

# **Getting Started with R for Economists**

2025-01-06

# Table of contents

<b>Preface</b>	<b>4</b>
About This Book . . . . .	4
Who Can Benefit from This Book . . . . .	4
Share Your Thoughts! . . . . .	5
<b>1 Introduction to R and RStudio</b>	<b>6</b>
1.1 Installing R and Rstudio . . . . .	6
1.1.1 Posit Cloud . . . . .	6
1.2 RStudio layout . . . . .	6
1.2.1 Console Pane . . . . .	7
1.2.2 Source pane . . . . .	7
1.2.3 Environment pane . . . . .	8
1.2.4 Output Pane . . . . .	8
1.3 Installing an R Package . . . . .	8
1.3.1 Alternative way to install R packages in Rstudio . . . . .	9
1.4 Loading an R Package . . . . .	9
1.5 Getting Started with R . . . . .	10
<b>2 R Programming Basics</b>	<b>11</b>
2.1 Data structures . . . . .	11
2.1.1 Vectors . . . . .	12
2.2 Data Frames . . . . .	14
2.3 Functions in R . . . . .	16
2.4 Subsetting . . . . .	16
2.5 Help file for Built-in functions . . . . .	17
2.6 Commenting . . . . .	18
2.7 Pipe operator ( <code> &gt;</code> ) . . . . .	19
<b>3 Scripts and Rstudio Projects</b>	<b>21</b>
3.1 Working with R Scripts . . . . .	21
3.2 Working with RStudio Projects . . . . .	22
<b>4 Reproducible Reporting with Quarto</b>	<b>24</b>
4.1 Dynamic documents . . . . .	24
4.2 Dynamic Documents with Quarto . . . . .	25
4.2.1 Installing Quarto . . . . .	25

4.2.2	Why Quarto ? . . . . .	25
4.3	Getting Started with Quarto . . . . .	25
4.4	Anatomy of a Quarto document . . . . .	27
4.4.1	YAML header . . . . .	28
4.4.2	Code Chunks . . . . .	28
4.4.3	Markdown text . . . . .	29
<b>5</b>	<b>Data Import and Export</b>	<b>30</b>
5.1	Tidy Workflow . . . . .	30
5.2	Import data . . . . .	31
5.2.1	Read data files into a tibble . . . . .	31
5.3	Export data . . . . .	31
<b>6</b>	<b>Data Wrangling</b>	<b>33</b>
6.1	Data Tidying . . . . .	33
6.2	Data Transformation . . . . .	37
6.2.1	Example . . . . .	39
<b>7</b>	<b>Data Visualization</b>	<b>52</b>
7.1	The ggplot2 API . . . . .	52
7.2	Data Layer . . . . .	54
7.3	Mapping Layer . . . . .	57
7.4	Geometries Layer . . . . .	58
<b>8</b>	<b>Introduction to Statistical Modelling</b>	<b>60</b>
	<b>References</b>	<b>61</b>

# Preface

**Author:** [Priyanga Dilini Talagala](#)

R is a free and powerful software environment for statistical computing and data visualization. It is widely used in academia and industry for data analysis and research. As one of the top programming languages for data science, R provides a variety of tools for statistical modeling, computing, and visualization.

Since empirical research is essential in economics, programming skills are crucial for conducting real-world data analysis. This textbook will introduce you to R and help you develop fundamental data science skills.

## About This Book

This textbook is designed for beginners, providing a strong foundation in R for economic research. It focuses on **the tidyverse** ecosystem, a collection of R packages that provide a simple yet powerful approach to data analysis. You will learn how to use tidy tools to manage and analyze data efficiently, covering the entire lifecycle of a data science project.

## Who Can Benefit from This Book

I wrote this textbook for two groups of readers. The first group is the participants of the workshop Getting Started with R for Economists: A Hands-On Workshop, which was organized for the 52nd Meeting of the Sri Lanka Forum of University Economists. This book serves as a companion to that workshop. Since the workshop lasts only one hour, the goal is to introduce participants to the tidyverse workflow and give them a strong foundation so they can continue learning on their own.

The second group is any economist who is new to programming and wants to start analyzing data using R. This book is a good starting point for your journey. However, to master each topic fully, I recommend reading more advanced textbooks on R, covering the different areas discussed here.

## Share Your Thoughts!

This book is still a work in progress, and some chapters (marked WIP) are still in the oven—they'll be ready soon! If you have any suggestions to improve the content, feel free to open a GitHub issue [here](#). I'd love to hear from you!

# 1 Introduction to R and RStudio

## 1.1 Installing R and Rstudio

- **Step 1:** First download R freely from the Comprehensive R Archive Network (CRAN) <https://cran.r-project.org/>. (At the moment of writing, R 4.4.2 is the latest version. Choose the most recent one.)
- **Step 2:** Then install R Studio's IDE (stands for integrated development environment), a powerful user interface for R from [<https://posit.co/download/rstudio-desktop/>](<https://posit.co/download/rstudio-desktop/>). Get the Open Source Edition of RStudio Desktop. RStudio allows you to run R in a more user-friendly environment.
  - You need to install **both** R and Rstudio to use RStudio.
  - If you have a pre-existing installation of R and/or RStudio, I highly recommend that you reinstall both and get as current as possible.
- **Step 3:** Then open **Rstudio**.

### 1.1.1 Posit Cloud

- In 2022, RStudio changed its corporate name to Posit with the aim of expanding its focus beyond R to include users of Python and Visual Studio Code.
- If you don't want to download or install R and R Studio, you can use RStudio on [Posit Cloud](https://posit.cloud/) (<https://posit.cloud/>) for free.

## 1.2 RStudio layout

The RStudio interface consists of four panes: See Figure 1)

1. **Source pane**
2. **Console pane**
3. **Environment pane**, containing the Environment, History, Connections, Build, and Tutorial tabs

4. **Output pane**, containing the Files, Plots, Packages, Help, Viewer, and Presentation tabs

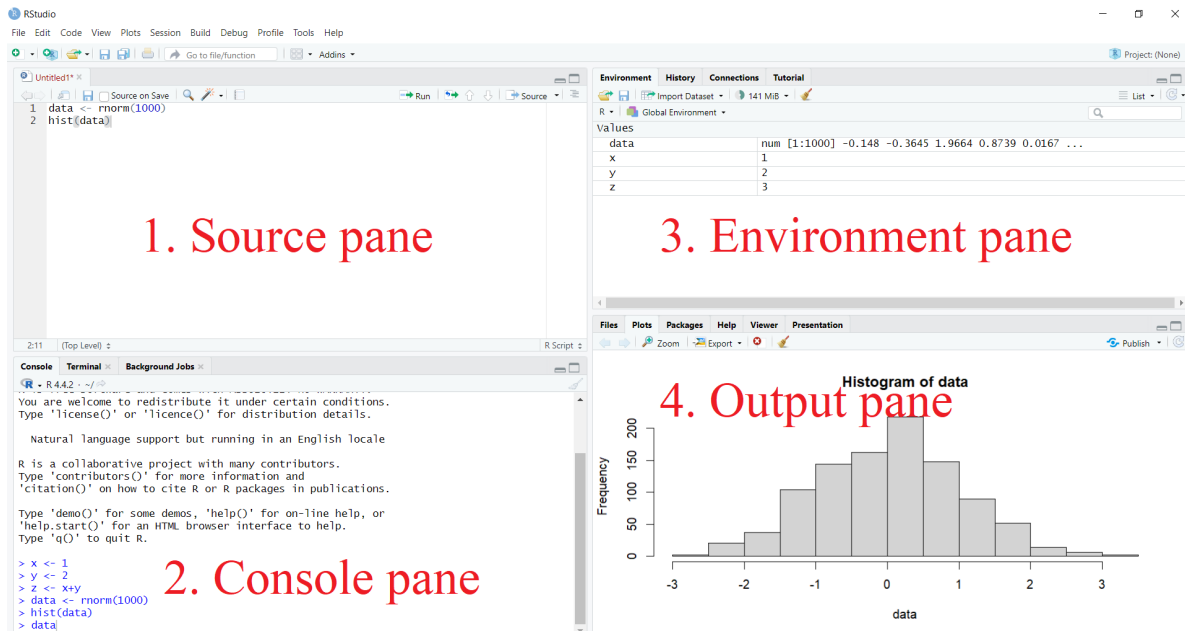


Figure 1.1: RStudio layout

### 1.2.1 Console Pane

- This is where you type and execute all your R commands.
- You can enter R commands after the ‘>’ prompt, and R will process and execute them.
- This is the most essential window, as it is where R performs computations and executes your instructions.

### 1.2.2 Source pane

- In this window, a collection of commands (scripts) can be edited and saved.
- If this window is not visible, you can open it via File → New File → R Script.
- Simply typing a command in the Source pane is not enough; it must be sent to the Console before R executes it.
- To run a line from the Source pane, place your cursor on the desired line or select multiple lines to execute, then click Run or press CTRL + ENTER to send them to the Console pane.

- Make sure to save the ‘Untitled1’ file as a \*.R script.

### 1.2.3 Environment pane

- This window contains multiple tabs: Environment, History, Connections, Build, and Tutorial.

The Environment tab displays all active objects.

- For data frames, clicking the grid symbol opens the full data frame in the Source pane.
- The History tab shows previously typed commands.
- To send a command in the history tab to the Source pane, select it and click the “To Source” icon, or click “To Console” to execute it in the Console.

### 1.2.4 Output Pane

- This window contains multiple tabs: Files, Plots, Packages, Help, Viewer, and Presentation.
- It allows you to open files, view plots (including previous ones), install and load packages, access help functions, and display web content such as Shiny apps and Quarto-generated web pages.

Now you are familiar with the layout. Let’s begin with R basics.

## 1.3 Installing an R Package

- The primary source for R packages is CRAN (Comprehensive R Archive Network).
- Packages can be installed using the `install.packages()` function in R.
- To install a single package, pass its name as the first argument to `install.packages()`.
- The following code installs the tidyverse package from CRAN:

```
install.packages("tidyverse")
```

- This command downloads and installs the `tidyverse` package from CRAN.
- Any dependencies required by the package will also be downloaded and installed.



- Installing the tidyverse package may take several minutes, but you only need to do this once. Think of it like installing a mobile app—you install it once on your smartphone and can use its features until a new version is released, at which point you may need to update it

### 1.3.1 Alternative way to install R packages in Rstudio

- An alternative way to install R packages is through the Packages tab in the Output Pane.
- Navigate to the Packages tab in the Output Pane and click Install.
- Under “Install from,” select “Repository (CRAN)”.
- In the Packages field, enter the name of the package you want to install.
- To install multiple packages at once, separate the package names with commas.
- Finally, click Install.

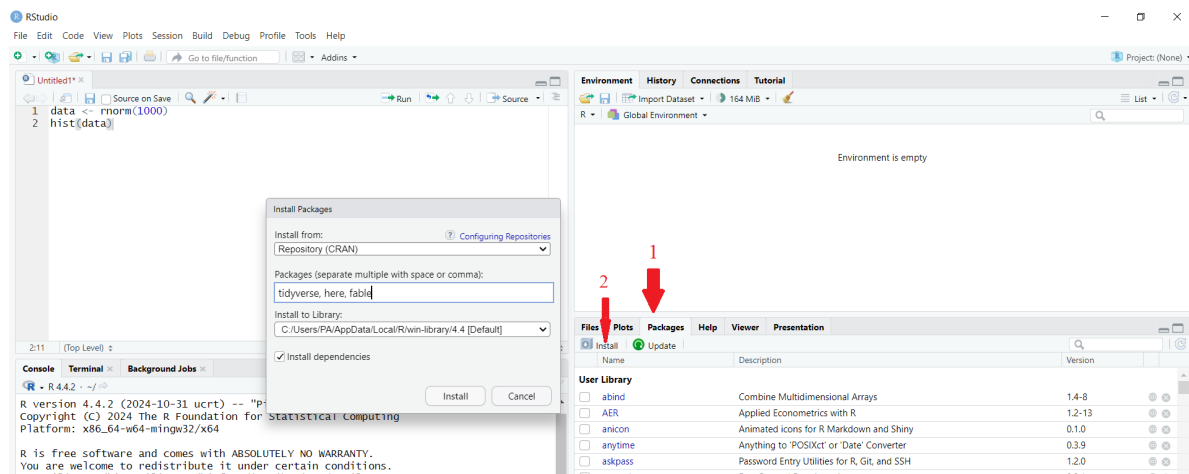


Figure 1.2: Alternative way to install R packages in Rstudio

## 1.4 Loading an R Package

- Installing a package does not automatically make it available for use; you must load it. It's like a mobile app—you need to open it to access its functionalities.
- The `library()` function is used to load installed packages into R.
- To load the tidyverse package, use:

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --  
v dplyr      1.1.4      v readr      2.1.5  
v forcats   1.0.0      v stringr   1.5.1  
v ggplot2    3.5.1      v tibble    3.2.1  
v lubridate  1.9.3      v tidyr     1.3.1  
v purrr      1.0.2
```

```
-- Conflicts ----- tidyverse_conflicts() --
```

```
x dplyr::filter() masks stats::filter()
```

```
x dplyr::lag()     masks stats::lag()
```

```
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

- **Note:** Do not put the package name in quotes when using `library()`.
- Some packages display messages when loaded, while others do not.

