Introduction to R: Part II

David Rieger and Elizabeth Waterfield

09/11/2026

Prerequisites

> **!** Important
>
> Before completing this session, please carefully read about the prerequisites.

Prerequisite	Description	Where to find it
R and RStudio installed	Latest versions of both R and RStudio	Download RStudio
Accessibility sheet for computers	Fundamentals of computer literacy	Download
Basic R skills	Introduction to R - Part I	Access here

- R: https://www.r-project.org/

Presenter Notes: These are the **presenter notes**. You will find a script for the presenter for every slide. In presentation mode, your audience will not be able to see these presenter notes, they are only visible to the presenter.

Welcome back! Before we dive into Part II, please make sure you still have R and RStudio installed and ready to go. You should also be familiar with the basic concepts we covered in Part I, like the RStudio interface and creating simple objects.

Instructor Notes: There are also **instructor notes**. For some slides, there will be pedagogical tips, suggestions for activities and troubleshooting tips for issues your audience might run into. You can find these notes underneath the presenter notes.

Quickly verify whether learners have the required software open. Remind them that they can open the R scripts they saved during the last session.

Accessibility Tips: Where applicable, this is a space to add any tips you may have to facilitate the accessibility of your slides and activities.

Consider pairing learners who might need a quick refresher on Part I with those who feel very confident.

Questions from previous sessions

Are there any open questions regarding **Introduction to R - Part I**?

Presenter Notes: Before we start with new topics, are there any lingering questions from Part I? For example, anything regarding data types, logical comparisons, or saving your scripts?

Instructor Notes: This slide is dedicated to clarifying questions from the previous session and/or to discuss assignments. It is critical for the learning process to ensure that learners are on the same page and have been able to achieve the learning objectives of the previous workshop.

Let's refresh our memories!

Question 1

Which of the following creates an object named `my_data` containing the number `42`?

a. `my_data == 42`

b. `my_data <- 42`

c. `my_data < - 42`

d. `42 <- my_data`

Question 2

What is the result of the following logical comparison?

```
(5 > 3) & (10 == 5)
```

a. TRUE

b. FALSE

c. Error

Question 3

How do you access a column named age from a data frame called people_data?

a. people_data[age]

b. people_data(age)

c. people_data$age

d. age$people_data

Presenter Notes: To get our brains back into "R mode", let's do a very quick quiz on two core concepts from our last session. Feel free to just call out the answers!

Question 1: The correct answer is B. The standard assignment arrow points to the left without any spaces. Option A uses a double equals sign, which is a logical operator that checks if an existing object is exactly 42, rather than assigning it. Option C contains a space between the less-than sign and the minus sign, which R interprets as a logical question asking if the object `my_data` is smaller than negative 42. Option D places the raw number on the left side of the arrow, which causes an error because the object name receiving the data must always be on the left.

Question 2: The correct answer is B, FALSE. While 5 is indeed greater than 3 (which is TRUE), 10 does not equal 5 (which is FALSE). Since we used the AND operator (&), both conditions must be true for the final result to be TRUE.

Question 3: The correct answer is C. The dollar sign ($) is used to extract a specific column from a data frame by its name. Option A is missing the necessary quotation marks for basic bracket indexing. Option B incorrectly uses parentheses, which are reserved for functions. Option D reverses the order; the data frame name must always precede the dollar sign.

Instructor Notes: Use these two quiz slides to actively engage learners instead of just reading out a recap. Ensure everyone understands the assignment operator and basic logical checks before moving on.

Before we start: Survey time!

Question 1

What is your level of familiarity with "Packages" in R?

a. I have never heard of them.
b. I have heard of them but never installed or loaded one.
c. I have basic experience installing and loading them.
d. I am very familiar and use them regularly.

Question 2

Have you ever imported a dataset (e.g., an Excel, CSV, or SPSS file) into a statistical software program?

a. No, never.
b. Yes, using menus (point-and-click).
c. Yes, using code/commands.

Question 3

How familiar are you with creating data visualizations (e.g., histograms or scatter plots)?

a. I have never done this before.
b. I have done it using point-and-click tools (e.g., Excel or SPSS).
c. I have done it using code (e.g., in R or Python).

Presenter Notes: Now that we've refreshed Part I, let's take a short survey regarding today's topics. In Part II, we are moving away from R as a simple calculator and starting to work with real data. In the next few slides, you'll see some questions about concepts we will cover today. As always, this isn't a test. If you have never heard these terms, that is exactly why we are here!

Instructor Notes: Set a low-stakes, supportive tone. Emphasize that uncertainty is expected and valuable for measuring learning progress.

For Question 3: Ask the learners about their experience with creating data visualizations, such as histograms or scatter plots. In Part II, we will cover how to create simple histograms.

Discussion of survey results

What do we see in the results?

Presenter Notes: Let us have a look at these results.

Instructor Notes: The aim is to understand the group's current standing in terms of knowledge and familiarity. Emphasize explicitly that the survey is not about correct or incorrect answers. Avoid evaluating answers.

Moving forward: From Part I to Part II

- **Part I (the basics)**:
  - R as a calculator
  - Data types
  - Basic functions
  - Writing basic scripts
- **Part II (the application)**: Transitioning from calculating to analyzing
  - Understanding where files are saved (*working directories*).
  - Using tools built by others (*packages*).
  - Bringing in real data (*importing & inspecting*).

Presenter Notes: In Part I, we learned the alphabet of R: data types, math operators, functions, and how to save a script. Today, we start writing sentences. We are shifting from treating R like a calculator to using it as a data analysis tool. By the end of this session, you will know how to load your own research data into R and start exploring it.

Learning objectives

At the end of this session, you will be able to…

- **Identify** and **set** the correct working directory on your computer
- **Install** and **load** R packages to extend R's functionality
- **Import** external data into the R environment
- **Inspect** data structures and **extract** specific columns
- **Compute** basic statistical summaries on a dataset
- **Visualize** data distributions using simple histograms

Presenter Notes: This slide outlines the learning objectives for today's session. Just like last time, think of them as a roadmap. They show where we're headed today. You can use these objectives to keep track of your learning and identify areas where you might want to ask questions later.

Key terms and definitions

- **Working directory**
- **Package**
- **Dataframe**
- **Subsetting**

Presenter Notes: These are key terms we will revisit throughout the lesson. What comes to mind when you see each of them? Which ones are familiar, and which feel new? Take a moment to think about what they might mean.

