First Steps with R and RStudio

Matt Steele

2023-03-21

Contents

Continue Learning with these Resources

R download and documentation

```
browseURL("https://cran.r-project.org/")
```

RStudio download and documentation

```
browseURL("https://rstudio.com/")
```

O'Reilly Learning Platform

```
browseURL("https://databases.lib.wvu.edu/connect/1540334373")
```

R Programming for Statistics and Data Science

```
browseURL("https://libwvu.on.worldcat.org/oclc/1062397089")
```

Getting Started

Start a New File

You can create new files (R Scripts or RMarkdown files) that allow you to document your code and the outputs of your code.

- File > New File > **Type of Document you want to create**

Set your Preferences

You can set your preferences in RStudio.

- Tools > Global Options

Setting your working directory

The working directory in R is the folder where you are working. Hence, it's the place (the environment) where you have to store your files of your project in order to load them or where your R objects will be saved.

Function: getwd()

See the current directory you are in

```
# HELP FOR getwd()

`?`(getwd)
```

```
# USING THE getwd() function

getwd()
```

Function: setwd(path)

You can set your working directory in RStudio by going to

- Session > Set Working Directory > Choose Directory
- Choose the folder you want to work out of

After you set your working directory, save the path by copy and pasting the file path from the console area into the source area using the setwd() function

```r
# Remember to save your working directory path to your script or markdown file

setwd("C:/Users/Matt/Documents/RWorkshop Development/workshop_firstSteps_R-main")
```

Overview of IDE in RStudio

RStudio is an Integrated Development Environment (IDE) that allows you to save you code, store your variables and environments and view outputs.

Console Pane

pane on the left-hand side of the screen

- This area is where your code is run and where outputs are displayed

Source Pane

this pane is opened when you create or open a markdown or script file.

- This area is where you can create code in script or markdown files
- These files can be a saved and accessed at any point

Environment Pane

view functions, objects, and data sets that are stored here

- Your environment can be saved and accessed at any point
- Save your environment to your working directory

Tips and Tricks

1. Keyboard Shortcuts - Tools > Keyboard Shortcut Tricks - Run Code: CTRL + ENTER (PC) - Run Code: CMD + RETURN (MAC) - Assignment Operator: ALT + - (<-)
2. Help + You can access the help tab in the button right. + Contains overviews of topics and functions. + You can also call it with code
    - Example: ?getwd
    - Example: help("read.table")
3. TAB Button + the TAB button auto-suggests
    - Functions
    - Arguments
    - Files in WD
    - Values
4. Commenting + # - single line + ''' **text** ''' - multiline + In script files or in coding areas, you can add text
    - This text will be ignored when the code is run
    - It is used for documentation purposes

Getting Started with Coding

Using Code with Calculations

R can be used to perform calculations

```
# Addition:
129 + 3483

# Subtraction:
23693 - 4536

# Multiplication:
23 * 45

# Division:
51/3

# Exponents:
2^4

# Logarithms (base e):
log(100)
```

Example Calculations

Inputing Ranged Values

You can also call a range of values using (:).

Example Range

```
1:50
```

Built-In Functions and Arguments

Like most computer software, R allows you to run commands. Commands in R are referred to as functions. There are several built-in functions in base R, however, when you install and call a new package, you will have access to more functions that you can use.

Base R Built-in Functions

```
browseURL("https://stat.ethz.ch/R-manual/R-devel/library/base/html/00Index.html")
```

```
sample(1:500)
```

Example Function

Function's Arguments

values and parameters that are acted on by the function.

- When a function is invoked, you pass a value to the argument.
- Arguments are optional; that is, a function may contain no arguments.
- Functions can have arguments with default values.
- The arguments of a function can be called in the order that they appear in their documentation

```
sample(1:500, size = 30, replace = TRUE)
```

Example Argument

Objects

Objects allow you to store information for future recall

- Things that can be objects:
  - single digit
  - a character
  - a vector
  - an imported data frame (e.g.csv)
  - a created data frame

Assignment Operator (<-)

The assignment operator allows you to create an object

- *syntax: objectName <- stored_value*

```
a <- 3

b <- 4

a + b
```

Naming Conventions

Objects must start with a lowercase letter and cannot start with a captial letter. Ideally, you should not name your object with the name of a function, to avoid confusion.

- If your object is using two words or more
  - Example 1 - longer.name
  - Example 2 - longer_name
  - Example 3 - longerName

```
first.example <- sample(3:15, size = 1)

second_example <- round(2.471, digits = 2)

thirdExample <- mean(500:4000)

first.example + second_example/thirdExample
```

Data Values

Double

regular numbers (large small, positive, negative, with digits after the decimal, or without)

```
str(400)
```

Integer

positive whole number with nothing after the decimal. Denoted using L

```
str(400L)
```

Character

non-numeric data to be interpreted as text. Denoted using (“ “)

```
str("Hello World")
```

Factor

data objects which are used to categorize the data and store it as levels.

```
# Line 333

tarantino <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/data/master/tarantino/tarantin
    stringsAsFactors = T)

str(tarantino$word)
```

Boolean

boolean data objects that denote TRUE or FALSE. Denoted with capital letters.

```
str(TRUE)
```

Packages

R is open source code after it’s initial development people began adding to it with packages. An R Package is something that you can plug into RStudio to extend the basic functionality that is built in with R. One of the reasons that R has become so popular is because it has this rich ecosystem of packages that really make R a comprehensive platform for data science.