## 1.5 Getting Started with R

For a detailed introduction to R, refer to:

An Introduction to R: <https://cran.r-project.org/doc/manuals/R-intro.pdf>

## 2 R Programming Basics

- R has two main parts: data structures and functions.
  - Data structures help store and organize data so it can be used efficiently.

```
income <- c(1000, 4000, 3400, 9700, 9800)
income
```

```
[1] 1000 4000 3400 9700 9800
```

- Functions tell R what to do.

```
mean(income)
```

```
[1] 5580
```

```
summary(income)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1000	3400	4000	5580	9700	9800

### 2.1 Data structures

- R's basic data structures can be grouped by:
  - Their dimensions (1D, 2D, or multi-dimensional).
  - Whether they store one type of data (homogeneous) or different types of data (heterogeneous).
- This gives us five common data structures in R:
  - Homogeneous (same type of data): Vectors, Matrices, Arrays
  - Heterogeneous (different types of data): Data Frames, Lists

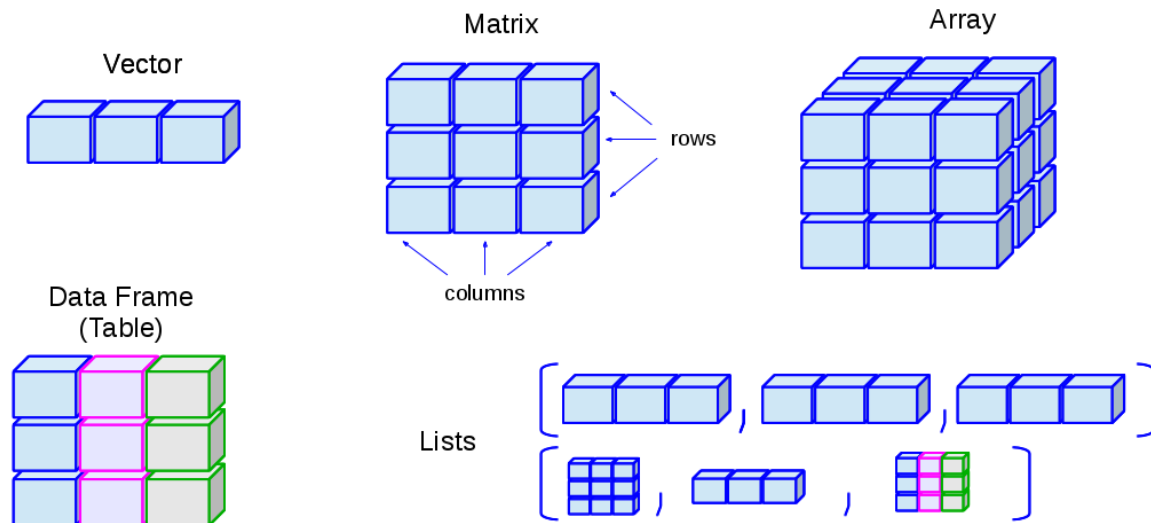


Figure 2.1: Data Structure Types. Image Source: <http://venus.ifca.unican.es/Rintro/dataStruct.html>

- Among these, vectors and data frames are the most commonly used data structures in practical applications.

### 2.1.1 Vectors

#### Creating vectors

Syntax

```
vector_name <- c(element1, element2, element3)
```

Example

```
x <- c(5, 6, 3, 1, 100)
x
```

```
[1] 5 6 3 1 100
```

#### Combine two vectors

```
p <- c(1, 2, 3)
p
```

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

```
q <- c(10, 20, 30)
q
```

```
[1] 10 20 30
```

```
r <- c(p, q)
r
```

```
[1] 1 2 3 10 20 30
```

### Vector with character elements

```
countries <- c("Sri Lanka", "Afghanistan", "Bangladesh", "Bhutan", "India", "Iran", "Maldives")
countries
```

```
[1] "Sri Lanka" "Afghanistan" "Bangladesh" "Bhutan" "India"
[6] "Iran" "Maldives" "Nepal" "Pakistan"
```

### Logical vector

```
result <- c(TRUE, FALSE, FALSE, TRUE, FALSE)
result
```

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

### Simplifying vector creation

- `rep` is a function in R that repeats the values in a vector

```
id <- 1:10
id
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
treatment <- rep(1:3, each=2)
treatment
```

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

## Vector operations

```
x <- c(1, 2, 3)
y <- c(10, 20, 30)
x+y
```

```
[1] 11 22 33
```

```
p <- c(100, 1000)
x+p
```

Warning in x + p: longer object length is not a multiple of shorter object length

```
[1] 101 1002 103
```

## 2.2 Data Frames

### Required R package to deal with data frames

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.4      v readr      2.1.5
v forcats    1.0.0      v stringr    1.5.1
v ggplot2    3.5.1      v tibble     3.2.1
v lubridate  1.9.3      v tidyr      1.3.1
v purrr      1.0.2
```

```
-- Conflicts ----- tidyverse_conflicts() --
```

```
x dplyr::filter() masks stats::filter()
```

```
x dplyr::lag()     masks stats::lag()
```

```
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

## Create a tibble

- A tibble is an improved version of a traditional data frame in R, designed to be more user-friendly and consistent.
- It is part of the **tidyverse** package and makes it easier to view and analyze data, especially when working with large datasets.

Lets consider the following example

### GDP per capita (current US\$) - South Asia

Data Source : <https://data.worldbank.org/>

Country	Most Recent Year	Most Recent Value
Afghanistan	2023	415.7
Bangladesh	2023	2,551.0
Bhutan	2023	NA
India	2023	2,480.8
Maldives	2023	12,530.4
Nepal	2023	1,377.6
Pakistan	2023	1,365.3
Sri Lanka	2023	3,828.0

```
country <- c("Afghanistan", "Bangladesh", "Bhutan", "India", "Maldives",  
            "Nepal", "Pakistan", "Sri Lanka")  
year <- c(rep(2023,8))  
value <- c(415.7, 2551.0, NA, 2480.8, 12530.4, 1377.6, 1365.3, 3828.0)  
  
final <- tibble(Country = country, Recent_Year = year, Value = value )  
final
```

```
# A tibble: 8 x 3  
  Country      Recent_Year  Value  
  <chr>          <dbl>   <dbl>  
1 Afghanistan    2023    416.  
2 Bangladesh     2023   2551  
3 Bhutan         2023     NA  
4 India          2023   2481.  
5 Maldives       2023  12530.  
6 Nepal          2023   1378.  
7 Pakistan       2023   1365.  
8 Sri Lanka      2023   3828
```

## 2.3 Functions in R

- In R, functions are blocks of code that perform specific tasks.
- They take input values (called arguments), process them, and return an output.
- Functions help automate repetitive tasks and make code more efficient.
- There are two main types of functions in R:
  - Built-in functions – Predefined in R (e.g., `mean()`, `sum()`, `rep()`).

```
summary(final$Value)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
415.7	1371.5	2480.8	3507.0	3189.5	12530.4	1

- User-defined functions – Created by users for specific needs using `function()`.

## 2.4 Subsetting

```
final
```

```
# A tibble: 8 x 3
  Country      Recent_Year Value
  <chr>          <dbl>   <dbl>
1 Afghanistan    2023    416.
2 Bangladesh     2023   2551
3 Bhutan         2023     NA
4 India          2023   2481.
5 Maldives       2023  12530.
6 Nepal          2023   1378.
7 Pakistan       2023   1365.
8 Sri Lanka      2023   3828
```

```
final[1, 1]
```

```
# A tibble: 1 x 1
  Country
  <chr>
1 Afghanistan
```



```
final[, 1]
```

```
# A tibble: 8 x 1
  Country
  <chr>
1 Afghanistan
2 Bangladesh
3 Bhutan
4 India
5 Maldives
6 Nepal
7 Pakistan
8 Sri Lanka
```

```
final[1, ]
```

```
# A tibble: 1 x 3
  Country      Recent_Year Value
  <chr>          <dbl> <dbl>
1 Afghanistan    2023  416.
```

```
final$Country
```

```
[1] "Afghanistan" "Bangladesh" "Bhutan"      "India"      "Maldives"
[6] "Nepal"       "Pakistan"    "Sri Lanka"
```

## 2.5 Help file for Built-in functions

- To access the help file for a built-in function in R, you can use either `?` or the `help()` function.
- Running either of these commands will open the help page for the specified function.

= For example, using `?summary` or `help(summary)` will display the help file for the `summary` function, which is part of the base R package.

```
?summary
```

```
# or
```

```
help(summary)
```

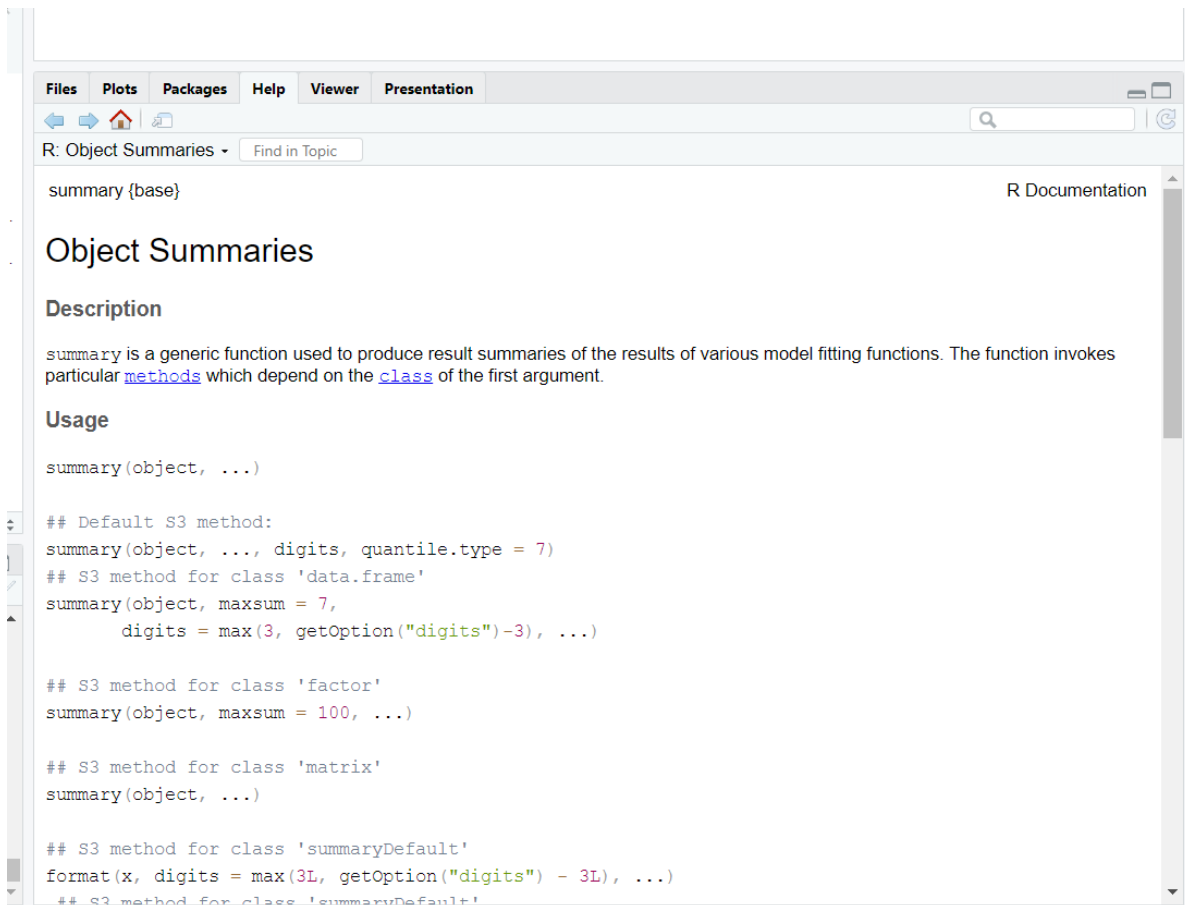


Figure 2.2: Help Page

## 2.6 Commenting

```
mean(final$Value) # Calculate the average GDP value
```

```
[1] NA
```

When you go to the help page for the `mean()` function, you'll find that by default, the `na.rm` argument is set to `FALSE`. This means that if the data contains missing values (NA), they will be included in the calculation, and the result will also be NA. To calculate the mean while ignoring missing values, you should set `na.rm = TRUE`.

```
mean(final$Value, na.rm = TRUE) # Calculate the average GDP value
```

```
[1] 3506.971
```

## 2.7 Pipe operator (|>)

- The pipe operator (|>) in the base R package helps improve the readability of your code.
- It takes the output of one function and passes it directly into another function as an argument, making the steps in your data analysis more connected.
- Instead of using nested function calls like `function(first_input, other_inputs)`, you can write the same command in a simpler format: `first_input |> function(other_inputs)`.

