Day 3, Session 1: R and RStudio basics

Jessica Williams-Nguyen and Brian D. Williamson

EPI/BIOST Bootcamp 2018

25 September 2018

Learning objectives

By the end of this session, you should be able to

- **organize** your files for a data analysis
- **download** data from the internet
- **load** data into an R workspace
- **explore** your data (subset, index, plot, summarize)
- **load** R packages

Goals

R and RStudio are powerful tools that make analyzing data and doing reproducible research easier (in the long run).

However, before you become familiar with these tools, they may be frustrating to use.

Goals

R and RStudio are powerful tools that make analyzing data and doing reproducible research easier (in the long run).

However, before you become familiar with these tools, they may be frustrating to use.

This module is meant to give you a space to engage with R and RStudio prior to diving into coursework. **All R functions required for your courses will be introduced in the specific course**.

Organization: helping future you

The first step in a data analysis isn't opening R or RStudio – instead, you should create a workspace for your analysis.

Organization: helping future you

The first step in a data analysis isn't opening R or RStudio – instead, you should create a workspace for your analysis.

This workspace is a **folder** on your computer that:

Organization: helping future you

The first step in a data analysis isn't opening R or RStudio – instead, you should create a workspace for your analysis.

This workspace is a **folder** on your computer that:

- is easy for you to find **now**

Organization: helping future you

The first step in a data analysis isn't opening R or RStudio – instead, you should create a workspace for your analysis.

This workspace is a **folder** on your computer that:

- is easy for you to find **now**
- will be easy for you to find in the **future**

Organization: helping future you

The first step in a data analysis isn't opening R or RStudio – instead, you should create a workspace for your analysis.

This workspace is a **folder** on your computer that:

- is easy for you to find **now**
- will be easy for you to find in the **future**
- will hold **all relevant files** for your analysis (e.g., the data file, R code, plots, results)

Organization: developing your workflow

It is important that you develop your own workflow, since *easy to find* means different things to everyone.

My workflow looks something like this:

Organization: developing your workflow

It is important that you develop your own workflow, since *easy to find* means different things to everyone.

My workflow looks something like this:

1. get involved in a new project

Organization: developing your workflow

It is important that you develop your own workflow, since *easy to find* means different things to everyone.

My workflow looks something like this:

1. get involved in a new project
2. set up a new folder for the project
    - under *Courses* if it relates to a course, e.g., `Courses/BIOST511`
    - under *the name of the project* if not, e.g., `Projects/HPTN063` refers to a clinical trial I have worked on

Organization: developing your workflow

It is important that you develop your own workflow, since *easy to find* means different things to everyone.

My workflow looks something like this:

1. get involved in a new project
2. set up a new folder for the project
   - under *Courses* if it relates to a course, e.g., `Courses/BIOST511`
   - under *the name of the project* if not, e.g., `Projects/HPTN063` refers to a clinical trial I have worked on
3. store all documents, data, code, and output in this folder (with subfolders for each of these)

Downloading data

The datasets for your coursework will typically be found either

- on **Canvas**, the UW's file-hosting site for courses, or
- on your professor's webpage, or
- on the webpage for a textbook

Your instructor will tell you **exactly where** to find the datasets necessary for your coursework.

Once you have set up a folder (workspace) for your project, download the data into this folder.

Exercise: downloading data

Create a new folder titled `fev_analysis` in the folder you are using for the files for this workshop.

Next, go to the Canvas webpage for this workshop and download the file `fev.txt`, located in the `Data` folder under the `Files` tab. Download these data into the `fev_analysis` folder.

Using R and RStudio

Once you have set up a workspace on your computer for your new project, and downloaded your data, it's time to use R!

Using R and RStudio

Once you have set up a workspace on your computer for your new project, and downloaded your data, it's time to use R!

R is a statistical programming language, and does all of the computing for your analysis (e.g., plots, summaries).

RStudio is a program that makes it easier to interact with R to perform your analysis.

Using R and RStudio

Once you have set up a workspace on your computer for your new project, and downloaded your data, it's time to use R!

R is a statistical programming language, and does all of the computing for your analysis (e.g., plots, summaries).

RStudio is a program that makes it easier to interact with R to perform your analysis.

I suggest that you **only interact with R through RStudio** – this is all you need for your Biostatistics coursework.