Instructor Notes: Use a think-pair-share approach. First, have learners reflect individually on their associations with each term for a few minutes, then discuss in pairs for a few minutes, and finally share with the group. Allow about 10 minutes for this activity.

Key terms and definitions

- **Working directory**: The default folder on your computer where R looks for files to read and where it saves your output.
- **Package**: A collection of additional R functions, data, and code that extends the basic capabilities of R.
- **Dataframe**: A table-like structure in R where rows represent observations and columns represent variables.
- **Subsetting**: The process of extracting a specific portion of a dataset, such as selecting specific columns or filtering rows.

Presenter notes: Let's go through what these key terms mean:

- The Working directory is the folder R is currently focused on. It reads files from here and saves outputs here.
- Packages are add-ons. They contain tools and functions created by the R community that expand what base R can do.
- A dataframe is the standard way to store data in R. Think of it like a spreadsheet where each column is a variable and each row is an observation.
- Subsetting allows us to isolate the exact data we need. Instead of working with an entire dataset, we extract only the relevant rows or columns.

The working directory



What *is* the working directory?

- The **working directory (WD)** is the default folder on your computer.
  - It is the *location where R looks for files to read.*
  - It is the *location where R saves files* you "produce" (e.g., plots, exported data).
- **Rule:** Always verify your working directory before beginning an analysis to get an idea in which environment you are in.

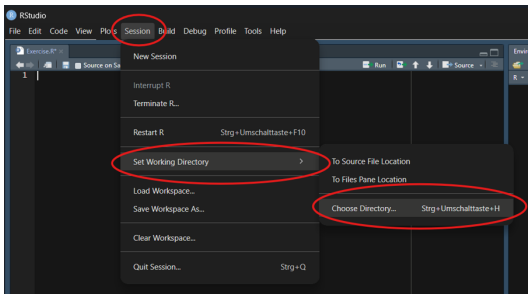
Presenter Notes: The working directory is your anchor point. If you tell R to open a dataset without specifying the full path on your computer, R looks in the working directory. If you tell R to save a plot, it saves it there. Setting it correctly prevents "file not found" errors.

Instructor Notes: Use the analogy of a physical desk. If you ask someone to hand you a paper, they will look on the desk they are currently standing at. If the paper is in another room, they won't find it unless you point them to the correct room.

Setting the working directory (I)

Method 1: Using the top menu bar in RStudio

1. Click **Session**
2. Hover over **Set Working Directory**
3. Select **Choose Directory...**
4. Navigate to your desired folder and click **Open**.



Open RStudio on your own machines. We will use it for later exercises.

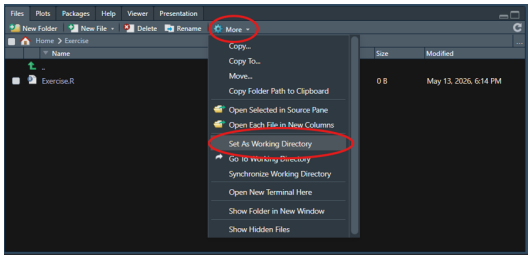
Presenter Notes: This is the standard graphical way to set your directory. It opens your operating system's standard file browser.

Instructor Notes: Briefly demonstrate this live. Ensure learners know how to navigate their local file system (e.g., finding their Documents or Desktop folder). Encourage learners to open their own RStudio to follow along during the later exercises.

Setting the working directory (II)

Method 2: Using the output pane (bottom right in default RStudio layout)

1. Navigate to your desired folder in the **output** pane.
2. Click the **More** button (gear icon).
3. Select **Set As Working Directory**.

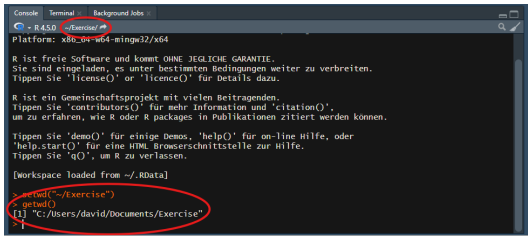


Presenter Notes: This method is often faster if you already see your files in the bottom right corner of RStudio. Clicking 'Set As Working Directory' locks R to that specific folder.

Instructor Notes: Remind learners that simply navigating to a folder in the output pane does not automatically set it as the working directory. They must explicitly execute the "Set As Working Directory" command.

Checking the working directory

Verify the current working directory by looking at the top of the **console** pane (bottom left):



> **Tip**
>
> Alternatively, type `getwd()` in the console and press Enter to print the exact working directory path.

Presenter Notes: You can always verify your current location by checking the gray text at the top of your console. If it shows the wrong path, your file imports will fail. You can also run the command `getwd()`, which stands for "get working directory".

Accessibility Tips: Read the console path structure aloud (e.g., "C drive, Users, Documents") to help learners orient themselves.

Practical exercise 1

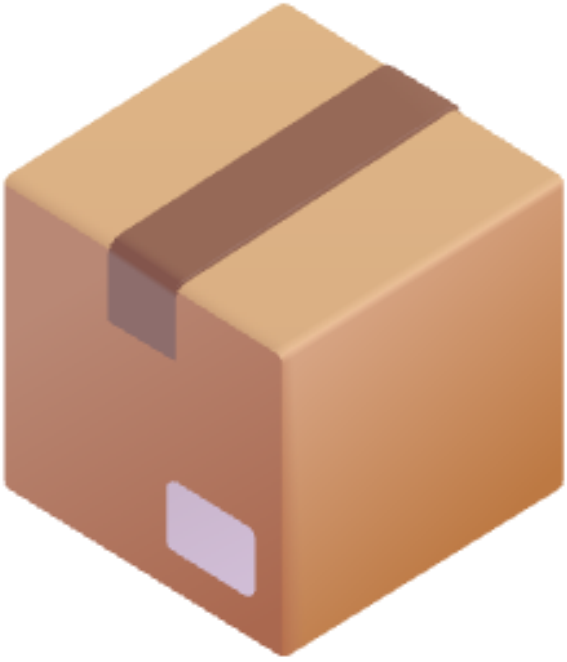
Setting the working directory

- ☐ Generate a new folder for this session's files
- ☐ Open RStudio
- ☐ In RStudio, set the new folder as your working directory using *either* Method 1 or Method 2
- ☐ Verify that the correct path is displayed at the top of your console

Presenter Notes: Open RStudio. Take a minute to create and set your working directory to the folder for today's session. We will need this set correctly when we start importing data later today.