Install Packages

You must install a package before you can call it. But you only need to install it one time

```
help("install.packages")
```

```
# INSTALL THE TIDYVERSE PACKAGE

install.packages("tidyverse")
```

Load Packages

For every new session, you must load it to use the package's functions

```
# Help for library function

help("library")
```

```
library(tidyverse)  # you need to call the library during each session
```

Creating Vectors

one-dimensional sequence of data elements

- use *c(. . . , . . .)* To make a vector
- separate elements using *, (comma)*
- syntax: *vectorSyntax <- c(object1, object2, object3)*

```
vec_one <- c(1, 2, 10)
vec_two <- c(5, 6, 7)
vec_three <- c("Morgantown", "Charleston", "Huntington")
```

```
mean(vec_one)
```

RBIND and CBIND

combine vectors to create two-dimensional arrays.Combinations with the same data value will create a *matrix*. Combinations with different values will create a *data frame*.

```
cbind(vec_one, vec_two, vec_three)
```

Creating Data Frames

data.frame documentation

two dimensional arrays that can multiple classes of values.

- syntax: *dataFrameSyntax <- data.frame(column1, column2, column3)*

```
# LINE 437

# create vectors
title <- c("Star Wars", "The Empire Strikes Back", "Return of the Jedi")
year <- c(1972, 1980, 1983)
length.min <- c(121, 124, 133)
box.office.mil <- c(787, 534, 572)

# combine these vectors with the data.frame() function

starWars.data <- data.frame(title, year, length.min, box.office.mil)
starWars.data
```

You get a faster and more efficient data frame using the package *data.table* which also provides more arguments for creating your data frame.

data.table package

Cleaning Data

Rename Variables

Change the variable names in the data frame.

- syntax: *data.frame(= , . . .)*

```
starWars.data <- data.frame(Title = title, Year = year, Length = length.min, Gross = box.office.mil)
starWars.data
```

Subsetting Variables

look at and/or work on individual variable in a data frame.

- syntax: *$*

```
starWars.data$Year
```

Changing Data Values

change how R interpret the values in a variable

```
starWars.data$Year <- as.character(starWars.data$Year)
str(starWars.data)
```

Changing Value of an Observation

Indexing is a way that you can call a column, row, or specific position in a data frame. You can use indexing to pinpoint a specific value and change that value.

- syntax: *data.frame[rownumber, columnnumber] = newvalue*

```
starWars.data[1, 2]
```

```
starWars.data[1, 2] = "1977"
starWars.data
```

Create New Variables

You can use observations in your data frame to create new variables or overwrite a current variable

- syntax: *= $*

```
starWars.data$Gross.billions = starWars.data$Gross/1000
starWars.data
```

Append new variables

Create new variables with observations and append them to your data frame

```
han <- c(18.7, 21.4, 16.3)

starWars.data$Han <- han

starWars.data
```

Merge new observations

Create new observations and merge it into a data frame

```
solo <- data.frame(Title = "Solo", Year = 2018, Length = 135, Gross = 393.2, Gross.billions = 0.392,
    Han = 0)

rbind(starWars.data, solo)
```

Exporting data to a saved file

```
write.csv(starWars.data, "starwars.csv")
```

Reading Data in R

Base R option

- **read.table(...) / read.csv(...)**

```
help("read.table")
```

read.table will provide you with a generic way to load data where you are setting most of the parameters. However, the most common file for data frames are .csv files. R provides the *read.csv() function* which allows you to read .csv files with parameters set to default.

```
avengers <- read.csv("avengers.csv")
str(avengers)
```

Tidyverse option

- **read_table(...) / read_csv(...)**

Using the tidyverse package you can read the data into a lighter frame.

```
fight_songs <- read_csv("fight-songs.csv")
fight_songs
```

Exploring data

Function: View()

view the full data frame

```
View(fight_songs)


# There is also a tidyverse option view(avengers)
```

Function: nrow()

view the number of rows/observations

```
nrow(fight_songs)
```

Function: ncol()

view the number of columns/variables

```
ncol(fight_songs)
```

Function: colnames()

view the names of columns/variables

```
colnames(fight_songs)
```

Function: str()

view the structure of data

```
str(fight_songs)
```

Function: summary()

basic descriptive statistics

```
summary(fight_songs)
```

Function: mean(), median(), mode(), count(), sum()

individual descriptive statistics

```
sum(fight_songs$number_fights)
```

Function: head() or tail()

view top and bottom of data

```
head(fight_songs)
```

You can perform more advanced analysis in base-r using the built in *stats package*

stats package

```
browseURL("https://stat.ethz.ch/R-manual/R-devel/library/stats/html/00Index.html")
```

Function: cor()

get correlation coefficients for two numeric variables

```
cor(fight_songs$bpm, y = fight_songs$number_fights)
```

Function: lm()

create a linear model to predict the value of an unknown variable based on independent variables.

```
model <- lm(data = fight_songs, bpm ~ sec_duration)
summary(model)
```

Explore and Manipulate Data more with Tidyverse

```
library(tidyverse)
view(fight_songs)
```

Pipe Operator - %>%

The pipe operator allows you to separate and unite individual operations on an object

- CTRL + SHIFT + M (windows)
- CMD + SHIFT + M (mac)

data %>% operation A %>% operation B

Function: arrange()

sort columns by values

```
arrange(fight_songs, desc(conference))  ## Most / z-a
arrange(fight_songs, conference)  ## Least / a-z
```

Function: filter()

limit observations in a variable based on criteria

```
fight_songs_big12 <- fight_songs %>%
    filter(conference == "Big 12")
mean(fight_songs$bpm)
mean(fight_songs_big12$bpm)
```

Function: select()

limit the number of variables in the data frame

```
songs_by_school <- fight_songs %>%
    select(school, song_name)
songs_by_school
```

Function: groupby() and summarise

create dummy variables for categorical variables and get summary statistics

```
conference_num_fights <- fight_songs %>%
    group_by(conference) %>%
    summarise(avg_fight = mean(number_fights))

conference_num_fights
```