Introduction to R - Young Researchers Fellowship Program

Lecture 1 - The bare basics

Daniel Sánchez Pazmiño

Laboratorio de Investigación para el Desarrollo del Ecuador

September 2024

Brief intro to R

What is R?

Tip

R is, at its heart, an elegant and beautiful language, well tailored for data analysis and statistics. Hadley Wickham, Chief Scientist at Posit (formerly RStudio), 2019.

What is R?

- A programming language (like Python, Java, C++, etc.)
  - Means that it has a syntax, grammar, and rules to follow.
  - Unlike Stata, SPSS, Minitab or Excel, which are software packages (that may allow you to write scripts).
  - Fully passes the Turing test (meaning it can do anything a computer can do).
- Well tailored for data analysis and statistics.
  - Has a lot of built-in functions and packages for data manipulation, visualization, and statistical analysis.
  - Was built by statisticians, though it has grown to be used in many other fields and purposes.

A brief history of R

- R was created by Ross Ihaka and Robert Gentleman at the University of Auckland in 1993.
    - They wanted to create a free and open-source alternative to the S programming language.
    - S was created at Bell Labs in the 1970s, and was commercialized by AT&T in the 1980s.
    - S was a language for data analysis and statistics, and was the precursor to R.
- R was released in 1995, and has been in continuous development since then.
    - It is now maintained by the R Core Team, a group of volunteers from around the world.
    - The R Foundation is a non-profit organization that supports the development of R.

A brief history of R

- Posit (formerly RStudio) was founded in 2009 by JJ Allaire, and is now the main company behind R.
  - They develop RStudio, the most popular IDE for R.
  - They also develop many packages for R, and contribute to the development of R itself (e.g. Quarto and the Tidyverse).
  - Hadley Wickham, Chief Scientist at Posit, is one of the most influential R developers.

Why R?

- **Free**: No cost to download, install, modify, distribute or use.
- **Open-source**: You can see the code that makes R work, and you can contribute to it.
  - Allows for a large community of developers and users.
  - Means that you can trust the software more, as it is open to scrutiny.
  - Also means that you can use it for any purpose, commercial or not.
  - Packages are easily developed and shared.

Why R for academic research?

- Scripting capabilities: complex analyses are the bread and butter of modern social science research.
    - R will allow you to automate these analyses, and to reproduce them easily.
    - Other click-and-point software (like SPSS, Minitab, etc.) are worse at this.
- Reproducibility: R allows you to write scripts that can be shared with others, and that can be run by others to reproduce your results.
    - This is a key aspect of the scientific method, and is becoming more and more important in academic research (journals require it).
    - Reproducibility allows for easier error detection and correction.
    - Other software make this very complicated, even those that allow you to write scripts (like Stata).

Why R for academic research?

- Free: less inequality in access to software, and less dependence on the whims of your institution, or on the availability of pirated software.
- Flexibility and availability of packages: R has a large community of developers, and a large repository of packages that can be used for a wide variety of purposes.
    - Often cutting-edge methods are available in R before they are available in other software as statisticians directly contribute to R rather than waiting for software developers to implement them.
    - This is especially important for interdisciplinary research.

The R ecosystem

- **R**: The base software, which you can download from CRAN.
- IDEs: Integrated Development Environments, which make it easier to write and run R scripts.
  - RStudio: The most popular IDE for R.
  - Jupyter: A notebook interface that allows you to write and run code in a more interactive way.
  - VS Code: A general-purpose code editor that can be used for R.
  - Many more...
- **Packages**: Libraries of functions that extend the capabilities of R.
  - CRAN: The main repository of R packages.
  - Bioconductor: A repository of packages for bioinformatics.
  - GitHub: where many developers share their packages before they are published in CRAN (if ever).

The R ecosystem

- Graphical user interfaces (GUIs): Software that allows you to use R without writing code.
  - R Commander: A GUI for R.
  - Rattle: A GUI for data mining.
  - RKWard: A GUI for R that integrates with the KDE desktop environment.
  - Many more...
- Analytical software based on R: Software that uses R in the background, but that is not R itself.
  - JASP: A software package that uses R in the background.
  - Jamovi: A software package that uses R in the background.
  - Many more...

The R ecosystem

- Database integrations and APIs: R can connect to many databases and APIs, allowing you to pull data directly into R for analysis.
- Document markup: R can be used to create reports, papers, and presentations.
  - R Markdown: A package that allows you to write reports, papers, and presentations in R.
  - Bookdown: A package that allows you to write books in R.
  - Quarto: A package that allows you to write reports, papers, and presentations in R.
  - Knitr: A package that allows you to create dynamic documents in R using LaTeX.
- Business intelligence software integrations: R can be used to create reports and visualizations integrated with business intelligence software, such as Tableau, Power BI, and Qlik.

The garden and the gardener: CRAN and package developers

- Expert progammers in R don't just use R, they also contribute to it.
    - They write packages that extend the capabilities of R.
    - They share these packages with the community by publishing them on CRAN, GitHub, or other repositories.
    - They maintain these packages, fixing bugs and adding features as needed.
- CRAN has very strict rules for package submission, which ensures that packages are of high quality.
    - This is one of the reasons why R is so popular in academia and industry.
    - It is also one of the reasons why R is so powerful and flexible.
- Other open-source languages (like Python) have similar ecosystems, but R's gardener, the R Core Team, is very active and has a lot of experience in maintaining a large and complex software project.
    - This is one of the reasons why R is so reliable and trustworthy.