Instructor Notes: Verifying the working directory is a critical step. Do not proceed until all learners have successfully set their directory. Check for common issues like OneDrive/Cloud sync folders causing path errors on Windows machines, or learners not typing the command correctly.

Packages



What are packages?

- R comes with many built-in functions (**Base R**)
- **Packages** provide thousands of additional, specialized functions
- They are created and shared by the R community to extend functionality

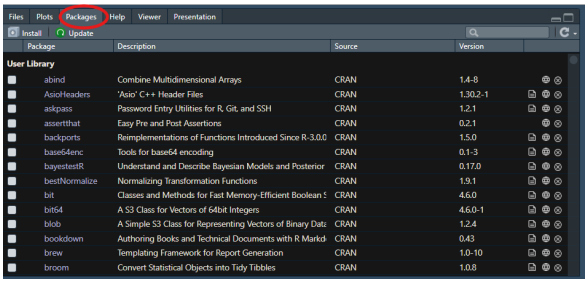
Presenter Notes: While Base R is powerful, the real strength of R lies in its packages. These are bundles of code written by other users that add new capabilities, from advanced statistical models to data import tools.

Installing packages

- Packages must be **installed once** (*or after updating R*).
- Use the `install.packages()` function. The package name must be in quotation marks.

```
install.packages("haven")
```

- Alternatively: Use the **Packages** tab in the **output** pane (bottom right) of RStudio and click **Install**.



Presenter Notes: You only need to install a package once on your computer. Note that the package name must be enclosed in quotation marks when installing. You can also use the Packages tab in RStudio to install them graphically.

Loading packages

- Once installed, a package must be **loaded every time** you open a new R session.
- Use the `library()` function. Quotation marks are not required here.

```
library(haven)
```

Presenter Notes: Think of installing a package like buying a book and putting it on your shelf. You only buy it once. Loading a package with the library function is like taking the book off the shelf to read it today. You have to do this every time you start a new session in R.

Instructor Notes: Emphasize the difference between installing (quotes needed, done once) and loading (no quotes needed, done every session). This is a very common point of confusion for beginners.

Practical exercise 2

Packages

- ☐ In your working directory, create a new R Script with `File > New File > R SCript`
- ☐ Install the package `haven`. This package is useful for reading in data files from other software like SPSS, Stata, and SAS.
- ☐ Load the package `haven` so it is ready to use in your current session.
- ☐ Add the command to install and load the package to your R script.

Presenter Notes: Take a moment to install the haven package and then load it. We will need this package later if we want to import specialized data formats.

Instructor Notes: Give learners about 3 minutes. Remind them to look at the console to see the installation progress.

Practical exercise 2: Solution

This is what you should add to your R script:

```
# 1. Install the package (run only once)
install.packages("haven")

# 2. Load the package (run every session)
library(haven)
```

Presenter Notes: First, we use `install.packages` with quotation marks to download the package to our computer. Then, we use `library` without quotation marks to load it into our current session.

Getting started with data



Preparation to work with data

- To start analyzing data, we first need to download it to our local machine.
- Click **here** to download the `characters.rds` file use the link below.

https://github.com/nickhaf/r_tutorial/tree/main/raw_data

Presenter Notes: Before we can load and inspect data, we need to have it on our computers. Please use the link or scan the QR code to download the `characters.rds` file. This dataset contains fun information about TV show characters, specifically from "Brooklyn Nine-Nine" and "Community", including variables like their show's name and a "notability" score. We will use this dataset to practice data exploration and visualization.

Instructor Notes: Ensure learners do not skip this step. The specific instructions for moving the file and setting the directory follow on the next slide.

Practical exercise 3

Setting up the data

- ☐ Download the dataset `characters.rds` using the link from the previous slide or click here
- ☐ Move the dataset into **your working directory**
- ☐ Verify your working directory once more

Presenter Notes: Once downloaded, make sure to move the file out of your general Downloads folder and into a new R folder with your files for this session. Finally, verify that RStudio is pointing to that exact folder as your working directory.

Instructor Notes: This step often causes friction as learners may struggle to locate their downloads or move files. Use the upcoming break to help anyone who is stuck here. Ensure they do not leave the file in their standard "Downloads" folder, as this will cause paths issues later.

Loading data into R

Once your **working directory is set**, you can **load your data** into RStudio. Check the file extension:

- **`.rds` files:** R's native data format. We use the `readRDS()` function to load them.
- **`.csv` files:** The most common text-based data format. Usually loaded with functions from the `readr` package.
- **Other formats (SPSS, Excel, SAS):** Require specific packages (e.g., the `haven` package you installed earlier!).

> Point-and-Click Import
>
> You don't always have to write the code from scratch! You can go to **File** → **Import Dataset** in the top menu to point-and-click your way through the import process. RStudio will even generate the correct code for you.

Presenter Notes: Now that R knows where to look, let's load some data. The function you use depends on the file type. Because we downloaded an `.rds` file, which is R's native format, we will use the `readRDS` function. For other common formats like CSVs, or files from SPSS and Excel, you will need different functions and sometimes extra packages like `haven`. If you ever forget how to load a specific file type, simply Google "how to open xlsx in RStudio". Alternatively, you can use the "Import Dataset" button in the menu, which is highly recommended because it gives you a preview of the data and writes the code for you.

Instructor Notes: Briefly demonstrate the "Import Dataset" feature via screen share if time permits, showing how it populates the console with code. Emphasize that Googling for specific file type imports is standard practice.

Practical exercise 4

Loading the data

- ☐ In your R script, create a new section using a comment (e.g., `# Loading Data`).
- ☐ Use the `readRDS()` function to read the `characters.rds` file. Remember to put the file name in quotation marks!
- ☐ **Assign** the output of this function to a new object called `characters`.
- ☐ Run the code and check your Environment pane.

Presenter Notes: It is time to start our main data analysis script! We will build on this script for the rest of the session, so you can save it and use it as a cheat sheet for your own future projects. Start by writing the code to load the `characters.rds` file and assign it to an object named `characters`.

Instructor Notes: Give learners about 3-5 minutes. Remind them that if they get a "file not found" error, it almost certainly means their Working Directory is not set to the folder where the file actually lives, or they made a typo in the file name.

Practical exercise 4: Solutions

This is what you should add to your R script:

```
# Loading Data
characters <- readRDS("characters.rds")
```

Presenter Notes: Here is the solution. We use the assignment arrow to store the data in an object called characters. The readRDS function tells R to look for the file named "characters.rds" inside our Working Directory. If everything worked correctly, you should now see a new object appear in the top right Environment pane, showing that it has hundreds of rows and several variables.