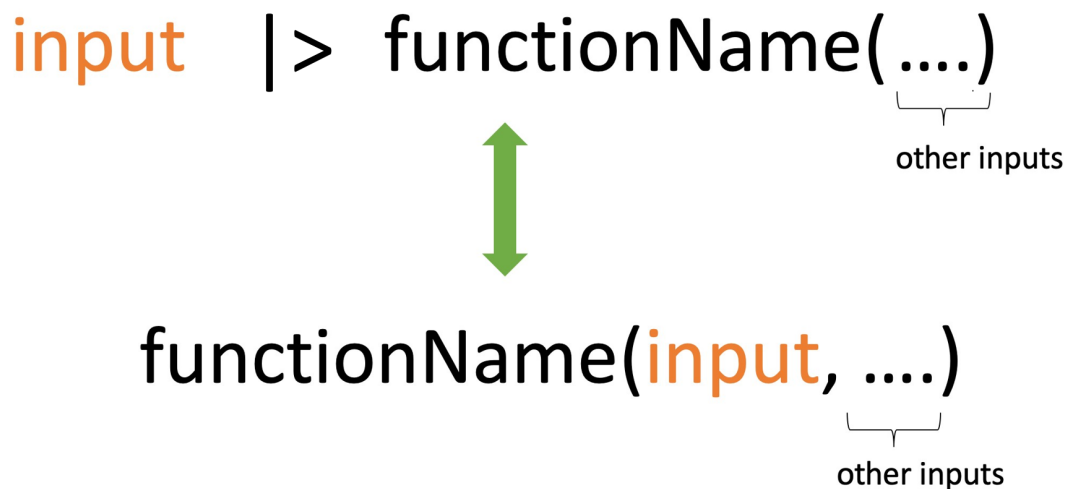


Figure 2.3: Pipe Operator

- Therefore, using the pipe operator allows you to chain multiple operations together in a way that is easier to read and understand compared to nested function calls.

```
# Nested function call  
mean(final$Value, na.rm = TRUE)
```

```
[1] 3506.971
```

```
# Using pipe operator)
final$Value |> mean(na.rm=TRUE)
```

```
[1] 3506.971
```

## 3 Scripts and Rstudio Projects

So far, we've been using the console to run code. One problem with the R console is that once you close the project, you won't be able to access the previous code you ran unless you save the workspace. This is where the script editor comes in.

### 3.1 Working with R Scripts

- To open a script file, go to File -> New File -> R Script.
- You can use a script file to save your code so you don't have to re-type everything when you start a new R session.
- Just typing a command in the script file isn't enough. It must be sent to the Console before R can execute it.
- To run a line of code from the script, place your cursor on the line (or select multiple lines), then click Run or press CTRL + ENTER to send them to the Console.
- Remember to save your script with the \*.R extension and give it a suitable name, not as 'Untitled1'.

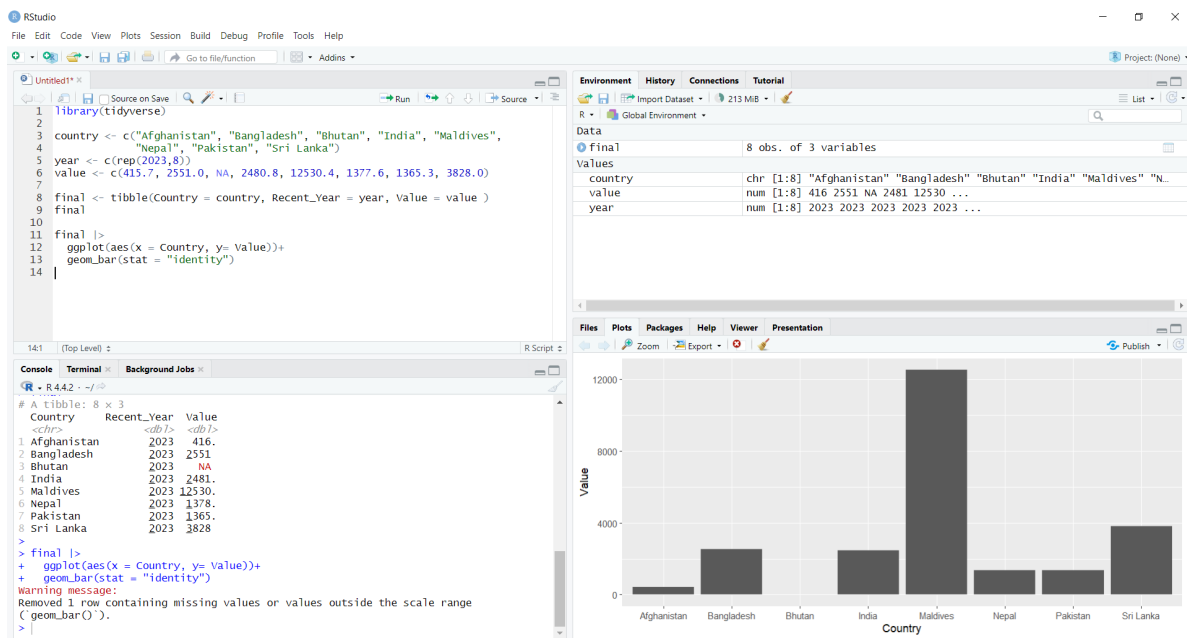


Figure 3.1: Working with R script files

## 3.2 Working with RStudio Projects

When you start working on large projects, you'll need to manage multiple files, such as input data, script files, and figures. Being organized is very important to manage your work efficiently. This is where RStudio projects help. RStudio projects allow you to keep all the documents related to a project in one folder.

To create an RStudio project, go to **File -> New Project -> New Directory -> New Project**, and then choose a suitable name for the folder to store your work. You also need to pick a location to save your project by setting the "Create project as subdirectory of" option. In the following example, I named my project "Test" and selected the Desktop to store it.

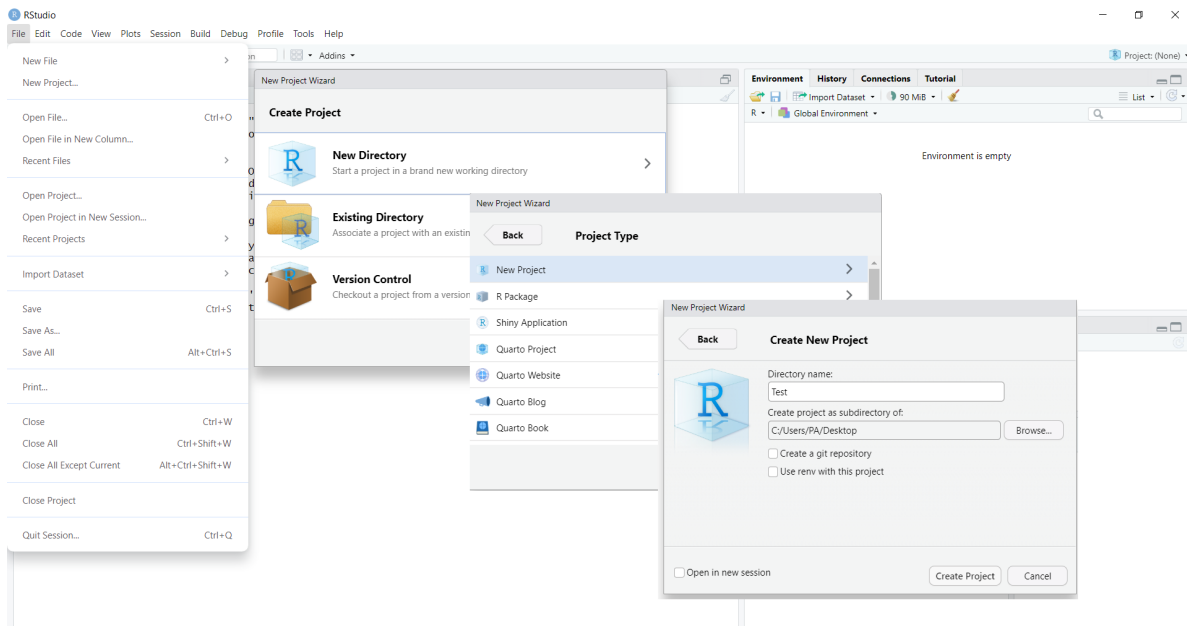


Figure 3.2: Working with RStudio Projects

Once this is done, I now have a new RStudio project called “Test” on my Desktop. This project looks like a regular folder where I can save all my work. If you open the folder, you will see a `.Rproj` file (R Project file). In my example, since I saved my project as “Test,” the R project file is called “Test.Rproj.” Inside this folder, you can create subfolders, like one for figures called **figures**, another for raw data files called **data**, and another for your R script files called **scripts**.

Let’s say you’re done working on the project for the day. Now, close the project and quit RStudio.

The next day, when you come back to the project, locate the folder (in my case, it’s on the Desktop), open it, and double-click the `.Rproj` file. This will reopen the project, and you’ll pick up right where you left off. You’ll have access to all the files you’ve saved inside the RStudio Project.

The only things you can’t retrieve are the commands you typed directly into the console and any unsaved outputs, like summary statistics or figures that were not saved.

But script files at least allow you to save time by preventing you from retyping commands or complex code you’ve already written in previous sessions. They’re better than typing directly into the console.

## 4 Reproducible Reporting with Quarto

As an economist, you work with complex data and statistics to give businesses useful financial advice. Clear and effective data communication is essential for this role.

So far, we have learned how to perform our analysis by writing commands in a script file. The advantage of this approach is that you can save all your previous code, avoiding the need to retype it from scratch each time. However, one major drawback is that the generated outputs (such as tables and graphs) are not saved automatically—you must manually save each one.

Even if you make an effort to save the output, manually copying and pasting results can sometimes lead to human errors. Additionally, you may only notice errors in the original dataset after generating the outputs. When this happens, you have to rerun the code step by step, regenerate all the outputs, and manually replace the previous results based on incorrect data. This process is time-consuming and increases the risk of mistakes.

This is where reproducible reporting or dynamic documents become useful. Dynamic documents allow you to store text, code, and output (e.g., tables, graphs) all in a single file. Unlike R script files, dynamic documents make it easy to save both the code and its results together, ensuring everything is available for future use.

### 4.1 Dynamic documents

Dynamic documents are important because they:

- They let us add R code directly into the document, and the code runs automatically to include the results.
- They update analyses and visuals automatically when data changes, saving time and reducing mistakes.
- They help recreate results easily, making analyses more transparent and consistent.
- They make it simple to share analysis and results with team members, improving collaboration and communication.



## 4.2 Dynamic Documents with Quarto

Both R Markdown and Quarto allow you to create dynamic documents. Quarto is the next generation of R Markdown, so in this book, I will focus on Quarto. For those already familiar with R Markdown, you'll find the transition to Quarto very easy, as the syntax is very similar.

### 4.2.1 Installing Quarto

A stable release of Quarto is included with RStudio version 2022.07.1 and later. Upgrading to future versions of RStudio will also upgrade the bundled version of Quarto.

### 4.2.2 Why Quarto ?

- Quarto supports multiple programming languages (R, Python, Julia, and more) within the same document, allowing for a more versatile workflow.
- Quarto provides a wider range of output formats, including HTML, PDF, Word, and slides, all from a single document. This flexibility makes it easier to publish content in various formats without needing separate files.
- Quarto is designed with modern development practices in mind, making it easier to integrate with version control systems like Git.
- Quarto offers advanced features for customization, including the ability to create custom formats and templates, which can help in producing highly tailored outputs.
- Quarto is actively developed with a focus on modern data science practices, which means it benefits from ongoing improvements and a growing community.

## 4.3 Getting Started with Quarto

Let's get started with Quarto:

- First, go to `File -> New File -> Quarto Document...`

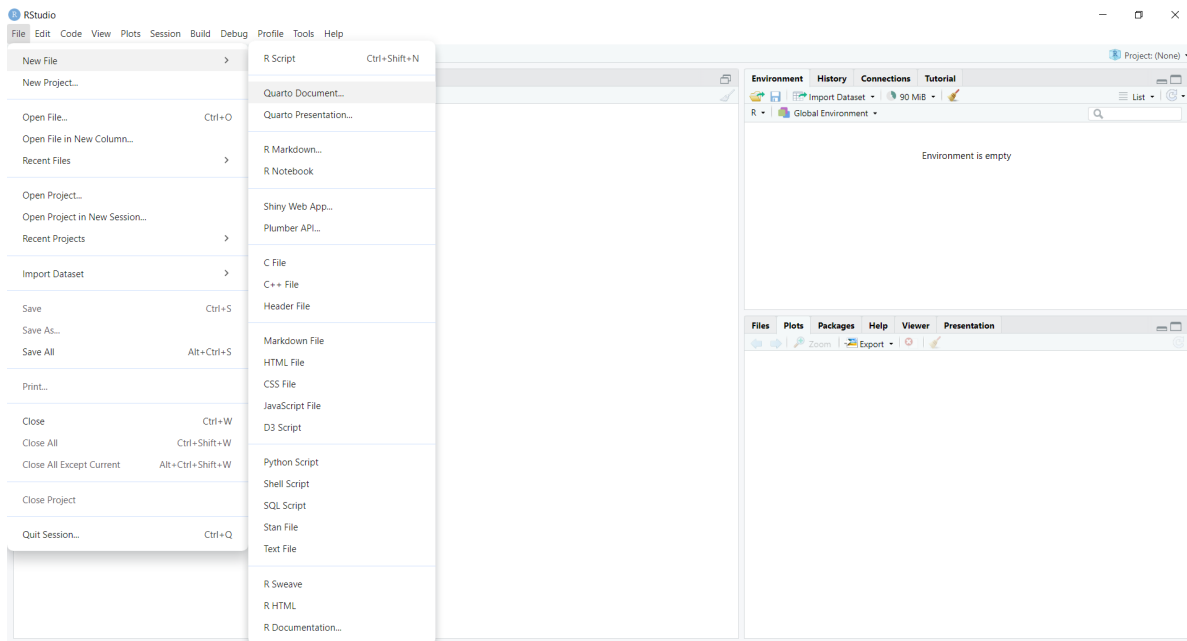


Figure 4.1: Getting Started with Quarto

- Assign a name for the document title. You can always change the title later.

