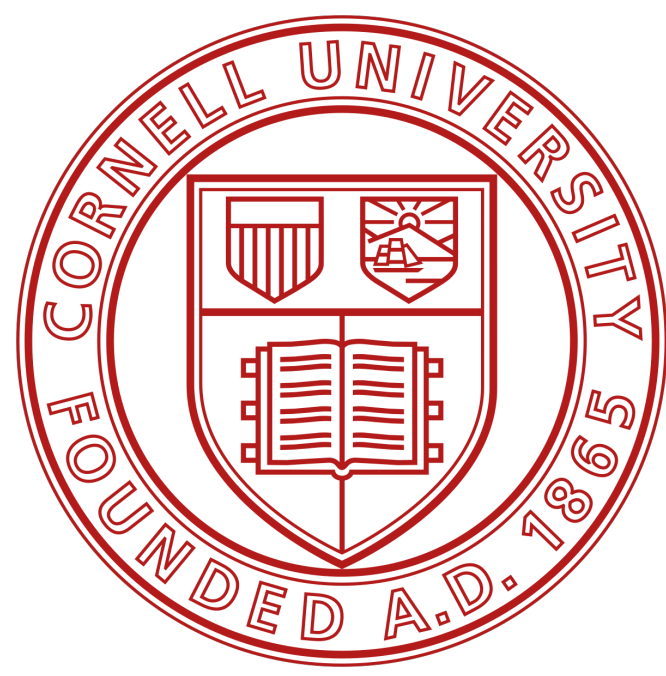
Arguments

- If you're not sure which names to use with a function, you can look up the function's arguments with `args`.
- To do this, place the name of the function in the parentheses behind `args`

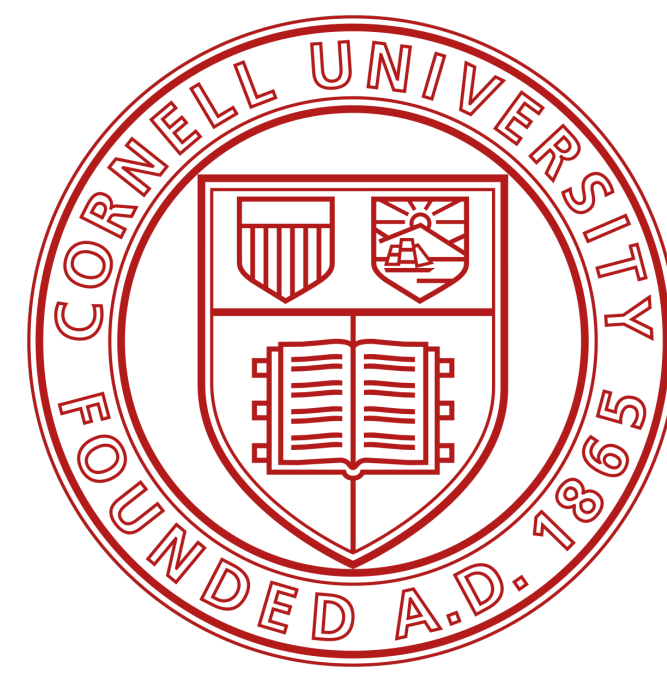
```
Console Terminal x
R 4.4.1 · ~/ ↵
> args(round)
function (x, digits = 0, ...)
NULL
```

Basics

Arguments



```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415)
[1] 3
> round(3.1415, digits = 2)
[1] 3.14
>
```