Where R is being used in academia

- **Statistics**: R was developed by statisticians, so R is a natural fit for researchers in statistics
  - Other languages such as Stan, Julia and sometimes Python are also used in these fields
- **Natural sciences**: R is used in many fields of the natural sciences, such as environmental sciences, biology, water resources, medical sciences, ecology, etc.
  - Life sciences are using R increasingly, replacing paid software like SPSS.
  - Other areas like physics, mathematics and chemistry rely on other software such as MATLAB & Mathematica.

Where R is being used in academia

- **Social science**: Paid solutions like Stata, SPSS, SAS and EViews are unfortunately still popular
  - Economics: Stata largely the leader, but R gaining traction very quickly.
  - Political science: code-based software like Stata still trying to overcome the click-and-point software, but some researchers starting to use open-source alternatives
  - Psychology: SPSS is still the leader, but R and Stata gain field

See more:

- 87% of social researchers say they want or need to learn R
- Why universities are switching to R for teaching social science

Where R is strong: lessons from usage in Economics

- The Academic Economics Scientific Computing Survey by Imola Csoka at JKU Linz finds that Stata largely leads in applied economic research, but R and other languages are gaining traction.
- The survey also finds that open source languages spark more interest in graduate students than respondents who have already completed their PhDs.

Where R is strong: lessons from usage in Economics

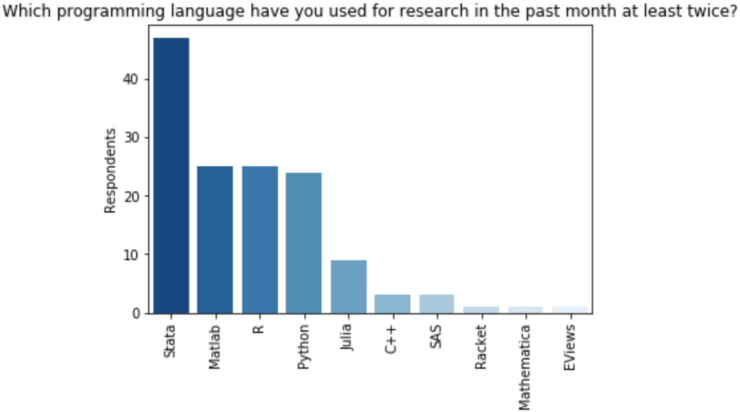


Figure 1: Software usage in academic economics

Where R is strong: lessons from usage in Economics

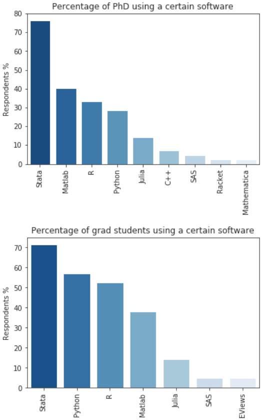
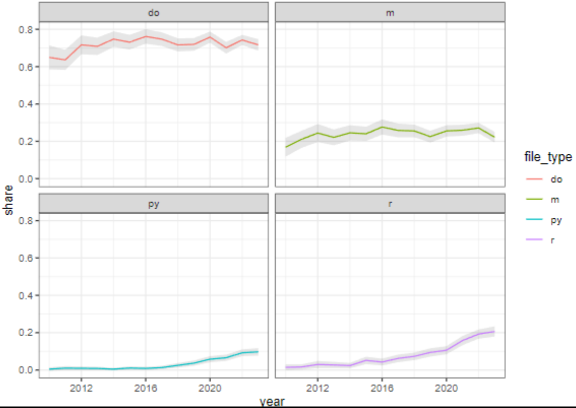


Figure 2: Softare usage by PhD status

Where R is strong: lessons from usage in Economics

- An analysis of replication files in [Sebastian Kranz'] replication database finds that both R and Python files have increased in the last years, yet are still a minority compared to Stata and Matlab.



Outside academia: R in industry

- The recent "data revolution" created the notion that R is in high demand in industry.
  - At best, this is a half-truth: R is in high demand in some industries, but not in all.
- R is very academic still, so industries that use it are those that are academia-adjacent
  - Health: R is used in large pharmaceutical companies, hospitals, and research institutions.
  - Insurance: R is used in some insurance companies, especially those that do actuarial work.

Outside academia: R in industry

- Government and non-profits: R is used in some government agencies and non-profits, especially those that do research or data analysis.
  - Governments employ many academics, so it is not surprising that they use R.
- Economics consulting firms are starting to use R more, but are still largely using very antiquated paid software (i.e. SPSS, EViews).
  - Legacy code, inflexible leadership and the priority of speed over quality keep private industry from innovating in workflows.
- Some industries like health and insurance are in regulatory capture, which is why paid software is still the norm.
  - SAS is only slowly being phased out, as government regulation requires pharmaceutical companies to provide analyses in "approved" tools.