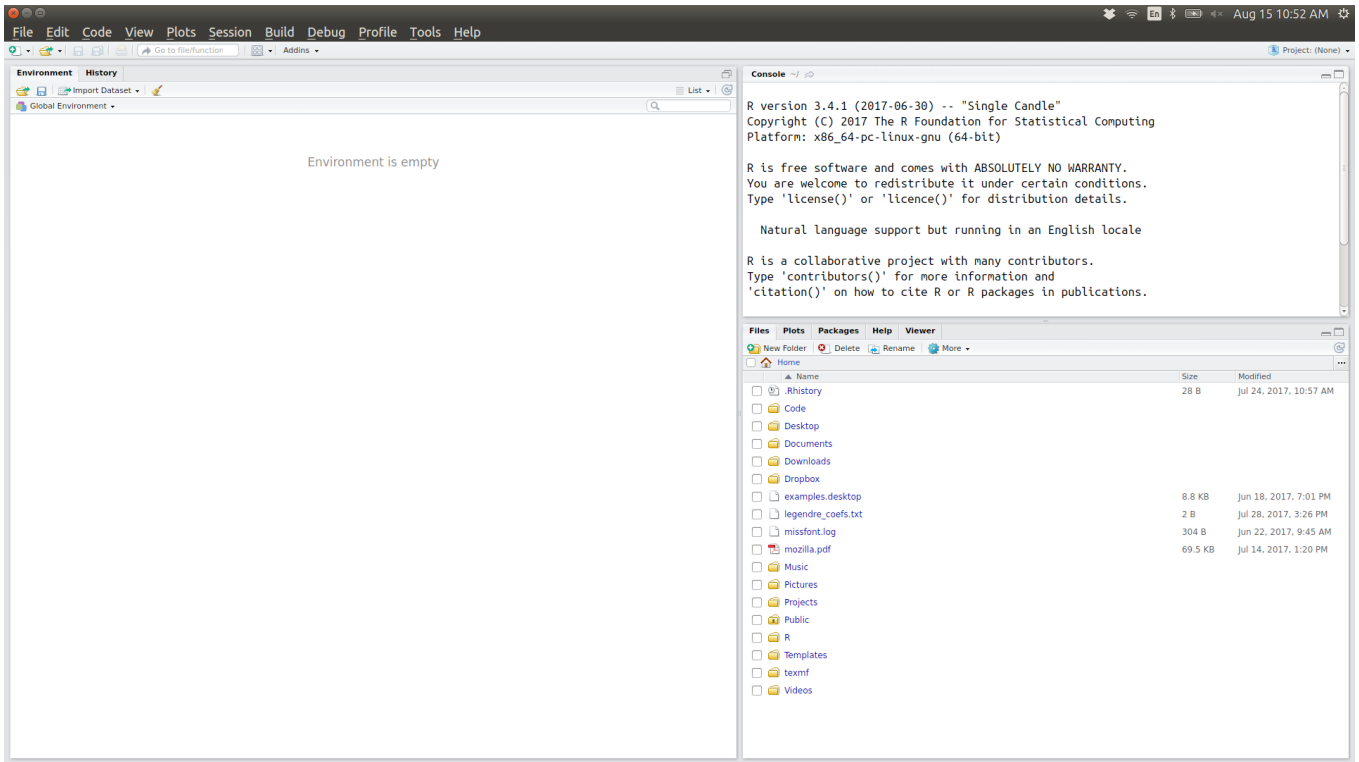
Using R and RStudio: the interface

We interact with both R and RStudio using an interface. For R, this is the command line. For RStudio, this is the graphical user interface.

Both of these concepts deserve a bit of introduction, since familiarity with the interface makes using R and RStudio much easier!

Using R and RStudio: RStudio

When you first open RStudio, you are met with a blank set of four panes:



Using R and RStudio: RStudio

You can edit pane layout under `Tools > Global Options > Pane Layout`. My preference is to have the source in the upper left, console in the upper right, environment in the lower left, and viewer in the lower right.

You'll notice that on the previous slide, there were only three panes visible: since we haven't opened or created any files to edit, the source pane is not currently open.

RStudio: the console

The console is our interface to the R command line.

Any commands that we wish to enter go through the console, and any results that are output by these commands will appear in the console.

Example: the console

Typing `47` at the > symbol (from now on, called the execution line) and hitting `Enter` on the keyboard yields the following:

```
> 47
[1] 47
```

Example: the console

Typing `47` at the > symbol (from now on, called the execution line) and hitting `Enter` on the keyboard yields the following:

```
> 47
[1] 47
```

Since R is a functional programming language, typing `47` and hitting `Enter` is the same thing as using the `print()` function:

```
> print(47)
[1] 47
```

Example: the console

Typing `47` at the > symbol (from now on, called the execution line) and hitting `Enter` on the keyboard yields the following:

```
> 47
[1] 47
```

Since R is a functional programming language, typing `47` and hitting `Enter` is the same thing as using the `print()` function:

```
> print(47)
[1] 47
```

The output is displayed as a vector, one of the fundamental data structures in R. The `[1]` helps to tell which element of the vector we are looking at (which is most useful if the vector spans multiple lines in the console).

Exercise: the console

Use the RStudio console to perform the following computations. Write down your answers and your process for arriving at your answers.

1. Multiply 55 and 389
2. Divide 500 by 15
3. Add 1525 and 3225

Exercise: the console

You just used your first R functions!

The *, + and −, and / symbols on your keyboard operate (unsurprisingly) as R functions to multiply, add and subtract, and divide.

RStudio: using the console

Anything that you type into the console is a part of the **current** R session – if you close RStudio and reopen it, these commands are not saved.

RStudio: using the console

Anything that you type into the console is a part of the **current** R session – if you close RStudio and reopen it, these commands are not saved.

For more complicated analyses, this can lead to a lot of confusion. Also, typing only in the console can lead you to forget what steps you took to get a certain figure or table.

RStudio: using the console

Anything that you type into the console is a part of the **current** R session – if you close RStudio and reopen it, these commands are not saved.

For more complicated analyses, this can lead to a lot of confusion. Also, typing only in the console can lead you to forget what steps you took to get a certain figure or table.

Organizing commands into scripts allows you to save exactly what you did during a given data analysis, which makes reproducible research easy. This helps both your future self and your collaborators, so that you can share what you did!

RStudio: the text editor ("Source")

The text editor (which RStudio calls the source) is the second most powerful part of the IDE. This allows us to edit and save files, and to run commands directly from the text file into the console.

RStudio: the text editor ("Source")

The text editor (which RStudio calls the source) is the second most powerful part of the IDE. This allows us to edit and save files, and to run commands directly from the text file into the console.

To create a new R script, either type `Ctrl+Shift+N` or go to `File > New > R script`. There are many other options here (most notably R markdown, which we won't cover but is incredibly useful). This brings up the source pane.

RStudio: the text editor ("Source")

The text editor (which RStudio calls the source) is the second most powerful part of the IDE. This allows us to edit and save files, and to run commands directly from the text file into the console.

To create a new R script, either type `Ctrl+Shift+N` or go to `File > New > R script`. There are many other options here (most notably R markdown, which we won't cover but is incredibly useful). This brings up the source pane.

At the top of the source pane are a set of buttons (with associated keyboard shortcuts) that will save the current file, run sections of code, or run the entire document.

R scripts

R scripts are collections of R commands and comments. These are run in the console to manipulate objects and produce output.

R scripts

R scripts are collections of R commands and comments. These are run in the console to manipulate objects and produce output.

To create a comment, type a # symbol. Anything on a line following a # will not be run by R.

