

R Training Session 1: Basics of R Programming

Data Avicenna

Department of Economics
Universitas Gadjah Mada

May 7, 2025

Outline

- 1 Introduction
- 2 Basic Building Blocks
- 3 Vectors
- 4 Functions
- 5 Conditional Statements
- 6 Assignment
- 7 References

What is R?

R is a powerful programming language and environment for statistical computing, it can be used for many data-related tasks and economic analysis:

- Load and manipulate data from almost any source
- Make descriptive statistics and advanced graphs
- Fit all sorts of statistical and econometric models
- Easily write simulations for statistical or other types of models
- Write your own functions/ programs and share them
- And more...

Why use R for economics?

- **Free and open-source** - No licensing costs unlike Stata or MATLAB
- **Powerful** - Advanced statistical and econometric capabilities
- **Flexible** - From simple calculations to complex economic models
- **Reproducible** - Code documents every step of your analysis
- **Industry-relevant** - Used in academia, economic research institutions, etc.
- **Growing community** - Extensive packages available
- **Publication-quality outputs** - Good-looking graphs and reports

Installing R

- R download link: <https://cran.rstudio.com>. RStudio download link: www.rstudio.com/products/rstudio/download/.
- Download the core programming language *R* itself and *RStudio* which is an *IDE* (integrated development environment), optimized for *R*. Using an IDE makes *R* more user-friendly, which helps you handle the program better.

The screenshot displays the RStudio IDE interface. The main editor window shows an R script with the following content:

```
1 ## 1. INSTALLATION R###
2
3 # LOAD DATA R###
4
5 library(datasets) # Load built-in datasets
6
7 # SAMPLE SIZE DATA R###
8
9 head(iris) # Show the first six lines of iris data
10 summary(iris) # Summary statistics
11 plot(iris) # Scatterplot
12
13 # CLEAN UP R###
14
15 # Clear packages
16 detach("package:datasets", unload = TRUE) # For base
17
18 # Clear plots
19 dev.off()
20
21 # Clear console
22 cat("\054") # ctrl+L
23
24
25 # 2. PACKAGES R###
26
27 ## 2.1. READ TABLE FOR TEXT FILES
```

The Environment pane on the right shows the following variables:

Variable	Value
my.dir	man [1:3] 3.48 3.18 2.15
my.samt	man [1:3] 0.316 2.828 1.463
old.dir	"/Users/dotson/cenno"
x	9
y	9
z	man [1:3] 1.1 9 3.14

The Files pane at the bottom shows the following content:

```
R: List the Files in a Directory/Folder - Find in Topic
List the Files in a Directory/Folder
Description
These functions produce a character vector of the names of files or directories in the named directory.
Usage
list.files(path = ".", pattern = NULL, all.files = FALSE,
full.names = FALSE, recursive = FALSE,
ignore.case = FALSE, include.dirs = FALSE, no.. = FALSE)
dir(path = ".", pattern = NULL, all.files = FALSE,
full.names = FALSE, recursive = FALSE)
```

The Four Panes in RStudio

- ❶ **Script pane** (top left): Where you write, edit, and save R codes. Create reproducible documents. Note: You may use comments (#) to write notes in your script.
- ❷ **Environment pane** (top right): List all objects in memory and tracks command history.
- ❸ **Console pane** (bottom left): Where commands are executed. Shows results of code execution. Direct interaction with R.
- ❹ **Files/Plots/Packages/Help** (bottom right):
 - File browser and working directory management
 - Display plots and visualizations
 - Install and load packages
 - Access documentation

Basic Operations

At its simplest level, R can function as an interactive calculator:

```
> 5+7  
[1] 12
```

Instead of repeating calculations, we can store results in variables:

```
> x <- 5+7
```

The assignment operator (`<-`) stores values but doesn't display them automatically. You can print the value:

```
> x  
[1] 12
```

More Arithmetic Operations

R supports all other common arithmetic operations useful for calculations:

```
# Multiplication
```

```
> 100*100  
[1] 10000
```

```
# Square root and division
```

```
> sqrt(25) + (200/5)  
[1] 45
```

```
# Exponentiation
```

```
> 1000 * (1.05)^3  
[1] 1157.625
```


