Introduction to R and R Studio

Jonathan Kropko (jkropko@virginia.edu)
District Data Labs

Data Software

Everyone who does quantitative research uses data analytic software. But the best software has changed over time:

Data Software

Everyone who does quantitative research uses data analytic software. But the best software has changed over time:

In the 1960s and 1970s, researchers had to write code on punchcards and manually feed them into computers that took up a whole room.

Data Software

Everyone who does quantitative research uses data analytic software. But the best software has changed over time:

In the 1960s and 1970s, researchers had to write code on punchcards and manually feed them into computers that took up a whole room.

Starting in the 1980s it became possible to run data analysis software on a desktop computer. There are different styles of software that break down along two dimensions:

- **open source or proprietary?**
- **code-based or with a graphical user interface?**

Data Software

Some well known examples of data analysis software can be described by their choice of license and interface:

		User interaction	
		Graphical interface	Code
Licensing	Proprietary	Excel SPSS Tableau	Stata SAS
	Open-source	Apache OpenOffice	R Python

Data Software

Some well known examples of data analysis software can be described by their choice of license and interface:

		User interaction	
		Graphical interface	Code
Licensing	Proprietary	Excel SPSS Tableau	Stata SAS
	Open-source	Apache OpenOffice	R Python

Many of you have experience with proprietary software with a graphical interface, like Excel. In this class we will be learning R, which is **open-source and code-based**.

Data Software

Some well known examples of data analysis software can be described by their choice of license and interface:

		User interaction	
		Graphical interface	Code
Licensing	Proprietary	Excel SPSS Tableau	Stata SAS
	Open-source	Apache OpenOffice	R Python

Many of you have experience with proprietary software with a graphical interface, like Excel. In this class we will be learning R, which is **open-source and code-based**.

Why open-source? Why use code?

Why Open-Source?

Open-source software has a copyright that allows anyone to use, study, change, and distribute the software to anyone and for any purpose.

Why Open-Source?

Open-source software has a copyright that allows anyone to use, study, change, and distribute the software to anyone and for any purpose.

Open-source software is free of charge, and is developed and maintained by communities of volunteers.

Why Open-Source?

Open-source software has a copyright that allows anyone to use, study, change, and distribute the software to anyone and for any purpose.

Open-source software is free of charge, and is developed and maintained by communities of volunteers.

If the community is large enough, then the software develops more rapidly than any proprietary product can.

Why Open-Source?

Open-source software has a copyright that allows anyone to use, study, change, and distribute the software to anyone and for any purpose.

Open-source software is free of charge, and is developed and maintained by communities of volunteers.

If the community is large enough, then the software develops more rapidly than any proprietary product can.

And the forums/blogs/guides for the software provide very in depth **user support**.

Why Open-Source?

Open-source software has a copyright that allows anyone to use, study, change, and distribute the software to anyone and for any purpose.

Open-source software is free of charge, and is developed and maintained by communities of volunteers.

If the community is large enough, then the software develops more rapidly than any proprietary product can.

And the forums/blogs/guides for the software provide very in depth **user support**.

Only a few software environments have a community this big, but R is one of these environments.

Why Open-Source?

Advantages of open-source:

- Free

Why Open-Source?

Advantages of open-source:

- Free
- Lots and lots of free extensions that can do anything you want to do

Why Open-Source?

Advantages of open-source:

- Free
- Lots and lots of free extensions that can do anything you want to do
- Develops fast and stays current with the cutting edge

Why Open-Source?

Advantages of open-source:

- Free
- Lots and lots of free extensions that can do anything you want to do
- Develops fast and stays current with the cutting edge
- A big community is willing to help each other out

Why Open-Source?

Advantages of open-source:

- Free
- Lots and lots of free extensions that can do anything you want to do
- Develops fast and stays current with the cutting edge
- A big community is willing to help each other out

Disadvantages of open-source:

- No guarantees: although the community has systems for checking extensions, bugs are possible

Why Open-Source?

Advantages of open-source:

- Free
- Lots and lots of free extensions that can do anything you want to do
- Develops fast and stays current with the cutting edge
- A big community is willing to help each other out

Disadvantages of open-source:

- No guarantees: although the community has systems for checking extensions, bugs are possible
- Although the blogs/forums are very helpful, there's no dedicated IT support

Why Open-Source?

Advantages of open-source:

- Free
- Lots and lots of free extensions that can do anything you want to do
- Develops fast and stays current with the cutting edge
- A big community is willing to help each other out

Disadvantages of open-source:

- No guarantees: although the community has systems for checking extensions, bugs are possible
- Although the blogs/forums are very helpful, there's no dedicated IT support
- Sometimes there's a learning curve to get started

Why Code-Based Data Analysis?

The leap from a point-and-click interface to a code-based interface can be very daunting. Code is one of those things that *seems* tougher than it actually is.

Why Code-Based Data Analysis?

The leap from a point-and-click interface to a code-based interface can be very daunting. Code is one of those things that *seems* tougher than it actually is.

Anxiety around code is common when people are just starting. BUT code gets easier the more you use it! It is **normal to write code with errors**, even for experienced coders, so please cultivate a forgiving attitude towards yourself about it.

Why Code-Based Data Analysis?

The leap from a point-and-click interface to a code-based interface can be very daunting. Code is one of those things that *seems* tougher than it actually is.

Anxiety around code is common when people are just starting. BUT code gets easier the more you use it! It is **normal to write code with errors**, even for experienced coders, so please cultivate a forgiving attitude towards yourself about it.

Some software, like R, can only be used with code. But code also helps you automate future work, avoid big mistakes, and share your work more easily.

Why Code-Based Data Analysis?

Advantages of code:

- Allows you to keep track of your work, so that repeating the work is a simple as clicking the button to run the code.

Why Code-Based Data Analysis?

Advantages of code:

- Allows you to keep track of your work, so that repeating the work is a simple as clicking the button to run the code.
- Usually helps you avoid small errors with big impacts – like one corrupted cell in a spreadsheet.

Why Code-Based Data Analysis?

Advantages of code:

- Allows you to keep track of your work, so that repeating the work is a simple as clicking the button to run the code.
- Usually helps you avoid small errors with big impacts – like one corrupted cell in a spreadsheet.
- Easy to share your work with collaborators (just email or share the script).

Why Code-Based Data Analysis?

Advantages of code:

- Allows you to keep track of your work, so that repeating the work is a simple as clicking the button to run the code.
- Usually helps you avoid small errors with big impacts – like one corrupted cell in a spreadsheet.
- Easy to share your work with collaborators (just email or share the script).
- Usually a super-marketable skill!

Why Code-Based Data Analysis?

Advantages of code:

- Allows you to keep track of your work, so that repeating the work is a simple as clicking the button to run the code.
- Usually helps you avoid small errors with big impacts – like one corrupted cell in a spreadsheet.
- Easy to share your work with collaborators (just email or share the script).
- Usually a super-marketable skill!

Disadvantages of code:

- May take a while to learn enough to be "fluent", like a foreign language.

Why Code-Based Data Analysis?

Advantages of code:

- Allows you to keep track of your work, so that repeating the work is a simple as clicking the button to run the code.
- Usually helps you avoid small errors with big impacts – like one corrupted cell in a spreadsheet.
- Easy to share your work with collaborators (just email or share the script).
- Usually a super-marketable skill!

Disadvantages of code:

- May take a while to learn enough to be "fluent", like a foreign language.
- Can make it tough to collaborate with people who aren't fluent in the coding language.

What is R?

R is open-source, code-based software for data analysis. It is one of the most popular tools for data analysis in the world, and the R developer community has hundreds of thousands of people.