R scripts

R scripts are collections of R commands and comments. These are run in the console to manipulate objects and produce output.

To create a comment, type a # symbol. Anything on a line following a # will not be run by R.

I like to include a preamble in all of my R scripts, which gives me information on when I created the file, what it does, whether or not it takes any input or produces output, and what I have changed since I started. Here is a sample:

R scripts

R scripts are collections of R commands and comments. These are run in the console to manipulate objects and produce output.

To create a comment, type a # symbol. Anything on a line following a # will not be run by R.

I like to include a preamble in all of my R scripts, which gives me information on when I created the file, what it does, whether or not it takes any input or produces output, and what I have changed since I started. Here is a sample:

```
##############################################################################
##
## FILE: <insert file name here>.R
##
## CREATED: <insert date here> by <your name here>
##
## PURPOSE: <Give a brief description of what the code in this file is supposed to do>
##
## INPUTS: <What inputs? Where does the data come from?>
##
## OUTPUTS: <What outputs? Plot/table names and locations>
##
## UPDATES:
## DDMMYY INIT COMMENTS
## ------ ---- --------
## <date> <inits> <What did you change?>
##############################################################################
```

Running commands from R scripts

There are many options for running commands using your R script, either in individual lines or in blocks.

You can run individual lines by placing your cursor on the line and

Running commands from R scripts

There are many options for running commands using your R script, either in individual lines or in blocks.

You can run individual lines by placing your cursor on the line and

- hitting the green `Run` button, or

Running commands from R scripts

There are many options for running commands using your R script, either in individual lines or in blocks.

You can run individual lines by placing your cursor on the line and

- hitting the green `Run` button, or
- hitting `Ctrl+Enter` (Windows/Linux) or `Cmd+Enter` (Mac)

Running commands from R scripts

There are many options for running commands using your R script, either in individual lines or in blocks.

You can run individual lines by placing your cursor on the line and

- hitting the green `Run` button, or
- hitting `Ctrl+Enter` (Windows/Linux) or `Cmd+Enter` (Mac)

Similarly, you run multiple lines by highlighting all desired lines and doing one of the two options described above.

Example: heights and weights

Enter some data on made-up heights and weights of five individuals:

Example: heights and weights

Enter some data on made-up heights and weights of five individuals:

```
ht <- c(72,65,84,73,68)
wt <- c(165,120,210,180,125)
```

Example: heights and weights

Enter some data on made-up heights and weights of five individuals:

```
ht <- c(72,65,84,73,68)
wt <- c(165,120,210,180,125)
```

into a new R script. The `c()` function concatenates data together, creating an object called a vector.

Example: heights and weights

Enter some data on made-up heights and weights of five individuals:

```
ht <- c(72,65,84,73,68)
wt <- c(165,120,210,180,125)
```

into a new R script. The `c()` function concatenates data together, creating an object called a vector.

Save this R script as `ht_wt_analysis.R`. This is a start to analyzing these data! The script now looks like this:

Example: heights and weights

Enter some data on made-up heights and weights of five individuals:

`ht <- c(72,65,84,73,68)`

`wt <- c(165,120,210,180,125)`

into a new R script. The `c()` function concatenates data together, creating an object called a vector.

Save this R script as `ht_wt_analysis.R`. This is a start to analyzing these data! The script now looks like this:

```
################################################################################
##
## FILE: ht_wt_analysis.R
##
## CREATED: 24 September 2018 by Brian Williamson
##
## PURPOSE: Learn some R basics using height and weight data on 5 imaginary individuals
##
## INPUTS: None
##
## OUTPUTS: None
##
## UPDATES:
## DDMMYY INIT COMMENTS
## ------ ---- --------
################################################################################

ht <- c(72,65,84,73,68)
wt <- c(165,120,210,180,125)
```

RStudio: environment, history, etc.

The basic environment pane allows us to do three things:

RStudio: environment, history, etc.

The basic environment pane allows us to do three things:

- view which objects (data sets, etc.) exist in the `Global Environment`

RStudio: environment, history, etc.

The basic environment pane allows us to do three things:

- view which objects (data sets, etc.) exist in the `Global Environment`
- view the history of which commands were entered in the console

RStudio: environment, history, etc.

The basic environment pane allows us to do three things:

- view which objects (data sets, etc.) exist in the `Global Environment`
- view the history of which commands were entered in the console
- import datasets (using the buttons on the top of the pane)

RStudio: environment, history, etc.

The basic environment pane allows us to do three things:

- view which objects (data sets, etc.) exist in the `Global Environment`
- view the history of which commands were entered in the console
- import datasets (using the buttons on the top of the pane)

The `Global Environment` is where commands that are executed in the console typically live — intermediate objects created within functions (but not returned) live within function-specific environments.

RStudio: environment, history, etc.

The basic environment pane allows us to do three things:

- view which objects (data sets, etc.) exist in the `Global Environment`
- view the history of which commands were entered in the console
- import datasets (using the buttons on the top of the pane)

The `Global Environment` is where commands that are executed in the console typically live — intermediate objects created within functions (but not returned) live within function-specific environments.

The full command history for a given session is accessed in the `History` tab; this can be cycled through in the console using the up/down arrow keys.