Outside academia: R in industry

- For data science (i.e. machine learning), while R has wonderful capabilities, Python is still the norm.
    - Python is often called the second best language at everything.
    - Software engineers, the workforce most likely to be hired for data science, are more likely to know Python than R.
- Not a lot of people know how to work with R *in production*.
    - Companies need to develop and deploy models live, and it is hard to do this with R because of the economy of scale that Python has.
- For this, R doesn't guarantee professional success, but knowing how to program well will translate into an ease of learning many other languages.

Some thoughts for the future

- R is likely to overcome paid software in several fields in the next few years, but will probably do so slowly.
  - The main reason is that paid software is very entrenched in academia, and it is hard to change the status quo.
- Python is not so much a competitor to R for academic work, but it is a competitor for data science work.
  - Python is more versatile and better suited for production work.
  - Yet, it is not as good as R for data analysis and statistics.

Some thoughts for the future

- Julia is a language that may prove to be the next great thing in academic research, as it is as good as R for data analysis and statistics, and offers some of the production capabilities of Python.
  - It is superior to R in reproducibility and speed.
  - Also offers better possibilities for numerical computing, which R lacks.
  - The tidyverse is being implemented in Julia! Check out Tidier.jl
- Posit (formerly RStudio), is slowly moving to a more general-purpose data science platform, with the introduction of Quarto and their new IDE, Positron.
  - Posit is a company after all, and academia is a limited market.
- Keep learning, and keep an open mind. The worst enemy of a good researcher is arrogance.

Getting started with R

Installing R

- To install R, go to the CRAN website, and download the version of R that is appropriate for your operating system.
- Once you have installed R, you can run it by clicking on the R icon, or by typing `R` in the terminal.
    - The terminal in Windows is called the Command Prompt, and in macOS and Linux it is called the Terminal.
    - One can also use Git Bash in Windows to simulate the terminal in macOS and Linux.
- The base R "program" is often called the RGui or the R console.

RGui

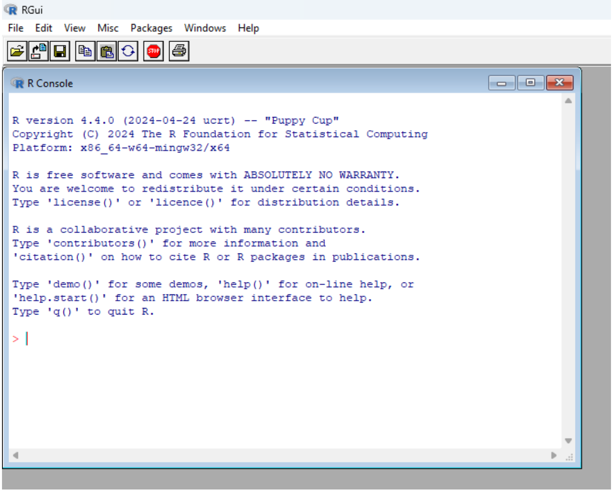


Figure 4: The RGui

RGui and RStudio

- We typically don't use the RGui for anything other than updating R.
    - It isn't very user-friendly, and doesn't have many features.
- Rather, we will download an IDE, which makes it easier to write and run R scripts.
    - RStudio is the most popular IDE for R, and is what we will use in this course.
    - As you become more adept at R, you may want to try other IDEs, like VS Code.
- You CANNOT run RStudio without having R installed on your computer.
    - RStudio is just a front-end for R, and needs R to be installed in order to work.

RStudio

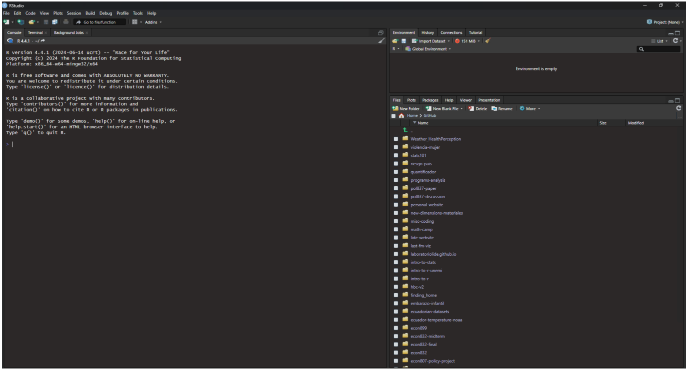


Figure 5: The RStudio IDE

You may see RStudio with a white background when you first open it. I have customized my RStudio to have a dark background. I suggest you do the same, as it is easier on the eyes.

Advantages of RStudio

- **Script editor**: A place to write and run R scripts.
  - Syntax highlighting, code completion, and other features to make writing code easier.
  - You can run code line-by-line, or all at once.
  - The console is integrated, so you can see the output of your code as you run it.
- **R Projects**: helps for an easy organization of your work, especially file paths.
- **Environment**: A place to see the objects in your workspace, and to manage them.

Advantages of RStudio

- **History**: A place to see the code that you have run in the past.
- **Files**: A place to see the files in your working directory, and to manage them.
- **Plots**: A place to see the plots that you have created.

Advantages of RStudio

- **Packages**: A place to see the packages that you have installed, and to manage them.
- Allows for **integrations** with other software, like Git, LaTeX/RMarkdown and Python.
- Very easy setup and **customization**.
    - VS Code can do the same, but requires much more setup.
- Many point-and-click features for those who don't like to write code
    - Code is provided for you, and you can run it by clicking a button, and later include it in your scripts.

