Day 1_session_01

Getting started with R and R-Studio

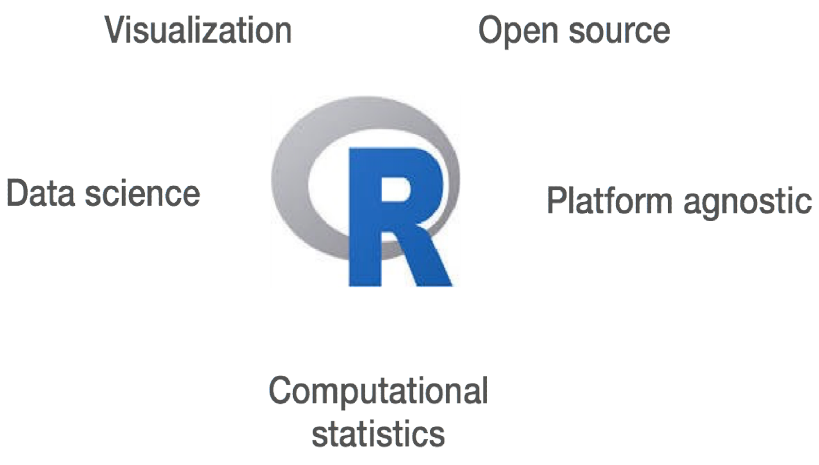
Module I:

Getting familiarity with the R language and R-Studio

- What R is and how it works
- R and R-Studio installation and interfaces
- Creating an R-Scriptsfile and new project directory
- Organizing your working directory & setting up R-Studio
- Interacting with R on R-Studio

1.1 What is R?

R is statistical software. R should not consider as a conventional programming language like C, C++, Python, Julia, Rust,Go, or Java, however, the R-software is as powerful as Python, MATLAB, and Julia. It was not created by software engineers for software development. Instead, it was developed by statisticians as an interactive environment for data analysis. The interactivity is an indispensable feature in data science because, as you will soon learn, the ability to quickly explore data is a necessity for success in this field.R is an interpreted language, not a compiled one, meaning that all commands typed on the keyboard are directly executed without requiring to build a complete program (this is like Python and unlike C, Fortran, Pascal, etc.). In R and R-studio (**Posit**is the current name), you can save your work as R-scripts (.R file) that can be easily executed at any moment. These R-scripts serve as a record of the analysis you performed, a key feature that facilitates reproducible work and open source (data) science research. If you are patient, you will come to appreciate the unequal power of R when it comes to data analysis, econometric and ML model estimation and, specifically, data visualization and presentation.



Visualization

Open source

Data science

Platform agnostic

Computational statistics

Why R?

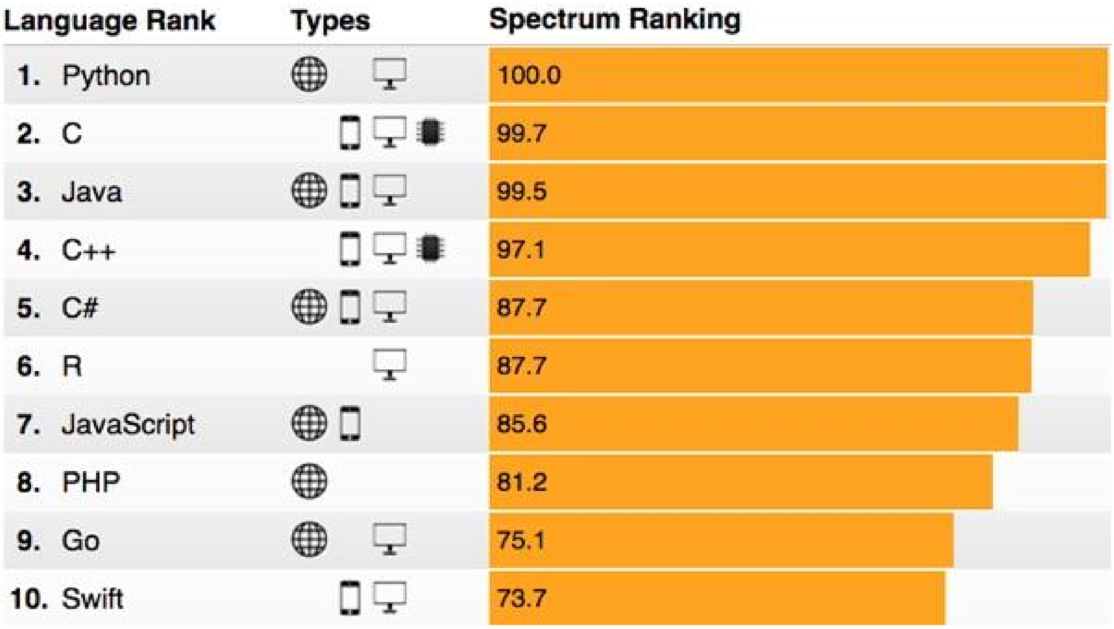
- The most popular software for data analysis
- Extremely flexible: can be used to manipulate, analyze, and visualize any kind of data
- Cutting edge statistical tools
- Publication quality graphics
- 20,000+ add on packages covering all aspects of statistics and machine learning
- Active community of users

Some attractive features of R relative to proprietary software STATA, EVIEWS, and MATLAB are:

- R is an open source and free to install, use, update, clone, modify, redistribute
- R can handle complex and large data
- R has technical merits in data science/machine learning
- Getting help from the R community is super easy
- R is popular and the standard language of choice for academics/econometrcians
- R is popular with employers
- Scripts and data objects can be shared seamlessly across platforms.
- There is a large, growing, and active community of R users and, as a result, there are numerous resources for learning and asking questions.

It is easy for others to contribute add-ons which enables developers to share software implementations of new data science methodologies. This gives R users early access to the latest methods and to tools which are developed for a wide variety of disciplines, including economics, finance, ecology, molecular biology, social sciences, and geography, just to name a few examples.

Below is a list of the most popular and best programming languages that will be in demand in 2023.



Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥	100.0
2. C	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	99.5
4. C++	📱 🖥 ▪	97.1
5. C#	🌐 📱 🖥	87.7
6. R	🖥	87.7
7. JavaScript	🌐 📱	85.6
8. PHP	🌐	81.2
9. Go	🌐 🖥	75.1
10. Swift	📱 🖥	73.7

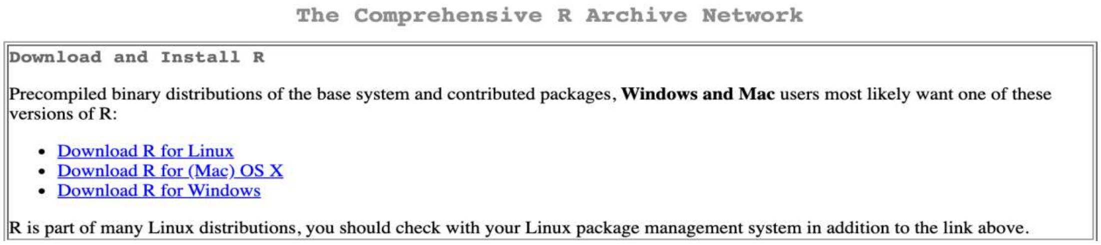
Fig. IEEE spectrum comes with the ranking sheet of Top 10 Programming Languages 2023 for Successful Development. Source: https://blog.sagipl.com/top-programming-languages/

Day1_Session1: Module II

In this tutorial we'll learn how to begin programming with R using RStudio. We'll install R, and RStudio R-Studio, an extremely popular development environment for R. We'll learn the key RStudio features in order to start programming in R on our own.

1. Install R

R is available to download from the official R website (CRAN-comprehensive R archive network). Look for this section of the web page:



The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

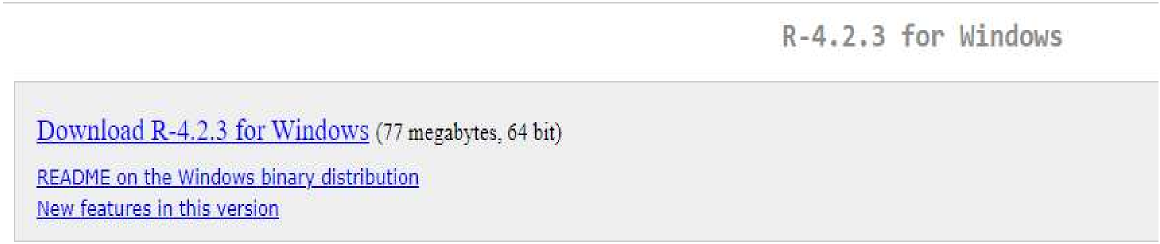
- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Search R download for windows/mac/Linux on your browser

Pick one of the links associated with your operating system

Click on the (**Download R 4.2.3 for windows** if you are windows user) and Download R for your system



R-4.2.3 for Windows

Download R-4.2.3 for Windows (77 megabytes, 64 bit)

README on the Windows binary distribution

New features in this version

Follow the typical downloading and installation prompts associated with your computer system

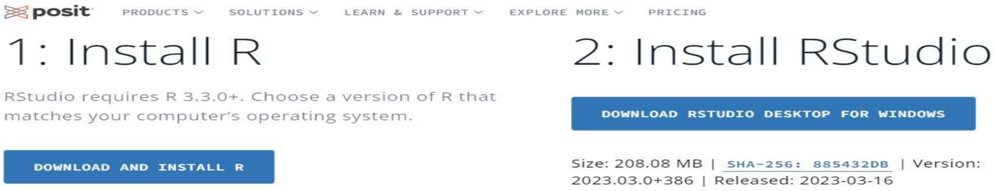
2. Install RStudio

Already have R the program on your computer, if you don't do that first.

Now that R is installed, we can install RStudio. Search R-studio download on google. Navigate to the RStudio downloads page. When we reach the RStudio downloads page, let's click the "Download" button of the RStudio Desktop Open-Source License Free option:

Go to: https://posit.co/download/rstudio-desktop/

In the right hand side click download



1: Install R

RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

DOWNLOAD AND INSTALL R

2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR WINDOWS

Size: 208.08 MB | SHA-256: 885432DB | Version: 2023.03.0+386 | Released: 2023-03-16

Pick the free version. Free version has all of the tools you will need for this course and most likely all of the classes you will take in Undergrad, and research in Honors, Masters, PhD, PostDoc, etc. Your operating system is usually detected automatically and so we can directly download the correct version for our computer by clicking the “Download RStudio” button. Follow your computer prompts to download and install.