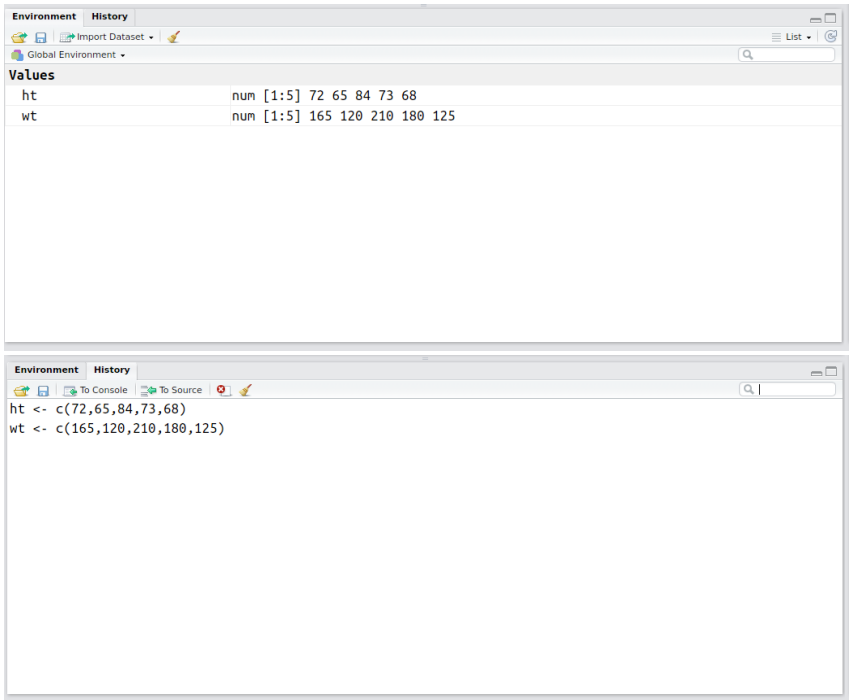
Example: heights and weights

Execute the commands in `ht_wt_analysis.R` using one of the two options we described earlier (`Run` button or `Ctrl/Cmd+Enter`).

Example: heights and weights

Execute the commands in `ht_wt_analysis.R` using one of the two options we described earlier (`Run` button or `Ctrl/Cmd+Enter`).

The environment pane then shows that `ht` and `wt` are `Values` (not data sets since they are vectors), and the history pane shows that we have input these two lines into the console.



RStudio: files, plot viewer, packages, help

The final pane shows us a variety of files/plots:

RStudio: files, plot viewer, packages, help

The final pane shows us a variety of files/plots:

- files in the current working directory

RStudio: files, plot viewer, packages, help

The final pane shows us a variety of files/plots:

- files in the current working directory
- plots generated from executed commands

RStudio: files, plot viewer, packages, help

The final pane shows us a variety of files/plots:

- files in the current working directory
- plots generated from executed commands
- loaded packages

RStudio: files, plot viewer, packages, help

The final pane shows us a variety of files/plots:

- files in the current working directory
- plots generated from executed commands
- loaded packages
- help files that we access

RStudio: files, plot viewer, packages, help

The final pane shows us a variety of files/plots:

- files in the current working directory
- plots generated from executed commands
- loaded packages
- help files that we access

We will cover packages and help files later on.

The working directory

Folders on your computer are also known as directories. These can contain data, documents, and subdirectories, among many other objects.

The working directory

Folders on your computer are also known as directories. These can contain data, documents, and subdirectories, among many other objects.

Your computer has a home directory, where all of your files (as the user) are stored, in different directories (e.g., `Documents`, `Pictures`, etc.). When you first log into your computer, you are in this directory.

The working directory

Folders on your computer are also known as directories. These can contain data, documents, and subdirectories, among many other objects.

Your computer has a home directory, where all of your files (as the user) are stored, in different directories (e.g., `Documents`, `Pictures`, etc.). When you first log into your computer, you are in this directory.

The current working directory is the directory where you are currently working. To be a bit less obtuse, consider the following example:

The working directory

Folders on your computer are also known as directories. These can contain data, documents, and subdirectories, among many other objects.

Your computer has a home directory, where all of your files (as the user) are stored, in different directories (e.g., `Documents`, `Pictures`, etc.). When you first log into your computer, you are in this directory.

The current working directory is the directory where you are currently working. To be a bit less obtuse, consider the following example:

You have a file called `ht_wt_analysis.R`, located in the `Epi-Biost-Workshop` directory, which is a subdirectory of `UW`, which is a subdirectory of `Documents`, which is part of your home directory. If you double-click `ht_wt_analysis.R`, which opens RStudio to edit the document, your current working directory is `<your home>/Documents/UW/Epi-Biost-Workshop`.

Changing the working directory using RStudio

Generally, when you open RStudio, your current working directory will be your home directory. This can make loading data that you have saved on your computer somewhat difficult.

Changing the working directory using RStudio

Generally, when you open RStudio, your current working directory will be your home directory. This can make loading data that you have saved on your computer somewhat difficult.

One way to remedy this is to manually set your working directory using R/RStudio: either

Changing the working directory using RStudio

Generally, when you open RStudio, your current working directory will be your home directory. This can make loading data that you have saved on your computer somewhat difficult.

One way to remedy this is to manually set your working directory using R/RStudio: either

- Using buttons:

Changing the working directory using RStudio

Generally, when you open RStudio, your current working directory will be your home directory. This can make loading data that you have saved on your computer somewhat difficult.

One way to remedy this is to manually set your working directory using R/RStudio: either

- Using buttons:
  1. In the file viewer, navigate to the folder where your dataset resides

Changing the working directory using RStudio

Generally, when you open RStudio, your current working directory will be your home directory. This can make loading data that you have saved on your computer somewhat difficult.

One way to remedy this is to manually set your working directory using R/RStudio: either

- Using buttons:
  1. In the file viewer, navigate to the folder where your dataset resides
  2. Click on the `More` button (with the settings-type wheel next to it) under the file viewer tab

Changing the working directory using RStudio

Generally, when you open RStudio, your current working directory will be your home directory. This can make loading data that you have saved on your computer somewhat difficult.

One way to remedy this is to manually set your working directory using R/RStudio: either

- Using buttons:
  1. In the file viewer, navigate to the folder where your dataset resides
  2. Click on the `More` button (with the settings-type wheel next to it) under the file viewer tab
  3. Click `Set as working directory`

Changing the working directory using RStudio

Generally, when you open RStudio, your current working directory will be your home directory. This can make loading data that you have saved on your computer somewhat difficult.