What is R?

R is open-source, code-based software for data analysis. It is one of the most popular tools for data analysis in the world, and the R developer community has hundreds of thousands of people.

R originated as a statistics engine, but it can now do so much more: make websites, interactive visualizations, animations, and maps, store databases, analyze texts, create artificial intelligence, and compose music.

What is R?

R is open-source, code-based software for data analysis. It is one of the most popular tools for data analysis in the world, and the R developer community has hundreds of thousands of people.

R originated as a statistics engine, but it can now do so much more: make websites, interactive visualizations, animations, and maps, store databases, analyze texts, create artificial intelligence, and compose music.

R and related software is 100% free.

What is R?

R is open-source, code-based software for data analysis. It is one of the most popular tools for data analysis in the world, and the R developer community has hundreds of thousands of people.

R originated as a statistics engine, but it can now do so much more: make websites, interactive visualizations, animations, and maps, store databases, analyze texts, create artificial intelligence, and compose music.

R and related software is 100% free.

R produces beautiful, logical, interactive graphics, and statisticians write R code to implement new methods and release it to the public. So R can do cutting-edge stuff.

What is R?

R is open-source, code-based software for data analysis. It is one of the most popular tools for data analysis in the world, and the R developer community has hundreds of thousands of people.

R originated as a statistics engine, but it can now do so much more: make websites, interactive visualizations, animations, and maps, store databases, analyze texts, create artificial intelligence, and compose music.

R and related software is 100% free.

R produces beautiful, logical, interactive graphics, and statisticians write R code to implement new methods and release it to the public. So R can do cutting-edge stuff.

R also makes it easy to clean data and to make websites that communicate your work to an audience.

Exercise

One of the most well-known metrics of the popularity of various programming languages is the TIOBE Programming Community Index. Their website is: https://www.tiobe.com/tiobe-index/.

"The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings."

According to TIOBE how popular is R? How is the popularity of R trending? How does it compare to SPSS, SAS, and Stata?

Exercise

One of the most well-known metrics of the popularity of various programming languages is the TIOBE Programming Community Index. Their website is: https://www.tiobe.com/tiobe-index/.

"The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings."

According to TIOBE how popular is R? How is the popularity of R trending? How does it compare to SPSS, SAS, and Stata?

We'll be using a web-based version of R in this course. But you can always download the latest version of R on your own computer from https://cran.r-project.org/.

R Studio

R is a programming language. We can write R code in any kind of text interface, even in a Word or Notepad file.

R Studio

R is a programming language. We can write R code in any kind of text interface, even in a Word or Notepad file.

But unlike Word and Notepad files, there are better tools that allow us to write code with some **extra features**:

- A button to run highlighted code

R Studio

R is a programming language. We can write R code in any kind of text interface, even in a Word or Notepad file.

But unlike Word and Notepad files, there are better tools that allow us to write code with some **extra features**:

- A button to run highlighted code
- A window to see results

R Studio

R is a programming language. We can write R code in any kind of text interface, even in a Word or Notepad file.

But unlike Word and Notepad files, there are better tools that allow us to write code with some **extra features**:

- A button to run highlighted code
- A window to see results
- A place to look at help documentation

R Studio

R is a programming language. We can write R code in any kind of text interface, even in a Word or Notepad file.

But unlike Word and Notepad files, there are better tools that allow us to write code with some **extra features**:

- A button to run highlighted code
- A window to see results
- A place to look at help documentation
- Tabs for different coding projects

R Studio

R is a programming language. We can write R code in any kind of text interface, even in a Word or Notepad file.

But unlike Word and Notepad files, there are better tools that allow us to write code with some **extra features**:

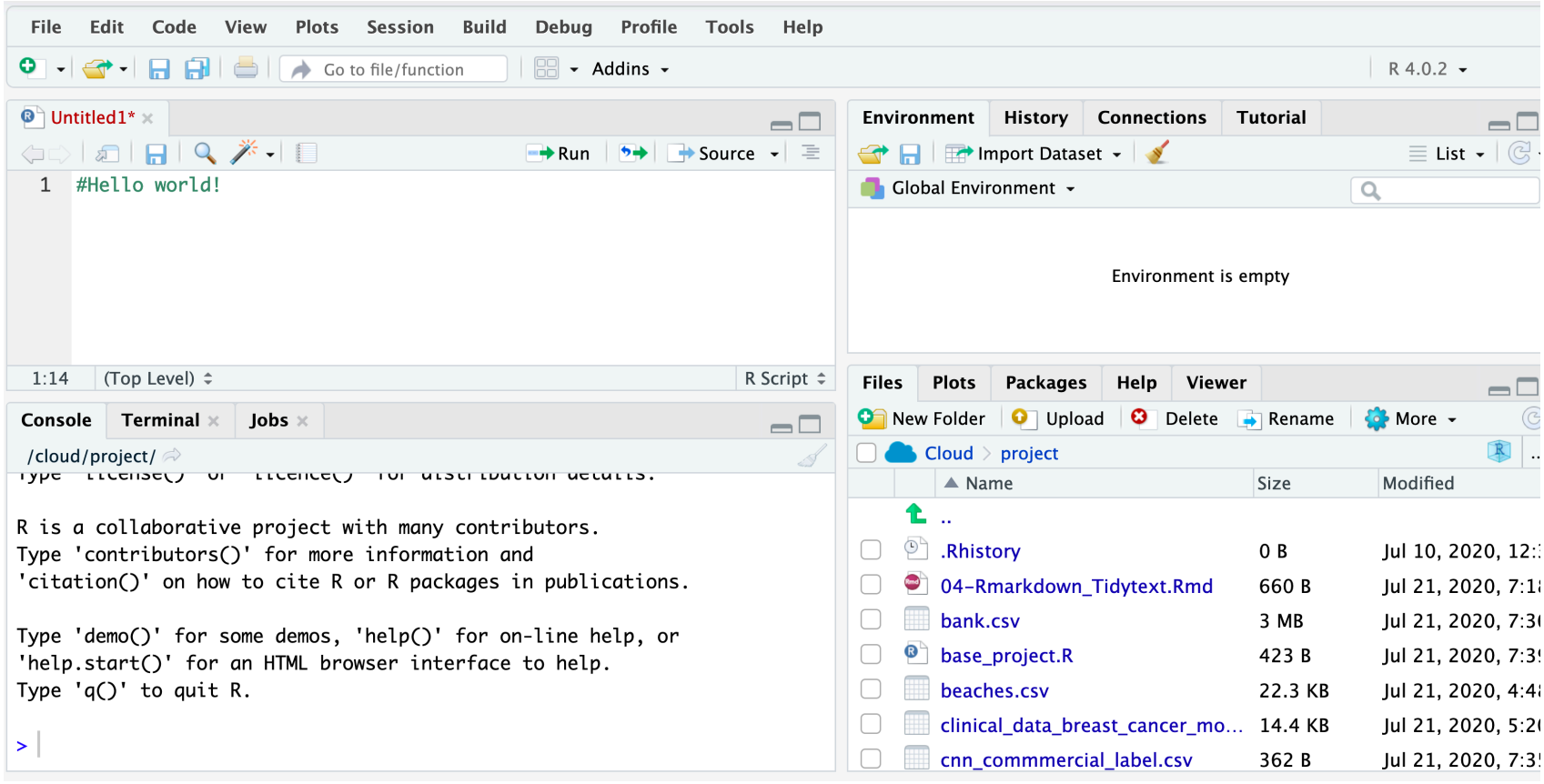
- A button to run highlighted code
- A window to see results
- A place to look at help documentation
- Tabs for different coding projects

The best tool for writing R code is a program called **R Studio**.