Instructor Notes: Have the participants confirm they see the dataset in their Environment. Do not move forward until everyone has successfully loaded the data, as the rest of the workshop depends on it.

Inspecting data structures

Before analyzing data, you must **familiarize yourself** with its **structure**. Built-in functions allow you to verify variables and data types:

- `names()`: Returns the column names.
- `str()`: Displays the internal structure (number of rows/columns, data types, and the first few values of each variable).
- `head()`: Returns the first six rows of the dataset for a quick preview.
- `summary()`: Provides a brief statistical summary of each variable (adapts output based on data type).

> Codebooks
>
> Variable names are often abbreviated. It is highly advisable to use a **codebook** for precise definitions. Check out our training material on data dictionaries: https://lmu-osc.github.io/data-documentation-validation-R/02-data-dictionaries/page1.html to learn more!

Presenter Notes: Once the data is loaded, we need to look at it. You should never start analyzing without knowing what variables you have. We use four main functions for this. `names` gives you the column headers. Keep in mind that column names are often short, so having a codebook explaining what each variable means is highly recommended. `str` shows the structure, including whether variables are numbers or text. `head` shows the first six rows. `summary` provides basic statistics, like means for numeric data or counts for categorical data.

Instructor Notes: Emphasize the importance of a codebook. Novices often assume variable names are self-explanatory, which leads to analytical errors.

Practical exercise 5

Inspecting the dataset

- ☐ Add a new comment in your R script (e.g., `# Inspecting Data`).
- ☐ Run `names(characters)`.
- ☐ Run `str(characters)`.
- ☐ Run `head(characters)`.
- ☐ Run `summary(characters)`.

Question: Based on the console outputs, can you guess what the data in this dataset is about?

Presenter Notes: Let us apply these functions to the dataset we just loaded. Run each function on the `characters` object and look closely at the output in the console. Based on what you see—the variable names and the specific text values—can anyone guess what this dataset is actually about?

Instructor Notes: Give learners 5 minutes to run the code and read the outputs. Ask the room to share their guesses regarding the dataset's content.

Practical exercise 5: Solutions

This is what you should add to your R script:

```
# Inspecting Data
names(characters)
str(characters)
head(characters)
summary(characters)
```

Presenter Notes: Running these four lines provides a complete overview of the dataset without having to open it like a spreadsheet.

Pre-break quiz

Question 1

What is the primary purpose of the Working Directory in RStudio?

a. It is the only folder where you are allowed to install R.
b. It is the default folder where R looks for files to read and saves outputs.
c. It is a specific R package used for data manipulation.
d. It is the console pane where R executes commands.

Question 2

Which function is used to load an already installed package into your current R session?

a. `install.packages()`
b. `load.package()`
c. `open()`
d. `library()`

Question 3

Which command correctly loads an .rds file named `characters.rds` into an object named `my_data`?

a. `my_data <- readRDS("characters.rds")`
b. `readRDS <- my_data("characters.rds")`
c. `my_data == readRDS("characters.rds")`
d. `my_data <- read_csv("characters.rds")`

Question 4

Which function provides the internal structure of a dataset, including the number of rows/columns and the data types of each variable?

a. `names()`
b. `head()`
c. `summary()`
d. `str()`

Question 5

What does the `head()` function output when applied to a dataset?

a. The column names of the dataset.
b. The first six rows of the dataset.
c. A statistical summary of the numeric variables.
d. The total number of variables in the dataset.

Presenter Notes: Before we head into the break, let's do a quick check-in to see how things are going so far. The questions are designed to help you reflect on your current understanding of the key concepts we just covered: working directories, packages, loading data, and inspecting data structures.

Question 1: The correct answer for this question is B. Question 2: The correct answer for this question is D. Question 3: The correct answer for this question is A. Question 4: The correct answer for this question is D. Question 5: The answer for this question is B.

Instructor Notes: Use this pre-break survey to gauge learners' understanding of the session's core concepts.

- If more than 80% of learners select the correct answer, briefly confirm the correct response and continue.
- If fewer than 80% answer correctly, give learners 1–2 minutes to discuss their reasoning with a partner before re-running the question. If results improve above 80%, continue; otherwise, explain the correct answer and address common misunderstandings.

- If a large portion of learners select the same distractor answer, pause to discuss why that option may seem plausible and clarify the misconception.

Accessibility Tip: This activity is designed for synchronous teaching settings and may be skipped or adapted if needed.

Break! 15 minutes

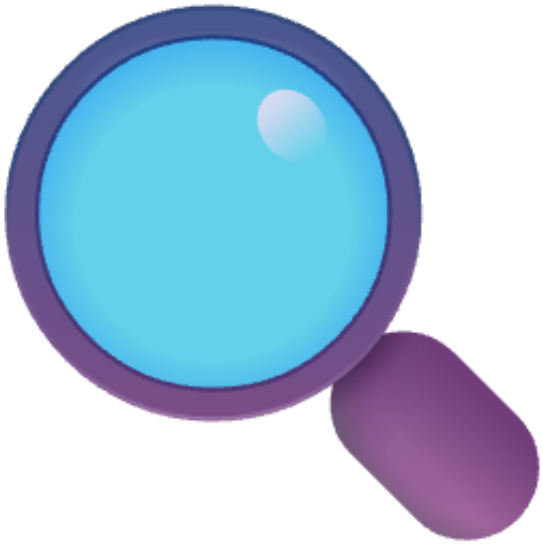
Post-break quiz discussion

What do we see in the results?

Presenter Notes: Let us have a look at these results.

Instructor Notes: Briefly examine the answers given to each question interactively with the group. Use visuals from the survey to highlight specific answers. Serves to clarify concepts and aspects that are not yet understood. Highlight specific answers given during the survey.

Data exploration



Accessing and subsetting columns

As discussed in **Part I**, we can use **indexing tools** (`$` and `[]`) to access specific parts of our data. In real-world analysis, we apply this to **extract entirely new subsets**.

- **Accessing a column:** Just like in Part I, use the `$` operator to isolate a single variable (e.g., `characters$uni_name`).
- **Filtering rows:** Use `dataframe[condition, ]` to select specific rows while keeping all columns.
- **Logical operators:** Build conditions using `==` (checks for exact equality) and `|` (logical OR, meaning either condition can be true).
- **Preserving data:** Always assign your subset to a **new object** (e.g., `relevant_data <- ...`). Overwriting the original object deletes your raw data from the environment.