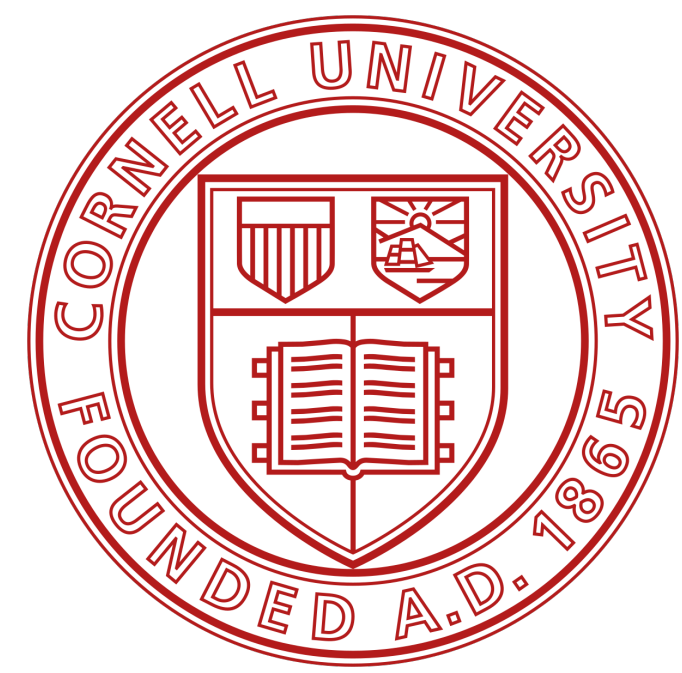


Basics

Arguments

- Did you notice that `args` shows that the `digits` argument of `round` is already set to 0?

```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415)
[1] 3
> round(3.1415, digits = 2)
[1] 3.14
>
```

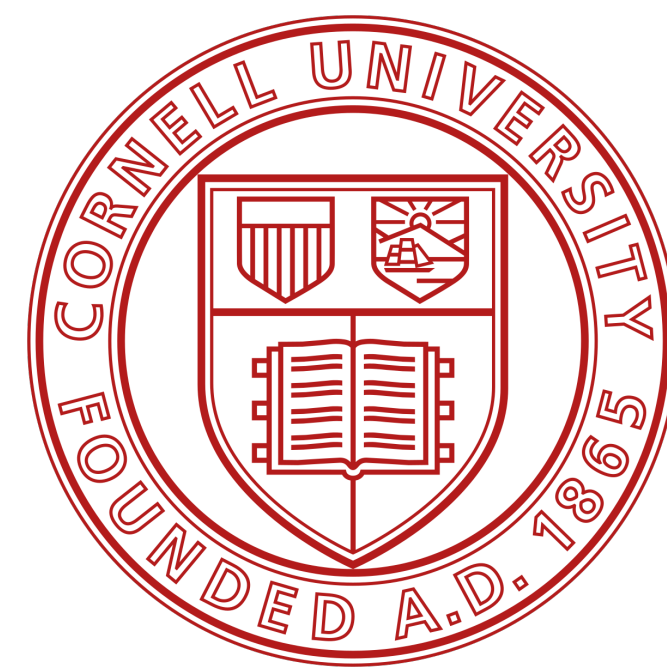


Basics

Arguments

- Did you notice that `args` shows that the `digits` argument of `round` is already set to 0?
- Frequently, an R function will take optional arguments like `digits`.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415)
[1] 3
> round(3.1415, digits = 2)
[1] 3.14
>
```

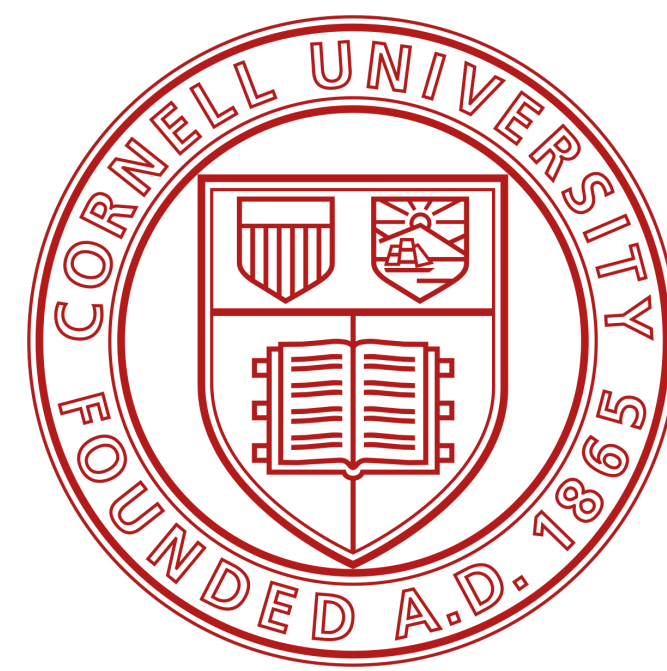


Basics

Arguments

- Did you notice that `args` shows that the `digits` argument of `round` is already set to 0?
- Frequently, an R function will take optional arguments like `digits`.
- These arguments are considered optional because they come with a default value.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415)
[1] 3
> round(3.1415, digits = 2)
[1] 3.14
>
```