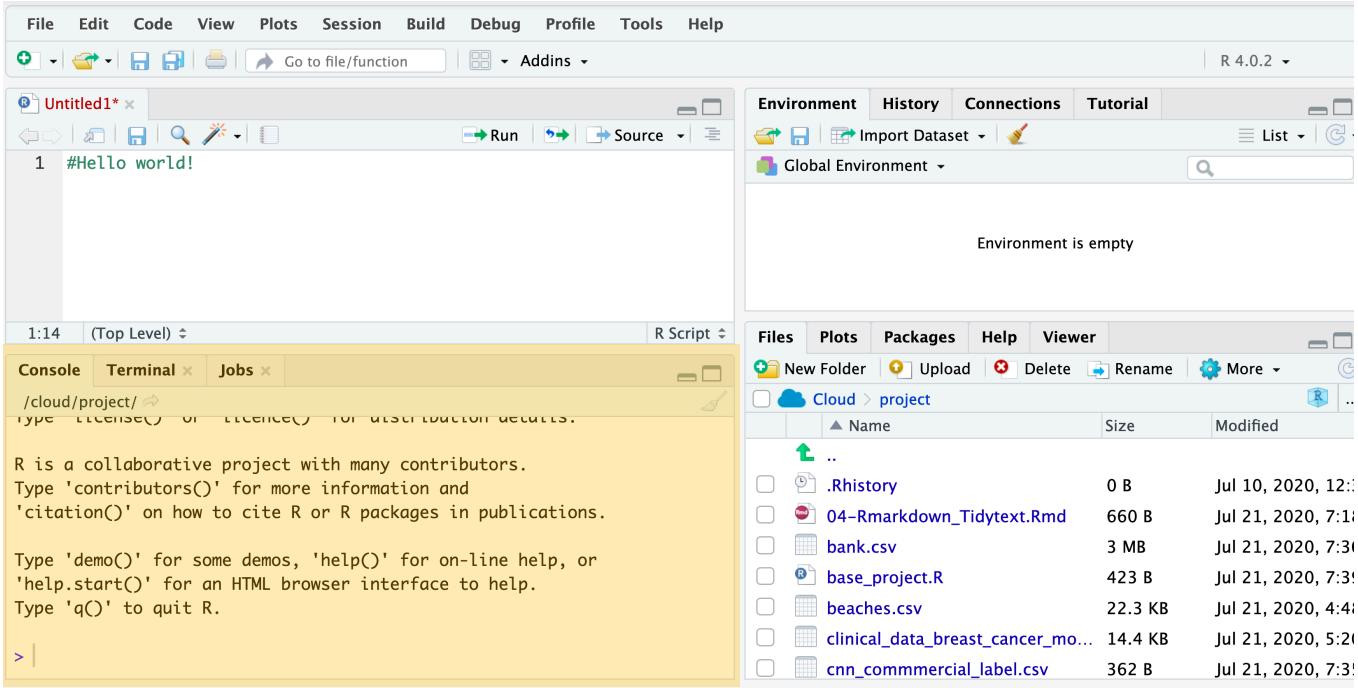
R Studio is a for profit company, but the R Studio software is free. If you want to download the latest version of R Studio on your own computer, go to https://www.rstudio.com/.

What's on the R Studio Screen?

The R Studio screen looks pretty crazy. Let's discuss what each window does.