Presenter Notes: As you might remember from Part I, indexing allows us to point R to specific data. When working with data frames, subsetting is central because we usually want to focus on specific subgroups rather than the whole dataset. We can access a specific column using the dollar sign, just like last time. To filter rows based on characteristics, we use the square brackets. By placing our logical conditions before the comma, we tell R which rows to keep. We build these conditions using double equals for exact matches and the vertical pipe for "OR". Most importantly, we save this subset as a new object so our original dataset remains untouched.

Instructor Notes: Clearly link this back to the indexing exercises from Part I to build confidence. Emphasize how `$` points to the variable inside the condition.

Practical exercise 6

Subsetting by conditions

- ☐ Create a new section in your R script using a comment.
- ☐ Apply your indexing knowledge from Part I to filter the rows of the `characters` data frame to include only observations where the `uni_name` is exactly `"Brooklyn Nine-Nine"` **OR** `"Community"`.
- ☐ Assign this subset to a new object named `relevant_data`.
- ☐ Run the code, then use the `table()` function on `relevant_data$uni_name` to check how many characters from each show are in your new subset.

> Syntax Hint
>
> Remember to specify the data frame for each condition (e.g., `data$column == "A" | data$column == "B"`).

Presenter Notes: Let's apply what we learned in Part I to create our subset. We want to isolate characters from Brooklyn Nine-Nine and Community. Use your indexing knowledge, the dollar sign, and the logical operators we just discussed. After you assign it to the new object `relevant_data`, use the `table()` function on the `uni_name` column of your new subset to verify that it worked.

Instructor Notes: Give learners 5-7 minutes. Monitor for the three common pitfalls: missing the trailing comma inside the brackets, using a single equals sign, or failing to repeat `characters$uni_name` after the OR operator.

Practical exercise 6: Solutions

This is an example of what you could have added to your R script:

```
# Accessing and subsetting columns
relevant_data <- characters[
    characters$uni_name == "Brooklyn Nine-Nine" |
        characters$uni_name == "Community",
]

# Verify the subset
table(relevant_data$uni_name)
```

Presenter Notes: Here is one correct way of solving this exercise. We specify the row conditions before the comma. Notice that we have to tell R exactly where to look for both sides of the OR condition by writing characters$uni_name twice. Finally, running table() on the newly created object confirms that we only have characters from those two shows.

Basic summary functions

After isolating the relevant data, we can **compute descriptive statistics** for specific variables.

- **Numeric variables:** We can apply functions like `mean()` or `sd()` to a single column using the `$` operator (e.g., `mean(data$column)`).
- **Categorical variables:** We can use the `table()` function to count the frequency of specific text or factor values.
- **Combined operations:** We can subset rows and select a specific column simultaneously within a function using `dataframe[condition, "column_name"]`.

Presenter Notes: Now that the subset is established, we can compute basic statistics. We combine summary functions with the dollar sign operator. For numeric data, we apply `mean()` and `sd()`. For categorical data, we use `table()` to count occurrences. Indexing can be nested directly inside functions to calculate statistics for specific subgroups without generating new objects.

Instructor Notes: Ensure learners recognize that executing `mean(data)` directly on a data frame fails. They must isolate a numeric vector using `$`.

Practical exercise 7

Comparing group means

- ☐ Add a new section in your script using a comment.
- ☐ Calculate the overall `mean()` and standard deviation (`sd()`) of the `notability` column in the `relevant_data` object.
- ☐ Calculate the mean notability exclusively for `"Brooklyn Nine-Nine"` characters.
- ☐ Calculate the mean notability exclusively for `"Community"` characters.

Question: Which show features characters with a higher average notability?

> Syntax Hint
>
> To compute the mean for a subgroup, place the row condition before the comma, and the target column name after the comma: `mean(data[condition, "column_name"])`.

Presenter Notes: Let's execute the commands to determine the overall dispersion and central tendency for the notability variable. We will follow this by extracting the mean notability for each show independently. We can utilize bracket subsetting logic directly inside the mean function.

Instructor Notes: Give learners 5 minutes. Monitor for syntax errors when nesting bracket indexing inside the parentheses of the `mean()` function.

Practical exercise 7: Solutions

This is an example of what you could have added to your R script:

```r
# Basic summary functions
mean(relevant_data$notability)
sd(relevant_data$notability)

# Mean notability for Brooklyn Nine-Nine
mean(relevant_data[
  relevant_data$uni_name == "Brooklyn Nine-Nine",
  "notability"
])

# Mean notability for Community
mean(relevant_data[relevant_data$uni_name == "Community", "notability"])
```

Presenter Notes: The dollar sign operator extracts the full vector for overall statistics. To calculate group-specific means, we nest the row condition inside the mean() function and specify "notability" in the column position after the comma.

Simple plotting: Histograms

Beyond calculating numbers, we can **visualize our data** to gain insights quickly. A great starting point for numeric data is a **histogram**.

- **Visualizing distributions:** A histogram groups numeric data into "bins" and shows the frequency (or count) of values falling into each bin.
- **The `hist()` function:** We apply this function directly to a numeric vector to generate a plot in the bottom-right **Plots** pane of RStudio.
- **Customizing plots:** You can add arguments like `main` for the title and `xlab` for the x-axis label to make your plots easier to read.
- **Why it matters:** While means and standard deviations tell us the average, a histogram shows us the actual shape of the data (e.g., are most values clustered in the middle, or skewed to one side?).

Presenter Notes: Numbers are great, but visualizing data often makes it much easier to understand. A histogram is one of the most common plots for numeric variables. It shows us how often different values occur in our dataset. We create it using the `hist()` function. When we run this code, the plot will automatically appear in the output pane in the bottom right corner of RStudio. We can also customize these plots, for example by changing the title or the axis labels, so they look clean and don't overlap. Histograms are incredibly useful because they show us the entire distribution and actual shape of the data, not just a single summary number like the mean.

Instructor Notes: Ensure learners know to look at the bottom-right pane for their output. If their output pane is too small, RStudio might throw a "figure margins too large" error. Tell them to simply drag the pane borders to make it larger if that happens.

Practical exercise 8

Visualizing distributions

- ☐ Add a new section in your script using a comment.
- ☐ Copy and run the hint below to allow two plots to be shown side-by-side.
- ☐ Create a histogram (`hist()`) showing the `notability` of `"Brooklyn Nine-Nine"` characters.
- ☐ Create a second histogram showing the `notability` of `"Community"` characters.