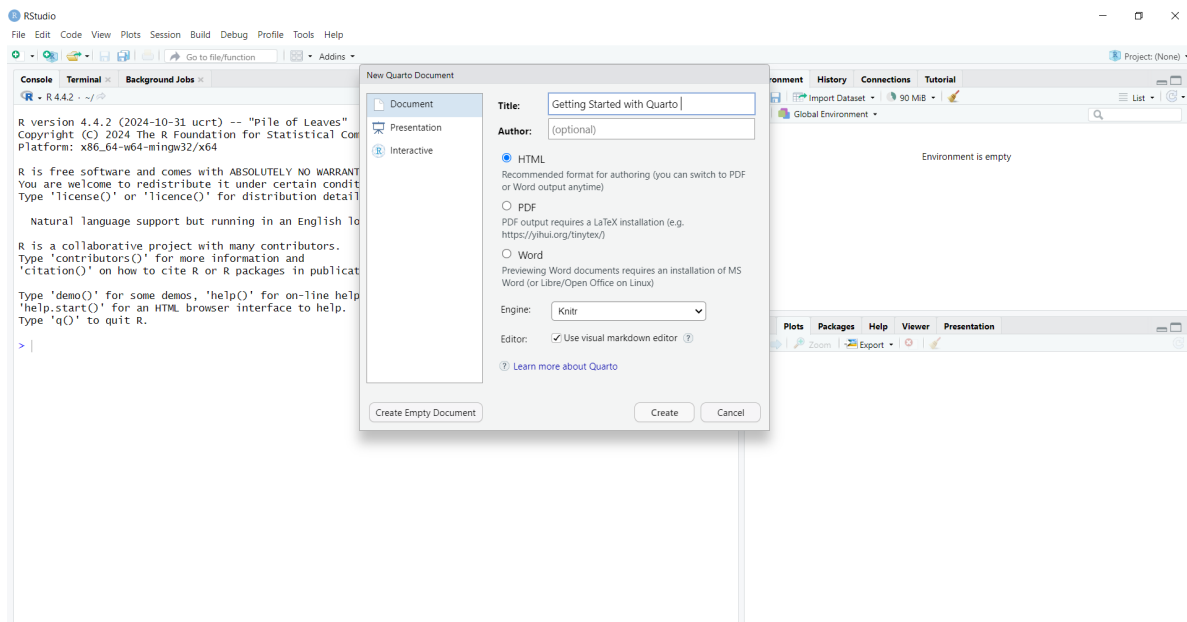


Figure 4.2: Assign a name for the document title

- Choose your preferred output format. In this example, I've selected HTML, which is the default setting. You can switch to PDF or Word output at any time using the same source file, with just minor changes to the output type.
- This will open a Quarto document called “Untitled1\*” with example content. First, save the document with an appropriate file name.
- To get a feel for Quarto, let's render the example document and check the output.
- Use the Render button in the RStudio IDE to render the file and preview the output.
- To view the output in the Viewer Pane, click the settings button next to the Render button on the right side, and select the option “Preview in Viewer Pane”.

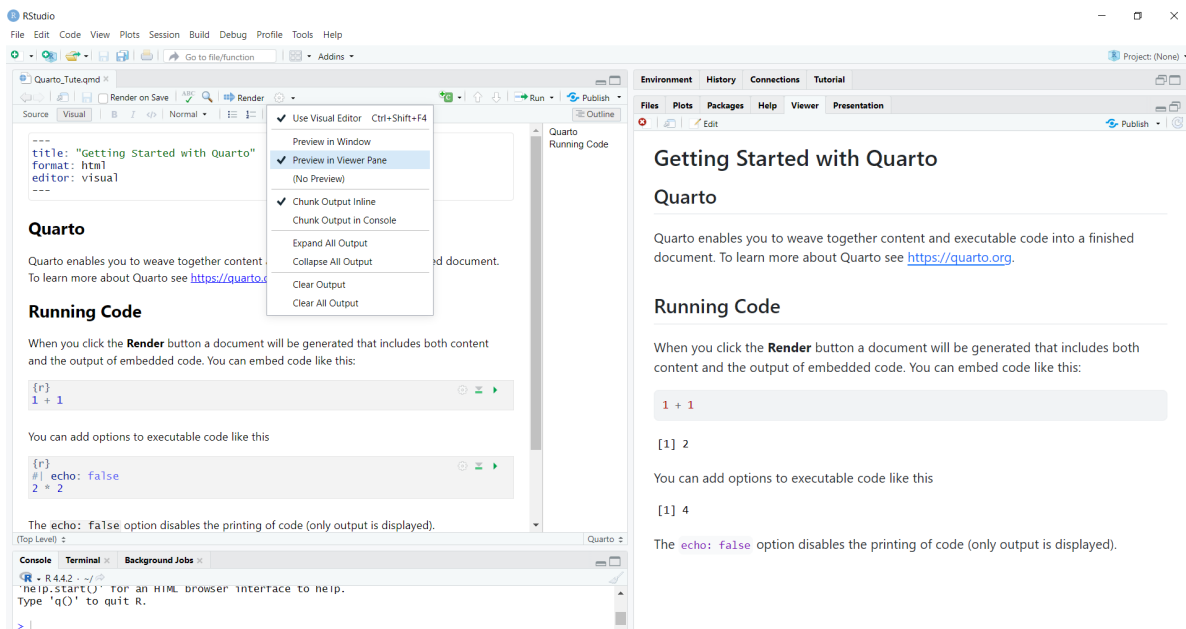


Figure 4.3: Save the document with an appropriate file name

- You will see the rendered output with both the code and the results in a single document.
- If you go to the directory where you saved the `.qmd` file (the Quarto source file), you will also find the generated HTML output with the same name in the same directory.

## 4.4 Anatomy of a Quarto document

Now that you know how to render a Quarto document, let's get familiar with the anatomy of a Quarto document. The file contains three types of content: a YAML header, code chunks, and markdown text.

In addition to that, we can view the same .qmd document in two modes of the RStudio editor: visual (on the left) and source (on the right). Open the source window so that you can easily identify the syntax..

#### 4.4.1 YAML header

- In Quarto, YAML (which stands for YAML Ain't Markup Language) is used for setting up metadata and configurations for your document.
- It is placed at the top of the Quarto file within three dashes (—), and it allows you to specify document options such as the title, author, output format (HTML, PDF, Word), date, and more.
- In YAML, the basic format is **key: value**, where you assign a value to a key.

```
---
title: "Getting Started with Quarto"
subtitle: "Report 1"
author: "Priyanga Talagala"
date: "2025-01-31"
output: html_document
---
```

#### 4.4.2 Code Chunks

- In Quarto, code chunks are sections of the document where you can write and run code. You can think of this part as a mini console window within the Quarto document.
- A code chunk in Quarto is enclosed between three backticks (“”) to separate it from the rest of the text.
- The code inside these chunks is executed when you render the document, and the output (such as tables, plots, or results) is displayed directly in the final document.
- These chunks can contain code written in different programming languages, such as R, Python, or Julia. In the following example, {r} indicates that we are going to run R code inside this code chunk.
- If you want to run Python code, you just need to replace {r} with {python} in the code chunk.

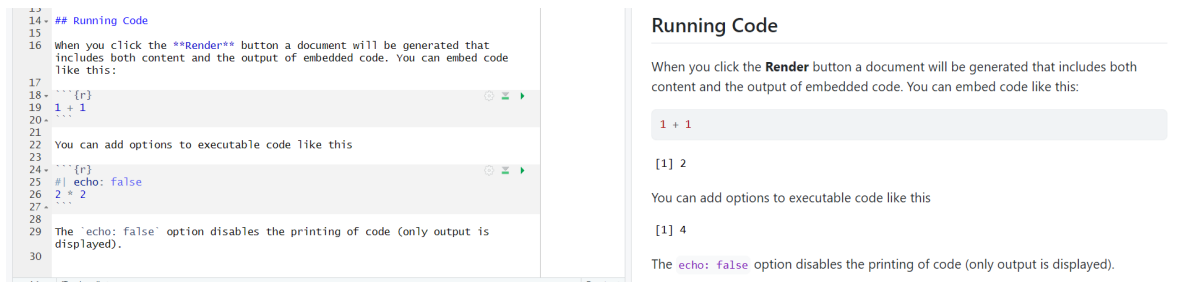


Figure 4.4: Code Chunks. LHS : Source file, RHS: Resulted output

### 4.4.3 Markdown text

- Markdown text includes formatted text, such as section headers, paragraphs, embedded images, and more. Quarto uses Markdown syntax for writing text, which is clear when you switch to the Source tab.
- To define section headings, you use the `#` symbol. The number of `#` symbols determines the level of the heading: one `#` for an H1 heading, two `##` for an H2 heading, three `###` for an H3 heading, and so on.
- In R Markdown syntax, you can format text as follows:
  - To make text bold, use **`**bold**`**.
  - To make text italic, use *`*italic*`*.

This is **`**bold**`** text and this is *`*italic*`* text.

This is **bold** text and this is *italic* text.

# 5 Data Import and Export

## 5.1 Tidy Workflow

As an economist, working with empirical research and data is crucial for making data-driven decisions, and data science helps transform raw data into understanding, knowledge, and insights to support this process. A **tidy workflow** focuses on the tools needed to carry out this process effectively.

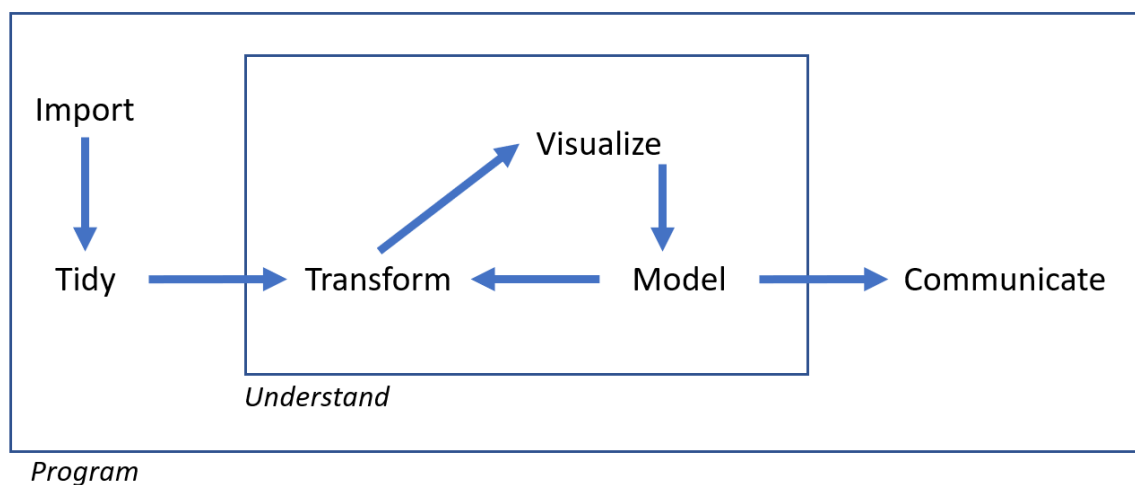


Figure 5.1: Tidy workflow, Image recreated from R for Data Science (2e) (<https://r4ds.hadley.nz/intro.html>)

When working with data, the first step is to import it into our data science environment. Next, we tidy up the data to make it clean and usable. Then, as data scientists, our main task is to understand the data using three key tools: transformation, visualization, and modeling. Finally, we communicate our results to the right people to support decision-making.

The **tidyverse** package is a collection of R packages that work together to make data analysis easier. When you load **tidyverse**, it also loads several useful packages, including: **dplyr**, **readr**, **forcats**, **stringr**, **ggplot2**, **tibble**, **lubridate**, **tidyr** and **purrr**

```
library(tidyverse)
```

Let's dive into each of these steps in the tidy workflow and explore the tools available in the R ecosystem.

## 5.2 Import data

When you work with data, you save your raw data in a separate file, like a CSV. You can easily load this external data into the data science environment using the `readr` package.

### 5.2.1 Read data files into a tibble

Download the dataset: [touristsl.csv](#)

```
data1 <- read_csv("touristsl.csv")
```

If the data file is in another folder within the **current working directory**, you can use the `here` function from the `here` package. With respect to the current working directory, the `here` function helps you define the file path starting from there.

For example, if you have a file called “touristsl.csv” inside a folder called “data” in the current working directory, and you are currently in the working directory, you would need to open the data folder to access the CSV file. In the `here` function, you define the path as `here("data", "touristsl.csv")`. Each folder you want to open is listed, and they are separated by commas.

```
library(here)
data <- read_csv(here("data", "touristsl.csv"))
```

## 5.3 Export data

Similarly, to save a data file, we can use the `write_csv` function.

```
weight <- c(50,44,60)
height <- c(150,160,163)
ds <- tibble(weight, height)
write_csv(ds, "ds.csv")
```

If you want to save the data file in another folder within the current working directory, use the `here` function to define the file path.

```
library(here)
write_csv(ds, here("data", "ds.csv"))
```

---



## 6 Data Wrangling

Data wrangling is the process of transforming and structuring raw data into a structured format to enhance its quality, making it easier to analyze. Data wrangling in the tidyverse workflow generally covers both tidy and transform steps in the tidy workflow.