RStudio Panes

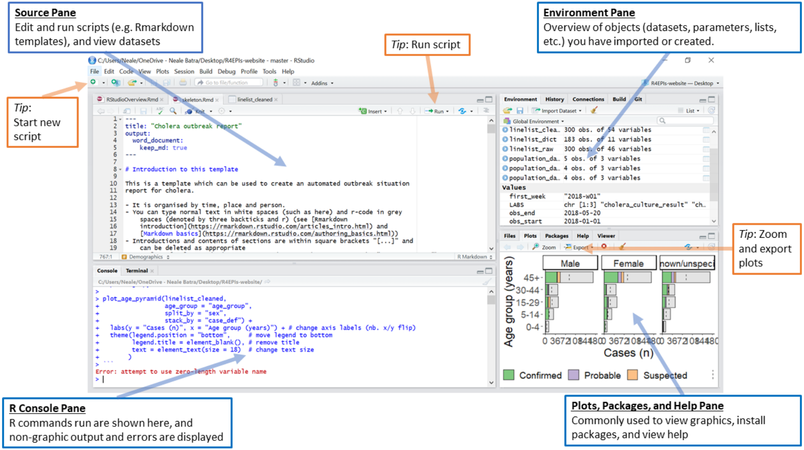


Figure 6: RStudio Panes

Posit Cloud

- Posit Cloud is a cloud-based version of RStudio.
  - It is free to use, and is a good option if you don't want to install R and RStudio on your computer.
  - It is also a good option if you want to collaborate with others on R projects.
- You will need to create an account to use Posit Cloud.
  - You can sign up for an account at cloud.rstudio.com.
- You can also use Posit Cloud to run R scripts on a server.
  - This is free to a certain extent, but you may need to pay for more resources if you use it a lot.

Datalab, formerly DataCamp Workspace

- Datalab is a cloud-based version of Jupyter Notebooks.
    - It is free to use to a certain extent, and is a good option if you don't want to install R and RStudio on your computer.
    - It is also a good option if you want to collaborate with others on R projects.
- Use a DataCamp account to access DataLab, and be sure to select R as the language you want to use (Python also available).
- Similar resources to run R online are available at repl.it.

Getting started: using the R Project

- When you open RStudio, it is a good idea to store your work in an R Project.
  - This will help you keep your work organized, and will make it easier to share your work with others.
- To create an R Project, click on **File** in the top menu, then click on **New Project**.
- You will be asked to choose a directory for your project.
  - Choose a directory where you want to store your project, and give your project a name.

Getting started: using the R Project

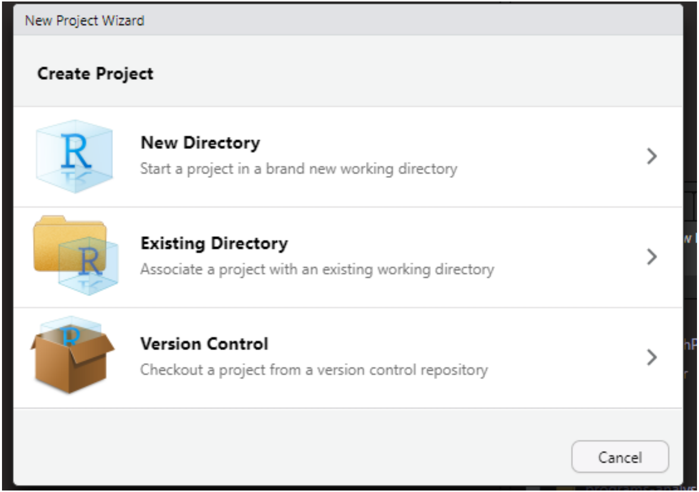


Figure 7: Creating an R Project

Getting started: using the R Project

- R Projects let you store your work in a single directory, and keep all of your files and data in one place.
  - This makes it easier to share your work with others, and to keep track of your work over time.
- Creating a project creates an `.Rproj` file in the project directory.
  - This file tells RStudio that the directory is an R Project, and allows RStudio to manage the project.
- When thinking about file paths, everything is relative to the project directory.
  - This means that you can move your project directory to another location, and everything will still work.

An analogy: RStudio as a kitchen

- The R console: the stove, where you cook your food.
    - The food needs to cooked in the stove, there is no way around this.
    - You may cook the food in the stove, remembering steps by heart, or you may use a **recipe/
- The script editor: the recipe book, where you write down the steps to cook your food.
    - You can write down the steps to cook your food, and then follow them to cook the food.
    - You can also share the recipe with others, so that they can cook the food too.
- The environment: the kitchen table, where you put your ingredients and intermediate outputs from the recipe.
    - You can see what ingredients you have, and what ingredients you need to buy.
    - You can also see how much of each ingredient you have.
    - As you cook intermediate steps, you may store the results in the kitchen table.

An analogy: RStudio as a kitchen

- The file viewer: the pantry, where you store your ingredients and recipes.
  - You can see what ingredients you have, and what ingredients you need to buy.
  - You can also see what recipes you have, and what recipes you need to buy.
  - As you cook, you take ingredients from the pantry (files) and put them in the kitchen table (environment).
- Libraries: kitchen utensils which you need to use to cook your food.
  - You may have a lot of kitchen utensils, but you only use the ones you need to cook your food.
  - In principle you can cook with just a stove, but it is much easier to cook with the right utensils.
  - If you see a recipe that requires a kitchen utensil you don't have, you can buy it or borrow it from someone else (install a package).