Question: By looking at the plots, how do the distributions of the two shows compare?

> Side-by-side plots hint
>
> Run `par(mfrow = c(1, 2))` *before* creating your histograms. This tells R to split the plotting window into 1 row and 2 columns so we can easily compare both graphs next to each other!

Presenter Notes: Let's visualize the data we just summarized. We will create two separate histograms—one for each show. Before you plot the data, make sure to run the `par` code provided in the hint to show them side-by-side. Then, nest your subsetting code inside the `hist()` function.

Instructor Notes: Give learners 5-7 minutes. Reassure them that they can reuse the exact subsetting syntax (`relevant_data[condition, "column_name"]`) they already wrote for the `mean()` exercise.

Practical exercise 8: Solutions

This is an example of what you could have added to your R script:

```
# Simple Plotting / Histograms

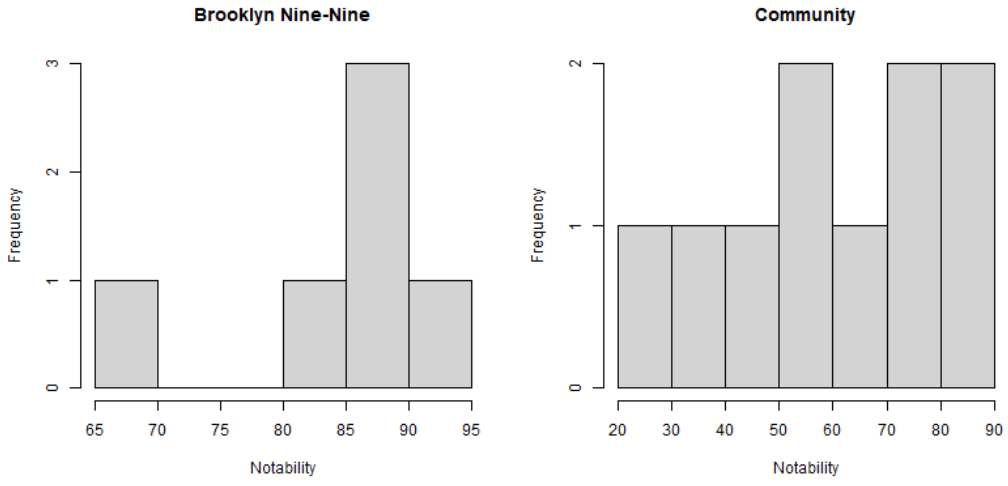
# Set up the plotting area to show 2 plots side-by-side
par(mfrow = c(1, 2))
```

```r
# Histogram for Brooklyn Nine-Nine
hist(
    relevant_data[relevant_data$uni_name == "Brooklyn Nine-Nine", "notability"],
    main = "Brooklyn Nine-Nine",
    xlab = "Notability"
)

# Histogram for Community
hist(
    relevant_data[relevant_data$uni_name == "Community", "notability"],
    main = "Community",
    xlab = "Notability"
)
```

Presenter Notes: Here is the solution. By reusing our subsetting logic, we can directly plot the notability for each specific show. Notice that we snuck in two extra arguments here: `main` and `xlab`. If you ran your code without them, you probably saw a very long, messy default title generated by R. By adding these arguments, we get clean, readable plots without overlapping text.

Practical exercise 8: R output



Presenter Notes: This is what your output should look like in the output pane. Looking at the plots, you can visually confirm what the means calculated earlier suggested: the distributions are different, and we can now actually see where the majority of characters fall on the notability scale for each universe. Here you can clearly see the side-by-side comparison of both distributions.

Best practices and context



Common mistakes and troubleshooting

Errors are a **normal part of programming**. When R throws an error, check for these common culprits first:

- **Wrong capitalization:** R is strictly case-sensitive. `Characters` is not the same as `characters`.
- **Forgetting quotation marks:** Text values and file names must be wrapped in quotes (e.g., `"Community"`, not `Community`).
- **Other typos:** A missing comma, an unclosed parenthesis, or spelling `notablity` instead of `notability` will cause failures.

- **Assignment issues:** Ensure the assignment arrow (`<-`) points to the object name on the left, and check if you accidentally overwrote your data.

Presenter Notes: Programming involves making mistakes constantly. When R gives you a red error message, do not panic. Most of the time, it is a simple typo. R is strictly case-sensitive, meaning a capital C is completely different from a lowercase c. Another frequent issue is forgetting quotation marks around text. If you type Community without quotes, R thinks you are looking for an object named Community, not the text string. Always double-check your spelling, missing commas, and ensure your assignment arrows are pointing the right way.

Instructor Notes: Normalize errors. Remind learners that the console output usually gives a hint about what went wrong (e.g., "object not found" usually means a typo or missing quotes).

Practical exercise 9: Spot the errors

What is wrong with the following lines of code?

```
# 1. Loading data
my_data <- readRDS(characters.rds)

# 2. Calculating a mean
Mean(my_data$notability)

# 3. Subsetting
brooklyn_data <- my_data[my_data$uni_name == Brooklyn Nine-Nine]
```

Write your correctd version into your R script.

Presenter Notes: Let's look at this code. There are three lines, and each contains typical beginner mistakes. Take a moment to read through them and see if you can spot what would cause R to throw an error.

Instructor Notes: Use this only if time permits. It is a highly effective way to test syntax recognition without requiring participants to write new code.

Practical exercise 9: Solutions

Some possible corrections of the previous code:

```
# 1. Loading data
my_data <- readRDS("characters.rds")

# 2. Calculating a mean
mean(my_data$notability)

# 3. Subsetting
brooklyn_data <- my_data[my_data$uni_name == "Brooklyn Nine-Nine", ]
```

Presenter Notes: Let's go through the solutions. Line 1 was missing quotation marks around the file name. Line 2 used a capital M for the mean function, but R functions are lowercase. Line 3 was missing quotation marks around the show name AND was missing the crucial comma inside the brackets.

The Big Picture: R in data science

Today, you **learned the foundational skills of data handling**.

- **Import:** Reading data from various sources (e.g., `.rds`, `.csv`, Excel). *(We did this!)*
- **Wrangle and transform:** Subsetting, cleaning, and reshaping data to fit your needs. *(We did this!)*
- **Visualize:** Creating plots to understand distributions and patterns. *(We did this!)*

Presenter Notes: Let's take a step back and look at the big picture. What we did today—loading data, subsetting it, and creating histograms—are the exact first steps of any professional data analysis pipeline. R is not just a calculator; it is an entire ecosystem. You start by importing data, then you wrangle and transform it. After that, you visualize it to understand what you are working with. Finally, you model the data statistically and communicate the results. In fact, this entire presentation was created using R!