A Brief Look at Objects and Functions in R

- In R, we create objects by assigning values to names:

```
a <- 1                # a single value object
```

```
a <- c(1, 2, 3)       # vector with multiple elements
```

- Functions process input objects and return output objects:

```
# Using the built-in sum function
```

```
# Calculates sum and stores in new object
```

```
sum_of_a <- sum(a)
```

Atomic Objects in R

Atomic Objects

R has five types of atomic objects:

- 1 **numeric** objects are all real numbers on a continuous scale
e.g., 1.23456
- 2 **integer** are all full numbers
e.g., 1, typed as 1L
If you want an integer you have to explicitly use the L suffix.
Otherwise, R will assign the number to the numeric class.
- 3 **complex** is used for complex numbers
e.g., $a + bi$, i.e., real + imaginary

Atomic Objects in R

Atomic Objects (continued)

- ④ **boolean** values are logical values like **TRUE** and **FALSE**.
- ⑤ **strings** are characters [e.g., **"Hello World!"**].
Use the **'** or the **"**.
Note that all other atomic objects can be converted into strings.

Data Structures in R

These data structures organize atomic objects for different analytical needs:

- **Vectors:** several elements of a single atomic type
- **Matrices:** collections of equal-length vectors
- **Factors:** categorical data (ordered, unordered)
- **Data frames:** a data set, collections of equal-length vectors of *different* types
- **Lists:** collections of unequal-length vectors of *different* types

We'll focus primarily on vectors, which are fundamental for understanding all other data structures.

Vectors

- Vectors are one of the most fundamental data structures in R.
- Create vectors using the `c()` function ("combine" or "concatenate")
- Useful for storing related data (like time series, multiple observations, etc.)

Vectors

```
# Numeric vector
```

```
> y <- c(1, 2, 3, 4, 5)
```

```
> y
```

```
[1] 1 2 3 4 5
```

```
# Character vector
```

```
> hello <- c("Hello", "World")
```

```
> hello
```

```
[1] "Hello" "World"
```

```
# Mixing types (converts to most flexible type)
```

```
> mixed <- c(1, "text", TRUE)
```

```
> mixed
```

```
[1] "1"      "text" "TRUE"
```

Vector Operations

```
# Creating a vector
> prices <- c(10, 15, 20, 25)
# Operations apply to each element
> prices * 2 # Double all prices
[1] 20 30 40 50
# Calculate total revenue if quantities sold are:
> quantities <- c(5, 3, 2, 1)
> revenue <- prices * quantities
> revenue
[1] 50 45 40 25
# Get the total
> sum(revenue)
[1] 160
```

Vectors of Atomic Objects

R automatically assigns the correct object type. Let's look at this in practice: (Remember: Vectors only contain a single atomic type!)

```
x <- c(1.1, 2.2, 3.3)
is.numeric(x)
> [1] TRUE
```

```
x <- c(1L, 2L, 3L)
is.integer(x)
> [1] TRUE
```

```
x <- c(1+0i, 2+4i, 3+6i)
is.complex(x)
> [1] TRUE
```


Vectors of Atomic Objects

```
x <- c(TRUE, FALSE, TRUE)
is.logical(x)
> [1] TRUE
```

```
x <- c("I", "like", "R")
is.character(x)
> [1] TRUE
```

Use the following commands to identify the object type:

```
typeof()
mode()
```

Exercise 1.1

- ① Answer the following (type your answers in an R script or directly in the console):
 - Create a vector containing the numbers 1.1, 9, and 3.14. Store the result in a variable called `z`.
 - Create a new vector that contains `z`, 555, then `z` again in that order. Don't assign this vector to a new variable, so that we can just see the result immediately.
 - Take the square root of `z - 1` and assign it to a new variable called `my_sqrt` (Hint: Use the `sqrt()` function).
 - What do you think `my_sqrt` contains? (Choose one from below)
 - a a vector of length 0 (i.e., an empty vector)
 - b a vector of length 3
 - c a single number (i.e., a vector of length 1)

Exercise 1.1

2 Vector recycling.

- For z containing the elements 1.1, 9, and 3.14, compute $z * 2 + 100$. After that, compute $z * c(2, 2, 2) + c(100, 100, 100)$. Are the answers different?
- Add $c(1, 2, 3, 4)$ and $c(0, 10)$. What do you observe?
- Add $c(1, 2, 3, 4)$ and $c(0, 10, 100)$. What do you observe?