Text editors, IDEs, & Notebooks

There are different ways of interacting with R. The two main ways are through:

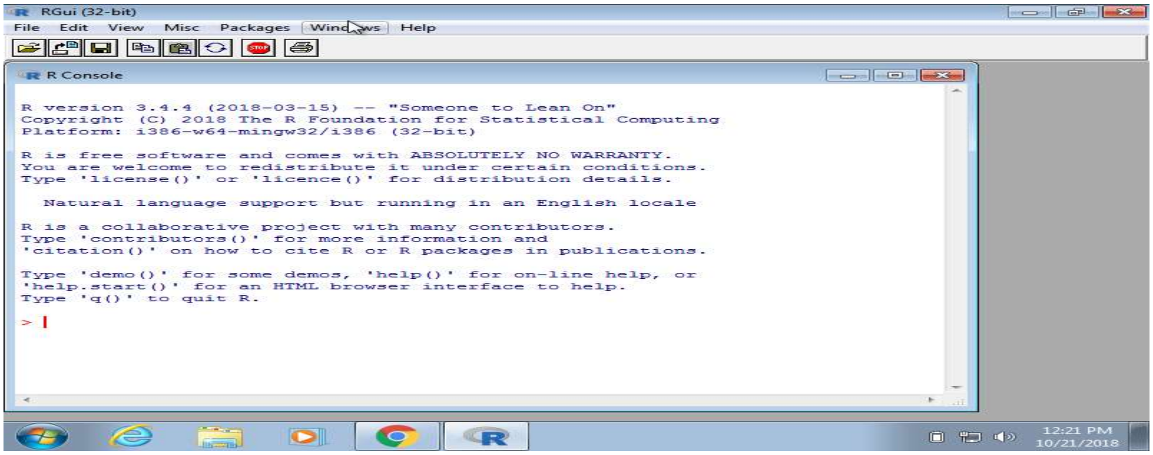
Text editors or **Integrated Development Environments (IDEs):** Text editors and IDEs are not really separate categories; as you add features to a text editor it becomes more like an IDE. Some editors/IDEs are language-specific while others are general purpose — typically providing language support via plugins. For these workshops we will use R-Studio; it is a good R-specific IDE with many useful features.

Here are a few popular editors/IDEs that can be used with R:

Editor / IDE	Features	Ease of use	Language support
R-Studio	Excellent	Easy	R only
Jupyter Lab	Good	Easy	Excellent
VS code	Excellent	Easy	Very good

3. The first look at R console

Interactive data analysis usually occurs on the R console that executes commands as you type them. There are several ways to gain access to an R console. One way is to simply start R on your computer. The console looks something like this:



```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

As a quick example, try using the console to calculate a multiplication of 15 times 5:

```
15 * 5
```

```
#> [1] 75
```

Note that in this book, grey boxes are used to show R code typed into the R console. The symbol #> is used to denote what the R console outputs.

4. Getting Started with R-Studio

R-Studio is an open-source tool for programming in R. R-Studio is a flexible tool that helps you create readable analyses, and keeps your code, images, comments, and plots together in one place. It's worth knowing about the capabilities of R-Studio for data analysis and programming in R.

4.1. First Look at R-Studio

The panes/panels

When you start RStudio for the first time, you will see three panes. The left pane shows the R console. On the right, the top pane includes tabs such as Environment and History, while the bottom pane shows five tabs: File, Plots, Packages, Help, and Viewer (these tabs may change in new versions). You can click on each tab to move across the different features.

When we open RStudio, R is launched as well. A common mistake by new users is to open R instead of RStudio. To open RStudio, search for RStudio on the desktop, and pin the RStudio icon to the preferred location (e.g. Desktop or toolbar).

Console: where you can type commands and see output. The console is all you would see if you ran R in the command line without RStudio.

Environment/History: environment shows all active objects and history keeps track of all commands run in console

Files/Plots/Packages/Help

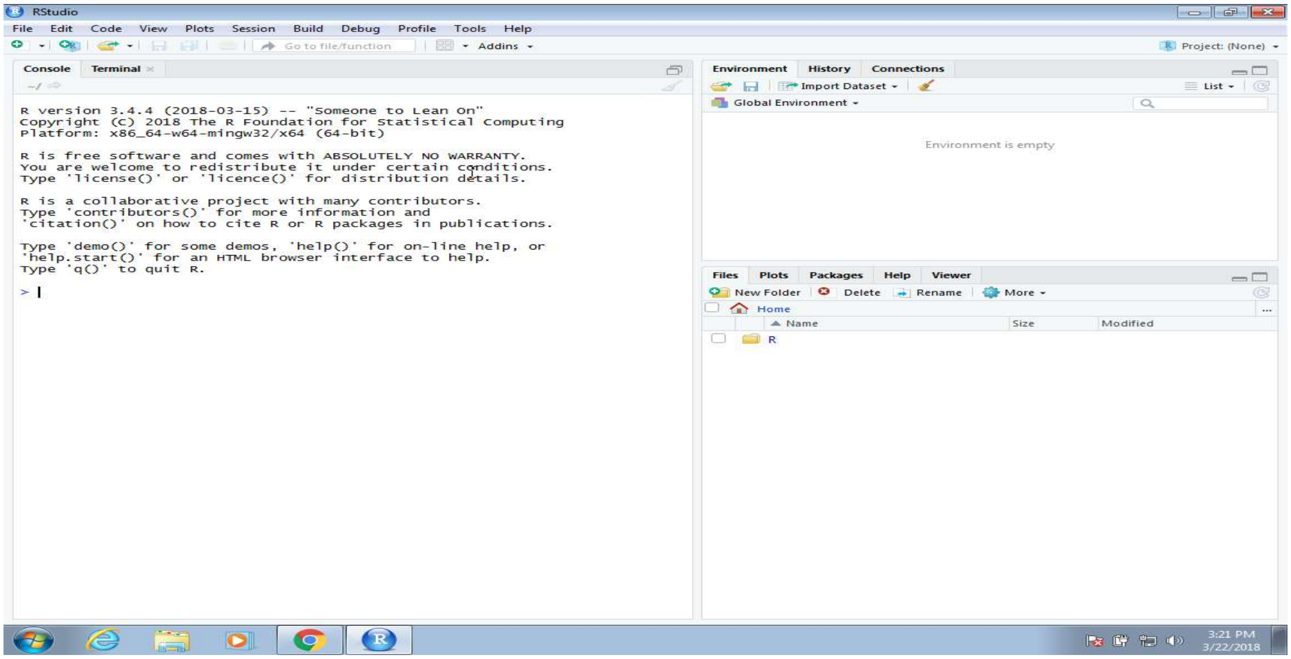
The Console

Let's start off by introducing some features of the Console. The Console is a tab in RStudio where we can run R code.Notice that the window pane where the console is located contains three tabs: Console, Terminal and Jobs (this may vary depending on the version of RStudio in use). We'll focus on the Console for now.

When we open RStudio, the console contains information about the version of R we're working with. Scroll down, and try typing a few expressions like this one. Press the enter key to see the result.

we can use the console to test code immediately. When we type an expression like 1 + 2, we'll see the output below after hitting the enter key.

When we open R-Studio for the first time, we'll probably see a layout like this:

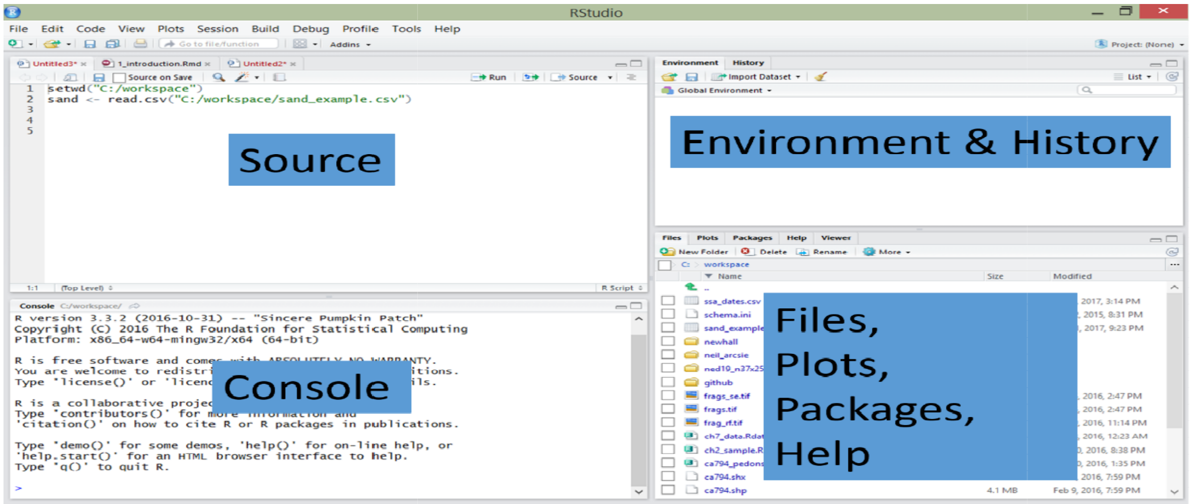


```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

The Global Environment

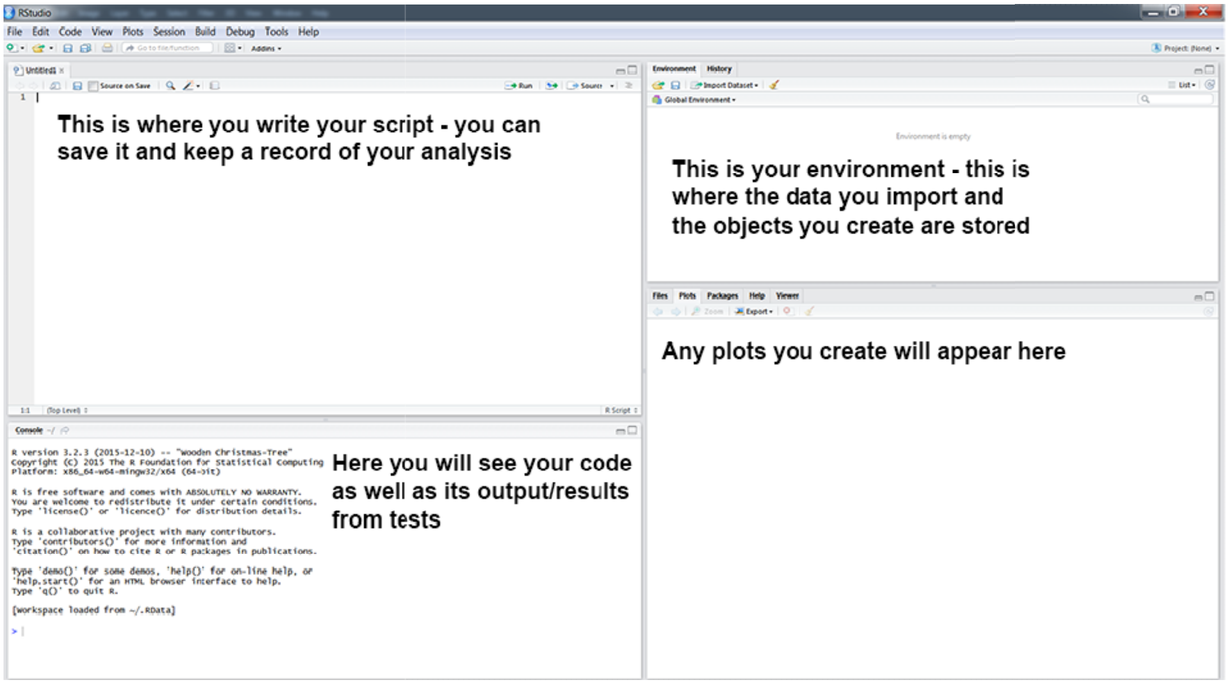
We can think of the global environment as our workspace. During a programming session in R, any variables we define, or data we import and save in a dataframe, are stored in our global environment. In RStudio, we can see the objects in our global environment in the Environment tab at the top right of the interface.