Instructor Notes: Use this slide to bridge the gap between the simple exercises done today and the real-world utility of R. Highlight that the skills learned today are the actual foundation of professional workflows.

Looking to the future

Import → Tidy / (*Part I*) Transform → Visualize / (*Part II*) Model → Communicate

Building up on your data handling skills in the future …

- **Model:** Running complex statistical analyses and machine learning algorithms.
- **Communicate:** Generating automated reports, interactive dashboards, or even **these exact presentation slides** using Quarto/RMarkdown!

Sneak peak of writing interactive reports and scientific manuscripts in Quarto for more experienced researchers: https://lmu-osc.github.io/introduction-to-Quarto/

Presenter Notes: What we did not cover yet are some more advanced aspects of using R for research. Building up on your data wrangling skills, you can also use R to run complex statistical analyses and machine learning algorithms. -You can also use R to generate automated reports, research manuscripts, interactive dashboards, or even **these exact presentation slides** using Quarto/RMarkdown! This last aspect is something we will cover in a following session.

The power of R: What could be next…?

Mastering the basics opens the door to highly **advanced, publication-ready tools**:

- **Advanced visualizations:** Packages like `ggplot2` https://ggplot2.tidyverse.org/ allow you to create stunning, highly customized graphics
- **Interactivity:** Packages like `shiny` https://shiny.posit.co/ let you build interactive web applications for your data
- **Reproducibility:** By scripting your analysis, anyone (including future you) can re-run your exact steps with zero guesswork

Presenter Notes: As you continue learning, you will discover that R is incredibly powerful. You can move from simple base R histograms to stunning, publication-ready graphics using packages like ggplot2. You can even build interactive websites using packages like shiny. If you're curious about either of these packages, both ggplot2 and shiny are clickable links on this slide that will take you straight to their official websites. Most importantly, by writing scripts instead of clicking through menus, your research becomes 100% reproducible.

Final words (of wisdom)

Learning R is exactly like learning a new language. It **takes time, practice, and patience**.

- **Syntax errors are normal:** Even advanced programmers make typos and get error messages every single day.
- **AI is a helpful tool:** Using AI to write code, navigate complex documentation, or troubleshoot errors is becoming increasingly widespread. However, the foundational understanding you built today is essential to verify and adapt the output.
- **Keep practicing:** Save the script you wrote today and use it as a cheat sheet for your next projects.

Great job today!

Presenter Notes: To wrap up, remember that learning R is like learning a new language. It feels unfamiliar at first, and syntax errors are completely normal. Even professionals use AI constantly to write scripts, bypass lengthy documentation, and troubleshoot errors. However, the foundational understanding you gained today is what allows you to actually evaluate and verify the solutions it provides. Do not get discouraged by red text in the console. Save the script you built today, use it as a reference, and keep practicing. Great job today!

Instructor Notes: End on a high note. Emphasize that the frustration of learning a new syntax is universal and temporary.

Using AI as a coding assistant

AI tools can hinder or help your learning progress - **do not outsource the thinking**! **If you do not know what you did, you cannot evaluate or explain the output, defeating the point of credible research.**

- **Verify everything:** AI can hallucinate or suggest outdated functions. You must still read, understand, evaluate and test the code it generates. For this, you must also **know what questions to ask**!
- **Data privacy is critical:** Never paste sensitive, personal, or unpublished research data into public AI chatbots!

Presenter Notes: As AI tools become deeply integrated into programming workflows, it's important to use them responsibly. They are excellent co-pilots for troubleshooting red text in the console or figuring out how to approach a new task. However, they are not infallible. You must understand the foundations of R to verify if the AI's code actually does what you need. When it comes to tools, you can use standalone chatbots, or integrations directly within your programming environment (also called an IDE, like RStudio). Integrations have the benefit of "seeing" your entire script, so you don't have to copy-paste as much context.

Most importantly, remind learners about data privacy: public AI models often train on user inputs, so sensitive data should never be shared with them!

Instructor Notes: Frame AI as an assistant, not a replacement for learning the basics. The core message is that acquiring foundational coding skills is the only way for transparent and credible research, because one must understand the question they are asking and be able to evaluate the output they are receiving.

Take-home messages

- Working directories and packages → **essential prerequisites** for accessing files and expanding R's capabilities.
- Subsetting data → allows you to **filter and isolate** the exact observations you need for your analysis.
- Summary functions and visualizations → provide a **quick and comprehensive understanding** of your data structures and distributions.

Presenter Notes: To conclude this part, there are three key ideas to remember. First, correctly setting your Working Directory and loading necessary packages are the non-negotiable first steps of any R session. Without them, you cannot access your data or advanced tools.

Second, subsetting is central to data analysis. You rarely work with a full raw dataset at once; instead, you filter specific rows based on logical conditions to answer targeted questions. And third, basic statistics and visualizations go hand in hand. Using summary functions alongside plots like histograms gives you a complete picture of your data's shape and distribution.

Instructor Notes: Keep this slide concise and use it as a clear closure of the R part 2 section.

Learning objectives: Check-in

At the end of this session, you should now be able to…

- ☐ **Identify** and set the correct Working Directory on your computer
- ☐ **Install** and **load** R packages to extend R's functionality
- ☐ **Import** external data into the R environment
- ☐ **Inspect** data structures and **extract** specific columns
- ☐ **Compute** basic statistical summaries on a dataset
- ☐ **Visualize** data distributions using simple histograms

Presenter Notes: Let us pause for a quick check-in. Take a moment to look back at the learning objectives and reflect on how comfortable you feel with each one. If you notice any gaps, this is the right moment to bring them up. Feel free to ask questions or start a discussion so we can address them before moving on.

Instructor Notes: Use this slide to encourage self-reflection rather than evaluation. Emphasize that learners are not expected to feel equally confident about all objectives at this stage. Keep the discussion supportive and brief, focusing on clarifying concepts rather than introducing new material.

Looking into the future

1. **Take a minute to think about the following questions, first for yourself, then in a pair:**

- Can you identify some **practical benefits** of using R/RStudio?
- What **obstacles** do you anticipate in using it consistently?
- How can R/RStudio support you in your **upcoming projects**?