One way to remedy this is to manually set your working directory using R/RStudio: either

- Using buttons:
  1. In the file viewer, navigate to the folder where your dataset resides
  2. Click on the `More` button (with the settings-type wheel next to it) under the file viewer tab
  3. Click `Set as working directory`
- Using code:

Changing the working directory using RStudio

Generally, when you open RStudio, your current working directory will be your home directory. This can make loading data that you have saved on your computer somewhat difficult.

One way to remedy this is to manually set your working directory using R/RStudio: either

- Using buttons:
  1. In the file viewer, navigate to the folder where your dataset resides
  2. Click on the `More` button (with the settings-type wheel next to it) under the file viewer tab
  3. Click `Set as working directory`
- Using code:
  1. Find the path to the folder where your data resides (e.g., `/home/brian/Documents/UW/Epi-Biost-Workshop`)

Changing the working directory using RStudio

Generally, when you open RStudio, your current working directory will be your home directory. This can make loading data that you have saved on your computer somewhat difficult.

One way to remedy this is to manually set your working directory using R/RStudio: either

- Using buttons:
  1. In the file viewer, navigate to the folder where your dataset resides
  2. Click on the `More` button (with the settings-type wheel next to it) under the file viewer tab
  3. Click `Set as working directory`
- Using code:
  1. Find the path to the folder where your data resides (e.g., `/home/brian/Documents/UW/Epi-Biost-Workshop`)
  2. Place the code `setwd("<path to your data>")` in your R script, or enter it at the command line

Introduction to R programming

We have already seen some examples of R programming: in the heights and weights example, we learned about the special `<-` command, which assigns a value to an object; and we learned about the `c()` command, which creates a vector.

Introduction to R programming

We have already seen some examples of R programming: in the heights and weights example, we learned about the special <- command, which assigns a value to an object; and we learned about the `c()` command, which creates a vector.

These commands relate functions to objects – both of these are fundamental concepts in R programming.

Functions

R functions take in arguments and return values. A function is accessed by typing its name, followed by an open and closed set of parentheses: e.g., `quantile()`, a function to compute desired sample quantiles (e.g., minimum, median, maximum).

Functions

R functions take in arguments and return values. A function is accessed by typing its name, followed by an open and closed set of parentheses: e.g., `quantile()`, a function to compute desired sample quantiles (e.g., minimum, median, maximum).

Arguments to functions go between the parentheses, and are separated by columns. You specify which object corresponds to which argument using =, e.g., `quantile(x = ht, probs = 0.5)` returns the median value of the `ht` object.

Functions

R functions take in arguments and return values. A function is accessed by typing its name, followed by an open and closed set of parentheses: e.g., `quantile()`, a function to compute desired sample quantiles (e.g., minimum, median, maximum).

Arguments to functions go between the parentheses, and are separated by columns. You specify which object corresponds to which argument using =, e.g., `quantile(x = ht, probs = 0.5)` returns the median value of the `ht` object.

Values are returned by functions, and are generally either a combination of objects, plots, and printout. The value of the result of entering `quantile(x = ht, probs = 0.5)` into the console is a vector containing the number 72.

Objects

R objects are (basically) the result of calling functions. However, there are two general ways this can be done:

Objects

R objects are (basically) the result of calling functions. However, there are two general ways this can be done:

- loading data (e.g., from a text or .csv file)

Objects

R objects are (basically) the result of calling functions. However, there are two general ways this can be done:

- loading data (e.g., from a text or .csv file)
- manipulating another object using a function

Objects

R objects are (basically) the result of calling functions. However, there are two general ways this can be done:

- loading data (e.g., from a text or .csv file)
- manipulating another object using a function

Both `ht` and `wt` are objects; we created them by manipulating raw numbers using the `c()` function.

Loading data

Data can be loaded using menus, buttons, or a function.

Loading data

Data can be loaded using menus, buttons, or a function.

The menus and buttons ultimately call a function; each data storage type (e.g., `.txt`, `.csv`, `.sas7bdat`, `.dta`) has its own function (and some functions come from special packages).

Loading data

Data can be loaded using menus, buttons, or a function.

The menus and buttons ultimately call a function; each data storage type (e.g., `.txt`, `.csv`, `.sas7bdat`, `.dta`) has its own function (and some functions come from special packages).

Regardless of how you load the data, once done it will show up in your environment. It can then be manipulated to display summary statistics, make plots, or do more complex analyses.

Loading data: menus

1. Go to `File > Import Dataset`

Loading data: menus

1. Go to `File > Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)

Loading data: menus

1. Go to `File > Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)
3. Browse to the directory or website where the data are stored

Loading data: menus

1. Go to `File > Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)
3. Browse to the directory or website where the data are stored
4. Specify any input options

Loading data: menus

1. Go to `File > Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)
3. Browse to the directory or website where the data are stored
4. Specify any input options
5. Edit the `Code Preview` section to rename the object, if necessary

Loading data: menus

1. Go to `File > Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)
3. Browse to the directory or website where the data are stored
4. Specify any input options
5. Edit the `Code Preview` section to rename the object, if necessary
6. Click `Import`

Loading data: buttons

1. In the `Environment` pane, click `Import Dataset`

Loading data: buttons

1. In the `Environment` pane, click `Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)

Loading data: buttons

1. In the `Environment` pane, click `Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)
3. Browse to the directory or website where the data are stored

Loading data: buttons

1. In the `Environment` pane, click `Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)
3. Browse to the directory or website where the data are stored
4. Specify any input options

Loading data: buttons

1. In the `Environment` pane, click `Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)
3. Browse to the directory or website where the data are stored
4. Specify any input options
5. Edit the `Code Preview` section to rename the object, if necessary

Loading data: buttons

1. In the `Environment` pane, click `Import Dataset`
2. Select the type of data from the menu (e.g., `From CSV`)
3. Browse to the directory or website where the data are stored
4. Specify any input options
5. Edit the `Code Preview` section to rename the object, if necessary
6. Click `Import`

Loading data: menus and buttons

Regardless of whether you use a menu or button to import data, there was a step where you had to look at the `Code Preview` box. When you click the `Import` button, this code is executed in the console, and the dataset shows up in your environment.