Sometimes, having too many named objects in the global environment creates confusion. Maybe we'd like to remove all or some of the objects. To remove all objects, click the broom icon at the top of the window. Finally, we can create an R-script (how? See the next section) and that will bring us to the following 4 panes:



```
setwd("C:/workspace")
sand <- read.csv("C:/workspace/sand_example.csv")
```

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and come
You are welcome to redistri                 itions.
Type 'license()' or 'licenc                 ils.

R is a collaborative projec
Type 'contributors()' for
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```



This is where you write your script - you can save it and keep a record of your analysis

This is your environment - this is where the data you import and the objects you create are stored

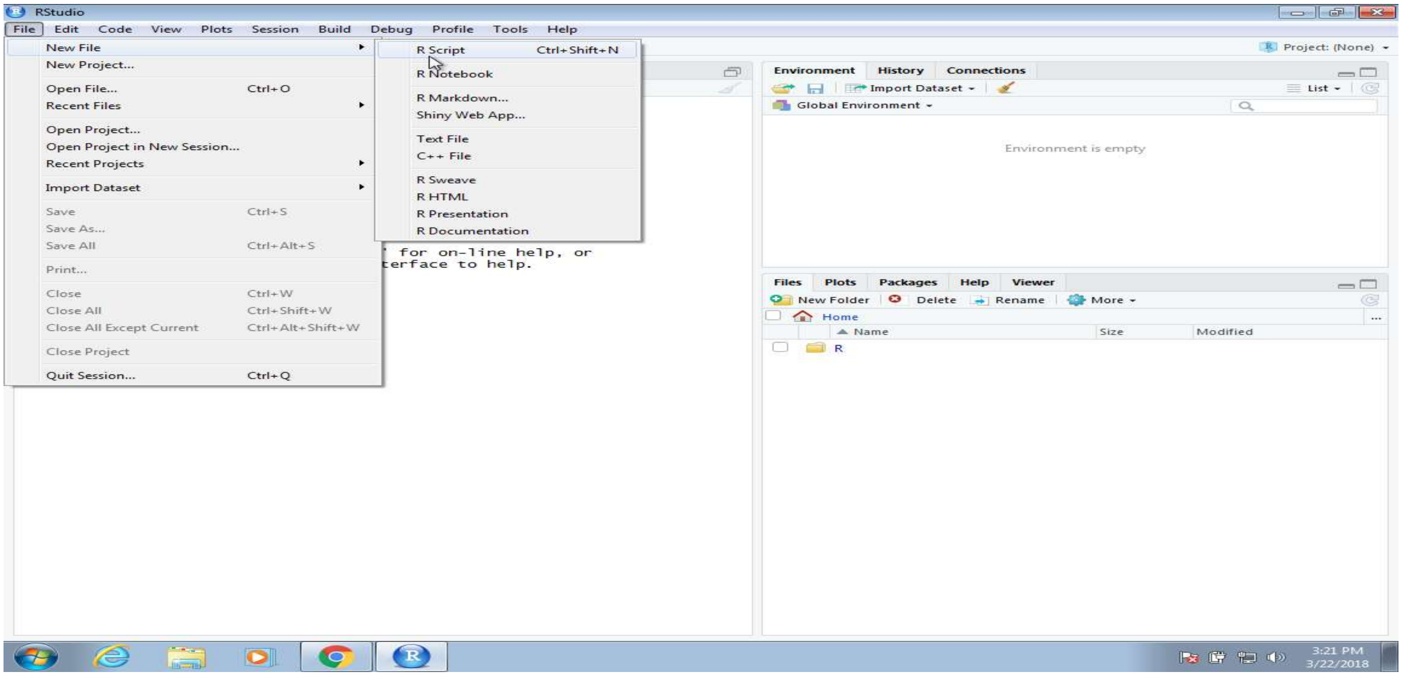
Any plots you create will appear here

Here you will see your code as well as its output/results from tests

```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> 
```

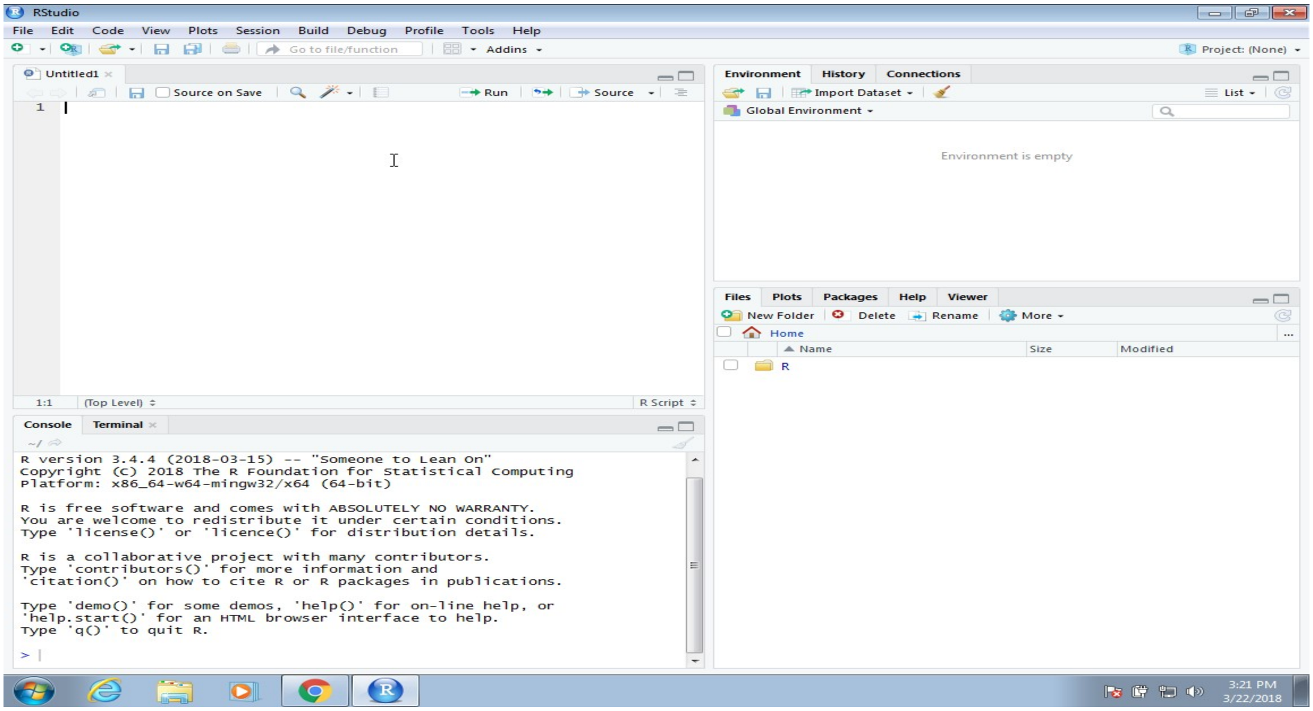
Scripts (R-Scripts)

One of the great advantages of R over point-and-click analysis software is that you can save your work as scripts. You can edit and save these scripts using a text editor. The material in this note was developed using the interactive integrated development environment (IDE) RStudio. RStudio includes an editor with many R specific features, a console to execute your code, and other useful panes, including one to show figures.To start a new script, you can click on File, then New File, then R Script.



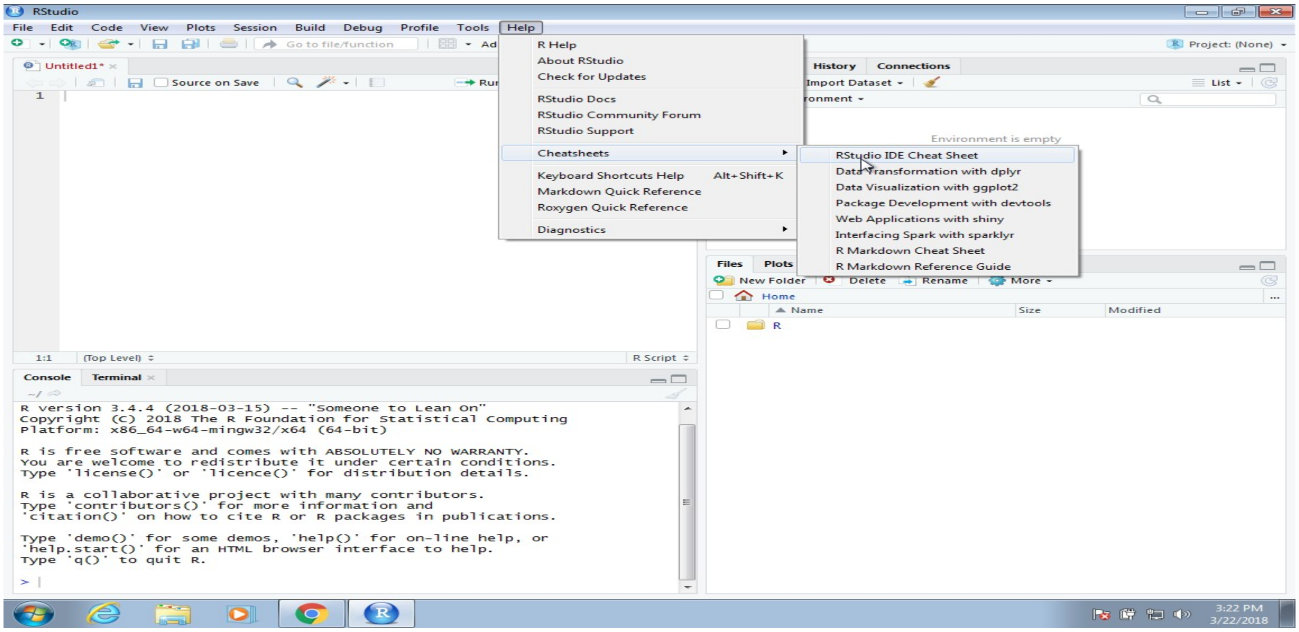
```
' for on-line help, or
erface to help.
```