R packages

- R packages are libraries of functions that extend the capabilities of R.
  - They are written by expert programmers, and are shared with the community.
  - They are published on CRAN, GitHub, or other repositories.
  - They are maintained by the package developers, who fix bugs and add features as needed.
- To install an R package, you can use the `install.packages()` function.
- Good practice is to install packages in the console, not write it in the script.
  - This is because you only need to install a package once, but you may need to run a script many times.
  - This makes code more readable and fast.

Installing packages

- To install a package, you can use the `install.packages()` function.

```
install.packages("dplyr")
install.packages("ggplot2")
install.packages("babynames")
```

- You may also install from the Packages pane in the bottom right corner of RStudio.

Loading packages

- To load a package, you can use the `library()` function.
- This is like taking the kitchen utensils out of the drawer and putting them on the kitchen table.
- You don't go to the store every time you need a kitchen utensil, you just take it out of the drawer, right?

```
library(dplyr)
library(ggplot2)
library(babynames)
```

Error, warning and message messages

- When you run code in R, you may see three types of messages: errors, warnings, and messages.
- **Errors**: These are messages that tell you that something went wrong.
  - You need to fix the error before you can continue running the code.
  - Errors are usually in red text, and are followed by a message that tells you what went wrong.
- **Warnings**: These are messages that tell you that something might be wrong.
  - You can usually continue running the code, but you should be aware that something might be wrong.
  - Warnings are usually in yellow text, and are followed by a message that tells you what might be wrong.

Package conflicts

- A typical warning message is that a package was built under a different version of R.
  - This is usually not a problem, but it may cause conflicts with other packages.
  - If you see this warning, you can usually ignore it, but you should be aware that there may be conflicts.
- Another typical warning message is a function mask.
  - This means that you have loaded a package that has a function with the same name as a function in another package.
  - This can cause conflicts, and you should be aware of it.

Package conflicts

- In order to get around this, people use the `::` operator to specify which package's function they want to use.
    - For instance, `dplyr::filter()` specifies that you want to use the `filter()` function from the `dplyr` package.
    - This is good practice, as it makes your code more readable and less error-prone when there are conflicts.
    - We also use the `::` operator to call functions from packages that we haven't loaded, and we may not want to load since we only need one function from them.

Dealing with file paths

- Whenever you work with files in R, you need to think about file paths.
  - A file path is the location of a file on your computer.
  - File paths can be absolute or relative.
  - Absolute file paths start at the root directory of your computer.
  - Relative file paths start at the working directory of your R session.
- Absolute file paths are those that start at the root directory of your computer.
  - They are usually very long, and are hard to read and write.
  - They are also hard to share with others, as they are specific to your computer.

```
C:/Users/daniel/Documents/MyProject/data.csv
```

Dealing with file paths

- Relative file paths are those that start at a specific directory on your computer.
  - They are usually shorter, and are easier to read and write.
  - They are also easier to share with others, as they are not specific to your computer.

```
data.csv
```

- Relative paths may cause problems when trying to run code from other computers.
  - Though, we've seen that R Projects help with this!

Dealing with file paths

- To set the working directory in R, you can use the `setwd()` function.
    - This becomes unnecessary when using R Projects, but it is good to know.
    - `setwd()` takes a single argument, which is the path to the directory you want to set as the working directory.
    - Similar to `cd` in the terminal.
- There are packages that can help you work with file paths in R.
    - The `here` package is a good option, as it allows you to work with file paths in a platform-independent way.

Good practices, pt. 1

- **Comment your code**: Write comments in your code to explain what you are doing.
  - Comments start with a # symbol, and are ignored by R.
  - Comments should explain what you are doing, not how you are doing it.
  - Comments should be concise and to the point.
- There is no undo button in R.
  - This means that if you do something to a file you didn't want to do, you can't just press `Ctrl+Z` to undo it.
  - Scripts should be a "beginning-to-end" process, so that you can run them from the beginning to the end to reproduce your results.
  - Do not overwrite files: if you need to change a file, save it with a different name. E.g. (`data_raw.csv`, `data_clean.csv`).
- Scripts should be organized in a logical way.
  - Ctrl+Shift+R in RStudio will create script sections, which can be collapsed.
  - You can always go back to a script and add more code to the relevant sections.

Getting to know R's quirks

- Unlike Python, R does not require a print statement to print output to the console.
    - This means that you can just write `1 + 1` in the console, and R will print 2 to the console.
    - This is because R is a functional programming language, and functions return values.
- As all programming languages, R is case-sensitive.
    - This means that `a` is not the same as `A`, and `data` is not the same as `Data`.
    - If you write the function wrong, you'll get an error.
- No assignation of variables means losing the output.
    - If you write `a <- 1 + 1`, R will print 2 to the console, but it will not store the value of 2 in the variable `a`.
    - You need to write `a <- 1 + 1` to store the value of 2 in the variable `a`.

