R- Basics

Prof. Tzu-Ting Yang

Institute of Economics, Academia Sinica

November 8, 2023

R script

- Please see **C2_basic**

- **R at various levels:**
  - R can be used as a (sophisticated) calculator.
  - R is good at statistical computing
  - R is a full-featured programming language.
  - R can create graphics on many devices.

R as a calculator

R as a calculator

```
1 + 1
[1] 2
2^3
[1] 8
```

- **Standard arithmetic operators**: +, -, *, /, and ^ are available
- **Details**:
  - In the output, [1] indicates the position of the first element of the vector returned by R.
- **Mathematical functions**: R has log(), exp(), sin(), asin(), cos(), acos(), tan(), atan(), sign(), sqrt(), abs(), min(), max(), ...

R as a calculator

```
log(exp(sin(pi/4)^2) * exp(cos(pi/4)^2))
[1] 1
```

- **Details**:
  - log(x, base = a) returns the logarithm of x to base a.
  - a defaults to exp(1).
  - Convenience functions: log10() and log2().
- **Further functions**: gamma(), beta(), and their logarithms and derivatives, are often useful in statistics and econometrics.

Objects

Objects

- **Fundamental design principle**: “Everything is an object”
- **Examples**: vectors and matrices are objects, but also functions and dataset

Create Objects

```
x <- 2
x
```

- Objects can easily be created by assigning a value (2) to a name (x), using the assignment operator <-
- Now, x is an object
- Typing an objects name (x) at the prompt, prints the object.

Create Objects

```
y = 3
y
```

- We can also use $=$ instead of $<-$
- People usually say it is better to use $<-$
- $<-$ is preferred for programming

Create Objects

```
a <-log(x = 16, base = 2)
a
```

- Argument in the functions could also use $=$

Create Objects

Create Vectors

```
z <- c(1.8, 3.14, 4, 88.169, 13)
z
Z
```

- **Basic unit**: Vector
  - All functions above operate directly on vectors
  - **c** stands for "combine" or "concatenate".
  - Use **c()** to generate a vector
- **Case-sensitivity**: z and Z are distinct.

Create Objects

Create Vectors

```
h <- seq(from = 2, to = 20, by = 2)
h
[1] 2 4 6 8 10 12 14 16 18 20

trend <- 1981:2005
trend
[1] 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 19
[13] 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2
[25] 2005
```

- **seq**: create a sequence of numbers from 2 to 20 (increase by 2)

Create Objects

Create Vectors

```
u <- c(z,h,x)
u
```

- We can combine z, h, and x vectors

Create Objects

Create Vectors

```
x1<- 2*x + 3
x2<- 1:5 * x + 5:1
log_x <-  log(x)
```

- Use x in subsequent calculations:
- **Details:**
  - **Line 1:** scalars 2 and 3 are recycled to the length of x.
  - **Line 2:** x is multiplied element-wise by the vector 1:5 (the integers from 1 through 5) and then the vector 5:1 is added element-wise.
  - **Line 3:** Application of mathematical functions.

Subsetting Vectors

```
z1 <- z[c(1, 4)]
z1

z2 <- z[-c(2, 3, 5)]
z2
```

- **Subsets of vectors**: Operator [can be used in several ways.
- **Details**:
  - Extract elements by their index
  - Exclude elements with negative index

Basic matrix algebra

- **Internally**: Matrices are vectors with an additional dimension attribute enabling row/column-type indexing.
- **Indexing**:
  - A[i,j] extracts element a_{ij} of matrix A.
  - A[i,] extracts i^{th} row.
  - A[,j] extracts j^{th} column.

Matrix operations

```
A <- matrix(1:6, nrow = 2)
A
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
```

- Creation: A 2×3 matrix containing the elements 1:6, by column, is generated via

```
B <- matrix(1:6, ncol = 3)
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
```

- Alternatively: Use **nrow** instead of **ncol**.

Basic matrix algebra

- **Illustration**:

```
A1 <- A[1:2, c(1, 3)]
A1
[,1] [,2]
[1,] 1 5
[2,] 2 6
```

- **Equivalently**:

```
A[, -2]
[,1] [,2]
[1,] 1 5
[2,] 2 6
```

List Objects

```
objects()

length(z)
```

- **Line 1:** List all objects in the global environment (i.e. the user's workspace)
- **Line 2:** Display the length of object

Remove Objects

```
remove(x)
```

- Removing objects with remove() or rm()

Functions and Packages

Functions and Packages

- R can be comprehended as a language that provides the basis for a system of code chunks, which have been developed by many people across the globe
- These code chunks are called **packages**
- And each of those packages usually consist of at least one **function**

Functions

- Functions are what makes R work
  - So it is necessary to understand what inputs they require from the user in order to work properly.
- Usually, a function has a **name** and can contain further **arguments**

Functions

- The syntax of a function is as follows:

  ```
  name_of_the_function(argument 1, argument 2, ...)
  ```

- Basically, a function requires the right arguments as inputs to produce some sort of output

Calling functions

```
y<-log(x = 16, base = 2)
z<-log(16, 2)
log(x = 16, base = 2)
[1] 4
```

- **Example**: The function log() has two arguments, x (a numeric scalar or vector), base (the base with respect to which logarithms are computed).
- The following calls all yield equivalent output:

```
log(16, 2)
log(x = 16, 2)
log(16, base = 2)
log(base = 2, x = 16)
```

Packages

- A **package** can be comprehended as a **collection of functions**
- Usually, a package serves a distinct purpose
- Example:
  - The package **ggplot2** is designed to generate nice graphics
  - The packages **foreign** is designed to load data files, which were originally stored in the file format of a different statistics software such as SAS, SPSS or Stata.

Install the Packages You Need

- The basic installation of R comes with a relatively small amount of pre-installed packages.
- Therefore, most packages must be installed by the user manually, before they can be used
- You could just execute the following line to install packages

```
install.packages(c("ggplot2", "foreign"))
```

```
install.packages("ggplot2")
install.packages("foreign")
```

一次灌多組套件

Load the Packages You Need


```
library(foreign)
library(ggplot2)
```

- Having installed all the necessary packages does not mean that we can use them without any further action
- Before we use the functions of a particular package, we have to load the package whenever R is started
- This is commonly done with the **library()** function

How to Use Help

How to Use Help

```
?lm
??lm
help("lm")
```

- **Documentation**: The help page for any function or data set can be accessed using either ? or help():

```
example("lm")
```

- **Examples**: At the bottom of a help page, there are typically practical examples of how to use that function.

Searching for help

- If the exact name of a command is not known, the functions to use are help.search() and apropos().

```
help.search("linear")
```

- help.search() returns help files with aliases or concepts or titles matching a "pattern" using fuzzy matching

```
apropos("lm")
```

- apropos() lists all functions whose names include the pattern entered.

Reserved words

- **Reserved words**: Basic grammatical constructs of the language that cannot be used in other meanings.
- **In R**: if, else, for, in, while, repeat, break, next, function, TRUE, FALSE, NA, NULL, Inf, NaN, ...).
- See ?Reserved for a complete list.

Quitting R

```
R> q()
```

- One exits R by using the q() function:
- R asks whether to save the workspace:
  - n (no): exit R without saving anything,
  - y (yes): save all currently defined objects in .RData and the command history in .Rhistory, both in the working directory.

Acknowledgement

- This slide is heavliy borrowed from Prof. Achim Zeileis
- Please visit his website to get more information
- `https://eeecon.uibk.ac.at/~zeileis/teaching/AER/`