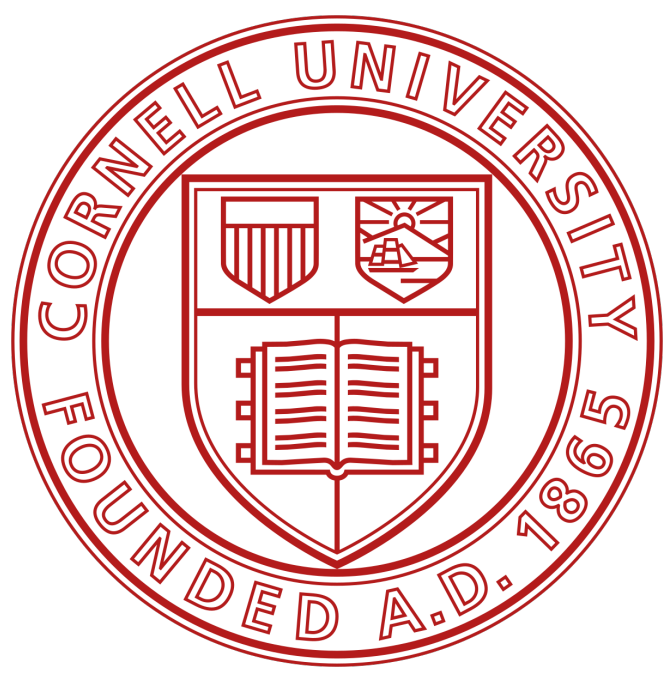


Basics

Arguments

- Did you notice that `args` shows that the `digits` argument of `round` is already set to 0?
- Frequently, an R function will take optional arguments like `digits`.
- These arguments are considered optional because they come with a default value.
- You can pass a new value to an optional argument if you want, and R will use the default value if you do not.

```
Console Terminal x
R 4.4.1 · ~/ ↵
> round(3.1415)
[1] 3
> round(3.1415, digits = 2)
[1] 3.14
>
```



Basics

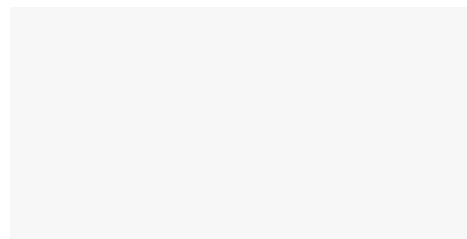
Sample with replacement

```
Console Terminal x
R 4.4.1 · ~/
> sample(die, size = 2)
[1] 1 6
> sample(die, size = 2)
[1] 5 3
> sample(die, size = 2)
[1] 2 1
> sample(die, size = 2)
[1] 4 3
> sample(die, size = 2)
[1] 1 3
> sample(die, size = 2)
[1] 2 1
> sample(die, size = 2)
[1] 4 5
> sample(die, size = 2)
[1] 1 2
> sample(die, size = 2)
[1] 5 2
> sample(die, size = 2)
[1] 3 4
> sample(die, size = 2)
[1] 4 6
> sample(die, size = 2)
[1] 6 3
> sample(die, size = 2)
[1] 5 6
> |
```

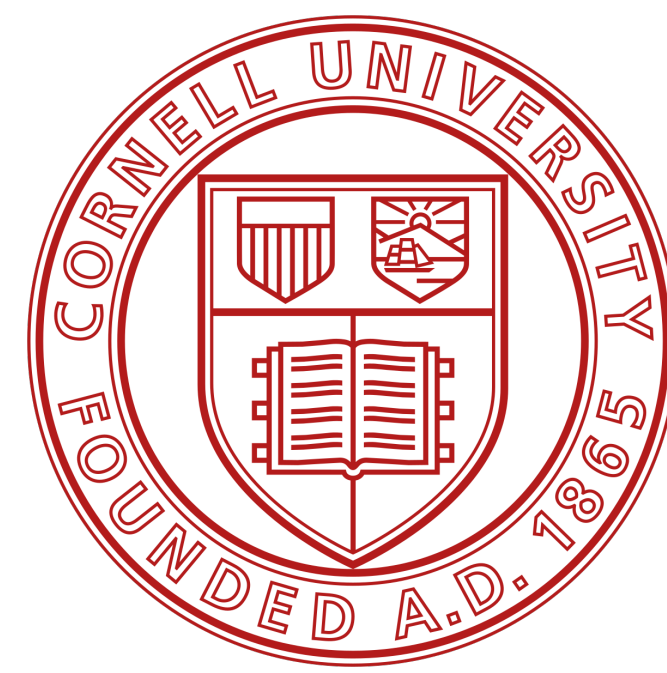
Basics

Sample with replacement

- With `sample` if you set `size = 2`, you can *almost* simulate a pair of dice.