Working with R in RStudio

Creating variables

- To create a variable or object in R, you can use the <- operator.
    - This is called the assignment operator, and is used to assign a value to a variable.
    - The variable name goes on the left side of the assignment operator, and the value goes on the right side.
    - The value can be a number, a character string, a logical value, or a vector.
    - The variable name can be any combination of letters, numbers, and underscores, but cannot start with a number.

Other assigners

- Some people use the = operator to assign variables in R.
  - This is called the equals operator, and is used to assign a value to a variable.
  - The = operator is less common than the <- operator, but is still used by some people.
  - Best practice is to use the <- operator, as it is more common and easier to read.
- The shortcut for <- is `Alt` + - in RStudio.

Other assigners

- You can assign the other way around:

```r
1 + 1 -> a

a
```

```
[1] 2
```

```r
# Don't do this, it is less readable!!

b = 1 + 1

b
```

```
[1] 2
```

Classes of objects in R

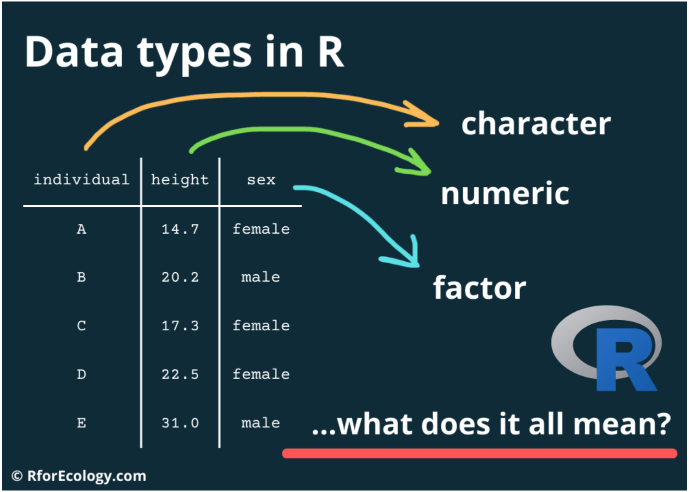


Figure 8: Classes of objects in R

Types of structures in R

- **Vectors**: A single column of data.
  - Can be numeric, character, or logical, and can mix types.
  - Can be of any length.
  - Can be created with the `c()` function (means concatenate).
- **Matrices**: A two-dimensional array of data.
  - Can be numeric, character, or logical, but all elements must be of the same type.
  - Can be of any size.
  - Can be created with the `matrix()` function.

Types of structures in R

- **Data frames**: A two-dimensional array of data, where each column can be a different type.
- **Lists**: A collection of objects, where each object can be of a different type.
    - Can be of any length.
    - Can be created with the `list()` function.

Structure of objects in R

- To see the structure of an object in R, you can use the `str()` function.
  - This function shows you the class of the object, and the structure of the object.
  - The structure of an object is the way that the object is organized in memory.
  - The structure of an object is important, as it determines how you can work with the object.

Vectors

- Single dimension of data.
  - Can be numeric, character, or logical.
  - Can mix types.
  - Can be of any length.
- Can be created with the `c()` function, indexing or functions that return vectors as output.
- We may access to specific elements of the vector using the [] operator (indexing).

Vector indexing

- To access an element of a vector, you can use the [] operator.
  - This operator takes a single argument, which is the index of the element you want to access.
  - The index can be a single number, a vector of numbers, or a logical vector.
  - The index can be positive, negative, or zero.
- Positive indices access elements from the beginning of the vector.
- Negative indices access elements from the end of the vector.
- Zero indices return an empty vector (R is 1-indexed, unlike Python).

Logical operators

- **Comparison operators**: These operators compare two values, and return a logical value.
    - ==: Equal to.
    - `!=`: Not equal to.
    - >: Greater than.
    - <: Less than.
    - >=: Greater than or equal to.
    - <=: Less than or equal to.
- When running a comparison, R will return `TRUE` if the comparison is true, and `FALSE` if the comparison is false.
    - One can also write `T` and `F` instead of `TRUE` and `FALSE`.
    - 1 is often used as `TRUE`, and 0 as `FALSE`, R will convert these to `TRUE` and `FALSE` respectively when needed.

Logical operators

- **Logical operators**: These operators combine two logical values, and return a logical value.
  - `&`: Logical AND.
  - `|`: Logical OR.
  - `!`: Logical NOT.
- The AND operator checks if both values are `TRUE`, and returns `TRUE` if they are.
- The OR operator checks if at least one value is `TRUE`, and returns `TRUE` if it is.
- The NOT operator checks if the value is `TRUE`, and returns `FALSE` if it is.

Working with functions

- A function in R is a block of code that performs a specific task.
    - Functions take arguments, which are the inputs to the function.
    - Functions return values, which are the outputs of the function.
    - Functions can be built-in, or user-defined.
- When we “write” the code of a function to invoke it, we say that we are “calling” the function.
    - Hence, you will hear call and invoke used interchangeably.

Working with functions

- The general form of a function in R is 'function_name(argument1, optional_argument1, optional_argument2, ...)
  - The function name is the name of the function.
  - The arguments are the inputs to the function.
  - The optional arguments are the inputs to the function that are not required, as they have default or predefined values.