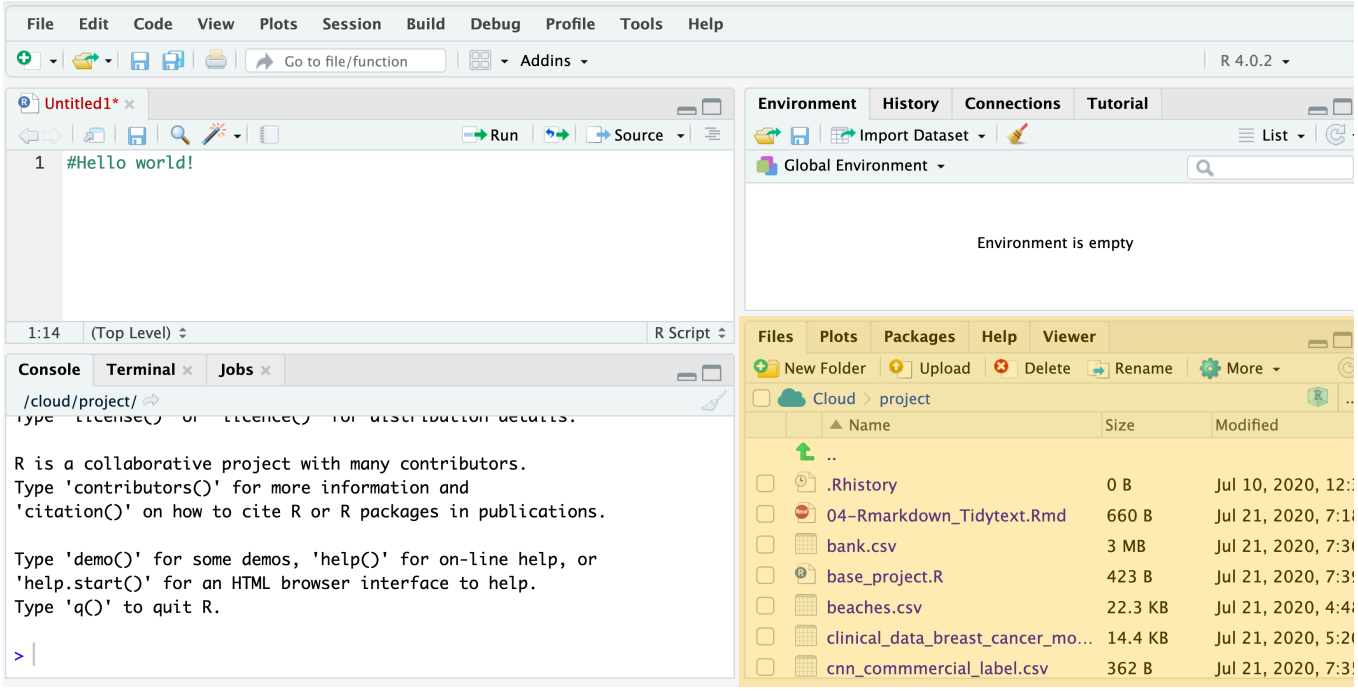


The **Console** Window



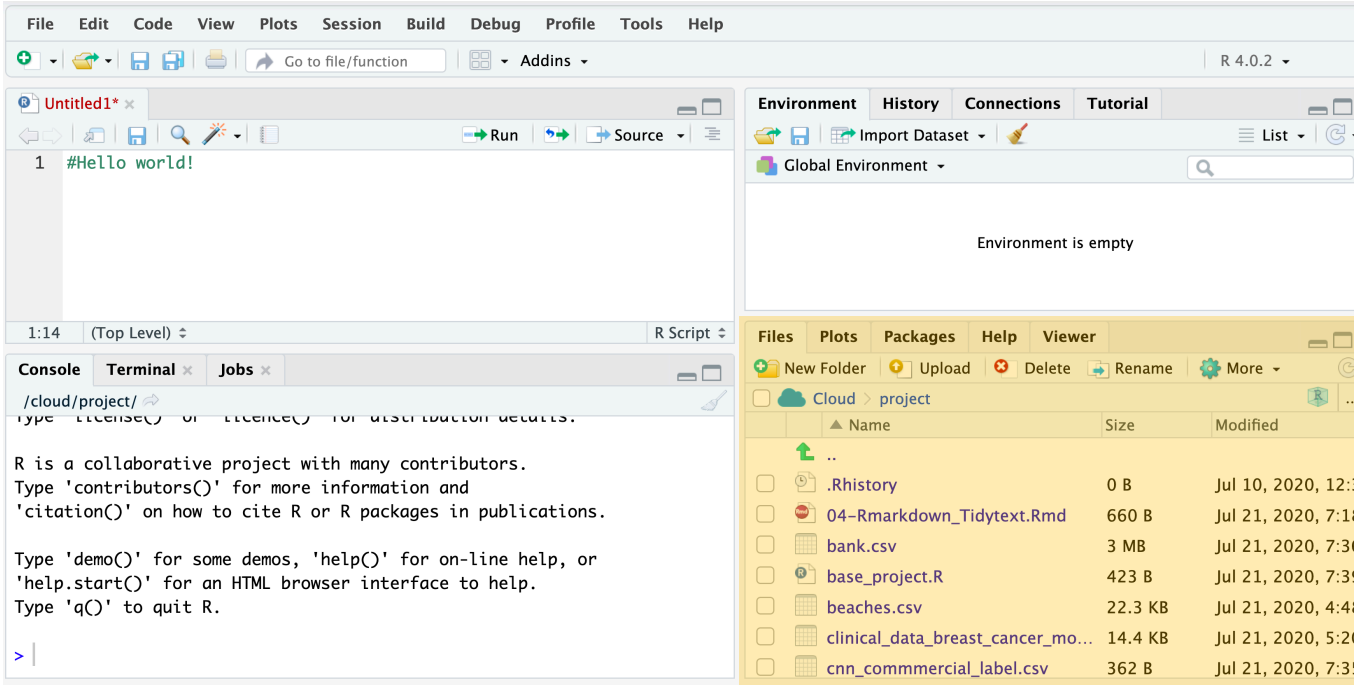
This is where R evaluates your code and displays results. You can type code (one line at a time) right in this window but most of the time we will type code in the **scripts** window just above instead.

The **Help/Plots** Window



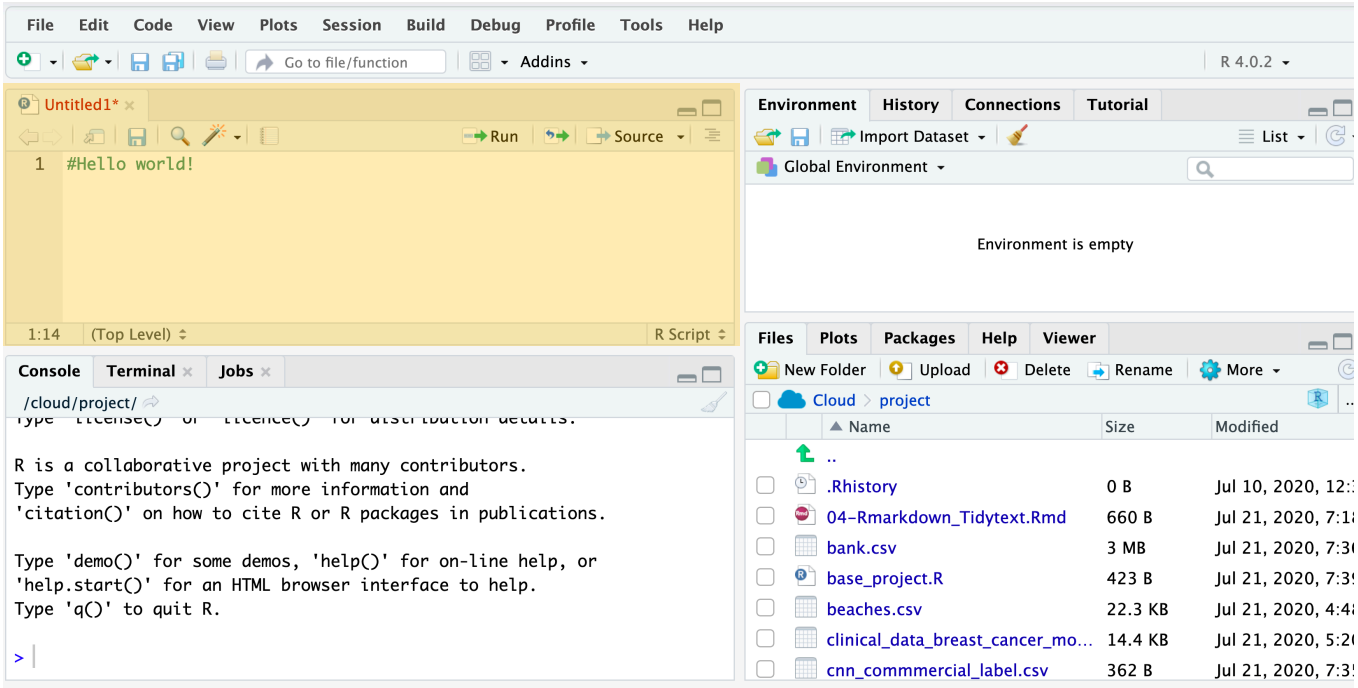
Every command in R has a **help** page, and when you call one up it displays here. This window switches to display graphics when you use a command to produce one. You can toggle between **Help** and **Plots** by clicking on the tabs at the top of this window.

The **Help/Plots** Window



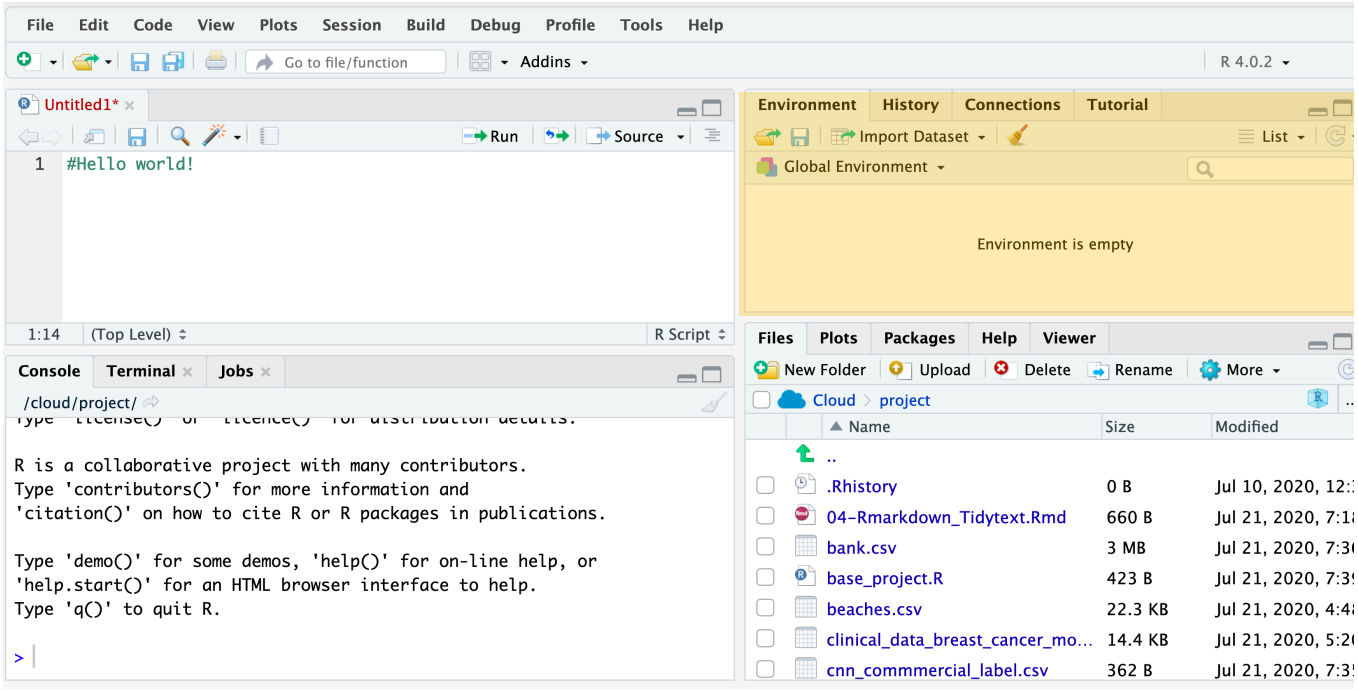
The **Files** tab shows you the files and folders that we've uploaded to R Studio Cloud for you to use. **Packages** and **Viewer** aren't used much. **Packages** shows you some of the libraries you can load. **Viewer** is a simple web browser for looking at HTML pages and graphics you might make.