Logical Operators in R

Logical operators are (vectors of) TRUE and FALSE statements. They are helpful for example for checking outcomes.

Operator	Description
<	Test for less than
<=	Test for less than or equal to
>	Test for greater than
>=	Test for greater than or equal to
==	Test for equality
!=	Test for if not equality
!x	Boolean negation, for vectors
x y	Boolean x OR y, for vectors
x & y	Boolean x AND y, for vectors
x y	Boolean x OR y, for scalars
x && y	Boolean x AND y, for scalars
isTRUE(x)	Boolean test if X is TRUE, for scalars

Note: scalars = vectors of length one.

Logical Operators in R

Examples:

```
x <- 5; y <- 10
```

```
x < y           # TRUE
```

```
x == (2+3)     # TRUE
```

```
(x > 3) & (y < 20) # TRUE (vector AND)
```

```
(x > 7) | (y == 10) # TRUE (vector OR)
```

```
is.logical(x < y)  # TRUE (returns logical value)
```

Factors: Categorical Vectors

- **Factors** are special vectors for categorical data (ordered and unordered). Examples:

```
# Creating a basic factor
```

```
education <- factor(c("High School", "Bachelor", "Master",  
                      "Bachelor", "PhD", "High School"))
```

```
# Examining the factor
```

```
levels(education) # Shows all unique categories
```

```
# [1] "Bachelor"      "High School"   "Master"        "PhD"
```

```
# Creating a factor with custom levels and order
```

```
income_level <- factor(c("Medium", "Low", "High", "Medium", "Low",  
                          levels = c("Low", "Medium", "High"),  
                          ordered = TRUE)
```

```
# Now we can use comparison operators
```

```
income_level[1] > income_level[2] # TRUE - Medium > Low
```

Functions

Just like mathematical functions, e.g. $y = f(x)$, an R function receives one or multiple inputs, then does something with these inputs, and returns something.

- `c()` & `sum()` are examples of functions!

```
> c(1, 2, 3)
```

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

```
> sum(1, 2, 3)
```

```
[1] 6
```

- You can create your own functions:

```
my_function <- function(x, y) {  
  # Function body goes here  
  return(x + y)  
}
```

Functions

- Create your own functions example:

```
> x <- c(1, 2); y <- c(2, 3)
> my_function <- function(x, y) {
  # Function body goes here
  return((x + y)/2)
}
> my_function(x, y)
[1] 1.5 2.5
```


Functions

- You can view function documentation with `?function_name`:

```
?sum # Opens help for the sum() function
```

```
?c   # Opens help for the c() function
```

- R functions are polymorphic - their behavior adapts to the input type:

```
# Summary of numeric vector
```

```
summary(1:10)
```

```
# Summary of factor
```

```
summary(factor(c("A", "B", "A")))
```

Common R Functions for Vectors

- `mean(x)` computes the mean
- `sd(x)` computes the sample standard deviation
- `var(x)` computes the sample variance
- `median(x)` computes the median of a vector
- `quantile(x, probs=...)` computes the supplied quantiles
- `summary(x)` summarizes the input object
- `cor(x,y)` computes the correlation of two vectors
- `cov(x,y)` computes the covariance of two vectors
- `abs(x)` takes the absolute value of a vector
- `sqrt(x)` takes the square root of a vector
- `log(x)` takes the natural logarithm
- `exp(x)` exponentiates the vector
- `min(x)` returns the minimum of a vector
- `max(x)` returns the maximum of a vector
- `sum(x)` returns the sum of a vector
- `prod(x)` returns the product of a vector
- `round(x, digits)` rounds to specified digits
- `trunc(x)` truncates the vector to an integer
- `cumsum(x)` returns the running sum of a vector

Functions in Practice

Examples:

```
gdp_growth <- c(2.1, 2.5, 3.0, 2.7, 1.8)
mean(gdp_growth)      # [1] 2.42
sd(gdp_growth)        # [1] 0.4764452
median(gdp_growth)    # [1] 2.5
summary(gdp_growth)   # Min: 1.80, 1st Qu: 2.10,
                      # Median: 2.50, Mean: 2.42,
                      # 3rd Qu: 2.70, Max: 3.00
```