### 6.1 Data Tidying

A tidy dataset follows these key principles:

- Each variable is placed in its own column.
- Each observation is placed in its own row.
- Each value is placed in its own cell.

The following examples are taken from published reports by various institutes and contain an untidy data structure. This is a common issue in real-world data. The problematic areas that make the data untidy are highlighted.

1. Image Source: Sri Lanka Export Development Board – Export Performance Indicators 2023

## 18. TOP 10 EXPORT SECTORS OF SRI LANKA - 2023

Table 18.1

	Sector	Export Value US\$ Mn.	Share in World exports %	Ranking in World exports	Contribution to total exports %	Exports to major markets as % of total exports of the product	Top export markets
1.	Textiles & Garments	4,865	1.1 % [HS.61] 0.8 % [HS.61]	15[HS.61] 21 [HS.62]	32.21	80%	United States, United Kingdom, Italy, Germany, Netherlands, Canada, India, France, Belgium, Australia
2.	Transport & Logistics	1,550	N/A	N/A	10.26	N/A	N/A
3.	Tea	1,310	16.8 % [HS.0902]	5	8.67	70%	Turkey, Iraq, Russian Federation, United Arab Emirates, China, Saudi Arabia, Azerbaijan, Syrian Arab Republic, Libyan Arab Jamahiriya, United States
4.	ICT/ BPM	1,227	N/A	N/A	8.12	N/A	N/A
5.	Rubber & Rubber based products	930	0.5%	29	6.16	70%	United States, Germany, Belgium, Italy, Brazil, France, Canada, United Kingdom, India, Australia
6.	Coconut & coconut-based products	709	0.85%	32	4.69	70%	United States, Germany, Netherlands, China, United Kingdom, India, United Arab Emirates, Canada, Mexico, Australia
7.	Electrical & electronics	487	0.01%	66	3.22	70%	Switzerland, India, United States, Bangladesh, Germany, Hong Kong, Singapore, United Kingdom, Mexico, Maldives
8.	Food & beverages	428	Since F&B is a vast sector comprising many HS chapters, unable to indicate World share & rank		2.83	70%	India, Maldives, Japan, United States, United Arab Emirates, Cyprus, Australia, Singapore, Malaysia, Netherlands
9.	Spices & Concentrates	398	2.42%	12	2.63	80%	India, Mexico, United States, Peru, Germany, Guatemala, Ecuador, Colombia, Bolivia, Spain
10.	Diamond, Gems & Jewelry	388			2.57	75%	Israel, Switzerland, United States, India, Thailand, Hong Kong, United Arab Emirates, Belgium, France, Italy

N/A – Not available

\*- According to ITC Trade Map Data 2022,

Sources: Central Bank of Sri Lanka, Sri Lanka Customs, ITC Trade map, EDB

Figure 6.1: Untidy Data, Image Source: Sri Lanka Export Development Board – Export Performance Indicators 2023

In this example, certain cells contain more than one value separated by a comma, making the dataset untidy.

## 2. Image Source: Central Bank of Sri Lanka – Annual Report 2022

### NATIONAL OUTPUT, EXPENDITURE, INCOME AND EMPLOYMENT

TABLE 28

#### Performance of Selected State Owned Industrial Enterprises

Corporation/Enterprise and Products	Unit	2018			2019			2020			2021 (a)			2022 (b)		
		Capacity	Production	Sales	Capacity	Production	Sales	Capacity	Production	Sales	Capacity	Production	Sales	Capacity	Production	Sales
1. Lanka Salt Ltd Common Salt	mt	80,000	68,834	47,317	70,000	57,290	62,275	70,000	60,076	56,988	100,000	63,000	78,726	100,000	111,000	60,923
2. State Timber Corporation (c)																
Sawn Timber	m <sup>3</sup> '000	5	4	250	4	5	295	7	3	222	8	4	309	9	3	373
Logs	"	127	130	3,249	109	123	3,445	101	107	3,299	104	128	4,929	108	103	3,624
Sleepers	Nos. '000	42	44	499	49	37	530	50	38	371	40	29	287	29	34	603
3. National Paper Co. Ltd (c)(d) Paper & Paper Products	mt	12,000	-	774	-	-	-	750	81	8.2	6,000	1,154	125	6,000	1,878	466
4. State Printing Corporation (c)																
80 pgs Exercise Books	mn	12	7	222	10	6	166	11	5	122	10	5	138	12	2	219
Text Books	"	13	3	289	13	10	977	20	14	1,096	20	15	1,030	22	15	-
Lottery Tickets	"	420	42	508	525	508	756	537	756	537	756	495	760	585	760	585
Commercial Printing Jobs	Nos.	800	44	360	1,200	448	427	1,200	306	332	1,200	306	270	1,200	280	490
5. Sri Lanka Ayurvedic Drugs Corporation																
Syrup and Oil	mt	724	419	262	640	507	405	692	425	326	755	545	344	845	402	316
Other Ayurvedic Drugs	mt	90	69	39	146	102	61	123	86	63	107	170	63	128	64	48
6. Ceylon Petroleum Corporation (e)																
Petrol (92 Octane)	mt	162,400	165,428	1,172,540	180,500	185,915	1,148,482	187,790	164,416	1,025,126	168,889	124,092	1,102,737	187,790	38,666	964,996
Kerosene	"	56,760	35,195	203,604	62,610	38,345	202,841	69,300	109,165	175,568	60,102	98,284	185,330	66,780	25,289	98,377
Naphtha	"	118,500	140,661	137,123	130,600	162,019	161,960	157,720	156,953	164,649	143,485	106,956	10,624	158,044	30,835	32,259
Auto Diesel	"	500,300	567,577	1,793,763	570,900	624,462	1,998,724	629,953	537,645	1,574,490	568,544	370,594	1,706,155	608,253	128,165	1,475,628
Avtur	"	227,040	237,270	498,841	250,430	258,986	473,780	277,200	157,279	188,731	240,408	130,572	223,854	267,120	57,346	245,838

Figure 6.2: Untidy Data, Image Source: Central Bank of Sri Lanka – Annual Report 2022

In this example, certain cells are merged and presented as a common value using curly braces. This format makes the data untidy and difficult to analyze.

## 3. Image Source: Disaster Management Centre, Ministry of Defense – River Water Level and Flood Warning Report



**Islandwide Water Level & Rainfall Situation in Major Rivers**

DATE : 26-Jan-2025

TIME : 9:30 AM

River Basin	Tributary/River	Gauging Station					Water Level at 8:00 am	Water Level at 9:00 am	Remarks	Water Level Rising or Falling	24 Hr RF in mm at 8.30 am
		Station	Unit	Alert Level	Minor Flood Level	Major Flood Level					
Kelani Ganga (RB 01)	Kelani Ganga	Nagalagam Street	ft	4.00	5.00	7.00	1.50	1.50	Normal		-
	Kelani Ganga	Hanwella	m	7.00	8.00	10.00	0.60	0.58	Normal		6.5
	Kelani Ganga	Glencourse	m	15.00	16.50	19.00	8.68	8.65	Normal		14.8
	Kelani Ganga	Kithulgala	m	3.00	4.00	6.00	1.55	1.50	Normal		0.0
	Garugoda Oya	Holombuwa	m	3.00	3.40	5.00	0.26	0.24	Normal		0.0
	Seethawaka Ganga	Deraniyagala	m	4.80	5.80	6.40	0.23	0.25	Normal		3.4
	Kehelgamu Oya	Norwood	m	1.50	3.00	4.50	0.69	0.68	Normal		0.5
Kalu Ganga (RB 03)	Kalu Ganga	Putupaula	m	3.00	4.00	5.00	0.55	0.46	Normal		0.0
	Kalu Ganga	Ellagawa	m	10.00	10.70	12.20	4.47	4.47	Normal		2.0
	Kalu Ganga	Rathnapura	m	5.20	7.50	9.50	1.04	1.03	Normal		4.1
	Maguru Ganga	Magura	m	4.00	6.00	7.50	1.16	1.16	Normal		0.0
	Kuda Ganga	Kalawellawa (Millakanda)	m	5.00	6.50	8.00	1.57	1.55	Normal		0.0
Gin Ganga (RB 09)	Gin Ganga	Baddegama	m	3.50	4.00	5.00	1.70	1.68	Normal		11.3
	Gin Ganga	Thawalama	m	4.00	6.00	7.50	1.25	1.23	Normal		1.1
Nilwala Ganga (RB 12)	Nilwala Ganga	Thalgahagoda	m	1.40	1.70	2.80	0.41	0.41	Normal		0.0
	Nilwala Ganga	Panadugama	m	5.00	6.00	7.50	2.68	2.68	Normal		0.0
	Nilwala Ganga	Pitabeddara	m	4.00	5.00	6.50	0.51	0.50	Normal		0.0

Figure 6.3: Untidy Data, Image Source: Disaster Management Centre, Ministry of Defense – River Water Level and Flood Warning Report

In this dataset, each cell in the last column contains both a value and a graphical representation, making the data untidy.

The tidy step in the tidy workflow ensures that the data adheres to the tidy principles of tidy data.

The `tidyr` package helps you structure data in a tidy format. This often involves:

- Pivoting: `pivot_longer()` and `pivot_wider()`
- Separating or uniting columns: `separate()` and `unite()`

For more details on its functionalities, refer to the `tidyr` package documentation: <https://tidyr.tidyverse.org/>.

A well-organized dataset saves time and ensures accurate results!

## 6.2 Data Transformation

After tidying up the data, when you start analyzing it, especially with secondary data, it may not always be in the exact form you need. Sometimes, you may need to add new variables using data from other variables. Other times, you may need to filter specific rows or columns from the original dataset. You might also need to summarize the data or rename the columns properly.

This is where the data transformation step comes in.

The `dplyr` package provides powerful tools to transform your data into the desired format. Some of the most commonly used functions in `dplyr` include:

- `filter()` – Select specific rows based on conditions
- `select()` – Choose specific columns
- `mutate()` – Create or modify columns
- `summarise()` – Calculate summary statistics
- `arrange()` – Sort rows by a column
- `group_by()` – Group data for grouped operations
- `rename()` – Change column names

By performing these tasks, you change the structure of the original dataset as demonstrated below, which is why it's called the data transformation step.

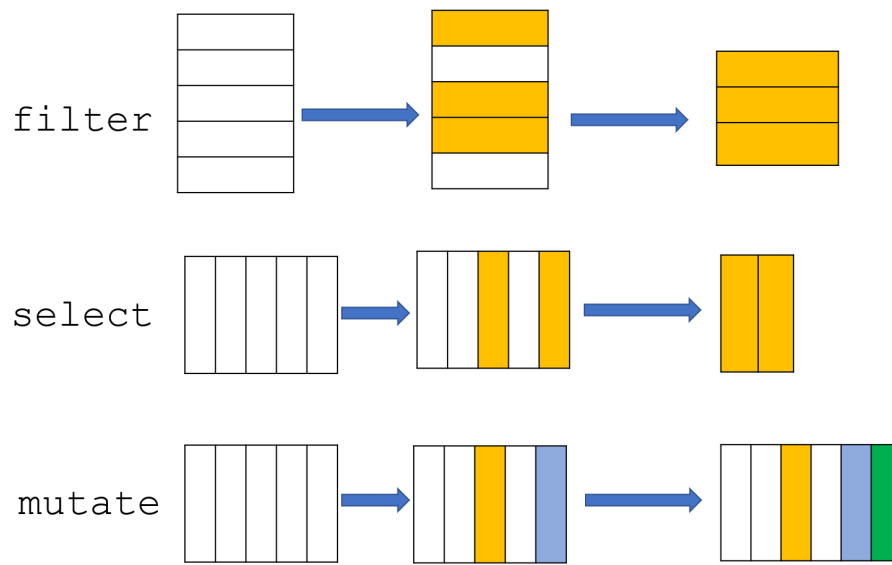


Figure 6.4: Data transformation, The original image, available here <https://perso.ens-lyon.fr/lise.vaudor/dplyr/>, has been updated with new additions.