This starts a new pane on the left and it is here where you can start writing your script.



```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

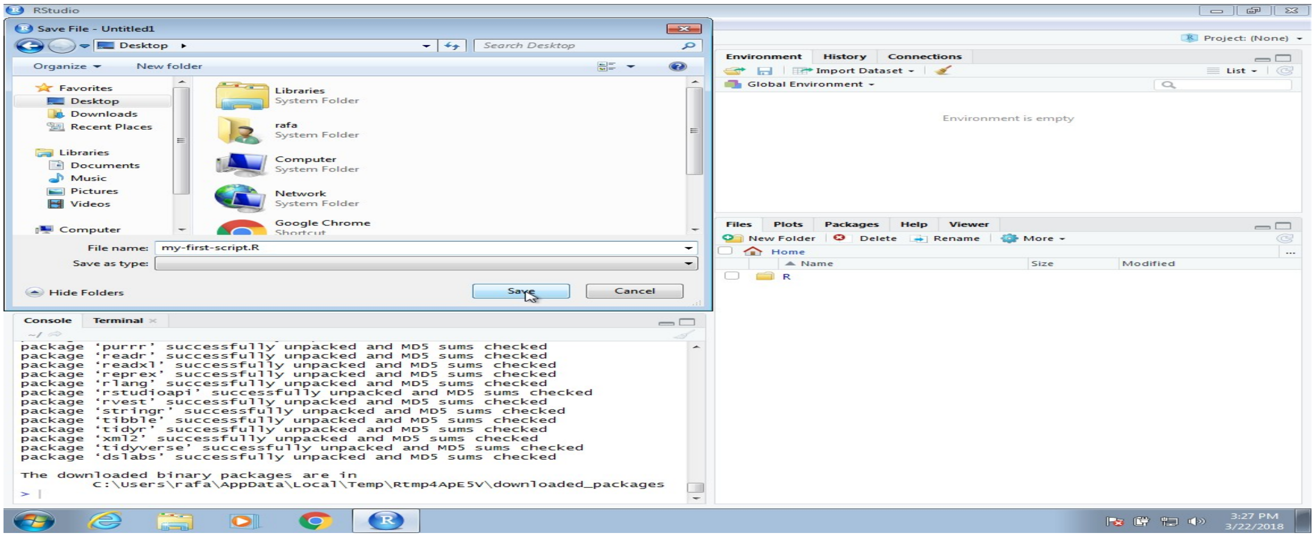
we just showed how to use the mouse to start a new script, but you can also use a key binding (key board short cut): Ctrl+Shift+N on Windows and command+shift+N on the Mac.Although in this tutorial we often show how to use the mouse, we highly recommend that you memorize key bindings for the operations you use most. RStudio provides a useful cheat sheet with the most widely used commands. You can get it from RStudio directly:



```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Running commands while editing scripts and saving you work file

Let’s start by opening a new script as we did before. A next step is to give the script a name. We can do this through the editor by saving the current new unnamed script. To do this, click on the save icon or use the key binding Ctrl+S on Windows and command+S on the Mac.When you ask for the document to be saved for the first time, RStudio will prompt you for a name. A good convention is to use a descriptive name, with lower case letters, no spaces, only hyphens to separate words, and then followed by the suffix .R. We will call this script **day1_session1**.R



File name: my-first-script.R

Save as type:

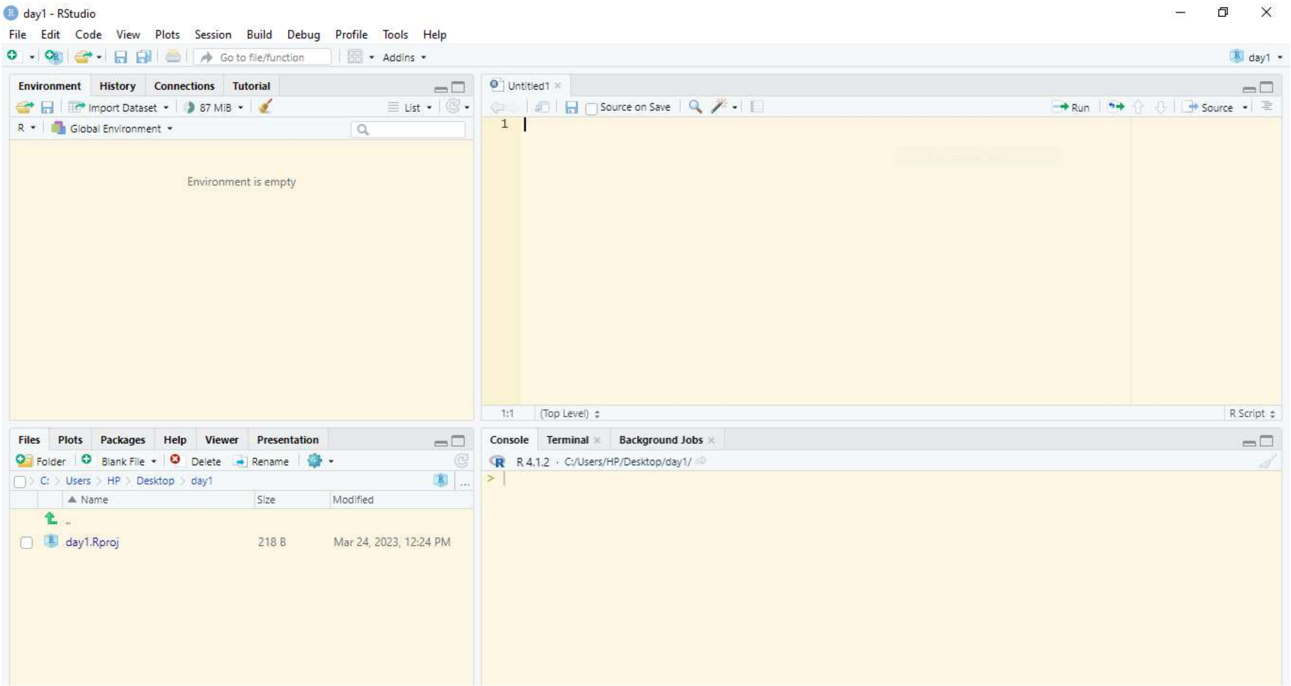
```
package ‘purrr’ successfully unpacked and MD5 sums checked
package ‘readr’ successfully unpacked and MD5 sums checked
package ‘readxl’ successfully unpacked and MD5 sums checked
package ‘reprex’ successfully unpacked and MD5 sums checked
package ‘rlang’ successfully unpacked and MD5 sums checked
package ‘rstudioapi’ successfully unpacked and MD5 sums checked
package ‘rvest’ successfully unpacked and MD5 sums checked
package ‘stringr’ successfully unpacked and MD5 sums checked
package ‘tibble’ successfully unpacked and MD5 sums checked
package ‘tidyr’ successfully unpacked and MD5 sums checked
package ‘xml2’ successfully unpacked and MD5 sums checked
package ‘tidyverse’ successfully unpacked and MD5 sums checked
package ‘dslabs’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\rafa\AppData\Local\Temp\Rtmp4ApE5V\downloaded_packages
> |
```

Now we are ready to start our coding and editing our first R-script.

Creating a new project directory in R-Studio

Let’s create a new project directory for our **day1_session1** today.

- Open R-Studio
- Go to the **File menu** and select **New Project**.
- In the New Project window, choose New Directory.
- Then, choose **New Project**. Name your new directory **day1** and then “**Create the project as subdirectory of:”** the Desktop (or location of your choice).
- Click on **Create Project**.
- After your project is completed, if the project does not automatically open in RStudio, then go to the File menu, select Open Project, and choose **day1.Rproj**.
- When RStudio opens, you will see three panels in the window.
- Go to the File menu and select New File, and select R Script. The R-Studio interface should now look like the screenshot below.


day1 - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
day1
Environment History Connections Tutorial
Import Dataset
87 MiB
List
R
Global Environment
Environment is empty
Untitled1
Source on Save
Run
Source
1:1
(Top Level)
R Script
Files Plots Packages Help Viewer Presentation
Folder Blank File Delete Rename
C: > Users > HP > Desktop > day1
Name Size Modified
day1.Rproj 218 B Mar 24, 2023, 12:24 PM
Console Terminal Background Jobs
R 4.1.2 · C:/Users/HP/Desktop/day1/

Organizing your working directory & setting up

Viewing your working directory

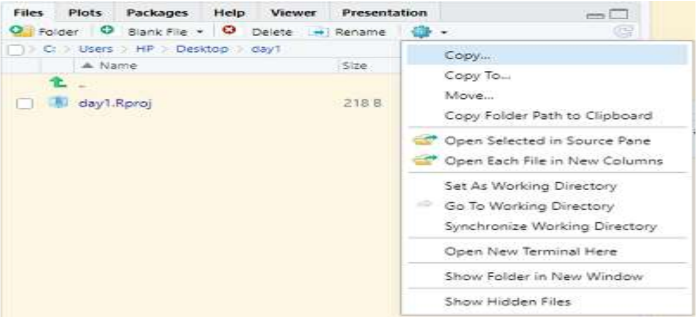
Before we organize our working directory, let’s check to see where our current working directory is located by typing into the console:

```
getwd()
```

Your working directory should be the ***day1*** folder constructed when you created the project. The working directory is where RStudio will automatically look for any files you bring in and where it will automatically save any files you create, unless otherwise specified.

You can visualize your working directory by selecting the Files tab from the **Files/Plots/Packages/Help window.**

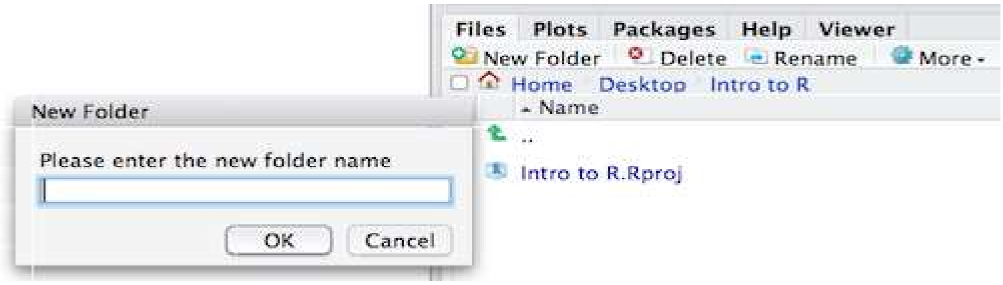
If you wanted to choose a different directory to be your working directory, you could navigate to a different folder in the Files tab, then, click on the **more** dropdown menu and select Set as Working Directory.