Conditional Statements in R

- Sometimes code should only run under certain conditions
- Conditional statements let your code make decisions
- In R, we use `if`, `else if`, and `else` statements

Basic if statement

The basic syntax of an if statement:

```
if (condition) {  
  # Code to execute when condition is TRUE  
}
```

Example: Print a message if GDP growth is positive

```
gdp_growth <- 2.5 # Annual GDP growth rate
```

```
if (gdp_growth > 0) {  
  print("The economy is growing")  
}  
# Output: [1] "The economy is growing"
```

if-else statements

Adding else for alternative actions:

```
if (condition) {  
  # Code to execute when condition is TRUE  
} else {  
  # Code to execute when condition is FALSE  
}
```

Example: Determine economic condition

```
gdp_growth <- -0.5 # Annual GDP growth rate
```

```
if (gdp_growth > 0) {  
  print("The economy is growing")  
} else {  
  print("The economy is contracting")  
}  
  
# Output: [1] "The economy is contracting"
```

if-else if-else statements

Multiple conditions with else if:

```
if (condition1) {  
  # Code for condition1 TRUE  
} else if (condition2) {  
  # Code for condition1 FALSE, condition2 TRUE  
} else {  
  # Code for all conditions FALSE  
}
```

Example: Classify economic conditions

```
gdp_growth <- 1.2 # Annual GDP growth rate
```

```
if (gdp_growth > 5) {  
  print("Strong economic growth")  
} else if (gdp_growth > 0) {  
  print("Moderate economic growth")  
} else if (gdp_growth > -1) {  
  print("Economic slowdown")  
} else {  
  print("Economic recession")  
}
```

The ifelse() Function

R has a vectorized `ifelse()` function for element-wise operations:

```
ifelse(test, yes, no)
```

This is much faster than using loops with if-else for vectors:

```
# Create vector of growth rates for different regions  
regional_growth <- c(2.1, -0.5, 3.2, 0.1, -1.2)
```

```
# Classify each region  
region_status <- ifelse(regional_growth > 0,  
                        "growing",  
                        "contracting")
```

```
region_status  
# [1] "growing" "contracting" "growing" "growing" "contracting"
```


Assignment 1

You need to submit your answers to this assignment to obtain a training certificate. Submit your R script to this link:

<https://tinyurl.com/R-Assignment-1-2025>. Contact dataavicenna@mail.ugm.ac.id for any questions.

Submission deadline: **13 May 2025 at 23.59pm**

Assignment 1

Write your answers in an R script:

- ① Create a vector consisting of integers from 0 to 1000 and assign it to a variable called A. Use the appropriate functions to calculate the mean, standard deviation, and median of A.
- ② Logical operations with vectors:
 - ⓐ Create a vector `gdp_growth` containing the values 2.1, 2.5, 3.0, 2.7, and 1.8.
 - ⓑ Create logical vectors that identify: (i) Which values are greater than 2.5; (ii) Which values are less than or equal to 2.0; (iii) Which values are exactly 2.7. Store each of the logical vector into an R object and name them however you want.
 - ⓒ Use the `sum()` function on each of these logical vectors. What does the result represent?
 - ⓓ Create a single logical vector that identifies which values are between 2.0 and 3.0 inclusive (those equal to 2 or 3 should also be included).

References

Schmidt, S. S., & Turbanisch, F. (n.d.). *Economic Analysis with R*. University of Göttingen. Retrieved from
<https://economic-analysis-with-r.uni-goettingen.de/>

Team swirl. (n.d.). *R Programming*. GitHub. Retrieved from
https://github.com/swirldev/swirl_courses

Appendix: R Time-Saving Tricks

- Use the up arrow ↑ on your keyboard to cycle through previous commands in your Console.
- If you forgot the name of a variable that you have created, type the first two letters of the variable name, then hit the Tab key. A list of variables will appear.
- R script shortcuts: (Use Command instead of control in MacOS)
 - a CTRL + SHIFT + N to create a new R script
 - b CTRL + SHIFT + ENTER to run all lines
 - c CTRL + ALT + B to run until current line (Command + Option + B in MacOS)
 - d CTRL + ENTER to run the code line by line while ignoring comments
 - e `rm(list = ls())` → code to remove all objects in the environment
 - f CTRL + L to clean the console