The **Script** Window



This window is the one we use to write code and to save our code in a document. There are two kinds of code documents. An **R script** is a plain text file that can only contain R code. An **R Markdown file** is a more advanced file that also contains text to be displayed on a webpage. We will discuss both in more detail this afternoon.

The **Environment/History** Window



The **Environment** tab lists all of the objects that you've created in R since starting the session. **History** lists the last several commands you ran. This is the least important of the four windows.

R Scripts

An R script is a text file. That's all it is. To open a new script in R Studio, click on File, then New File, then R Script.

R Scripts

An R script is a text file. That's all it is. To open a new script in R Studio, click on File, then New File, then R Script.

The only difference between this file and a plain text file is that **you can quickly run commands** that are typed here:

R Scripts

An R script is a text file. That's all it is. To open a new script in R Studio, click on File, then New File, then R Script.

The only difference between this file and a plain text file is that **you can quickly run commands** that are typed here:

1. Highlight the part of the script you want to run. If you want the whole thing, press **command (or control) and A** on your keyboard to highlight everything.

R Scripts

An R script is a text file. That's all it is. To open a new script in R Studio, click on File, then New File, then R Script.

The only difference between this file and a plain text file is that **you can quickly run commands** that are typed here:

1. Highlight the part of the script you want to run. If you want the whole thing, press **command (or control) and A** on your keyboard to highlight everything.
2. Press **command (or control) and enter**, or click the button marked "run", to run the highlighted code.

R Scripts

An R script is a text file. That's all it is. To open a new script in R Studio, click on File, then New File, then R Script.

The only difference between this file and a plain text file is that **you can quickly run commands** that are typed here:

1. Highlight the part of the script you want to run. If you want the whole thing, press **command (or control) and A** on your keyboard to highlight everything.
2. Press **command (or control) and enter**, or click the button marked "run", to run the highlighted code.

I usually type quick commands to explore the data directly into the console. But I put all the commands that are essential for an analysis into the script, and I save this script often.

R Scripts

An R script is a text file. That's all it is. To open a new script in R Studio, click on File, then New File, then R Script.

The only difference between this file and a plain text file is that **you can quickly run commands** that are typed here:

1. Highlight the part of the script you want to run. If you want the whole thing, press **command (or control) and A** on your keyboard to highlight everything.
2. Press **command (or control) and enter**, or click the button marked "run", to run the highlighted code.

I usually type quick commands to explore the data directly into the console. But I put all the commands that are essential for an analysis into the script, and I save this script often.

R scripts are saved with the extension **.R**

R Scripts: Commenting

A comment is text in a script that is **not read as code**.

R Scripts: Commenting

A comment is text in a script that is **not read as code**.

Comments are used for

- Providing notes to collaborators

R Scripts: Commenting

A comment is text in a script that is **not read as code**.

Comments are used for

- Providing notes to collaborators
- Reminding yourself what each command does

R Scripts: Commenting

A comment is text in a script that is **not read as code**.

Comments are used for

- Providing notes to collaborators
- Reminding yourself what each command does
- Keeping notes for yourself

R Scripts: Commenting

A comment is text in a script that is **not read as code**.

Comments are used for

- Providing notes to collaborators
- Reminding yourself what each command does
- Keeping notes for yourself
- Sectioning the script to make it easier to navigate

R Scripts: Commenting

A comment is text in a script that is **not read as code**.

Comments are used for

- Providing notes to collaborators
- Reminding yourself what each command does
- Keeping notes for yourself
- Sectioning the script to make it easier to navigate
- Finding a line that produces an error by selectively commenting-out lines ("de-bugging")

R Scripts: Commenting

A comment is text in a script that is **not read as code**.

Comments are used for

- Providing notes to collaborators
- Reminding yourself what each command does
- Keeping notes for yourself
- Sectioning the script to make it easier to navigate
- Finding a line that produces an error by selectively commenting-out lines ("de-bugging")

To comment in a script, type #. Everything that appears after # on that line will not be run.

R Scripts: Commenting

A comment is text in a script that is **not read as code**.

Comments are used for

- Providing notes to collaborators
- Reminding yourself what each command does
- Keeping notes for yourself
- Sectioning the script to make it easier to navigate
- Finding a line that produces an error by selectively commenting-out lines ("de-bugging")

To comment in a script, type #. Everything that appears after # on that line will not be run.

Sometimes I put # *after* some code, to make a specific note about that line.

R is "Object Oriented" Programming

Excel, SPSS, and Stata are spreadsheet programs. **R** works with data objects of all kinds, not just spreadsheets, and R can have all these different data objects open in memory at once.

R is "Object Oriented" Programming

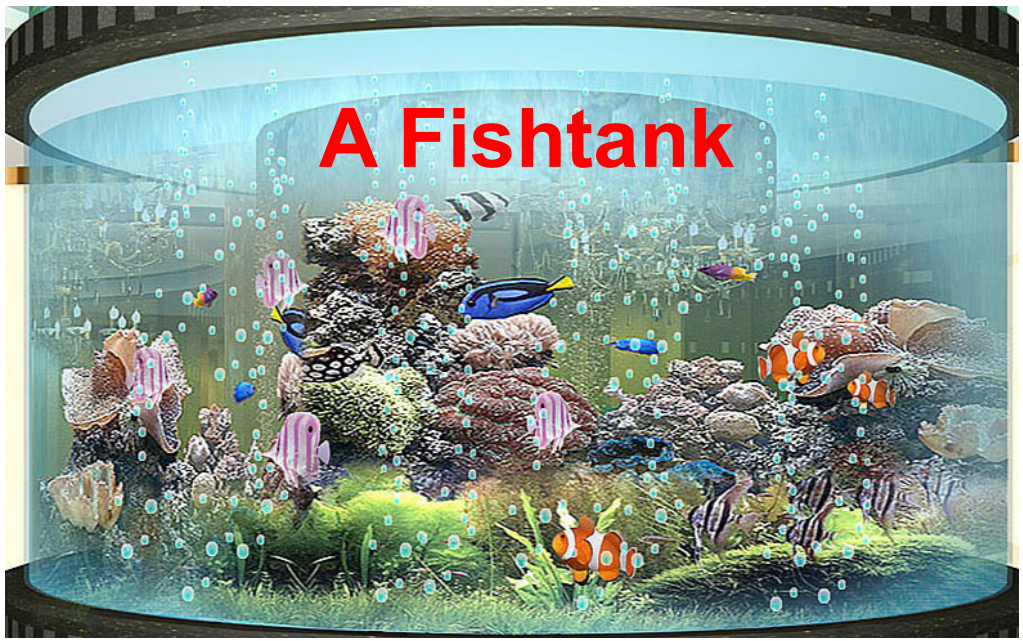
Excel, SPSS, and Stata are spreadsheet programs. **R** works with data objects of all kinds, not just spreadsheets, and R can have all these different data objects open in memory at once.

