Intro_to_R

Kevin Healy

20/08/2023

Intro to R intro

This is the first intro to R. The best way to learn R is to use it. The learning curve at the start is steep but once you get used to the basics of coding the rest is mainly googling.

R is just a fancy calculator

You can add, multiply etc

```
1+1
```

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

```
#multiply
20*18
```

```
## [1] 360
```

To print names use " "

```
#print text
#It is important that you use " " for text

"Hello"
```

```
## [1] "Hello"
```

If you want R to remember something you need to save it as an object using the <- symbol. Do not use the equal symbol = for storing objects. Notice when you create an object it appears in the Global environment in the top right box.

```
#no. of oranges
oranges <- 10

#no. of apples
apples <- 5
```

If you want to print out the object simply type the name

```
#if you want to print what is in the object
oranges
```

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

R, like any code, will read everything in the most literal way so any difference in spelling, upper and lower case or spaces between words will lead to an error.

Its hard to break R so feel free to make errors.

```
orange
```

```
## Error in eval(expr, envir, enclos): object 'orange' not found
```

```
Oranges
```

```
## Error in eval(expr, envir, enclos): object 'Oranges' not found
```

Vectors, databases and other objects.

There are several classes of data, we will deal with Numeric, Character and Logic here. Numeric is numbers, Character is words and Logic is a special type used to make arguments. We cal ask what classa data type is in R using class().

```
#numeric data
class(1)
```

```
## [1] "numeric"
```

```
#character data
class("hello")
```

```
## [1] "character"
```

```
#logical data
class(TRUE)
```

```
## [1] "logical"
```

We can also store these data as vectors using c().

Lets say we say we observed the number of birds over 5 days, we could store the data like below.

```
no_birds <- c(2,5,10,24,12)
no_birds
```

```
## [1]  2  5 10 24 12
```

Numbers are stored as "numeric" however, we can also store words as characters. Lets say we recorded the weather as above.

```
weather <- c("rain","rain", "sun", "rain", "sun")
weather
```

```
## [1] "rain" "rain" "sun"  "rain" "sun"
```

Logical class data arises when we ask if something is true or false. For example, we can ask when is TRUE that an entrey in the vector weather is rain using the below line.

```
weather == "rain"
```

```
## [1]  TRUE  TRUE FALSE  TRUE FALSE
```

Dataframes combine data into rows and column in the same way you would deal with them in excel

```
#data.frames
bird_watching <- data.frame(no_birds, weather)
bird_watching
```

```
##   no_birds weather
## 1        2    rain
## 2        5    rain
## 3       10     sun
## 4       24    rain
## 5       12     sun
```

We can query the rows and column of the dataframe using square brackets dataframe[row numbers, column numbers]

```
#row 1 column 1
bird_watching[1,1]
```

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

```
#row 3 column 2
bird_watching[3,2]
```

```
## [1] "sun"
```

```
#all of row 1
bird_watching[1,]
```

```
##   no_birds weather
## 1        2    rain
```

```
#all of column 1
bird_watching[,1]
```

```
## [1]  2  5 10 24 12
```

We will cover this in more detail next week but we can also subset from data.frames using the name of the column.

```
#Can also use the name of the column
bird_watching[,"weather"]
```

```
## [1] "rain" "rain" "sun"  "rain" "sun"
```

We can also subset our data.frames using simple arguments. We can ask R which entries are equal to a given query. For example, for which rows was it raining.

```
#This asks which entries match "rain"
bird_watching[,"weather"] == "rain"
```

```
## [1]  TRUE  TRUE FALSE  TRUE FALSE
```

We can do these queries within the square brackets and R will return only the rows/column for which the query is TRUE

```
#The data.frame will only return results that are TRUE
bird_watching[bird_watching[,"weather"] == "rain",]
```

```
##   no_birds weather
## 1        2    rain
## 2        5    rain
## 4       24    rain
```

We can also ask to return values when it is not TRUE that it is raining by using !=

```
# !=  means not TRUE
bird_watching[,"weather"] != "rain"
```

```
## [1] FALSE FALSE  TRUE FALSE  TRUE
```

```
#The dataframe will only return results that are TRUE
#but here we will get values for when it is TRUE that
#the value is not rain.
bird_watching[bird_watching[,"weather"] != "rain",]
```

```
##   no_birds weather
## 3       10     sun
## 5       12     sun
```

There are many ways to subset data. However, while you might see it in some intro courses do not use the attach() function. This automatically creates a object for the each column. This can lead to overwriting previous objects.

```
#There are many ways to subset data the same way

bird_watching[,"weather"]
```

```
## [1] "rain" "rain" "sun"  "rain" "sun"
```

```
bird_watching[,1]
```

```
## [1]  2  5 10 24 12
```

```
bird_watching$weather
```

```
## [1] "rain" "rain" "sun"  "rain" "sun"
```

```
bird_watching[,]$weather
```

```
## [1] "rain" "rain" "sun"  "rain" "sun"
```

Functions

One of the most powerful reasons to use R is the use of functions. A function is some process where we do something to some set of data. For example, a function could be as simple as adding 1 to a list of numbers. However, we often want to do something more interesting. For example, we might want to find the mean of some set of numbers.

```
#create some list of numbers
input_data <- c(20,10,15)

mean(input_data)
```

```
## [1] 15
```

If you want to know more about a function you can type ? before the function and a help file will appear.

```
?mean
```