Getting help with R

- To get help with R, you can use the `help()` function.
  - This function takes a single argument, which is the name of the function you want help with.
  - The help file for the function will be displayed in the Help pane in RStudio.
  - You can also use the `?` operator to get help with a function.
  - Typically, it will show you the arguments of the function, and some examples of how to use it, as well as predefined values for optional arguments.
- To get help with a package, you can use the `help(package = "package_name")` function.
- To get help with a topic, you can use the `help.search()` function.

Getting help with R

- You can use many outside resources to get help with R.
  - The R documentation is a good place to start.
  - The RStudio Community is a good place to ask questions.
  - The Stack Overflow is a good place to ask questions.
  - The R-bloggers is a good place to find tutorials and examples.
- Reddit has plenty of R communities, such as r/rstats where basic questions are welcome (as Stack Overflow is a bit more strict).
- The RStudio Cheat Sheets are a good resource for learning R.
- Using Slack, Discord, or other chat platforms to ask questions is also a good idea
  - Never hesitate to ask questions, as it is the best way to learn.
  - Use the quote marks in Slack to write code, so that it is easier to read when asking questions. “‘”

The LLM scene in R

- Of course, R can be learned by asking one of the new chatbots that are available.
  - ChatGPT codes well in R.
  - Gemini is a good option for R help, and recently added integration with RStudio.
  - Claude is a good option for R help, and is free to use.
  - Copilot is a good option for R help, and is free to use.
- GitHub Copilot is free for students and teachers, integrating amazingly with RStudio.
  - Check out the GitHub Copilot quickstart guide
  - Predicts code and comments, and can be used to write entire scripts.
- Learning R can be done by using AI, but it is better to learn the basics first, and then use AI to help with more complex tasks.

Dataframes in R

Ol' reliable: the dataframe

- The single most important object for an empirical researcher in R is the dataframe.
  - A dataframe is a two-dimensional array of data, where each column can be a different type.
  - Dataframes are used to store data in R, and are the primary object used in data analysis.
  - Dataframes can be created with the `data.frame()` function, or by importing data from a file.
- Think of the dataframe as a spreadsheet, where each column is a variable, and each row is an observation.

Creating dataframes

- To create a dataframe in R, you can use the `data.frame()` function.
  - This function takes a series of arguments, which are the columns of the dataframe.
  - The columns can be vectors, matrices, or other dataframes.
  - The columns must be the same length, or R will recycle the shorter columns to make them the same length.
- You can also create a dataframe by importing data from a file.
  - This is the most common way to create a dataframe in R, as it allows you to work with real data.

Creating a dataframe from vectors

```r
# Create vectors

name <- c("Alice", "Bob", "Charlie", "David", "Eve")
age <- c(25, 30, 35, 40, 45)
height <- c(1.60, 1.70, 1.80, 1.90, 2.00)

# Create dataframe

df <- data.frame(name, age, height)
```

Working with dataframes

- There are some key functions for working with dataframes in R.
    - `head()`: Shows the first few rows of the dataframe.
    - `tail()`: Shows the last few rows of the dataframe.
    - `str()`: Shows the structure of the dataframe.
    - `summary()`: Shows a summary of the dataframe.
- For accesing information within the dataframe, we may work with the following functions:
    - `nrow()`: Shows the number of rows in the dataframe.
    - `ncol()`: Shows the number of columns in the dataframe.
    - `dim()`: Shows the dimensions of the dataframe.
    - `names()`: Shows the names of the columns in the dataframe.
    - `colnames()`: Shows the names of the columns in the dataframe.
    - `rownames()`: Shows the names of the rows in the dataframe.
    - `unique()`: Shows the unique values in a column of the dataframe.
    - `table()`: Shows the frequency of values in a column of the dataframe.

Working with dataframes

- Indexing also works for dataframes, we need to mantain the two-dimensional structure of the dataframe when using indices.
- For example:
  - `df[1, 2]` will return the value in the first row and second column of the dataframe.
  - `df[1, ]` will return the first row of the dataframe.
  - `df[, 2]` will return the second column of the dataframe.
  - `df[1:3, ]` will return the first three rows of the dataframe.
  - `df[, c(1, 3)]` will return the first and third columns of the dataframe.

Working with dataframes

- We may also use logical operators to subset the dataframe.
  - `df[df$age > 30, ]` will return the rows of the dataframe where the age is greater than 30.
  - `df[df$age > 30 & df$height > 1.80, ]` will return the rows of the dataframe where the age is greater than 30 and the height is greater than 1.80.

Working with dataframes

- If we need to access a column by name, we can use the $ operator.
  - `df$name` will return the column of the dataframe with the name `name`.
  - `df$age` will return the column of the dataframe with the name `age`.
  - `df$height` will return the column of the dataframe with the name `height`.
  - This is simply a shortcut for `df[, "name"]`, `df[, "age"]`, and `df[, "height"]`.
- We can also use the `subset()` function to subset the dataframe.
  - `subset(df, age > 30)` will return the rows of the dataframe where the age is greater than 30.
  - `subset(df, age > 30 & height > 1.80)` will return the rows of the dataframe where the age is greater than 30 and the height is greater than 1.80.
  - Again, this is a base R shorthand for `df[df$age > 30, ]` and `df[df$age > 30 & df$height > 1.80, ]`.