A good analogy is that R is like

R is "Object Oriented" Programming

Excel, SPSS, and Stata are spreadsheet programs. **R** works with data objects of all kinds, not just spreadsheets, and R can have all these different data objects open in memory at once.

A good analogy is that R is like



R is "Object Oriented" Programming

The "tank" is the memory on your computer (or on the Cloud) that R sets aside for data, graphs, results, etc.

R is "Object Oriented" Programming

The "tank" is the memory on your computer (or on the Cloud) that R sets aside for data, graphs, results, etc.

The "fish" are the objects. When you start R, there are no objects. But once you create an object, it gets stored in the memory that R uses. You can create as many objects as you like.

R is "Object Oriented" Programming

The "tank" is the memory on your computer (or on the Cloud) that R sets aside for data, graphs, results, etc.

The "fish" are the objects. When you start R, there are no objects. But once you create an object, it gets stored in the memory that R uses. You can create as many objects as you like.

There are many kinds of objects. Some are very big, like a gigantic dataset, or a high resolution image. Some are very little, like a 2×2 table, or a single number.

R is "Object Oriented" Programming

The "tank" is the memory on your computer (or on the Cloud) that R sets aside for data, graphs, results, etc.

The "fish" are the objects. When you start R, there are no objects. But once you create an object, it gets stored in the memory that R uses. You can create as many objects as you like.

There are many kinds of objects. Some are very big, like a gigantic dataset, or a high resolution image. Some are very little, like a 2×2 table, or a single number.

Objects can be

- numbers
- words
- logical values
- lists
- vectors
- matrices
- tables of results
- whole datasets
- graphics

Creating Objects

Every object in R must have a **name**. Every object in R must contain **information of some kind**.

Creating Objects

Every object in R must have a **name**. Every object in R must contain **information of some kind**.

The basic syntax for creating an object is

```
name <- information
```

The assignment operator <- is an arrow. It means "the information goes into an object named" whatever you've written as the name.

Creating Objects

Every object in R must have a **name**. Every object in R must contain **information of some kind**.

The basic syntax for creating an object is

```
name <- information
```

The assignment operator <- is an arrow. It means "the information goes into an object named" whatever you've written as the name.

Careful: the arrow must point to the left.

Creating Objects

Every object in R must have a **name**. Every object in R must contain **information of some kind**.

The basic syntax for creating an object is

```
name <- information
```

The assignment operator <- is an arrow. It means "the information goes into an object named" whatever you've written as the name.

Careful: the arrow must point to the left.

Some people use equal signs instead, `name = information`, but I recommend against that. It's too easy to confuse this equal sign with the ones we use for function arguments and logic.

Creating Objects

Simple examples: assign single numbers to objects, like this:

Creating Objects

Simple examples: assign single numbers to objects, like this:

- `x <- 7`
- `y <- -3`
- `z <- 2.134`
- `howdy <- 1708`
- `theAnswer <- 42`
- `problems <- 99`
- `ways.to.leave.your.lover <- 50`
- `uva_student_enrollment <- 22391`

Creating Objects

Simple examples: assign single numbers to objects, like this:

- `x <- 7`
- `y <- -3`
- `z <- 2.134`
- `howdy <- 1708`
- `theAnswer <- 42`
- `problems <- 99`
- `ways.to.leave.your.lover <- 50`
- `uva_student_enrollment <- 22391`

No spaces or dashes in object names. Dots, underscores, and capital letters are fine (but remember that object names are case sensitive).

Creating Objects

Simple examples: assign single numbers to objects, like this:

- `x <- 7`
- `y <- -3`
- `z <- 2.134`
- `howdy <- 1708`
- `theAnswer <- 42`
- `problems <- 99`
- `ways.to.leave.your.lover <- 50`
- `uva_student_enrollment <- 22391`

No spaces or dashes in object names. Dots, underscores, and capital letters are fine (but remember that object names are case sensitive).

Chose names that are descriptive but not too hard to type.

Creating Objects

Now that you've defined these objects, you can use them. If you type the object name next to the $>$ prompt in the console, it reports the contents of the object.

```
> theAnswer
[1] 42
```

Creating Objects

Now that you've defined these objects, you can use them. If you type the object name next to the > prompt in the console, it reports the contents of the object.

```
> theAnswer
[1] 42
```

You can also do simple math with objects like this. Some of the **math operators** are:

Creating Objects

Now that you've defined these objects, you can use them. If you type the object name next to the > prompt in the console, it reports the contents of the object.

```
> theAnswer
[1] 42
```

You can also do simple math with objects like this. Some of the **math operators** are:

- `+` addition
- `-` subtraction or negation
- `*` multiplication
- `/` division
- `^` exponentiation
- `sqrt()` square root
- `log()` natural log
- `log10()` log-base-10
- `exp()` *e* to the power of

Creating Objects

Now that you've defined these objects, you can use them. If you type the object name next to the > prompt in the console, it reports the contents of the object.

```
> theAnswer
[1] 42
```

You can also do simple math with objects like this. Some of the **math operators** are:

- + addition
- - subtraction or negation
- * multiplication
- / division
- ^ exponentiation
- `sqrt()` square root
- `log()` natural log
- `log10()` log-base-10
- `exp()` *e* to the power of

```
> theAnswer*problems
[1] 4158
```

Vectors

A vector is a one-dimensional list of values which are often (but not necessarily) numbers. Vectors help you work with lots of data at once.

Vectors

A vector is a one-dimensional list of values which are often (but not necessarily) numbers. Vectors help you work with lots of data at once.

To define a vector, use the `c()` command:

Vectors

A vector is a one-dimensional list of values which are often (but not necessarily) numbers. Vectors help you work with lots of data at once.

To define a vector, use the `c()` command:

```
starWarsEps <- c(4, 5, 6, 1, 2, 3, 7, 3.5, 8)
```

Vectors

A vector is a one-dimensional list of values which are often (but not necessarily) numbers. Vectors help you work with lots of data at once.

To define a vector, use the `c()` command:

```
starWarsEps <- c(4, 5, 6, 1, 2, 3, 7, 3.5, 8)
```

You can do math on the whole vector at once:

```
> starWarsEps^2
[1] 16.00 25.00 36.00  1.00  4.00  9.00 49.00 12.25 64.00
```

Vectors

A vector is a one-dimensional list of values which are often (but not necessarily) numbers. Vectors help you work with lots of data at once.