Loading data: menus and buttons

Regardless of whether you use a menu or button to import data, there was a step where you had to look at the `Code Preview` box. When you click the `Import` button, this code is executed in the console, and the dataset shows up in your environment.

This means that, under the hood, both methods use R code to import data!

Also, both of these methods are hard to reproduce: each time you want to analyze a dataset (which may be multiple times on a problem set or during the course of a project/paper), you have to click through these buttons.

Loading data: menus and buttons

Regardless of whether you use a menu or button to import data, there was a step where you had to look at the `Code Preview` box. When you click the `Import` button, this code is executed in the console, and the dataset shows up in your environment.

This means that, under the hood, both methods use R code to import data!

Also, both of these methods are hard to reproduce: each time you want to analyze a dataset (which may be multiple times on a problem set or during the course of a project/paper), you have to click through these buttons.

To make your code fully reproducible, embed the code to load the data directly in your R script! While this involves finding the correct functions, at the end of the day you have more control over how your data are handled.

Loading data: functions

1. Find the path to your data file

Loading data: functions

1. Find the path to your data file
   - e.g., `"/home/brian/Documents/UW/Epi-Biost-Workshop/ht_wt_data.txt"`
2. Based on the file extension (e.g., `.txt`, `.csv`, `.dta`), choose the appropriate function

Loading data: functions

1. Find the path to your data file
   - e.g., `"/home/brian/Documents/UW/Epi-Biost-Workshop/ht_wt_data.txt"`
2. Based on the file extension (e.g., `.txt`, `.csv`, `.dta`), choose the appropriate function
3. Put the function call with the file path in your R script with the desired options

Loading data: functions

1. Find the path to your data file
   - e.g., `"/home/brian/Documents/UW/Epi-Biost-Workshop/ht_wt_data.txt"`
2. Based on the file extension (e.g., `.txt`, `.csv`, `.dta`), choose the appropriate function
3. Put the function call with the file path in your R script with the desired options
   - e.g., `read.table("/home/brian/Documents/UW/Epi-Biost-Workshop/ht_wt_data.txt", sep = "", header = TRUE)`

Loading data: functions

1. Find the path to your data file
   - e.g., `"/home/brian/Documents/UW/Epi-Biost-Workshop/ht_wt_data.txt"`
2. Based on the file extension (e.g., `.txt`, `.csv`, `.dta`), choose the appropriate function
3. Put the function call with the file path in your R script with the desired options
   - e.g., `read.table("/home/brian/Documents/UW/Epi-Biost-Workshop/ht_wt_data.txt", sep = "", header = TRUE)`
4. Run this line to load the data

Loading data: common functions and packages

File extension	Package	Function
.txt	utils (always loaded)	read.table()
		read.delim()
	readr	read_table()
		read_delim()
.csv	utils	read.csv()
	readr	read_csv()
.dta	haven	read_stata()
.sas7bdat	haven	read_sas()
.xlsx	readxl	read_excel()
.Rdata	base (always loaded)	load()
.Rds	base	readRDS()

Exercise: Reading in data

Load the height and weight data by placing the following code in your R script `ht_wt_analysis.R` (replace `<path to here>` with the path to your folder):

```
dat <- read.table("<path to here>/Epi-Biost-Workshop/
                  day_3_session_1/data/ht_wt_data.txt",
                   header = TRUE)
```

Look at the data by typing `View(dat)` on a new line in your R script, and executing the line in R.

What do you notice about this new dataset? How does it relate to the two vectors `ht` and `wt` that we created earlier?

Exercise: reading data

Now read in the FEV data, again using the function `read.table`. *Hint: you may copy the code from the height and weight example, but you will have to change the path to the data*.

Again, use the `View()` function to take a look at the data, and answer the following questions:

1. How many columns do the data have?
2. Write down what you think the meaning of at least 3 of the variables is, based on the column names.

Manipulating data

A large component of any data analysis is getting the data into a useable format for our analysis: this usually involves a combination of creating new variables and taking subsets of the data.

Manipulating data

A large component of any data analysis is getting the data into a useable format for our analysis: this usually involves a combination of creating new variables and taking subsets of the data.

The two most common types of data you will see are vectors

Manipulating data

A large component of any data analysis is getting the data into a useable format for our analysis: this usually involves a combination of creating new variables and taking subsets of the data.

The two most common types of data you will see are vectors and data frames.

Manipulating data

A large component of any data analysis is getting the data into a useable format for our analysis: this usually involves a combination of creating new variables and taking subsets of the data.

The two most common types of data you will see are vectors and data frames. We have already been introduced to vectors in the heights and weights example;

Manipulating data

A large component of any data analysis is getting the data into a useable format for our analysis: this usually involves a combination of creating new variables and taking subsets of the data.

The two most common types of data you will see are vectors and data frames. We have already been introduced to vectors in the heights and weights example; data frames are generally datasets, where the columns are variables (each of these is a vector), and the rows are observations.

Manipulating data

A large component of any data analysis is getting the data into a useable format for our analysis: this usually involves a combination of creating new variables and taking subsets of the data.

The two most common types of data you will see are vectors and data frames. We have already been introduced to vectors in the heights and weights example; data frames are generally datasets, where the columns are variables (each of these is a vector), and the rows are observations.

By manipulating the columns of a data frame we create new variables or change existing ones; and by manipulating the rows of our data we restrict our attention to different subsets of the observations.

Manipulating data: accessing values in vectors

The easiest way to access data is from a vector: each vector is its own variable, and hence we just need to extract the correct element(s).

Manipulating data: accessing values in vectors

The easiest way to access data is from a vector: each vector is its own variable, and hence we just need to extract the correct element(s).

To access the *i*th element in the vector `vec`, we use square brackets; `vec[i]` accomplishes this. For example, `ht[1]` returns the first element in the `ht` vector, which is the height of our first imaginary individual.