Files Plots Packages Help Viewer Presentation
Folder Blank File Delete Rename
C: > Users > HP > Desktop > day1
Name Size
day1.Rproj 218 B
Copy...
Copy To...
Move...
Copy Folder Path to Clipboard
Open Selected in Source Pane
Open Each File in New Columns
Set As Working Directory
Go To Working Directory
Synchronize Working Directory
Open New Terminal Here
Show Folder in New Window
Show Hidden Files

Structuring your working directory

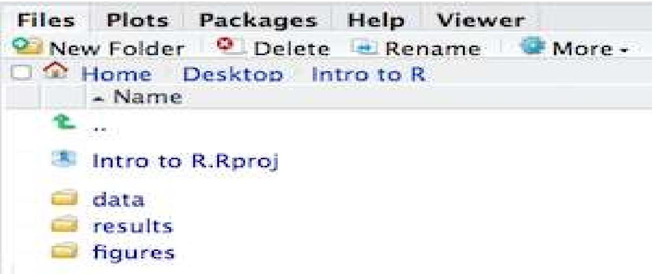
To organize your working directory for a particular analysis, you should separate the original data (raw data) from intermediate datasets. For instance, you may want to create a data/ directory within your working directory that stores the raw data and have a results/ directory for intermediate datasets and a figures/ directory for the plots you will generate.

Let's create these three directories within your working directory by clicking on New Folder within the Files tab.



Please enter the new folder name

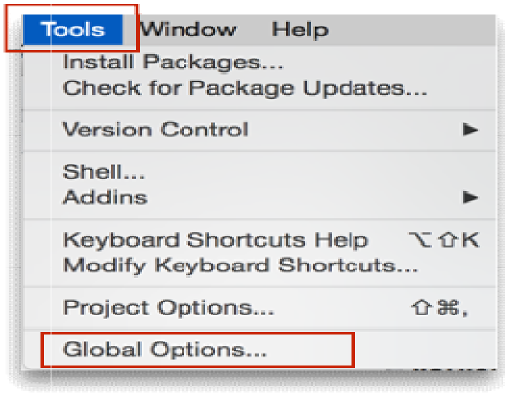
When finished, your working directory should look like:


Files Plots Packages Help Viewer
New Folder Delete Rename More
Home Desktop Intro to R
Name

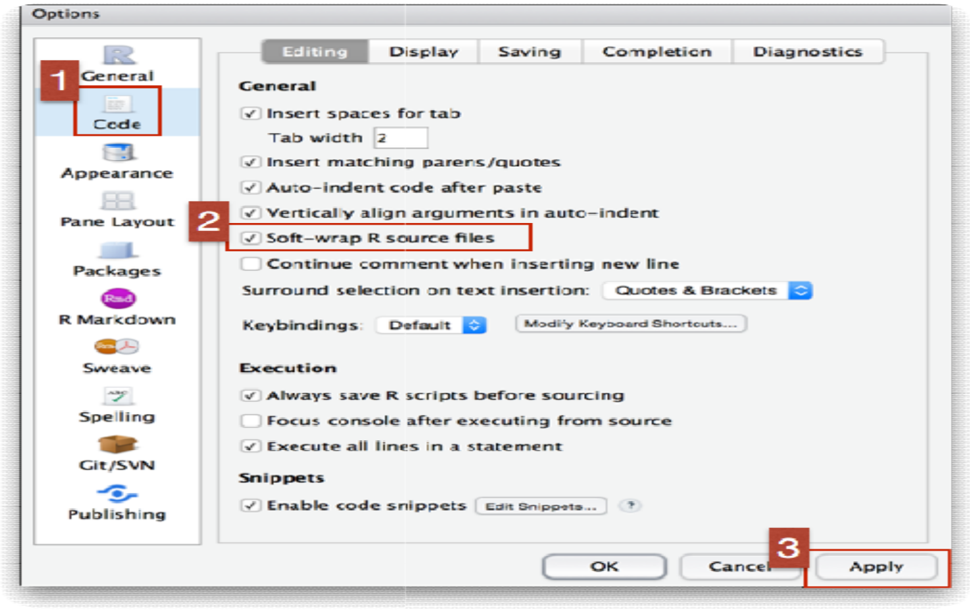
..
Intro to R.Rproj
data
results
figures

Setting up R-Studio (to change color, font, and panel/pane's location)

This is more of a housekeeping task. Just click on the global options and feel free to change color, font, pane's location of your choice. Also, we will be writing long lines of code in our script editor and want to make sure that the lines "wrap" and you don't have to scroll back and forth to look at your long line of code.Click on "Tools" at the top of your RStudio screen and click on "Global Options" in the pull down menu.



On the left, select “Code” and put a check against “Soft-wrap R source files”. Make sure you click the “Apply” button at the bottom of the Window before saying “OK”.



General

☑ Insert spaces for tab

Tab width 2

☑ Insert matching parens/quotes

☑ Auto-indent code after paste

☑ Vertically align arguments in auto-indent

☑ Soft-wrap R source files

☐ Continue comment when inserting new line

Surround selection on text insertion: Quotes & Brackets

Keybindings: Default Modify Keyboard Shortcuts...

Execution

☑ Always save R scripts before sourcing

☐ Focus console after executing from source

☑ Execute all lines in a statement

Snippets

☑ Enable code snippets Edit Snippets...

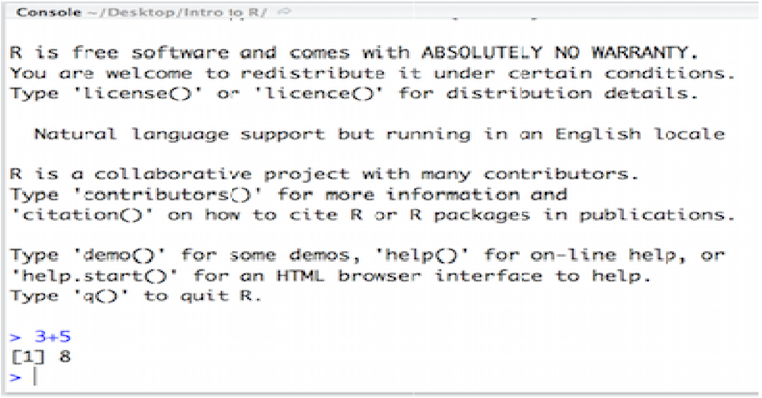
Interacting with R

Now that we have our interface and directory structure set up, let’s start playing with R! There are two main ways of interacting with R in RStudio: using the console or by using **script editor** (plain text files that contain your code).

Console window

The console window (in RStudio, the bottom left panel) is the place where R is waiting for you to tell it what to do, and where it will show the results of a command. You can type commands directly into the console, but they will be forgotten when you close the session.Let’s explore it out:

```
3 + 5
```



```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 3+5
[1] 8
> |
```

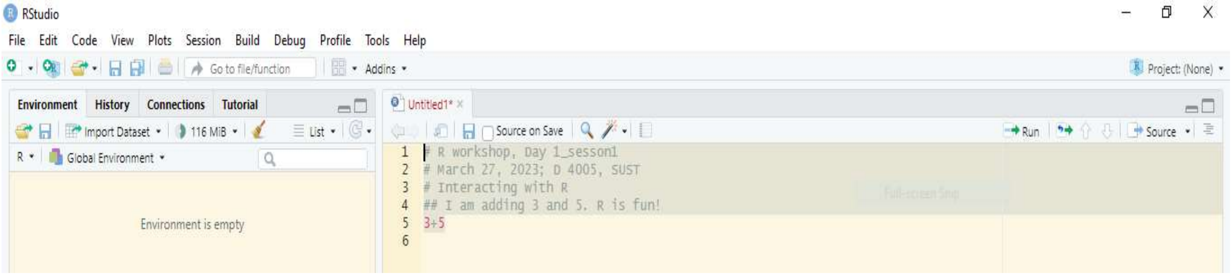
Script editor (Source)

Best practice is to enter the commands in the script editor and save the script. You are encouraged to commentgenerously to describe the commands you are running using #. This way, you have a complete record of what you did, you can easily show others how you did it and you can do it again later on if needed.

The Rstudio script editor allows you to 'send' the current line or the currently highlighted text to the R console by clicking on the Run button in the upper-right hand corner of the script editor. Alternatively, you can run by simply pressing the Ctrl and Enter keys at the same time as a shortcut.

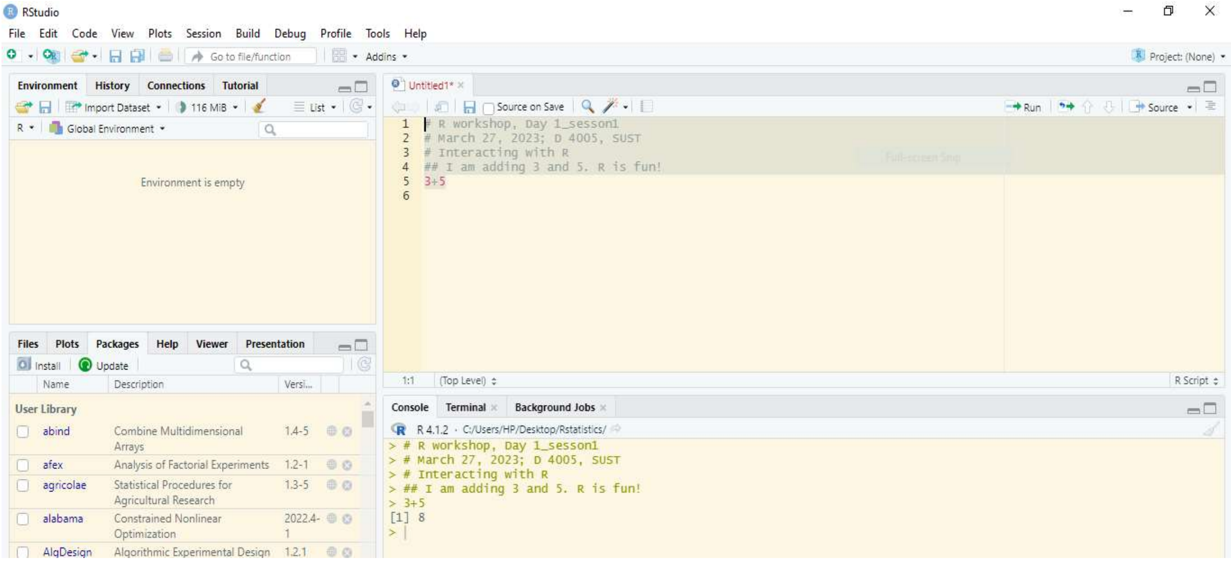
Now let's try entering commands to the script editor and using the comments character # to add descriptions and highlighting the text to run:

```
# R workshop, Day 1_session1
# March 27, 2023; D 4005, SUST
# Interacting with R
## I am adding 3 and 5. R is fun!
3+5
```



```
# R workshop, Day 1_sesson1
# March 27, 2023; D 4005, SUST
# Interacting with R
## I am adding 3 and 5. R is fun!
3+5
```

You should see the command run in the console and output the result.



```
# R workshop, Day 1_sesson1
# March 27, 2023; D 4005, SUST
# Interacting with R
## I am adding 3 and 5. R is fun!
3+5
```

Name	Description	Version
abind	Combine Multidimensional Arrays	1.4-5
afex	Analysis of Factorial Experiments	1.2-1
agricolae	Statistical Procedures for Agricultural Research	1.3-5
alabama	Constrained Nonlinear Optimization	2022.4-1
AlgDesign	Algorithmic Experimental Design	1.2.1

```
R 4.1.2 · C:/Users/HP/Desktop/Rstatistics/
> # R workshop, Day 1_sesson1
> # March 27, 2023; D 4005, SUST
> # Interacting with R
> ## I am adding 3 and 5. R is fun!
> 3+5
[1] 8
>
```

What happens if we do that same command without the comment symbol #? Re-run the command after removing the # sign in the front:

```
I am adding 3 and 5. R is fun!
```

```
3+5
```

Now R is trying to run that sentence as a command, and it doesn't work. We get an error in the console ***"Error: unexpected symbol in "I am"***means that the R interpreter did not know what to do with that command."