2. **Share your thoughts with the group.**

Presenter Notes: To wrap up, let's take a moment to bring everything together and think about how this connects to your own work going forward. First, take about one minute to reflect individually on these questions. Then discuss them with a partner. Think about where R/RStudio could realistically support you, for example, in upcoming essays, group projects, or your thesis work. What benefits do you see in using it? What might make it difficult to use it consistently? And how could it fit into your current workflow?
After that, we'll briefly share some of your thoughts with the group.

Instructor Notes: Allocate about 5-10 minutes total for this think-pair-share activity (1-2 minutes for individual reflection, 3-4 minutes in pairs, and the remainder for group discussion). Facilitate an open discussion focusing on application rather than theory. Encourage concrete examples from learners' own academic context. Actively prompt for both benefits and barriers. Keep the discussion time-bound to avoid overextension at the end of the session.

Accessibility Tip: Allow silent reflection time before discussion. Summarize key contributions aloud for clarity.

Assignment: Consolidating what you learned

Create an R script that applies the data handling skills covered today

- Set your Working Directory and load the `characters.rds` dataset.
- Create a new subset of the data using a logical condition of your choice (e.g., a different show).
- Calculate a summary statistic (e.g., `mean()` or `table()`) on a variable within your subset.
- Create a histogram (`hist()`) to visualize a numeric variable and customize its title.
- Include clear comments (`#`) explaining each step.

> Self-Assessment
>
> Use this assignment to assess your learning of, and progress with, R/RStudio; reflect on any challenges you faced and how you overcame them.

Presenter Notes: To finish, you have a short home assignment that brings together everything we covered in Part II. You will create a script where you first set your working directory and load the characters dataset. Then, you will practice subsetting by filtering the data based on a condition of your choosing—perhaps explore characters from a different show. Next, compute a basic summary statistic like a mean or a frequency table on your new subset. Finally, visualize a numeric variable using a histogram and give it a custom title. Make sure to comment your code so you know exactly what each line does when you look at it later.

Instructor Notes: Clarify that this is a practice exercise rather than a graded assessment. Encourage learners to explore variables in the dataset they haven't looked at yet. Reassure them that running into errors during subsetting is normal and part of the learning process.

To conclude: Survey time!

Question 1

How confident do you feel setting a Working Directory and loading external data into R?

a. Not confident at all

b. Slightly confident

c. Moderately confident

d. Very confident

e. Extremely confident

Question 2

How comfortable do you feel using logical conditions (`==`, `|`) to subset data frames?

a. Not comfortable at all

b. Slightly comfortable

c. Moderately comfortable

d. Very comfortable

e. Extremely comfortable

Question 3

Compared to the beginning of the workshop, how intimidated do you feel about working with R?

a. Much more intimidated

b. Slightly more intimidated

c. About the same

d. Slightly less intimidated

e. Much less intimidated

Question 4

What part of today's lesson still feels confusing, and which exercise helped your understanding the most?

Presenter Notes: To conclude, we'll take a short survey. This post-session survey helps us understand your current level of knowledge after completing the session. Please answer honestly based on how confident you feel now.

Instructor Notes: The purpose is to compare pre- and post-session responses to evaluate learning progress. Encourage the learnrs to share in class. Use this space to answer doubts, and provide learning or practice tips if the learners still feel under-confident in any areas.

Discussion of survey results

What do we see in the results?

Presenter Notes: Let us have a look at these results.

Additional resources

- Datacamp offers a free, interactive course on R online (https://www.datacamp.com/courses/free-introduction-to-r).
- The LMU Open Science Center provides a comprehensive, self-paced R tutorial (https://lmu-osc.github.io/introduction-to-R/).

Presenter Notes: If you'd like to continue practicing R after this session, there are optional additional resources available.

Datacamp offers a free, interactive introduction to R, where you can practice directly in your browser. We also highly recommend the LMU OSC's own Introduction to R tutorial, which is a great comprehensive guide. These are useful ways to reinforce what we covered today and get more comfortable with writing code at your own pace.

You are not required to complete them, but they can be helpful if you want extra practice or a more structured introduction to R outside of class.

Instructor Notes: Present these as optional extensions, not part of the core learning requirements. Avoid suggesting them as mandatory. Emphasise self-paced learning and reinforcement of basic R skills using both the interactive Datacamp course and the in-depth OSC material.

Accessibility Tips:
Ensure the links are accessible through the course platform or slide deck.

Contribution statement

Creator: Rieger, David (0009-0004-3701-3101)

Creator: Waterfield, Elizabeth (0009-0006-3725-6730)

Consultant: Sanchez Steinhagen, Caterina Luz (0009-0006-4783-1110)

Reviewer: Middleton, Sara (0000-0001-5307-8029)

Reviewer: von Grebmer zu Wolfsthurn, Sarah (0000-0002-6413-3895)

Licence and citation statement

This work was originally created by Nicklas Hafiz. His work was subsequently adapted by David Rieger, Elizabeth Waterfield, Caterina Luz Sanchez Steinhagen, Sara Lil Middleton and Sarah von Grebmer zu Wolfsthurn. The current work is licensed under a Creative Commons Attribution 4.0 SA International License (CC-BY-SA 4.0). Code snippets are dedicated to the public domain and are licensed under a Creative Commons Universal Licence (CC0 1.0).

Please cite these materials as: Rieger, D., Waterfield, E., Sanchez-Steinhagen, C. L., Middleton, S., & Von Grebmer Zu Wolfsthurn, S. (2026). *Introduction to R: Part II* [Quarto slides]. LMU Open Science Center. https://lmu-osc.github.io/teaching-material-for-novices/open-research-tools/r-part-II

Presenter Notes: The Creative Commons Attribution 4.0 International SA License, or CC BY-SA 4.0, allows others to copy, share, and adapt a work in any medium, including for commercial purposes. These permissions are broad and cannot be withdrawn as long as the license terms are followed. The main requirement is attribution: users must give appropriate credit to the original creator, provide a link to the license, and clearly indicate whether any changes were made, without implying endorsement by the original author. In addition, the ShareAlike condition means that if someone modifies or builds upon the work, the resulting material must be distributed under the same CC BY-SA 4.0 license, or a compatible one. Finally, users are not allowed to apply legal or technical restrictions, such as DRM, that would prevent others from exercising these same rights.

Code snippets are dedicated to the public domain and licensed under a Creative Commons Universal License (CC0 1.0). You may use, modify, distribute, and sell the code snippets for any purpose, without permission or attribution. The code snippets are provided “as is”, without warranty of any kind.