Manipulating data: accessing values in vectors

The easiest way to access data is from a vector: each vector is its own variable, and hence we just need to extract the correct element(s).

To access the *i*th element in the vector `vec`, we use square brackets; `vec[i]` accomplishes this. For example, `ht[1]` returns the first element in the `ht` vector, which is the height of our first imaginary individual.

To access a continuous range of elements *i* through *l*, use square brackets with a colon; `vec[i:l]` accomplishes this. For example, `ht[1:4]` returns the first four elements of `ht`.

Manipulating data: accessing values in vectors

The easiest way to access data is from a vector: each vector is its own variable, and hence we just need to extract the correct element(s).

To access the *i*th element in the vector `vec`, we use square brackets; `vec[i]` accomplishes this. For example, `ht[1]` returns the first element in the `ht` vector, which is the height of our first imaginary individual.

To access a continuous range of elements *i* through *l*, use square brackets with a colon; `vec[i:l]` accomplishes this. For example, `ht[1:4]` returns the first four elements of `ht`.

To access a set of elements (*i*, *k*, *n*), use square brackets with `c()`; `vec[c(i, k, n)]` accomplishes this. For example, `ht[c(1, 3, 5)]` returns the first, third, and fifth elements of `ht`.

Manipulating data: accessing values in data frames

Data frames have both rows and columns. The dimensions of the data frame are first rows, then columns.

Manipulating data: accessing values in data frames

Data frames have both rows and columns. The dimensions of the data frame are first rows, then columns. We can access a single value in a data frame by first specifying the desired row, then the desired column, within square brackets;

Manipulating data: accessing values in data frames

Data frames have both rows and columns. The dimensions of the data frame are first rows, then columns. We can access a single value in a data frame by first specifying the desired row, then the desired column, within square brackets; for a data frame `dat` with 5 rows and 2 columns, we can get the first observation from the second column using `dat[1, 2]`.

Manipulating data: accessing values in data frames

Data frames have both rows and columns. The dimensions of the data frame are first rows, then columns. We can access a single value in a data frame by first specifying the desired row, then the desired column, within square brackets; for a data frame `dat` with 5 rows and 2 columns, we can get the first observation from the second column using `dat[1, 2]`.

For example, in the heights and weights data, `dat[1, 2]` returns the weight of the first individual.

Manipulating data: accessing values in data frames

Data frames have both rows and columns. The dimensions of the data frame are first rows, then columns. We can access a single value in a data frame by first specifying the desired row, then the desired column, within square brackets; for a data frame `dat` with 5 rows and 2 columns, we can get the first observation from the second column using `dat[1, 2]`.

For example, in the heights and weights data, `dat[1, 2]` returns the weight of the first individual.

We can use the same rules as those for vectors to select groups of rows and/or columns.

Manipulating data: accessing variables in data frames

There are three ways to access a variable in a data frame. For a data frame `dat`, with columns `x` and `y`, we can use:

Manipulating data: accessing variables in data frames

There are three ways to access a variable in a data frame. For a data frame `dat`, with columns `x` and `y`, we can use:

- `$`; `dat$y` returns the variable `y` from the data frame `dat`

Manipulating data: accessing variables in data frames

There are three ways to access a variable in a data frame. For a data frame `dat`, with columns `x` and `y`, we can use:

- `$`; `dat$y` returns the variable `y` from the data frame `dat`
- the column number; `dat[, 2]` returns the variable `y` from `dat`

Manipulating data: accessing variables in data frames

There are three ways to access a variable in a data frame. For a data frame `dat`, with columns `x` and `y`, we can use:

- `$`; `dat$y` returns the variable `y` from the data frame `dat`
- the column number; `dat[, 2]` returns the variable `y` from `dat`
- the column name; `dat[, "y"]` returns the variable `y` from `dat`

Manipulating data: accessing variables in data frames

There are three ways to access a variable in a data frame. For a data frame `dat`, with columns `x` and `y`, we can use:

- `$`; `dat$y` returns the variable `y` from the data frame `dat`
- the column number; `dat[, 2]` returns the variable `y` from `dat`
- the column name; `dat[, "y"]` returns the variable `y` from `dat`

The methods using the square brackets introduce a new concept: we are selecting all rows from `dat` (since there is nothing between the first bracket and the comma), and are selecting certain columns from `dat`.

Manipulating data: subsetting data frames

Often, we don't know exactly which rows we want to select. Using logical operations helps us select participants with certain characteristics. Logical operations return `TRUE` or `FALSE`.

Manipulating data: subsetting data frames

Often, we don't know exactly which rows we want to select. Using logical operations helps us select participants with certain characteristics. Logical operations return `TRUE` or `FALSE`.

The common logical operations are:

Operation	Meaning	Result
|	or	`TRUE` if any argument is `TRUE`
&	and	`TRUE` only if all arguments are `TRUE`
!	negation	`FALSE` if argument returns `TRUE`, `TRUE` else
==	equals	`TRUE` if the two arguments are equal

We can also make comparisons, using $<$, $>$, $<=$, $>=$.

Example: heights and weights

The following code loads the data as a data frame, and then: prints out the height column, and displays the data from participants whose weight is below 160 lbs (replace <path to here> with your own path):

```
## load using read.table
dat <- read.table("<path to here>/Epi-Biost-Workshop/
                  day_3_session_1/data/ht_wt_data.txt", header = TRUE)
## print out the height column multiple ways
dat$height
dat[, "height"]
dat[, 2]

## print out the observations with weight under 160
dat[dat$weight < 160, ]
```

Exercise: manipulating data

These questions all pertain to the `fev` data.

1. The logical expression `fev$age <= 8` returns `TRUE` for those individuals who are less than or equal to 8 years old, and `FALSE` for those who are older than 8 years old. Create a new dataset called `sub_less_equal_8` that contains the data for all participants with age less than or equal to 8 years old.
2. What does `fev$sex == 1` return?
3. The `$` allows you to both access variables and create new variables within a dataset. Create a new variable called `fev$female` that takes value 0 if the participant is male and 1 if the participant is female, using the command `fev$female <- ifelse(fev$sex == 1, 0, 1)`. What is the sex of the participant with `fev$subjid == 451`?