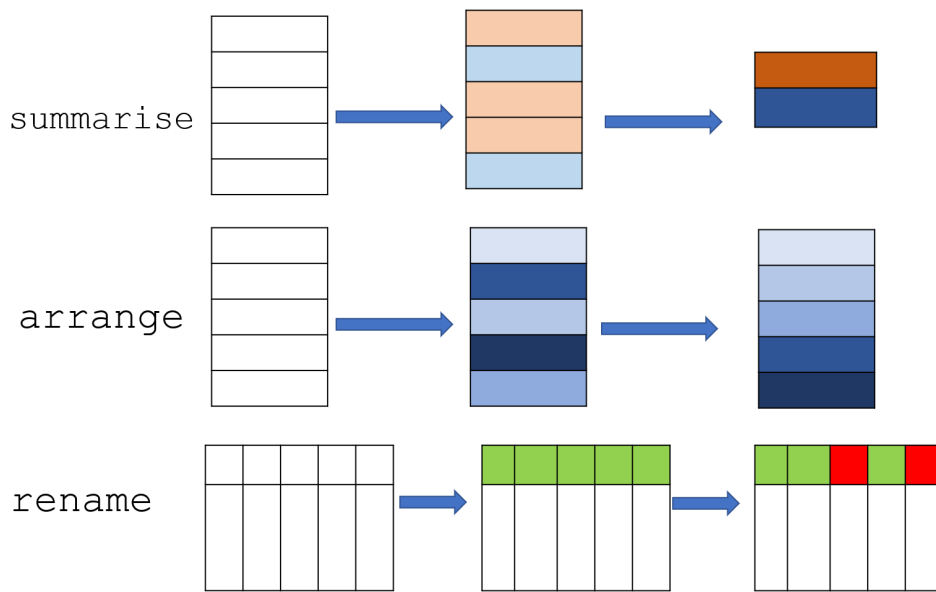


Figure 6.5: Data transformation, The original image, available here <https://perso.ens-lyon.fr/lise.vaudor/dplyr/>, has been updated with new additions.

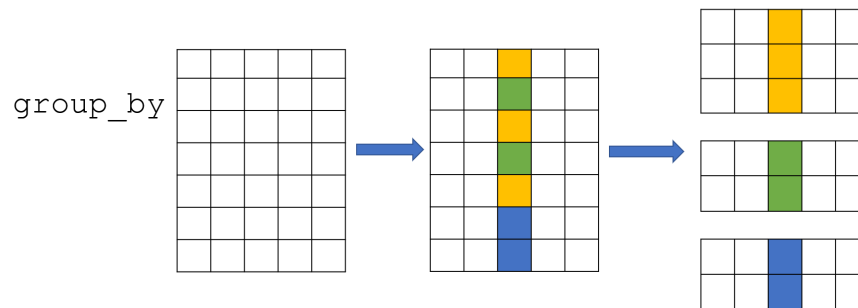


Figure 6.6: Data transformation, The original image, available here <https://perso.ens-lyon.fr/lise.vaudor/dplyr/>, has been updated with new additions.

### 6.2.1 Example

Let's consider the following dataset which contains synthetic airline data.

[Download the dataset: airline\\_data.csv](#) Data Source: <https://www.kaggle.com/datasets/iamsouravbanerjee/airline-dataset/data>

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.4      v readr      2.1.5
v forcats    1.0.0      v stringr    1.5.1
v ggplot2     3.5.1      v tibble     3.2.1
v lubridate  1.9.3      v tidyr      1.3.1
v purrr       1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

```
data <- read_csv(here::here("data", "airline_data.csv"))
```

```
Rows: 98619 Columns: 15
```

```
-- Column specification -----
Delimiter: ","
chr (14): Passenger ID, First Name, Last Name, Gender, Nationality, Airport ...
dbl (1): Age
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dim(data)
```

```
[1] 98619      15
```

```
colnames(data)
```

```
[1] "Passenger ID"      "First Name"        "Last Name"
[4] "Gender"            "Age"               "Nationality"
[7] "Airport Name"      "Airport Country Code" "Country Name"
[10] "Airport Continent" "Continents"         "Departure Date"
[13] "Arrival Airport"   "Pilot Name"        "Flight Status"
```

## Dataset Glossary (Column-wise)

Passenger ID - Unique identifier for each passenger



First Name - First name of the passenger

Last Name - Last name of the passenger

Gender - Gender of the passenger

Age - Age of the passenger

Nationality - Nationality of the passenger

Airport Name - Name of the airport where the passenger boarded

Airport Country Code - Country code of the airport's location

Country Name - Name of the country the airport is located in

Airport Continent - Continent where the airport is situated

Continents - Continents involved in the flight route

Departure Date - Date when the flight departed

Arrival Airport - Destination airport of the flight

Pilot Name - Name of the pilot operating the flight

Flight Status - Current status of the flight (e.g., on-time, delayed, canceled)

This synthetic dataset covers global airline operations. This dataset is useful for economists because it provides information on passenger travel patterns, flight routes, and airport locations. It helps analyze how air travel affects trade, tourism, and jobs. By looking at flight status and passenger details, economists can study the impact of aviation on the economy and make decisions about transportation policies and infrastructure development.

```
data <- data |> as_tibble()
head(data)
```

```
# A tibble: 6 x 15
  `Passenger ID` `First Name` `Last Name` Gender   Age Nationality
  <chr>          <chr>        <chr>      <chr> <dbl> <chr>
1 ABVWIg        Edithe        Leggis    Female   62 Japan
2 jkXXAX        Elwood        Catt      Male     62 Nicaragua
3 CdUz2g        Darby        Felgate   Male     67 Russia
4 BRS38V        Dominica      Pyle      Female   71 China
5 9kvTLo        Bay          Pencost   Male     21 China
6 nMJKVh        Lora         Durbann   Female   55 Brazil
# i 9 more variables: `Airport Name` <chr>, `Airport Country Code` <chr>,
#   `Country Name` <chr>, `Airport Continent` <chr>, Continents <chr>,
#   `Departure Date` <chr>, `Arrival Airport` <chr>, `Pilot Name` <chr>,
```

```
# `Flight Status` <chr>
```

```
summary(data)
```

Passenger ID	First Name	Last Name	Gender
Length:98619	Length:98619	Length:98619	Length:98619
Class :character	Class :character	Class :character	Class :character
Mode :character	Mode :character	Mode :character	Mode :character

Age	Nationality	Airport Name	Airport Country Code
Min. : 1.0	Length:98619	Length:98619	Length:98619
1st Qu.:23.0	Class :character	Class :character	Class :character
Median :46.0	Mode :character	Mode :character	Mode :character
Mean :45.5			
3rd Qu.:68.0			
Max. :90.0			

Country Name	Airport Continent	Continents	Departure Date
Length:98619	Length:98619	Length:98619	Length:98619
Class :character	Class :character	Class :character	Class :character
Mode :character	Mode :character	Mode :character	Mode :character

Arrival Airport	Pilot Name	Flight Status
Length:98619	Length:98619	Length:98619
Class :character	Class :character	Class :character
Mode :character	Mode :character	Mode :character

### 6.2.1.1 filter: Select specific rows based on conditions

- Takes logical expressions and returns the rows for which all are TRUE.

```
filter(data, Age > 45)
```

```
# A tibble: 49,383 x 15
```

```

  `Passenger ID` `First Name` `Last Name` Gender   Age Nationality
<chr>           <chr>         <chr>         <chr>  <dbl> <chr>
1 ABVWIg        Edithe        Leggis        Female   62 Japan
2 jkXXAX        Elwood        Catt          Male    62 Nicaragua
3 CdUz2g        Darby         Felgate       Male    67 Russia
4 BRS38V        Dominica      Pyle          Female   71 China
5 nMJKVh        Lora          Durbann       Female   55 Brazil
6 8IPFPE        Rand          Bram          Male    73 Ivory Coast
7 sBf524        Briant        De La Haye    Male    71 Russia
8 PlwJZT        Kalie         Scoble        Female   47 Sweden
9 iU75x3        Catriona      Beaument      Female   77 Russia
10 eOH5LI       Jerrine       Peeters       Female   87 Philippines
# i 49,373 more rows
# i 9 more variables: `Airport Name` <chr>, `Airport Country Code` <chr>,
#   `Country Name` <chr>, `Airport Continent` <chr>, Continents <chr>,
#   `Departure Date` <chr>, `Arrival Airport` <chr>, `Pilot Name` <chr>,
#   `Flight Status` <chr>

```

```
filter(data, Nationality == "Sri Lanka")
```

```

# A tibble: 138 x 15
  `Passenger ID` `First Name` `Last Name` Gender   Age Nationality
<chr>           <chr>         <chr>         <chr>  <dbl> <chr>
1 fhD57Z        Jordan        Fierman        Female   11 Sri Lanka
2 5joYmi        Valdemar      Marccone       Male     7 Sri Lanka
3 2R0pnA        Jordon        Sallowaye      Male   43 Sri Lanka
4 eKkekT        Dael          Edlestone      Female  58 Sri Lanka
5 5TxM24        Hiram         Memory         Male   21 Sri Lanka
6 wQVX9G        Sherm         Kippie         Male   60 Sri Lanka
7 fKtiLJ        Filippa       Prestige        Female  53 Sri Lanka
8 q7uPji        Wainwright    Dunkerton      Male   47 Sri Lanka
9 Tn9LGe        Matty         Alflat         Male    6 Sri Lanka
10 fZsoAD        Rowney        Messitt        Male   43 Sri Lanka
# i 128 more rows
# i 9 more variables: `Airport Name` <chr>, `Airport Country Code` <chr>,
#   `Country Name` <chr>, `Airport Continent` <chr>, Continents <chr>,
#   `Departure Date` <chr>, `Arrival Airport` <chr>, `Pilot Name` <chr>,
#   `Flight Status` <chr>

```

### 6.2.1.2 select: Choose specific columns by their names.

```
select(data, `Passenger ID`:Gender)
```

```
# A tibble: 98,619 x 4
  `Passenger ID` `First Name` `Last Name` Gender
  <chr>          <chr>        <chr>    <chr>
1 ABVWIg        Edithe        Leggis   Female
2 jkXXAX        Elwood        Catt     Male
3 CdUz2g        Darby         Felgate  Male
4 BRS38V        Dominica      Pyle     Female
5 9kvTLo        Bay          Pencost  Male
6 nMJKVh        Lora         Durbann  Female
7 8IPFPE        Rand         Bram     Male
8 pqixbY        Perceval     Dallosso Male
9 QNAs2R        Aleda        Pigram   Female
10 3jmudz        Burlie       Schustl  Male
# i 98,609 more rows
```

```
select(data, `Pilot Name`, `Flight Status`)
```

```
# A tibble: 98,619 x 2
  `Pilot Name`      `Flight Status`
  <chr>            <chr>
1 Fransisco Hazeldine On Time
2 Marla Parsonage    On Time
3 Rhonda Amber       On Time
4 Kacie Commucci     Delayed
5 Ebonee Tree        On Time
6 Inglis Dolley      On Time
7 Stanislas Tiffin   Cancelled
8 Sharyl Eastmead    Cancelled
9 Daryn Bardsley     On Time
10 Alameda Carlyle   On Time
# i 98,609 more rows
```

When you pass a vector of column names with `-c()`, it omits the specified columns and returns the dataframe with the remaining columns. This option is very useful when working with large datasets that have many columns, and you only need to remove a few columns to finalize the dataset.

```
select(data, -c(`Passenger ID`, `First Name`, `Last Name`))
```

```
# A tibble: 98,619 x 12
  Gender   Age Nationality `Airport Name` `Airport Country Code` `Country Name`
  <chr>   <dbl> <chr>         <chr>         <chr>         <chr>
1 Female    62 Japan      Coldfoot Airp~ US          United States
2 Male     62 Nicaragua Kugluktuk Air~ CA          Canada
3 Male     67 Russia    Grenoble-Isèr~ FR          France
4 Female    71 China      Ottawa / Gati~ CA          Canada
5 Male     21 China      Gillespie Fie~ US          United States
6 Female    55 Brazil     Coronel Horác~ BR          Brazil
7 Male     73 Ivory Coast Duxford Aerod~ GB          United Kingdom
8 Male     36 Vietnam    Maestro Wilso~ BR          Brazil
9 Female    35 Palestinia~ Venice Marco ~ IT          Italy
10 Male     13 Thailand   Vermilion Air~ CA          Canada
# i 98,609 more rows
# i 6 more variables: `Airport Continent` <chr>, Continents <chr>,
#   `Departure Date` <chr>, `Arrival Airport` <chr>, `Pilot Name` <chr>,
#   `Flight Status` <chr>
```

### 6.2.1.3 mutate: Create or modify columns

- This function allows you to creates new variables from an existing variable.

```
data_new <- data |> mutate(New_Age = Age + 2)
data_new
```

```
# A tibble: 98,619 x 16
  `Passenger ID` `First Name` `Last Name` Gender   Age Nationality
  <chr>          <chr>       <chr>      <chr>   <dbl> <chr>
1 ABVWIg        Edithe      Leggis    Female    62 Japan
2 jkXXAX        Elwood      Catt      Male     62 Nicaragua
3 CdUz2g        Darby       Felgate   Male     67 Russia
4 BRS38V        Dominica    Pyle      Female    71 China
5 9kvTLo        Bay         Pencost   Male     21 China
6 nMJKVh        Lora        Durbann   Female    55 Brazil
7 8IPFPE        Rand        Bram      Male     73 Ivory Coast
8 pqixbY        Perceval    Dallosso  Male     36 Vietnam
9 QNAs2R        Aleda       Pigram    Female    35 Palestinian Territory
10 3jmudz        Burlie      Schustl   Male     13 Thailand
```

```
# i 98,609 more rows
# i 10 more variables: `Airport Name` <chr>, `Airport Country Code` <chr>,
#   `Country Name` <chr>, `Airport Continent` <chr>, Continents <chr>,
#   `Departure Date` <chr>, `Arrival Airport` <chr>, `Pilot Name` <chr>,
#   `Flight Status` <chr>, New_Age <dbl>
```

```
colnames(data_new)
```

```
[1] "Passenger ID"      "First Name"        "Last Name"
[4] "Gender"            "Age"               "Nationality"
[7] "Airport Name"      "Airport Country Code" "Country Name"
[10] "Airport Continent" "Continents"         "Departure Date"
[13] "Arrival Airport"   "Pilot Name"         "Flight Status"
[16] "New_Age"
```