Console command prompt

Interpreting the command prompt can help understand when R is ready to accept commands. Below lists the different states of the command prompt and how you can exit a command:

Console is ready to accept commands: **>**

If R is ready to accept commands, the R console shows a **>** prompt.

When the console receives a command (by directly typing into the console or running from the script editor (**Ctrl-Enter**), R will try to execute it.

After running, the console will show the results and come back with a **new >** prompt to wait for new commands.

Console is waiting for you to enter more data: **+**

If R is still waiting for you to enter more data because it isn't complete yet, the console will show a + prompt. It means that you haven't finished entering a complete command. Often this can be due to you having not 'closed' a parenthesis or quotation.

Escaping a command and getting a new prompt: **esc**

If you're in Rstudio and you can't figure out why your command isn't running, you can click inside the console window and press esc to escape the command and bring back a new prompt >.

Best practices

Before we move on to more complex concepts and getting familiar with the language, we want to point out a few things about best practices when working with R which will help you stay organized in the long run:

- Code and workflow are more reproducible if we can document everything that we do.
- Our end goal is not just to "do stuff", but to do it in a way that anyone can easily and exactly replicate our workflow and results.
- All code should be written in the script editor and saved to file, rather than working in the console.
- The R console should be mainly used to inspect objects, test a function or get help.
- Use # signs to comment. Comment liberally in your R scripts. This will help future you and other collaborators know what each line of code (or code block) was meant to do.
- Anything to the right of a # is ignored by R.
- A shortcut for this is Ctrl + Shift + C if you want to comment an entire chunk of text.

Day1_Session1_Module III

The foundations of the R language.

- R syntax and Assignment operator
- Create objects/variables
- Data Types
- Explore data types used in R
- Construct data frame and data structures to store data

## Create a New R-Script file

The R syntax

A computer language is described by its syntax and semantics; where syntax is about the grammar of the language and semantics the meaning behind the sentence.

- The comments **#**. Write something meaningful after Hash tag sign as R will not execute those lines after the **#**

- create an object (variable)
- The assignment operator **<-**
- The**=** for arguments in functions (however, this is not a must do thing, just an R-style)

Assignment operator

## The **<-** is called the assignment operator. This operator assigns values to variables. The command below is translated into a sentence as: The result variable **x gets the value of 3**. We need to assign values to variables using the assignment operator, **<-.**

For example, we can use the assignment operator to assign the value of 8 to an object called number. Also, an object can get multiple values like the object x1 below. **C** stand for **concatenation**:

```
X       <-      3
number  <-      8
x1      <-      c(3,4,6)
```

The assignment operator (<-) assigns values on the right to variables on the left.In RStudio, typing **Alt and -** (push Alt at the same time as the - key, on Mac type option and -) will write ` <- ` in a single keystroke.

Variables

A variable is a symbolic name for (or reference to) information. Variables in computer programming are analogous to “buckets”, where information can be maintained and referenced. On the outside of the bucket is a name. When referring to the bucket, we use the name of the bucket, not the data stored in the bucket.

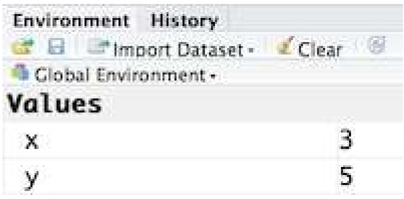
In the example above, we created a variable, or a 'bucket' called x. Inside we put a value, 3.

Let's create another variable called y and give it a value of 5.

When assigning a value to a variable, R does not print anything to the console. You can force to print the value by using parentheses or by typing the variable name.

```
        y       <-      5
        y
```

You can also view information on the variable by looking in your Environment window in the upper right-hand corner of the RStudio interface.



Values

x	3
y	5

Now we can reference these buckets by name to perform mathematical operations on the values contained within. What do you get in the console for the following operation?

```
x+y
```

Try assigning the results of this operation to another variable called number.

```
        number  <-      x+y
```

Tips on variable names

Variables can be given almost any name, such as x, current_temperature, or subject_id. However, there are some rules / suggestions you should keep in mind:

- Make your names explicit and not too long.
- Avoid names starting with a number (2x is not valid but x2 is)
- Avoid names of fundamental functions in R (e.g., if, else, for, see here for a complete list). In general, even if it's allowed, it's best to not use other function names (e.g., c, T, mean, data) as variable names. When in doubt check the help to see if the name is already in use.
- Avoid dots (.) within a variable name as in my.dataset. There are many functions in R with dots in their names for historical reasons, but because dots have a special meaning in R (for methods) and other programming languages, it's best to avoid them.
- Use nouns for object names and verbs for function names

- Keep in mind that **R is case sensitive** (e.g., asset_length is different from Asset_length) and ignores white space.
- Be consistent with the styling of your code (where you put spaces, how you name variable, etc.). In R, two popular style guides are Hadley Wickham's style guide and Google's.

Data Types

Variables can contain values of specific types within R.

The six **data types** that R uses include:

- "numeric" for any numerical value
- "character" or "string" for text values, denoted by using quotes ("") around value
- "integer" for integer numbers (e.g., 2L, the L indicates to R that it's an integer)
- "logical" for TRUE and FALSE (the Boolean data type)
- "complex" to represent complex numbers with real and imaginary parts (e.g., 1+4i)
- "raw" that is not part of this course

The table below provides examples of each of the commonly used data types:

Data Type	Examples
Numeric:	1, 1.5, 20, pi
Character:	"Top Gun", "5", "TRUE"
Integer:	2L, 500L, -17L
Logical:	TRUE, FALSE, T, F

Data Structures

Data Structures

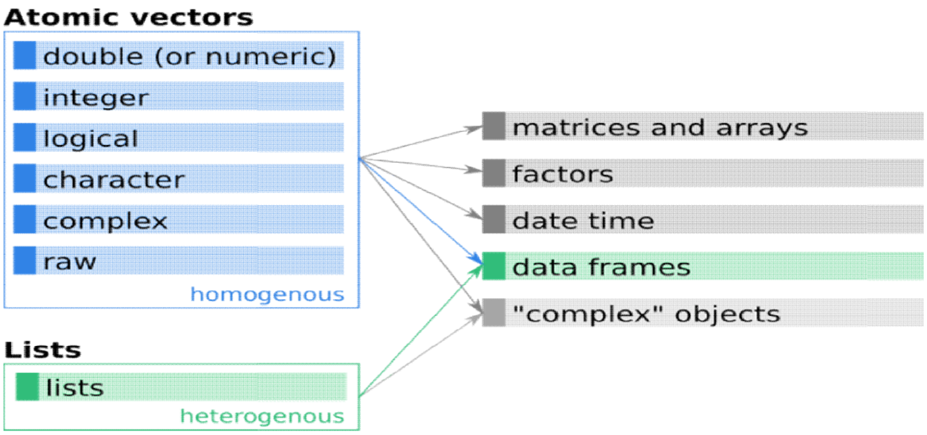
Variables can store more than just a single value; they can store a multitude of different data structures. These include, but are not limited to, vectors (c), factors (factor), matrices (matrix), data frames (data.frame) and lists (list).

Vectors

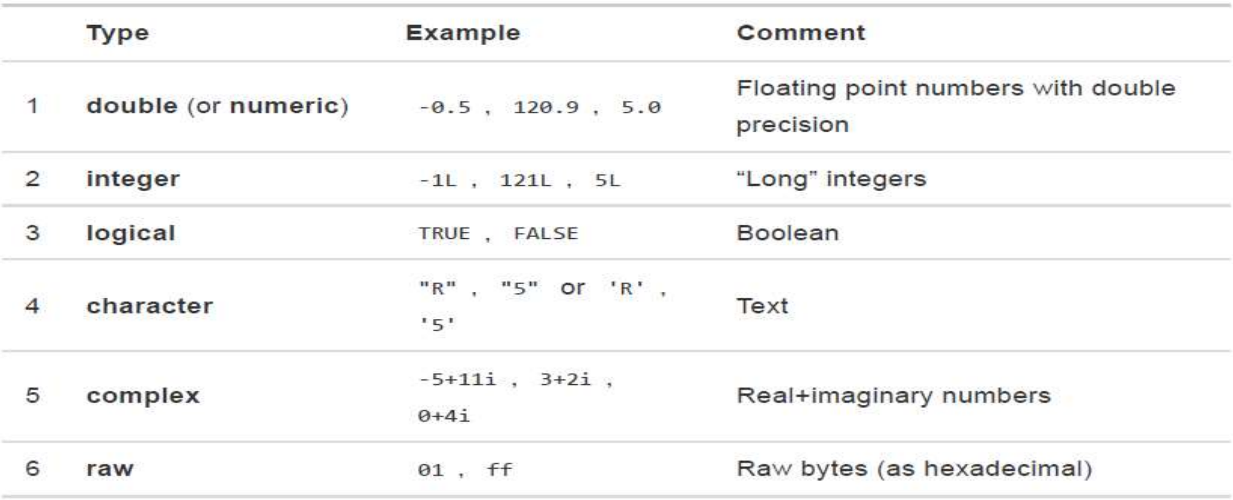
A vector is nothing else than a sequence of elements of a certain type. R distinguishes vectors with two different modes.

Atomic vectors: All elements must have the same basic type (e.g., numeric, character, …).

Lists: Special vector mode. Different elements can have different types.