To define a vector, use the `c()` command:

```
starWarsEps <- c(4, 5, 6, 1, 2, 3, 7, 3.5, 8)
```

You can do math on the whole vector at once:

```
> starWarsEps^2
[1] 16.00 25.00 36.00  1.00  4.00  9.00 49.00 12.25 64.00
```

To refer to a particular value, use brackets around the position of the value you want. For example, the 4th Star Wars movie was Episode 1:

```
> starWarsEps[4]
[1] 1
```

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Some of the classes are:

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Some of the classes are:

- `numeric` — for single numbers or vectors of numbers

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Some of the classes are:

- `numeric` — for single numbers or vectors of numbers
- `logical` — for values that can only be either `TRUE` or `FALSE`

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Some of the classes are:

- `numeric` — for single numbers or vectors of numbers
- `logical` — for values that can only be either `TRUE` or `FALSE`
- `character` — for words

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Some of the classes are:

- `numeric` — for single numbers or vectors of numbers
- `logical` — for values that can only be either `TRUE` or `FALSE`
- `character` — for words
- `factor` — for categories, which may or may not be ordered

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Some of the classes are:

- `numeric` — for single numbers or vectors of numbers
- `logical` — for values that can only be either `TRUE` or `FALSE`
- `character` — for words
- `factor` — for categories, which may or may not be ordered
- `matrix` — for matrices

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Some of the classes are:

- `numeric` — for single numbers or vectors of numbers
- `logical` — for values that can only be either `TRUE` or `FALSE`
- `character` — for words
- `factor` — for categories, which may or may not be ordered
- `matrix` — for matrices
- `table` — like matrices, but with more functionality for clean display

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Some of the classes are:

- `numeric` — for single numbers or vectors of numbers
- `logical` — for values that can only be either `TRUE` or `FALSE`
- `character` — for words
- `factor` — for categories, which may or may not be ordered
- `matrix` — for matrices
- `table` — like matrices, but with more functionality for clean display
- `data.frame` — a matrix or spreadsheet that contains a dataset. **This is the most important kind of object in this course.**

Object Classes

There are different types of objects for different kinds of information. The object's type is called its **class**.

Some of the classes are:

- `numeric` — for single numbers or vectors of numbers
- `logical` — for values that can only be either `TRUE` or `FALSE`
- `character` — for words
- `factor` — for categories, which may or may not be ordered
- `matrix` — for matrices
- `table` — like matrices, but with more functionality for clean display
- `data.frame` — a matrix or spreadsheet that contains a dataset. **This is the most important kind of object in this course.**
- `list` — a loose, ordered collection of other objects

Object Classes

The `class()` command lets you see the class of an object. R will try to guess the object's class when you define it:

```
> CLEcavs <- c("LeBron", "Kyrie", "KLove", "JR", "TT")
> class(CLEcavs)
[1] "character"
```

Object Classes

The `class()` command lets you see the class of an object. R will try to guess the object's class when you define it:

```
> CLEcavs <- c("LeBron", "Kyrie", "KLove", "JR", "TT")
> class(CLEcavs)
[1] "character"
```

Sometimes R gets it wrong. I want the following **categorical** object to be read as a `factor`:

```
> dem.types <- c("FPTP", "PR", "PR", "FPTP")
> class(dem.types)
[1] "character"
```

Object Classes

Changing an object from one class to another is called **coercing** the object.

Object Classes

Changing an object from one class to another is called **coercing** the object.

To coerce a character vector to a factor, use the `as.factor()` command:

```
> dem.types <- as.factor(dem.types)
> class(dem.types)
[1] "factor"
> dem.types
[1] FPTP PR   PR   FPTP
Levels: FPTP PR
```

Object Classes

Changing an object from one class to another is called **coercing** the object.

To coerce a character vector to a factor, use the `as.factor()` command:

```
> dem.types <- as.factor(dem.types)
> class(dem.types)
[1] "factor"
> dem.types
[1] FPTP PR   PR   FPTP
Levels: FPTP PR
```

If you coerce an object, ALWAYS CHECK TO MAKE SURE IT WORKED. Sometimes R can't coerce. You might get an error. You might not get an error but the object will be wonky and wrong.

Object Classes: Data frames

A data frame is the same thing as a single page spreadsheet in Microsoft Excel, or a single dataset in Stata. Unlike Stata, **R can work with many data frames** at once.

Object Classes: Data frames

A data frame is the same thing as a single page spreadsheet in Microsoft Excel, or a single dataset in Stata. Unlike Stata, **R can work with many data frames** at once.

You can define a data frame directly with the `data.frame()` command.

```
> sandwich <- data.frame(grp=c("sweet", "savory", "French"),
        type=c("PBJ", "Hoogie", "Croque Monsieur"),
        rating=c(10,7,8))
> sandwich
     grp             type rating
1  sweet              PBJ     10
2 savory           Hoogie      7
3 French Croque Monsieur      8
```

Object Classes: Data frames

Most of the time, however, you will **load a data frame from an outside file**, instead of enter one directly.

Object Classes: Data frames

Most of the time, however, you will **load a data frame from an outside file**, instead of enter one directly.

If the data is saved as a text file, you can use the `read.csv()` command to open it. We will get into more detail later, but here's a simple example:

```
nats <- read.csv("nationals.csv")
```

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

- `dim(nats)` — report on the number of rows and columns

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

- `dim(nats)` — report on the number of rows and columns
- `names(nats)` — see all of the variable names

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

- `dim(nats)` — report on the number of rows and columns
- `names(nats)` — see all of the variable names
- `summary(nats)` — get quick summary stats for every variable

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

- `dim(nats)` — report on the number of rows and columns
- `names(nats)` — see all of the variable names
- `summary(nats)` — get quick summary stats for every variable
- `str(nats)` — see the **class** and the first several values of each variable

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

- `dim(nats)` — report on the number of rows and columns
- `names(nats)` — see all of the variable names
- `summary(nats)` — get quick summary stats for every variable
- `str(nats)` — see the **class** and the first several values of each variable
- `head(nats)` — see the first 5 rows for every variable

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

- `dim(nats)` — report on the number of rows and columns
- `names(nats)` — see all of the variable names
- `summary(nats)` — get quick summary stats for every variable
- `str(nats)` — see the **class** and the first several values of each variable
- `head(nats)` — see the first 5 rows for every variable
- `nats[,17]` — see the entire 17th column

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

- `dim(nats)` — report on the number of rows and columns
- `names(nats)` — see all of the variable names
- `summary(nats)` — get quick summary stats for every variable
- `str(nats)` — see the **class** and the first several values of each variable
- `head(nats)` — see the first 5 rows for every variable
- `nats[,17]` — see the entire 17th column
- `nats$HR` — see the entire variable named `HR`

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

- `dim(nats)` — report on the number of rows and columns
- `names(nats)` — see all of the variable names
- `summary(nats)` — get quick summary stats for every variable
- `str(nats)` — see the **class** and the first several values of each variable
- `head(nats)` — see the first 5 rows for every variable
- `nats[,17]` — see the entire 17th column
- `nats$HR` — see the entire variable named `HR`
- `View(nats)` — see the data spreadsheet displayed

Object Classes: Data frames

Here are some things you can do with data frames. Try these on your computer:

- `dim(nats)` — report on the number of rows and columns
- `names(nats)` — see all of the variable names
- `summary(nats)` — get quick summary stats for every variable
- `str(nats)` — see the **class** and the first several values of each variable
- `head(nats)` — see the first 5 rows for every variable
- `nats[,17]` — see the entire 17th column
- `nats$HR` — see the entire variable named `HR`
- `View(nats)` — see the data spreadsheet displayed

These are not all designed to be pretty, but they all are useful for quick information.

I usually work through all of these as soon as I load a new dataset.

R Packages

An R package is a zipped folder with a bunch of R scripts that define new commands. Packages can be downloaded from the internet directly and loaded into an R session.

R Packages

An R package is a zipped folder with a bunch of R scripts that define new commands. Packages can be downloaded from the internet directly and loaded into an R session.

If R can't do something, there's an excellent chance that you can **download a package** to give R the ability to do that thing.

R Packages

An R package is a zipped folder with a bunch of R scripts that define new commands. Packages can be downloaded from the internet directly and loaded into an R session.

If R can't do something, there's an excellent chance that you can **download a package** to give R the ability to do that thing.

