Using R for Data Processing

Using R for Data Processing

Glenn Williams

University of Sunderland

2021-10-25 (updated: 2021-10-25)

Overview

- **Data Types**: How is data stored in R, and how does this affect your choices?
- **Objects and Functions**: How to save data to a variable you can work with, and how to work with it.
- **Packages**: Adding functionality to R with additional packages.

This will give you some basic experience in R which we will build upon in future sessions for doing serious data processing and graphing.

Data Types

Basic Types

R can read your data in and parse it in different ways. There are a few basic data types available to R:

- **Strings** = **Text** data, e.g. "Dog", "Cat", "Dogs and Cats".
- **Integers** = **Whole numbers**, e.g. 1, 2, 3.
- **Doubles/Floats** = **Decimal** numbers, e.g. 0.1, 0.22, 0.14132.
- **Factors** = Ordered or unordered **factors with levels**. These take numbers and associated labels, e.g. 1 = "Dog", 2 = "Cat".
- **Booleans** = These are TRUE or FALSE statements, and only take those values.

These can be stored on their own in **objects** called vectors.

Data Types

Using R to Print Data

You can use R for basic things like printing out your data or using it as a calculator.

```
"cat"
```

```
## [1] "cat"
```

```
1 + 1
```

```
## [1] 2
```

Data Types

Using R to Print Data

R will wait for you to finish an expression before executing the operation, e.g. if your line ends with a + or you haven't finished a quote for a string.

```
1 + 1 + 1 + 1 + 2 +
  1 + 2
```

```
## [1] 9
```

```
"this is a really long string on
two separate lines."
```

```
## [1] "this is a really long string on\ntwo separate lines."
```

This isn't very useful on its own though...

Data Types

Storing Data

In R, we can assign values to an object using the assignment operator, `<-`.

When we do this, we can perform operations on the object and access it later to retrieve its values:

```
# make an object called my_object containing the string,
# "hello, class!"
my_object <- "hello, class!"

# access the value of my_object
my_object
```

```
## [1] "hello, class!"
```

Note: We can include comments to our code using the symbol #, which is ignored by R. Comments are good for explaining the aim of your code.

Data Types

Storing Data Together

What if we have several strings or numbers we'd like to store together? We can combine them by **concatenation** with `c()`.

This is a **function** for combining data. Like all R functions, it has a name, and begins and ends with `()`.

```
my_strings <- c("A", "B")
my_numbers <- c(1, 2, 3)

# print them out
my_strings
```

```
## [1] "A" "B"
```

```
my_numbers
```

```
## [1] 1 2 3
```

Be careful when **mixing data types**. If you combine a string and a number, R will make everything a string. Why? You can write out a "1", or "2" as text, but you can't make "hello, class" into a number.

```
c(my_strings, my_numbers)
```

```
## [1] "A" "B" "1" "2" "3"
```

Data Types

Storing More Data Together

We can store data together in different ways.

The most common way in R is to use a `data.frame`. This lets you to make a table of data with columns of different data types.

```
my_frame <- data.frame(
  instructor = c("Glenn", "Becci", "Shannon"),
  has_cat = c(TRUE, TRUE, FALSE),
  n_cats = c(2, 1, 0)
)

my_frame
```

```
##   instructor has_cat n_cats
## 1      Glenn    TRUE      2
## 2      Becci    TRUE      1
## 3    Shannon   FALSE      0
```

Data Types

Storing More Data Together

We can also store data in matrices if they are numbers:

```
matrix(
  c(1:6),
  nrow = 2,
  ncol = 3
)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Or in lists where we can have nested levels of detail:

```
list(
  instructor = c("Glenn", "Becci
  is_mint = c(TRUE, TRUE, TRUE)
)
```

```
## $instructor
## [1] "Glenn"   "Becci"   "Shannon"
##
## $is_mint
## [1] TRUE TRUE TRUE
```

We won't focus on these in the following sessions.

Operations

You've seen we can do things with data. Here's some basic operations we can perform:

R has the simple arithmetic operations, e.g.:

- add: `x + y`,
- subtract: `x - y`,
- multiply: `x * y`
- divide: `x/y`,

R also has some logical operations built in. For example:

- less than or equal to: `x <= y`
- exactly equal to: `x == y`
- not equal to: `x != y`
- x OR y: `x | y`
- x AND y: `x & y`

We can do additional operations with functions.

Functions

R has a number of inbuilt functions, which can be expanded by loading new packages.

Some basic arithmetic can be seen below:

```
sum(c(1, 2, 3, 4))
```

```
## [1] 10
```

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

This is just the same:

```
a <- c(1, 2, 3, 4)
mean(a)
```

```
## [1] 2.5
```

Accessing Data from Vectors

When we have data stored in an R object, we can **pull out specific values**.

With vectors, we do this by their index (position) in the vector, starting with 1. These values are accessed with square brackets after the vector name.

```
my_numbers <- c(1, 2, 5, 10, 20)
my_numbers[3]
```

```
## [1] 5
```

- We can also change these values if we want.

```
my_numbers[3] <- 100
my_numbers
```

```
## [1]   1   2 100  10  20
```

Accessing Data from Data Frames

When we have data stored in an R object, we can **pull out specific values**.

With **data frames**, we can do this by their **column name, position**, or some mix of the two. Values are indexed by square brackets after the data.frame name, separated like: **[row number, column number]**.

```
df <- data.frame(
  a = c(1, 2, 3),
  b = c("x", "y", "z")
)

df # show the data
```

```
##   a b
## 1 1 x
## 2 2 y
## 3 3 z
```

```
df[2, 1] # row 2, column 1
```

```
## [1] 2
```

```
df[2, "a"] # row 2, column "a"
```

```
## [1] 2
```

```
df$a # access column "a"
```

```
## [1] 1 2 3
```

If you want all rows from column b, use `df[, "b"]`. If you want a specific row for all columns, use e.g. `df[1, ]`.

Using Packages

We've seen how to install and load packages. Once loaded (e.g. `library(tidyverse)`), you can use functions from it:

```
library(tidyverse)

df <- data.frame(
  a = c(1, 2, 3, 10, 20),
  b = c("a", "b", "c", "d", "e")
)

filter(df, a > 2)
```

```
##    a b
## 1  3 c
## 2 10 d
## 3 20 e
```

Here, **filter subsets your data frame** only to observations where column a has values greater than 2.

Conclusion

In the last 3 sessions you've learned a lot, including:

- **What R is**, and why it's important and helpful in your research.
- How to **install R**, navigate the RStudio interface, and set up a **project**.
- How to **create a notebook** to document your work in R.
- How to **perform basic operations on data in R**, including storing data in objects, accessing that data, and creating tables of data.

Next week, you'll learn about how to read and write data in R, how to inspect your data, and how to perform operations on the data you have.