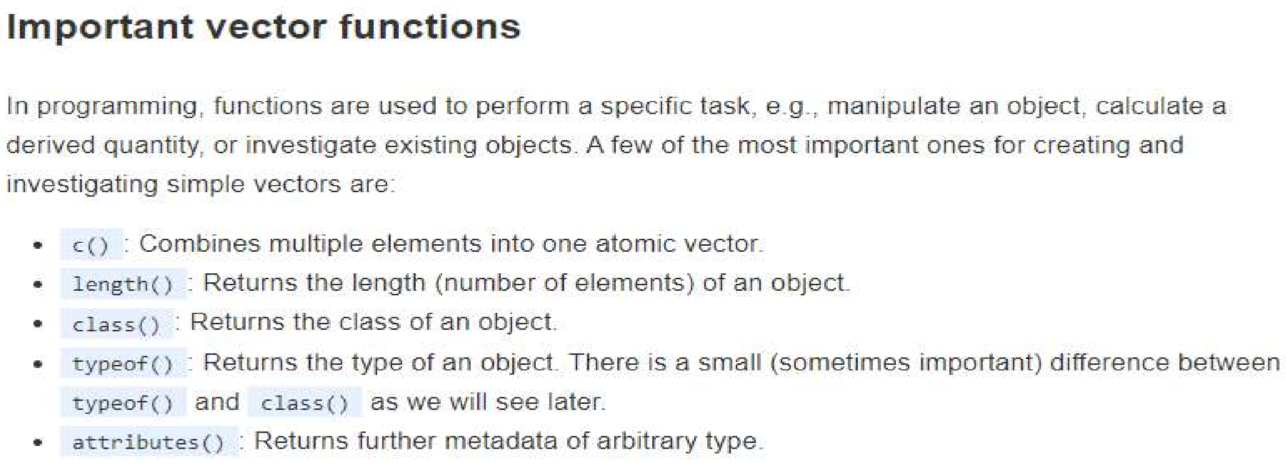


Six types of atomic vectors in R. Only the first four (double, integer, logical, and character) are discussed and used in this workshop.



	Type	Example	Comment
1	double (or numeric)	`-0.5` , `120.9` , `5.0`	Floating point numbers with double precision
2	integer	`-1L` , `121L` , `5L`	"Long" integers
3	logical	`TRUE` , `FALSE`	Boolean
4	character	`"R"` , `"5"` or `'R'` , `'5'`	Text
5	complex	`-5+11i` , `3+2i` , `0+4i`	Real+imaginary numbers
6	raw	`01` , `ff`	Raw bytes (as hexadecimal)

A vector is the most common and basic data structure in R, and is pretty much the workhorse of R.



Important vector functions

In programming, functions are used to perform a specific task, e.g., manipulate an object, calculate a derived quantity, or investigate existing objects. A few of the most important ones for creating and investigating simple vectors are:

- `c()` : Combines multiple elements into one atomic vector.
- `length()` : Returns the length (number of elements) of an object.
- `class()` : Returns the class of an object.
- `typeof()` : Returns the type of an object. There is a small (sometimes important) difference between `typeof()` and `class()` as we will see later.
- `attributes()` : Returns further metadata of arbitrary type.

Numeric Example#01

```
length(x)       # Length of the vector.
## [1] 1
class(x)        # What is the class of the object?
## [1] "numeric"
typeof(x)       # What is the type of this vector?
## [1] "double"
```

Concatenating values c()

Create a vector: cgpa

Let’s create a vector of cgpa lengths and assign it to a variable called cgpa.

Each element of this vector contains a single numeric value, and 5 values will be combined together into a vector using c() (the combine function). All of the values are put within the parentheses and separated with a comma.

```
        cgpa    <-      c(4.0,3.75,3.5,3.25, 3)
        cgpa
        y       <-      c(3,4,5)
```

Note your environment shows the cgpa variable is numeric and tells you the cgpa vector starts at element 1 and ends at element 5 (i.e. your vector contains 5 values).

Numeric Example#02

```
x <-c(111L, 555L)     # Create the vector; overwrite 'x'
length(x)              # Checking length
## [1] 2
class(x)               # Checking class
## [1] "integer"
typeof(x)              # Checking the type
## [1] "integer"
```

Double (or numeric)

One atomic data class is the double class (double precision floating point values). Double is actually not often used in R, objects of type double are simply called numeric and the class of floating point numbers will be "numeric"

```
x <-1.5
x
## [1] 1.5
```

Note: the number 10 will also be numeric (double) as 10 is interpreted as 10.000.

```
(x <-10)
```

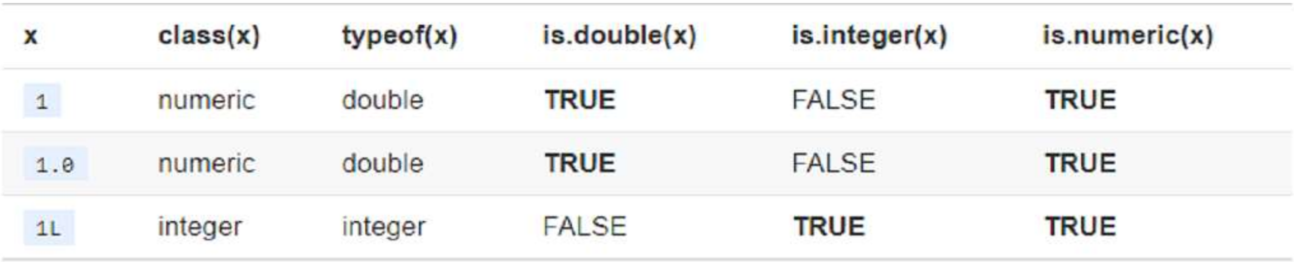
```
## [1] 10
class(x)
## [1] "numeric"

# Double (numeric) vector of length 5
vector("double", length =5)
```

Double vs. numeric

If you are new to programming this might be a bit confusing. However, at some point it will get important to know that there is a small but important difference between “numeric” and “double”.If you are coming from another programming language you might be familiar with objects of type “double”, “real”, “float”, and “integer”. A “double” is a double precision floating point numeric value (e.g., 0.001234). Depending on the programming language they might be referred to as “double”, “real”, or “float” which are basically the same.

An “integer” is also a numeric value but from ±***No*** (`[..., -3, -2, -1, 0, 1, 2, 3, ...]`; positive and negative natural numbers including zero). As we have learned, *R* is based on *C* and, thus, shares the data types “double” and “integer” with *C*. However, there is a difference between the class of an object in *R* and the type (which is the underlying type).

Let us check what `class()` and `typeof()` returns for `1`, `1.0`, and `1L`:



x	class(x)	typeof(x)	is.double(x)	is.integer(x)	is.numeric(x)
`1`	numeric	double	**TRUE**	FALSE	**TRUE**
`1.0`	numeric	double	**TRUE**	FALSE	**TRUE**
`1L`	integer	integer	FALSE	**TRUE**	**TRUE**

The table shows (left to right) the value which is checked (`x`), the *R* class of the object `x`, the type of `x`, and three checks: whether the value `x` is double, numeric, or integer (`is.*()`).

- `1` and `1.0`: are of class “numeric”. The underlying data type in *C* is “double”, thus both `is.double()` and `is.numeric()` return `TRUE`.
- `1L`: this is how an integer is specified in *R*. `1L` is of class “integer” and the underlying *C* type is also “integer”, however, an integer in *R* is also threated as “numeric”! Thus `is.numeric()` returns `TRUE` as well.

Why? Well, the philosophy is that one can use both for mathematical operations. In *R* there is no difference between `5L^2L`, `5L^2.0`, and `5.0^2L` as in some other programming languages. Thus, `is.numeric()` basically tells us whether or not we can use this specific object (`x`) for mathematical calculations.

Integers

A second atomic class is the integer class. Integers can explicitly be defined using the L ("Long") suffix. While 10 will be double (or numeric; 10.0), 10L defines an integer (natural number).

```
 (x <-10L)
class(x)
## [1] "integer"
```

This is not important most of the time when you are using *R*. Except when it is! There are some situations where integers are required!

Logical

Logicals are rather simple as they can only take up two different values: either TRUE or FALSE.

```
(x <-TRUE)
## [1] TRUE
(x <-FALSE)
## [1] FALSE
class(x)
## [1] "logical"
```

Note: TRUE and FALSE must be written without quotation marks (they
are **logicals** not **characters**).

Characters

The last atomic class we will deal with is the *character* class. A character is basically just a text and can be of any length.

```
(x <- "Austria")
## [1] "Austria"
(x <- "Max Konstantin Mustermann (AT)")
## [1] "Max Konstantin Mustermann (AT)"
class(x)
## [1] "character"
```

```
y <-c("this is", "just some text")
length(y)
## [1] 2
class(y)
## [1] "character"
typeof(y)
## [1] "character"
```

Create another vector called semester with five elements,

```
semester      <-      c("one","two","three", “four”, “special” )
semester
```

```
# Character vector of length 3
vector("character", length =3)
```

Creating vector with sequences

Regular sequences can be created using the function seq() and its shortcuts seq.int(), seq_along(), and seq_len().

```
# Equidistant numeric sequence
#from: Where to start the sequence.
#to: Where to end the sequence.
#length or by: either define the length of the resulting vector, or the increment
#by which the values change.

seq(from =1.5, to =2.5, length.out =5)  # Specify length
seq(from =4, to =-4, by =-0.5)          # Specify increment/interval
```

```
# Explicitly define from/to/by as integers
(x <-seq(from = 10L, to = 100L, by = 10L))
class(x)
# 'from' defined as a numeric value (10.0)
(y <-seq(from =10, to = 100L, by = 10L))
class(y)
```

```
# Create character vector 'cities'
cities <-c("Vienna", "Paris", "Berlin", "Rome", "Bern")
seq_along(cities)
seq_len(10)
```

Character Sequences:

```
# First 7 letters of the alphabet
LETTERS[1:7]
## [1] "A" "B" "C" "D" "E" "F" "G"
letters[1:7]
```

Replicating elements

```
rep(2L, 5)
cities <-c("Vienna", "Bern", "Rome")
rep(cities, each =3)     # Case (2)
rep(cities, times =3)    # Case (3)
rep(cities, times =c(3, 2, 5))
```

```
rep_len(c(4, 5, 6), length.out =5)    # Resulting vector has length 5
rep_len(c(4, 5, 6), length.out =9)    # Resulting vector has length 9
```

Checking class/type

- `is.double()`
- `is.numeric()`
- `is.integer()`
- `is.logical()`
- `is.character()`
- `is.vector()`
- … and many more.