This is one of the greatest things about R. R's functionality increases VERY rapidly. New packages are released every day, so often it is hard to keep up. There are reputable journals devoted entirely to documentation for new R packages.

R Packages

Packages are stored on The Comprehensive R Archive Network (CRAN): https://cran.r-project.org/. You can see a list of the available packages sorted by name or the date the package was lasted updated.

R Packages

Packages are stored on The Comprehensive R Archive Network (CRAN): https://cran.r-project.org/. You can see a list of the available packages sorted by name or the date the package was lasted updated.

To download a package from CRAN, find the name of a package you want to install and type the following into the console:

```
install.packages("package")
```

R Packages

Packages are stored on The Comprehensive R Archive Network (CRAN): https://cran.r-project.org/. You can see a list of the available packages sorted by name or the date the package was lasted updated.

To download a package from CRAN, find the name of a package you want to install and type the following into the console:

```
install.packages("package")
```

You only need to install a package **once ever** on a computer (until you update to a new version of R).

R Packages

Packages are stored on The Comprehensive R Archive Network (CRAN): https://cran.r-project.org/. You can see a list of the available packages sorted by name or the date the package was lasted updated.

To download a package from CRAN, find the name of a package you want to install and type the following into the console:

```
install.packages("package")
```

You only need to install a package **once ever** on a computer (until you update to a new version of R).

To load an installed package into your R session type

```
library(package)
```

in your script, console, or markdown file. You need to do this **once for every R session**.

R Packages

The website fivethirtyeight.com writes about politics, sports, and culture from a data point-of-view. They have an R package just for sharing the data they use for their articles.

R Packages

The website fivethirtyeight.com writes about politics, sports, and culture from a data point-of-view. They have an R package just for sharing the data they use for their articles.

To download the `fivethirtyeight` package from CRAN, for example, type `install.packages("fivethirtyeight")`.

R Packages

The website fivethirtyeight.com writes about politics, sports, and culture from a data point-of-view. They have an R package just for sharing the data they use for their articles.

To download the `fivethirtyeight` package from CRAN, for example, type `install.packages("fivethirtyeight")`.

Once the package is installed, open it in R by typing `library(fivethirtyeight)` in the script. **Now you can use all of the commands in this package.**

R Packages

The website fivethirtyeight.com writes about politics, sports, and culture from a data point-of-view. They have an R package just for sharing the data they use for their articles.

To download the `fivethirtyeight` package from CRAN, for example, type `install.packages("fivethirtyeight")`.

Once the package is installed, open it in R by typing `library(fivethirtyeight)` in the script. **Now you can use all of the commands in this package.**

To see the commands available in a package you've installed, use the `help()` command in the console, like this:

```
help(package="fivethirtyeight")
```

Getting Help: R Help and R Search

R help pages and **R search pages** are built into R. If you know the command you need, but don't know how to use it, type a `?` in front of the command name.

Getting Help: R Help and R Search

R help pages and **R search pages** are built into R. If you know the command you need, but don't know how to use it, type a `?` in front of the command name.

If you don't know the command you need, type `??` and a word that seems close to what you need.

Getting Help: R Help and R Search

R help pages and **R search pages** are built into R. If you know the command you need, but don't know how to use it, type a `?` in front of the command name.

If you don't know the command you need, type `??` and a word that seems close to what you need.

The command to run a regression is `lm()`. To see the help page for this command, type `?lm`. If you didn't know about `lm()`, you could discover it by typing something like `??linearmodel`.

Getting Help: R Help and R Search

R help pages and **R search pages** are built into R. If you know the command you need, but don't know how to use it, type a `?` in front of the command name.

If you don't know the command you need, type `??` and a word that seems close to what you need.

The command to run a regression is `lm()`. To see the help page for this command, type `?lm`. If you didn't know about `lm()`, you could discover it by typing something like `??linearmodel`.

The search page shows you commands from different packages, code demonstrations (example R scripts), and vignettes (short papers).

Getting Help: Reading an R Help Page

An R help page looks intimidating. But it has a particular anatomy. **Knowing how help pages are organized will make it much easier to use them**.

Getting Help: Reading an R Help Page

An R help page looks intimidating. But it has a particular anatomy. **Knowing how help pages are organized will make it much easier to use them**.

Part 1: command name and package

The name in the curly braces is the name of the package that provides this command to R. We will talk later about getting new packages. The `lm()` command, for example, comes from the **stats** package, which is one of the ones that is included with R when you first download it.

Getting Help: Reading an R Help Page

An R help page looks intimidating. But it has a particular anatomy. **Knowing how help pages are organized will make it much easier to use them**.

Part 1: command name and package

The name in the curly braces is the name of the package that provides this command to R. We will talk later about getting new packages. The `lm()` command, for example, comes from the **stats** package, which is one of the ones that is included with R when you first download it.

Part 2: short and long description

There will be a few words in bold that give you a short description of the command. Then a brief paragraph summarizing what this command does.

Getting Help: Reading an R Help Page

Part 3: usage

This is the "full" version of the command, complete with every single option available for the command, and the default values if you don't change the option.
(Some commands also have "..." as an option — which just means that it can accept arguments from other commands that call this one. That's not something we need to worry about now.)

Getting Help: Reading an R Help Page

Part 3: usage

This is the "full" version of the command, complete with every single option available for the command, and the default values if you don't change the option.
(Some commands also have "..." as an option — which just means that it can accept arguments from other commands that call this one. That's not something we need to worry about now.)

Part 4: arguments

Every argument is listed here with a sentence or two describing what the option does and what values the option can be set to.

Getting Help: Reading an R Help Page

Part 5: details

This is a much longer description of the command. It might get into the statistics that underlie the command, and might display equations.

Getting Help: Reading an R Help Page

Part 5: details

This is a much longer description of the command. It might get into the statistics that underlie the command, and might display equations.

Part 6: value

Many commands, like `lm()`, will output results in a big list, with many components. The items of this list are described here.

Getting Help: Reading an R Help Page

Part 5: details

This is a much longer description of the command. It might get into the statistics that underlie the command, and might display equations.

Part 6: value

Many commands, like `lm()`, will output results in a big list, with many components. The items of this list are described here.

Part 7: additional details and notes.

If there are particular dangers or problems to be aware of, the authors of the command will discuss those here.

Getting Help: Reading an R Help Page

Part 8: authors and references

Getting Help: Reading an R Help Page

Part 8: authors and references

Part 9: see also

A list of similar commands, in case this one isn't quite right. It includes hyperlinks to the help pages for these other commands.

Getting Help: Reading an R Help Page

Part 8: authors and references

Part 9: see also

A list of similar commands, in case this one isn't quite right. It includes hyperlinks to the help pages for these other commands.

Part 10: examples.

Examples are designed to be run, not looked at. Copy and paste the code into the console and see what happens. Unless otherwise stated, these examples are completely self-contained and are ready to run.

Getting Help: Reading an R Help Page

Part 8: authors and references

Part 9: see also

A list of similar commands, in case this one isn't quite right. It includes hyperlinks to the help pages for these other commands.

Part 10: examples.

Examples are designed to be run, not looked at. Copy and paste the code into the console and see what happens. Unless otherwise stated, these examples are completely self-contained and are ready to run.

Part 11: index. There is a link to the package itself. If you click on it, you will see an index of help pages for every command in this package.

Exercise

I know that the `write.table()` function can be used to output data to a external file to share with other people. But I don't yet know *how* to use this function.

1. Call up the help page for the `write.table()` function.

2. Suppose I want to label the rows in the data file I save. Does there appear to be a way to do that with this function?