Working with dataframes

- R does not allow for numerically named columns, so we need to use the apostrophe character to access them.
  - `df$3will return the column of the dataframe with the name3`.
- In general, any "invalid" name (e.g. starting with a number, having spaces, etc.) can be accessed with the ‘ character.
  - Good practice is to avoid these names, as they can be hard to work with.
  - The best names are lowercase, with underscores separating words, and no spaces or special characters.

Working with dataframes

- The infamous `attach()` function can be used to attach the dataframe to the search path.
    - This means that you can access the columns of the dataframe without using the $ operator.
    - This is not recommended, as it can lead to confusion and errors.
    - It is better to use the $ operator, as it is more explicit and less error-prone.
- The `detach()` function can be used to detach the dataframe from the search path.
- `with()` is a better alternative to `attach()`, as it allows you to access the columns of the dataframe without using the $ operator, but only within the `with()` function.
    - Generally, it is better to just stick to the $ operator.
    - We will later use tidyverse functions to access columns without the $ operator.

The `mtcars` dataset

- The `mtcars` dataset is a built-in dataset in R.
    - It is often used in examples and tutorials, as it is small and easy to work with.
    - For this, it's a good idea to get used to it
- It contains data on 32 cars, and 11 variables.

The `mtcars` dataset

- The variables are:
  - `mpg`: Miles per gallon.
  - `cyl`: Number of cylinders.
  - `disp`: Displacement (cubic inches).
  - `hp`: Gross horsepower.
  - `drat`: Rear axle ratio.
  - `wt`: Weight (1000 lbs).
  - `qsec`: 1/4 mile time.
  - `vs`: V/S.
  - `am`: Transmission (0 = automatic, 1 = manual).
  - `gear`: Number of forward gears.
  - `carb`: Number of carburetors.
- In Stata, the `auto` dataset is often used in the same way as the `mtcars` dataset in R.

The mtcars dataset

- Access the `mtcars` dataset by typing `mtcars` in the console. No need to load it, as it is built-in.

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3
```

Importing data from a file

- We will later learn the specifics about importing files into R as dataframes.
  - This is the most common way to work with dataframes in R.
- We can use base R functions to load the used car sales data in the `data` directory of the GitHub repo.

```r
# Load the data using read.csv, the base R function for loading

cars <- read.csv("data/used_cars_ecuador.csv")

str(cars)
```

One object to rule them all: the list

- A list is a collection of objects, where each object can be of a different type.
  - Lists can be of any length, and can contain any type of object.
  - Lists are used to store objects in R, and are the primary object used in programming.
  - Lists can be created with the `list()` function, or by combining objects with the `c()` function.
- Think of the list as a box, where each object is a different type of object, and each object can be accessed by its position in the box.

Creating lists

- To create a list in R, you can use the `list()` function.
    - This function takes a series of arguments, which are the objects in the list.
    - The objects can be vectors, matrices, dataframes, or other lists.
    - The objects can be of any type, and can be of any length.
- You can also create a list by combining objects with the `c()` function.

Creating lists

```r
name <- c("Alice", "Bob", "Charlie", "David", "Eve")

age <- matrix(c(25, 30, 35, 40, 45), nrow = 5, ncol = 1)

height <- data.frame(height = c(1.60, 1.70, 1.80, 1.90, 2.00))

# Create list

lst <- list(name, age, height)

str(lst)
```

```
List of 3
 $ : chr [1:5] "Alice" "Bob" "Charlie" "David" ...
 $ : num [1:5, 1] 25 30 35 40 45
 $ :'data.frame':   5 obs. of  1 variable:
  ..$ height: num [1:5] 1.6 1.7 1.8 1.9 2
```

Working with lists

- There are some key functions for working with lists in R.
    - `length()`: Shows the number of objects in the list.
    - `names()`: Shows the names of the objects in the list.
    - `str()`: Shows the structure of the list.
    - `summary()`: Shows a summary of the list.
- For accesing information within the list, we may work with `[[`.
    - Double brackets `[[` are used to access the elements of the list.
- Indexing by list works similarly to indexing by dataframe, but we need to use the `[[` operator to access elements of the list.

Working with lists

- For example:
  - `lst[[1]]` will return the first element of the list.
  - `lst[[2]]` will return the second element of the list.
  - `lst[[3]]` will return the third element of the list.
  - `lst[[1]][1]` will return the first element of the first element of the list.
  - `lst[[2]][1, 1]` will return the first element of the first element of the second element of the list.
  - `lst[[3]][1, 1]` will return the first element of the first element of the third element of the list.

Working with lists

- We can also use the $ operator to access elements of the list.
  - `lst$name` will return the element of the list with the name `name`.
  - `lst$age` will return the element of the list with the name `age`.
  - `lst$height` will return the element of the list with the name `height`.
  - This is simply a shortcut for `lst[[1]]`, `lst[[2]]`, and `lst[[3]]`.

Working with lists

```
lst[[1]]
```

```
[1] "Alice"   "Bob"     "Charlie" "David"   "Eve"
```

```
lst[[2]]
```

```
     [,1]
[1,]   25
[2,]   30
[3,]   35
[4,]   40
[5,]   45
```

Working with lists

```r
lst[[3]]
```

```
  height
1    1.6
2    1.7
3    1.8
4    1.9
5    2.0
```

```r
lst[[3]]$height
```

```
[1] 1.6 1.7 1.8 1.9 2.0
```