- The same mutate function also allows you to update an existing variables.

```
data_new <- data |> mutate(`Departure Date` =as.Date(`Departure Date`, format = "%m/%d/%Y"))
summary(data_new)
```

Passenger ID	First Name	Last Name	Gender
Length:98619	Length:98619	Length:98619	Length:98619
Class :character	Class :character	Class :character	Class :character
Mode :character	Mode :character	Mode :character	Mode :character

Age	Nationality	Airport Name	Airport Country Code
Min. : 1.0	Length:98619	Length:98619	Length:98619
1st Qu.:23.0	Class :character	Class :character	Class :character
Median :46.0	Mode :character	Mode :character	Mode :character
Mean :45.5			
3rd Qu.:68.0			
Max. :90.0			

Country Name	Airport Continent	Continents	Departure Date
Length:98619	Length:98619	Length:98619	Min. :2022-01-01
Class :character	Class :character	Class :character	1st Qu.:2022-04-01
Mode :character	Mode :character	Mode :character	Median :2022-07-01
			Mean :2022-07-01
			3rd Qu.:2022-09-30
			Max. :2022-12-30

Arrival Airport	Pilot Name	Flight Status
Length:98619	Length:98619	Length:98619
Class :character	Class :character	Class :character
Mode :character	Mode :character	Mode :character

#### 6.2.1.4 summarise(British) or summarize (US) : Calculate summary statistics

- this function collapses many values down to a single summary.

```
data_new |>
  summarise(
    Departure_Date_start =min(`Departure Date`),
    Departure_Date_end =max(`Departure Date`),
    Age_median=median(Age),
    Age_mean=mean(Age))
```

```
# A tibble: 1 x 4
  Departure_Date_start Departure_Date_end Age_median Age_mean
  <date>              <date>              <dbl>    <dbl>
1 2022-01-01          2022-12-30              46      45.5
```

Since it contains missing values, they should be removed before calculating the summary statistics

```
data_new |>
  summarise(
    Departure_Date_start =min(`Departure Date`, na.rm = TRUE),
    Departure_Date_end =max(`Departure Date`, na.rm = TRUE),
    Age_median=median(Age, na.rm = TRUE),
    Age_mean=mean(Age, na.rm = TRUE))
```

```
# A tibble: 1 x 4
  Departure_Date_start Departure_Date_end Age_median Age_mean
  <date>              <date>              <dbl>    <dbl>
1 2022-01-01          2022-12-30              46      45.5
```

### 6.2.1.5 arrange() – Sort rows by a column

```
arrange(data, desc(Age))
```

```
# A tibble: 98,619 x 15
  `Passenger ID` `First Name` `Last Name` Gender   Age Nationality
  <chr>          <chr>        <chr>      <chr> <dbl> <chr>
1 to75jW        Pamela      Eaden      Female   90  Indonesia
2 Zlsjqw        Sydney     MacGhee     Male    90  Guatemala
3 aogYwD        Vernen     Ivakhnov    Male    90  China
4 X0basI        Mariann    Moogan      Female   90  Philippines
5 N7DLfQ        Jermaine   Escott      Male    90  Portugal
6 fZhPqR        Valdemar   Whate       Male    90  Brazil
7 usnzLP        Raul       Mantha      Male    90  Argentina
8 L49UDI        Neall      Zamudio     Male    90  Czech Republic
9 L1ZBXS        Anthea     Blodget     Female   90  Indonesia
10 CjAY9o        Valene     Megarry     Female   90  Indonesia
# i 98,609 more rows
# i 9 more variables: `Airport Name` <chr>, `Airport Country Code` <chr>,
#   `Country Name` <chr>, `Airport Continent` <chr>, Continents <chr>,
#   `Departure Date` <chr>, `Arrival Airport` <chr>, `Pilot Name` <chr>,
#   `Flight Status` <chr>
```

### 6.2.1.6 group\_by() – Group data for grouped operations

- This function takes an existing tibble and converts it into a grouped tibble where operations are performed “by group”. `ungroup()` removes grouping.

```
customers_grouped <- data |> group_by(Continents )
customers_grouped
```

```
# A tibble: 98,619 x 15
# Groups:   Continents [6]
  `Passenger ID` `First Name` `Last Name` Gender   Age Nationality
  <chr>          <chr>        <chr>      <chr> <dbl> <chr>
1 ABVWIg        Edithe     Leggis     Female   62  Japan
2 jkXXAX        Elwood     Catt       Male    62  Nicaragua
3 CdUz2g        Darby      Felgate    Male    67  Russia
4 BRS38V        Dominica   Pyle       Female   71  China
5 9kvTL0        Bay       Pencost    Male    21  China
```



```

6 nMJKVh      Lora      Durbann      Female      55 Brazil
7 8IPFPE      Rand      Bram      Male      73 Ivory Coast
8 pqixbY      Perceval      Dallosso      Male      36 Vietnam
9 QNAs2R      Aleda      Pigram      Female      35 Palestinian Territory
10 3jmudz      Burlie      Schustl      Male      13 Thailand
# i 98,609 more rows
# i 9 more variables: `Airport Name` <chr>, `Airport Country Code` <chr>,
#   `Country Name` <chr>, `Airport Continent` <chr>, Continents <chr>,
#   `Departure Date` <chr>, `Arrival Airport` <chr>, `Pilot Name` <chr>,
#   `Flight Status` <chr>

```

```
data |> summarise(age_mean=mean(Age, na.rm=TRUE))
```

```

# A tibble: 1 x 1
  age_mean
  <dbl>
1    45.5

```

```
customers_grouped |> summarise(age_mean=mean(Age, na.rm=TRUE))
```

```

# A tibble: 6 x 2
  Continents      age_mean
  <chr>          <dbl>
1 Africa          45.6
2 Asia            45.7
3 Europe          45.2
4 North America   45.6
5 Oceania         45.4
6 South America   45.4

```

### 6.2.1.7 rename() – Change column names

```

data <- rename(data, Passenger_ID = `Passenger ID`,
               Airport_code = `Airport Country Code`) # new_name = old_name
data

```

```

# A tibble: 98,619 x 15
  Passenger_ID `First Name` `Last Name` Gender   Age Nationality `Airport Name`
  <chr>        <chr>        <chr>    <chr> <dbl> <chr>        <chr>

```

```

1 ABVWIg      Edithe      Leggis      Female      62 Japan      Coldfoot Airp~
2 jkXXAX      Elwood      Catt        Male        62 Nicaragua  Kugluktuk Air~
3 CdUz2g      Darby       Felgate     Male        67 Russia     Grenoble-Isèr~
4 BRS38V      Dominica    Pyle        Female      71 China      Ottawa / Gati~
5 9kvTLo      Bay         Pencost     Male        21 China      Gillespie Fie~
6 nMJKVh      Lora        Durbann     Female      55 Brazil     Coronel Horác~
7 8IPFPE      Rand        Bram        Male        73 Ivory Coast Duxford Aerod~
8 pqixbY      Perceval    Dallosso    Male        36 Vietnam    Maestro Wilso~
9 QNAs2R      Aleda       Pigram      Female      35 Palestinia~ Venice Marco ~
10 3jmudz      Burlie      Schustl     Male        13 Thailand   Vermilion Air~
# i 98,609 more rows
# i 8 more variables: Airport_code <chr>, `Country Name` <chr>,
#   `Airport Continent` <chr>, Continents <chr>, `Departure Date` <chr>,
#   `Arrival Airport` <chr>, `Pilot Name` <chr>, `Flight Status` <chr>

```

### 6.2.1.8 Combine multiple operations

You can even combine multiple verbs at once to obtain the dataset in the desired structure. This is where the pipe operator becomes very handy.

```

data |>
  filter(Nationality == "Sri Lanka") |>
  head(2)

```

```

# A tibble: 2 x 15
  Passenger_ID `First Name` `Last Name` Gender   Age Nationality `Airport Name`
  <chr>        <chr>        <chr>    <chr> <dbl> <chr>        <chr>
1 fhD57Z      Jordan      Fierman   Female   11 Sri Lanka  Yoshkar-Ola Ai~
2 5joYmi      Valdemar    Marccone  Male     7 Sri Lanka  Mungeranie Air~
# i 8 more variables: Airport_code <chr>, `Country Name` <chr>,
#   `Airport Continent` <chr>, Continents <chr>, `Departure Date` <chr>,
#   `Arrival Airport` <chr>, `Pilot Name` <chr>, `Flight Status` <chr>

```

```

data |>
  filter(Nationality == "Sri Lanka") |>
  summarise(Age_mean=mean(Age, na.rm=TRUE))

```

```

# A tibble: 1 x 1
  Age_mean
  <dbl>
1    43.9

```

```
data_new |>
  filter(Nationality == "Sri Lanka") |>
  group_by(Gender) |>
  filter(Age > 45) |>
  arrange(desc(`Departure Date`))
```

```
# A tibble: 65 x 15
```

```
# Groups:   Gender [2]
```

	`Passenger ID`	`First Name`	`Last Name`	Gender	Age	Nationality
	<chr>	<chr>	<chr>	<chr>	<dbl>	<chr>
1	Garp7s	Elwood	Seczyk	Male	61	Sri Lanka
2	bmMaNv	Mellie	Werrett	Female	76	Sri Lanka
3	W9kjm9	Lewes	Gollard	Male	75	Sri Lanka
4	C5sSaq	Jose	Duffit	Male	87	Sri Lanka
5	YpH6kH	Xenos	McEntagart	Male	71	Sri Lanka
6	lXSEsW	Ad	Fernihough	Male	48	Sri Lanka
7	50PpWn	Marieann	Strelitzki	Female	77	Sri Lanka
8	oCZkZ2	Aaron	MacCaffery	Male	83	Sri Lanka
9	L7ut1f	Dell	Shreenan	Male	62	Sri Lanka
10	soCbR1	Matthieu	Matches	Male	47	Sri Lanka

```
# i 55 more rows
```

```
# i 9 more variables: `Airport Name` <chr>, `Airport Country Code` <chr>,
#   `Country Name` <chr>, `Airport Continent` <chr>, Continents <chr>,
#   `Departure Date` <date>, `Arrival Airport` <chr>, `Pilot Name` <chr>,
#   `Flight Status` <chr>
```

## 7 Data Visualization

In R, there are many ways to create visualizations, as it provides various built-in functions for generating charts and graphics. . The graphics package, which comes with R by default, includes basic functions for creating different types of plots, such as line graphs, scatter plots, histograms, and bar charts. In this approach, each type of graph has its own dedicated function.

However, when there are too many plotting functions to remember, managing them becomes difficult. Every time you need a new type of graph, you have to learn a new function. This challenge is what visualization expert Leland Wilkinson aimed to address through the Grammar of Graphics.

Instead of treating plots as separate, unrelated charts, the Grammar of Graphics provides a unified framework for creating visualizations using a consistent set of rules. This approach moves beyond simply naming charts (e.g., pie chart, line chart, scatter plot) and focuses on understanding their structure or anatomy. By using layered components, we can build effective visualizations in a more systematic way.

The key idea behind the Grammar of Graphics is that any plot can be explained using eight layered components. This theoretical design system serves as the foundation for the ggplot2 package in R, which is an implementation of these principles.

### 7.1 The ggplot2 API

The ggplot2 package structures plots using eight key components (layers):

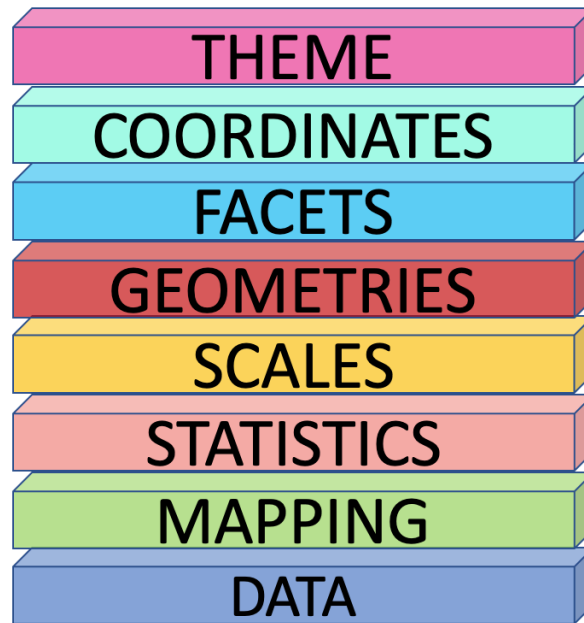


Figure 7.1: The ggplot2 API

In each layer, we aim to answer a specific question:

- **Data** – What dataset should be used for the visualization?
- **Mapping (Aesthetics - aes)** – How should the data be mapped to visual properties? (e.g., x-axis, y-axis, color, size)\*
- **Statistics (stat)** – Should any transformations or summaries be applied? (e.g., smoothing, binning, aggregation)
- **Scales** – How should data values be translated into visual properties? (e.g., color gradients, axis scales)
- **Geometries (geom)** – What type of plot best represents the data? (e.g., points, lines, bars)
- **Facets** – Should the data be divided into multiple panels for better comparison?
- **Coordinates (coord)** – Which coordinate system should be used? (e.g., Cartesian, polar)
- **Theme** – How should the overall appearance of the plot be styled? (e.g., fonts, background, gridlines)

By layering these components, ggplot2 provides a flexible and structured approach to building effective visualizations

Now, let's see how to create a plot using ggplot2.