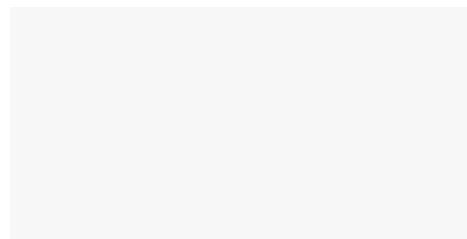
```
Console Terminal x
R 4.4.1 · ~/
> sample(die, size = 2)
[1] 1 6
> sample(die, size = 2)
[1] 5 3
> sample(die, size = 2)
[1] 2 1
> sample(die, size = 2)
[1] 4 3
> sample(die, size = 2)
[1] 1 3
> sample(die, size = 2)
[1] 2 1
> sample(die, size = 2)
[1] 4 5
> sample(die, size = 2)
[1] 1 2
> sample(die, size = 2)
[1] 5 2
> sample(die, size = 2)
[1] 3 4
> sample(die, size = 2)
[1] 4 6
> sample(die, size = 2)
[1] 6 3
> sample(die, size = 2)
[1] 5 6
> |
```



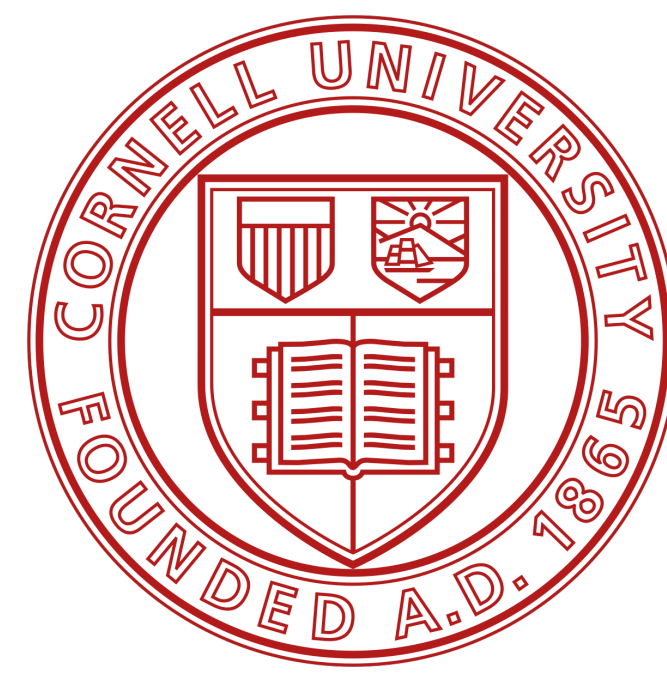
Basics

Sample with replacement

- With `sample` if you set `size = 2`, you can *almost* simulate a pair of dice.
- “Almost” because if you use it many times, you’ll notice that the second die never has the same value as the first die.



```
Console Terminal x
R 4.4.1 · ~/
> sample(die, size = 2)
[1] 1 6
> sample(die, size = 2)
[1] 5 3
> sample(die, size = 2)
[1] 2 1
> sample(die, size = 2)
[1] 4 3
> sample(die, size = 2)
[1] 1 3
> sample(die, size = 2)
[1] 2 1
> sample(die, size = 2)
[1] 4 5
> sample(die, size = 2)
[1] 1 2
> sample(die, size = 2)
[1] 5 2
> sample(die, size = 2)
[1] 3 4
> sample(die, size = 2)
[1] 4 6
> sample(die, size = 2)
[1] 6 3
> sample(die, size = 2)
[1] 5 6
> |
```



Basics

Sample with replacement

- With `sample` if you set `size = 2`, you can *almost* simulate a pair of dice.
- “Almost” because if you use it many times, you’ll notice that the second die never has the same value as the first die.
- By default, `sample` builds a sample without replacement.

```
Console Terminal x
R 4.4.1 · ~/
> sample(die, size = 2)
[1] 1 6
> sample(die, size = 2)
[1] 5 3
> sample(die, size = 2)
[1] 2 1
> sample(die, size = 2)
[1] 4 3
> sample(die, size = 2)
[1] 1 3
> sample(die, size = 2)
[1] 2 1
> sample(die, size = 2)
[1] 4 5
> sample(die, size = 2)
[1] 1 2
> sample(die, size = 2)
[1] 5 2
> sample(die, size = 2)
[1] 3 4
> sample(die, size = 2)
[1] 4 6
> sample(die, size = 2)
[1] 6 3
> sample(die, size = 2)
[1] 5 6
> |
```