Summarizing data

The first part of my data analysis workflow (after reading in the data) is typically **exploratory**. Here, I investigate whether I can find any trends in the data that support my initial hypothesis.

Summarizing data

The first part of my data analysis workflow (after reading in the data) is typically **exploratory**. Here, I investigate whether I can find any trends in the data that support my initial hypothesis.

Two powerful ways of summarizing the data are:

Summarizing data

The first part of my data analysis workflow (after reading in the data) is typically **exploratory**. Here, I investigate whether I can find any trends in the data that support my initial hypothesis.

Two powerful ways of summarizing the data are:

- summary statistics (e.g., mean, median)

Summarizing data

The first part of my data analysis workflow (after reading in the data) is typically **exploratory**. Here, I investigate whether I can find any trends in the data that support my initial hypothesis.

Two powerful ways of summarizing the data are:

- summary statistics (e.g., mean, median)
- graphical summaries (e.g., boxplots, scatterplots)

Computing summary statistics

Quick summary statistics are available in R using the `summary()` function. This function automatically computes the minimum, mean, maximum, and 25th, 50th (median) and 75th percentile values of each variable in your data.

Example: heights and weights

```
summary(dat)
```

Plotting data

Quick plots are also powerful ways to convey information and get a glimpse at trends in the data.

The most common type of plot that I use is a scatterplot. You create a scatterplot using the `plot()` function.

Plotting data

Quick plots are also powerful ways to convey information and get a glimpse at trends in the data.

The most common type of plot that I use is a scatterplot. You create a scatterplot using the `plot()` function.

The two main arguments, `x` and `y`, tell R the data to plot on the x- and y-axis, respectively. Other arguments help to make the plot understandable by adding informative labels and titles.

Example: heights and weights

```
# plot of height (on x-axis) vs weight
plot(dat$height, dat$weight)
```

Exercise: summarizing data

All of these questions involve either the full `fev` data, or the subset of participants with age less than or equal to eight years old `sub_less_equal_8`.

1. What is the average age of participants in the `fev` data?
2. What is the proportion of subjects that are female? That are male?
3. What is the average FEV measurement in the full data?
4. What is the average FEV measurement in those who are less than or equal to eight years old?
5. If your answers to 3 and 4 are different, why do you think that is?
6. Plot FEV on the y-axis versus age on the x-axis for the full data. What do you notice about the trend between age and FEV? Does that help you answer 5?

R packages

Functions in R are available through packages. Each package contains a specific set of functions, and must be both installed and loaded before the function can be used.

R packages

Functions in R are available through packages. Each package contains a specific set of functions, and must be both installed and loaded before the function can be used.

Each package only has to be installed once, unless you upgrade your version of R. Packages are installed using the `install.packages()` function.

R packages

Functions in R are available through packages. Each package contains a specific set of functions, and must be both installed and loaded before the function can be used.

Each package only has to be installed once, unless you upgrade your version of R. Packages are installed using the `install.packages()` function.

Each time you open a new R session, you have to re-load all R packages you wish to use in the session. Packages are loaded using the `library()` function.

R packages

Functions in R are available through packages. Each package contains a specific set of functions, and must be both installed and loaded before the function can be used.

Each package only has to be installed once, unless you upgrade your version of R. Packages are installed using the `install.packages()` function.

Each time you open a new R session, you have to re-load all R packages you wish to use in the session. Packages are loaded using the `library()` function.

However, some packages are always loaded each time you open R. These are `stats`, `graphics`, `grDevices`, `utils`, `datasets`, `methods`, and `base`.

Finding new R packages

In general, R packages are stored on one (or more) of three repositories:

Finding new R packages

In general, R packages are stored on one (or more) of three repositories: CRAN, under the `Packages` side-menu;

Finding new R packages

In general, R packages are stored on one (or more) of three repositories: CRAN, under the `Packages` side-menu; Bioconductor, under the `Explore packages` menu;

Finding new R packages

In general, R packages are stored on one (or more) of three repositories: CRAN, under the `Packages` side-menu; Bioconductor, under the `Explore packages` menu; and individual programmer's GitHub pages (e.g., the repository for the `devtools` package, for easy R package creation).

Finding new R packages

In general, R packages are stored on one (or more) of three repositories: CRAN, under the `Packages` side-menu; Bioconductor, under the `Explore packages` menu; and individual programmer's GitHub pages (e.g., the repository for the `devtools` package, for easy R package creation).

You can install packages from CRAN using `install.packages()`;

Finding new R packages

In general, R packages are stored on one (or more) of three repositories: CRAN, under the `Packages` side-menu; Bioconductor, under the `Explore packages` menu; and individual programmer's GitHub pages (e.g., the repository for the `devtools` package, for easy R package creation).

You can install packages from CRAN using `install.packages()`; from Bioconductor by first running `source("https://bioconductor.org/biocLite.R")` and then using `biocLite()`;

Finding new R packages

In general, R packages are stored on one (or more) of three repositories: CRAN, under the `Packages` side-menu; Bioconductor, under the `Explore packages` menu; and individual programmer's GitHub pages (e.g., the repository for the `devtools` package, for easy R package creation).

You can install packages from CRAN using `install.packages()`; from Bioconductor by first running `source("https://bioconductor.org/biocLite.R")` and then using `biocLite()`; and from GitHub using the function `install_github()` from the `devtools` package.

Example: heights and weights

The following code reads in the height and weight data using the `read_table()` function from `readr`:

```
## install if necessary
# install.packages("readr")
## load the package; do this each time you re-open R!
library("readr")
## read in the data
dat <- read_table("<path to here>/Epi-Biost-Workshop/
                  day_3_session_1/ht_wt_data.txt", header = TRUE)
```