Let's start with the foundational layer: the data layer

## 7.2 Data Layer

This layer asks the question: What dataset should be used for the visualization?

In this example, I am going to use South Asian economic indicators from the World Bank, covering the years 1960 to 2017.

[Download the dataset: SAEconomy.csv](#)

### Dataset Glossary (Column-wise)

Country: The country or region of the series.

GDP: Gross domestic product (in \$USD February 2019).

Growth: Annual percentage growth in GDP.

CPI: Consumer price index (base year 2010).

Imports: Imports of goods and services (% of GDP).

Exports: Exports of goods and services (% of GDP).

Population: Total population.

For the initial exploration, I am going to filter the economic health data specifically related to Sri Lanka and analyze the country's economic health .

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.4      v readr      2.1.5
v forcats    1.0.0      v stringr    1.5.1
v ggplot2    3.5.1      v tibble     3.2.1
v lubridate  1.9.3      v tidyr      1.3.1
v purrr      1.0.2

-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

```
SAeconomy_data <- read_csv(here::here("data", "SAeconomy.csv" ))
```

Rows: 464 Columns: 9

-- Column specification -----

Delimiter: ","

chr (2): Country, Code

dbl (7): Year, GDP, Growth, CPI, Imports, Exports, Population

i Use `spec()` to retrieve the full column specification for this data.

i Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

```
SLeconomy <- SAeconomy_data |>
  filter(Country == "Sri Lanka")
```

SLeconomy

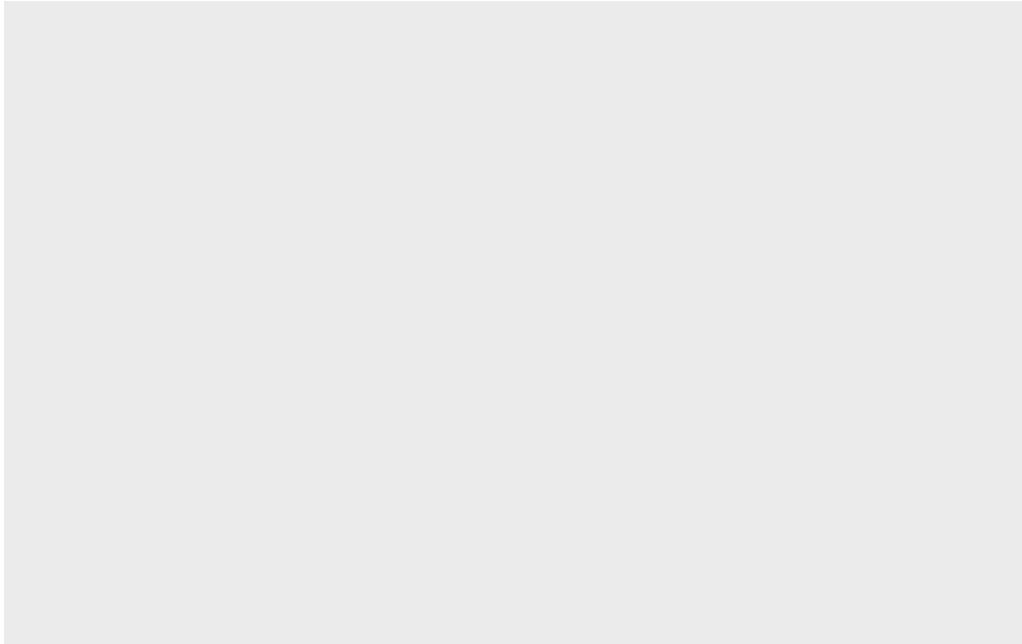
# A tibble: 58 x 9

	Country	Code	Year	GDP	Growth	CPI	Imports	Exports	Population
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	Sri Lanka	LKA	1960	1409873950.	NA	1.49	32.9	30.0	9874481
2	Sri Lanka	LKA	1961	1444327731.	NA	1.50	28.5	27.5	10111646
3	Sri Lanka	LKA	1962	1434156379.	3.82	1.53	29.2	27.8	10352188
4	Sri Lanka	LKA	1963	1240672269.	2.52	1.56	27.5	25.8	10597520
5	Sri Lanka	LKA	1964	1309747899.	3.91	1.61	26.7	24.6	10849979
6	Sri Lanka	LKA	1965	1698319328.	2.54	1.62	25.5	25.9	11110828
7	Sri Lanka	LKA	1966	1751470588.	5.02	1.61	25.8	22.4	11380683
8	Sri Lanka	LKA	1967	1859465021.	6.44	1.65	23.3	20.5	11657660
9	Sri Lanka	LKA	1968	1801344538.	5.80	1.74	23.6	20.6	11937611
10	Sri Lanka	LKA	1969	1965546218.	7.72	1.87	24.6	18.4	12214968

# i 48 more rows

A ggplot function call begins with the `ggplot()` function.

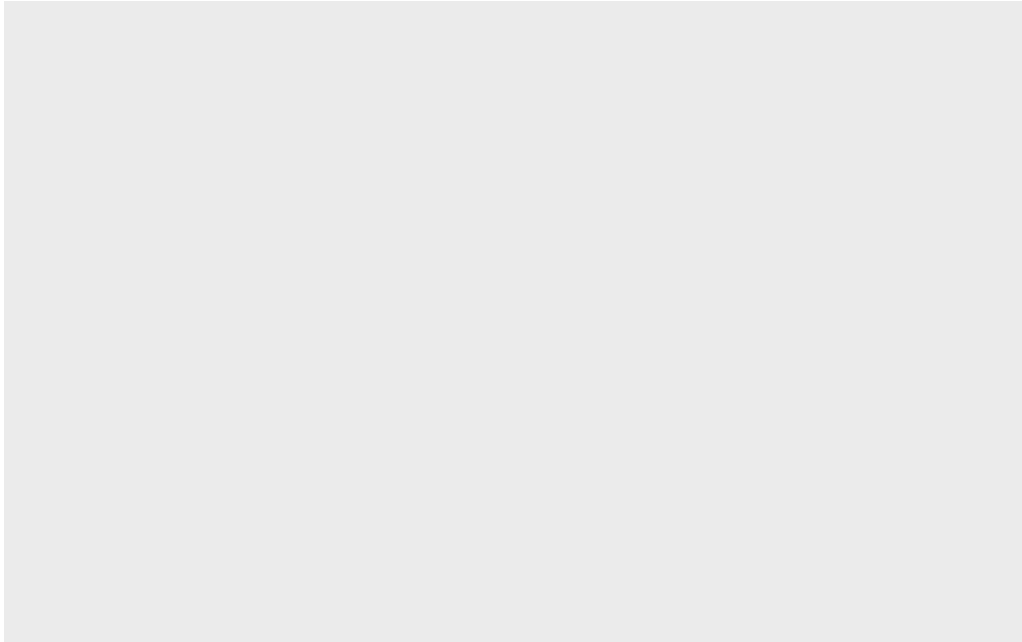
```
ggplot()
```



This call is similar to taking a blank piece of paper to draw a plot, where you set the foundation for what will be visualized. In the data layer, you have to specify what dataset you want to use for the visualization. In this case, the dataset will be the economic health data for Sri Lanka.

```
ggplot(data = SLeconomy)
```





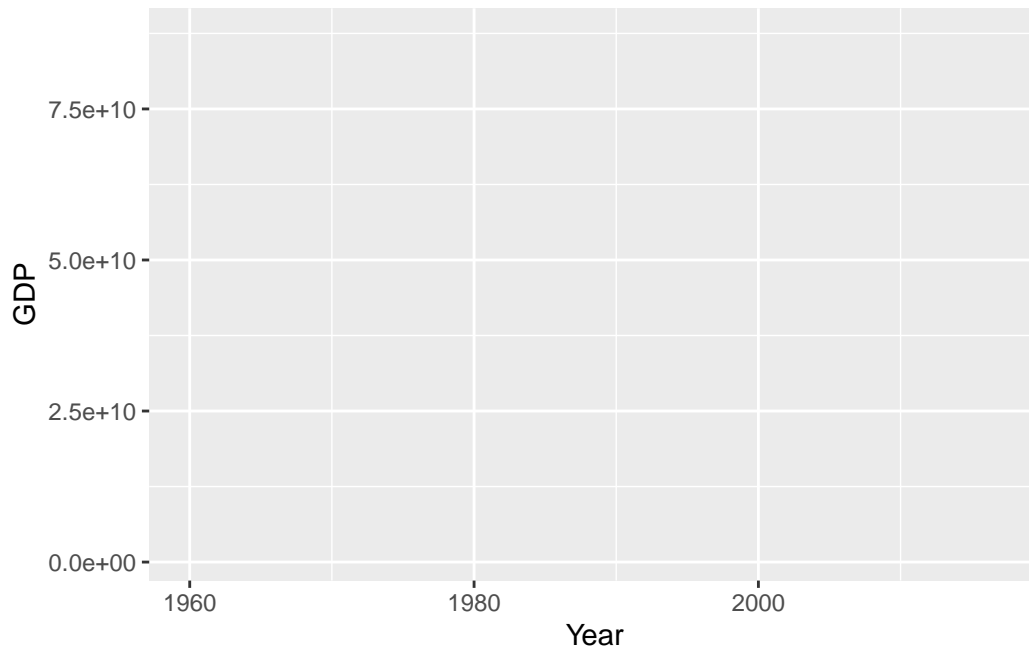
You're still getting the same result because `ggplot()` doesn't know how to work with the dataset until you tell it what to plot and how to map the data to the axes and visual elements.

This is done using the second layer, the Mapping layer.

## 7.3 Mapping Layer

In this layer you define how the data should be represented visually, such as which variables to map to the x-axis, y-axis, and other visual elements.

```
ggplot(data = SLeconomy,  
       mapping = aes(x = Year,  
                     y = GDP))
```



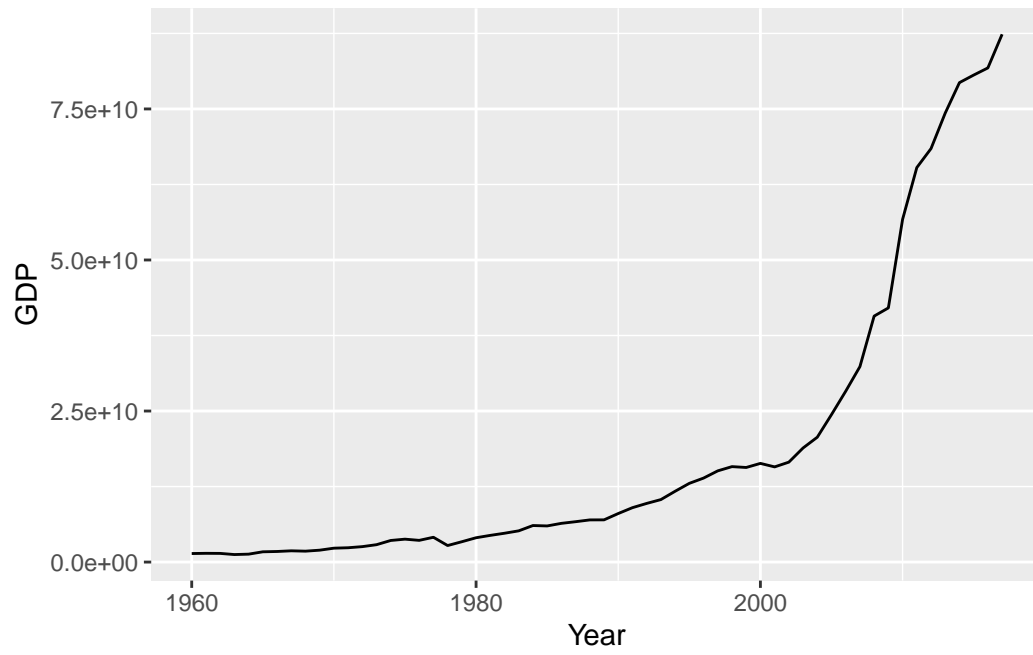
With the Mapping layer, we define what we want to see on the graph. For example, if I want to see the trend in the GDP series over the years, I select the relevant columns and map them to the x-axis and y-axis. This will label the axes with the selected variable names. However, you still can't see a visual representation in the plotting panel. That's because you need to specify how to draw the plot and what kind of geometry to use to represent the data values. This is where the Geometries layer comes in.

## 7.4 Geometries Layer

The Geometries layer is concerned with the shape of the data. In this case, since I have a time series, I can represent the data by using a line to show how the values change over the years.

We connect each layer with a plus sign (+), because it essentially “adds layers” to the existing plot, building on top of what was already defined.

```
ggplot(data = SLeconomy,  
       mapping = aes(x = Year,  
                     y = GDP)) +  
  geom_line()
```



## 8 Introduction to Statistical Modelling

WIP

# References

R for Data Science (2e) : <https://r4ds.hadley.nz/workflow-scripts.html>

Advanced R: <https://adv-r.hadley.nz/>