- `is.integer(1.5)`
- `is.double(1.5)`
- `is.character("Bruno")`
- `is.vector(c(1, 2, 3, 4))`

Logical Comparison

```
3 < 4
3 > 4
#contrast with 3 = 4;
#see section about variables below
3 == 4
```

```
#!= means "not equal to"
3 != 4
4 >= 5
4 <= 5
2 + 2 == 5
10 - 6 == 4
```

Function: Logical Comparison

Operator	Description	Operator	Description
>	greater than	!x	Not x
>=	greater than or equal to	x | y	x OR y
<	less than	x & y	x AND y
<=	less than or equal to	isTRUE(x)	test if X is TRUE
==	equal to		
!=	not equal to		
&	and		
|	or		

Factors

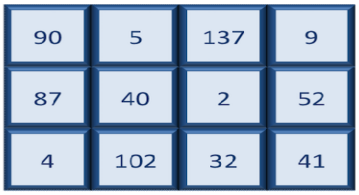
A factor is a special type of vector that is used to store categorical data. Each unique category is referred to as a factor level (i.e. category = level).

```
Let's create a factor vector and explore a bit more.

# Create a vector as input.
```

```
data <- c("East","West","East","North","North","East","West","West","West","East","North")

print(data)
print(is.factor(data))

# Apply the factor function.
factor_data<- factor(data)

print(factor_data)
print(is.factor(factor_data))

food <- factor(c("low", "high", "medium", "high", "low", "medium", "high"))
levels(food)
```

Matrix

A matrix in R is a collection of vectors of same length and identical datatype. Vectors can be combined as columns in the matrix or by row, to create a 2-dimensional structure.



90	5	137	9
87	40	2	52
4	102	32	41

Matrices are used commonly as part of the mathematical machinery of statistics. They are usually of numeric datatype and used in computational algorithms to serve as a checkpoint.

# Generating matrix, A, from one vector with all values

```
v       <-      c(2,-4,-1,5,7,0)
A       <-       matrix(v,nrow=2)
```

# Generating matrix, A, from two vectors corresponding to rows:

```
row1    <-      c(2,-1,7);
row2    <-       c(-4,5,0)
( A<- rbind(row1, row2) )
```

# Generating matrix, A, from three vectors corresponding to columns:

```
col1    <-      c(1,6)
col2     <-      c(2,3)
col3     <-      c(7,2)
```

```
            ( US40Chart <- cbind(col1, col2, col3) )
```

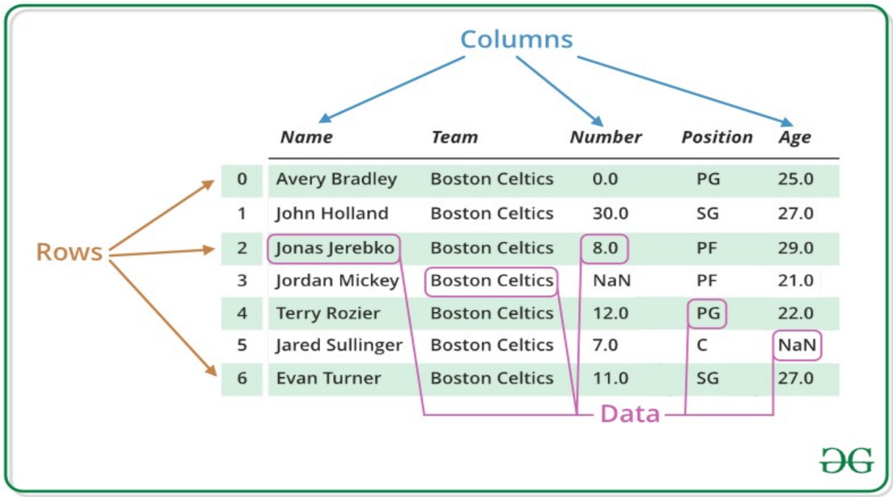
# Giving names to rows and columns:

```
            colnames(US40Chart) <-          c(“Drake",“Maroon5",“Dua Lipa")
            rownames(US40Chart) <-          c(“weekTop",“totPeak")
            US40Chart
```

Data Frame

A data.frame is the de facto data structure for most tabular data and what we use for statistics and plotting. A data.frame is similar to a matrix in that it’s a collection of vectors of the same length and each vector represents a column. However, in a dataframe each vector can be of a different data type (e.g., characters, integers, factors).

Year	sales	revenue	economy
2010	24	120	"trough"
2012	45	650	"expansion"
2014	67	980	"slowdown"
2016	78	890	"expansion"
2018	89	990	"peak"



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Source: GeeksforGeeks; R - Data Frames – GeeksforGeeks; Images may be subject to copyright.

A data frame is the most common way of storing data in R, and if used systematically makes data analysis easier.

We can create a dataframe by bringing vectors together to form the columns. We do this using the data.frame() function, and giving the function the different vectors we would like to bind together. This function will only work for vectors of the same length.

```
df<-data.frame(semester,cgpa)
```

Beware of data.frame()'s default behavior which turns character vectors into factors. Print your data frame to the console:

```
df
```

Upon inspection of our dataframe, we see that although the semester vector was a character vector, it automatically got converted into a factor inside the data frame (the removal of quotation marks). We will show you how to change the default behavior of a function in the next lesson.Note that you can view your data.frame object by clicking on its name in the Environment window.

# define year vector:

```
        year    <-   c(2008,2009,2010,2011,2012,2013)
```

# Define a matrix of product values:

```
        product1   <-          c(0,3,6,9,7,8)
        product2   <-          c(1,2,3,5,9,6)
        product3   <-          c(2,4,4,2,3,2)
```

# Create a sales matrix

```
        sales_mat  <-          cbind(product1,product2,product3)
```

## Give row names

```
        rownames(sales_mat)          <-          year
```

# The matrix looks like this:

```
        sales_mat
```

# Create a data frame and display it:

```
        sales      <-          as.data.frame(sales_mat)
        sales
```

# Accessing a single variable:

```
        sales$product2
```

# Generating a new variable in the data frame:

```
        sales$totalvalue       <-          sales$product1 + sales$product2 + sales$product3
```

# Subset: all years in which sales of product 3 were >=3

```
subset(sales, product3>=3)
```

```
# Create the vectors for data frame.
height <- c(132,151,162,139,166,147,122)
```

```
weight <- c(48,49,66,53,67,52,40)
gender <- c("male","male","female","female","male","female","male")

# Create the data frame.
input_data<- data.frame(height,weight,gender)
print(input_data)

# Test if the gender column is a factor.
print(is.factor(input_data$gender))

# Print the gender column so see the levels.
print(input_data$gender)
```

Coercion (mixing objects)

What happens if we try to mix elements of different classes, like numeric values and characters as in this example?

```
# Numeric and character = character
(x <- c(1.7, "1", "A"))
## [1] "1.7" "1"   "A"
class(x)
## [1] "character"
```

Numeric and logical

- Every TRUE converted to numeric will be 1 (or 1L).
- Every FALSE converted to numeric will be 0 (or 0L).

```
# Combine TRUE, 5.5, and 10.0.
(x <- c(FALSE, 5.5, 10.0))
## [1]  0.0  5.5 10.0
class(x)
```

Explicit coercion

- as.integer()
- as.numeric()
- as.character()
- as.logical()
- as.matrix()

```
• # Let `x` be an integer vector
• # with elements 0, 1, 2, 3, 4.
• (x <- 0:4)
• ## [1] 0 1 2 3 4
• # Coerce to character
• as.character(x)
• ## [1] "0" "1" "2" "3" "4"
• # Coerce to logical
• as.logical(x)
```

Lists

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements.

#List is created using list() function.

# Create a list containing strings, numbers, vectors and a logical

# Values.

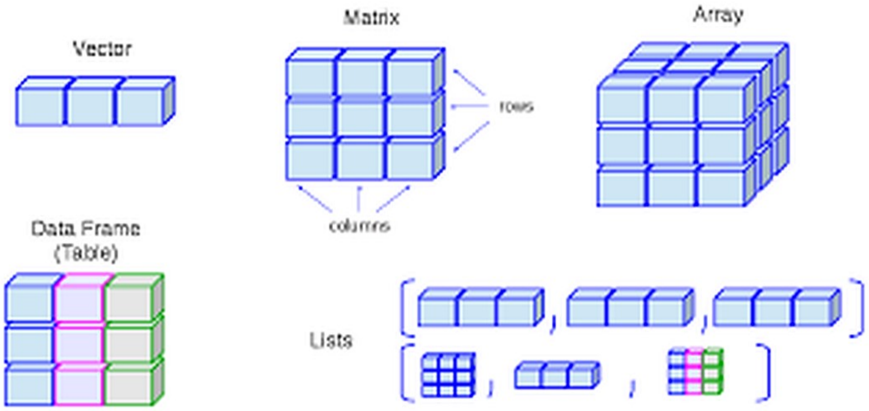
```
list_data<- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
print(list_data)
```

If you have variables of different data structures you wish to combine, you can put all of those into one list object by using the list() function and placing all the items you wish to combine within parentheses:

```
list<-list(semester,df, number, utility)
```

Print out the list to screen to take a look at the components:

```
list
```



Missing values

One additional important element is the missing values. *R* knows two types of missing values: NaN and NA. NaN is “not a number” and results from mathematical operations which are illegal/invalid. NA on the other hand stands for ‘not available’ and indicates a missing value. NAs can occur in all the vectors discussed above, and keep their class.

- NaN: Mathematically not defined (and always of class numeric).
- NA: Missing value, NA’s still have classes!
- NaN is also NA but not vice versa.

```
(x <- 0 / 0)
```

```
## [1] NaN
is.nan(x)     # Check if 'x' is NaN
## [1] TRUE
is.na(x)      # Check if 'x' is NA
## [1] TRUE
class(x)   # The value from above
## [1] "numeric"
```

When defining a new object and storing a single missing value on it (with x <- NA), it will be 'logical' by default. We can, however, convert them (if needed).

```
# By default - logical
(x <- NA)
## [1] NA
class(x)
## [1] "logical"
# Convert to integer
(y <- as.integer(NA))
## [1] NA
class(y)
## [1] "integer"
# Convert to character
(z <- as.character(NA))
## [1] NA
class(z)
## [1] "character"
```

```
# let `x` be a character vector
(x <- c("a", "b", "c", "d"))
## [1] "a" "b" "c" "d"
# Coerce to integer
as.integer(x)
## Warning: NAs introduced by coercion
## [1] NA NA NA NA
# But ...
x <- c("1", "100", "a", "b", "33")
as.integer(x)
## Warning: NAs introduced by coercion
## [1]   1 100  NA  NA  33
```

To conclude,

```
x <- c(0.5, 0.6, 0.7)            # double
x <- c(TRUE, FALSE, TRUE)        # logical
x <- c(T, F, T)                  # logical (!!!)
x <- c("a", "b", "c")            # character
x <- c(1L, 2L, 3L, 4L)           # integer
x <- 15:20                       # integer
x <- c(1+0i, 2+4i, 5+2i)         # complex
```

- Matrices are vectors with an additional dimension attribute for rows and columns.
- As based on vectors, they must be homogeneous (numeric, character, …).
- An alternative object is needed for empirical data which typically contain heterogeneous columns.
- Data frame: Most commonly used data type to represent rectangular data in R.
- Data frames are always two-dimensional objects.
- Columns of a data frame correspond to variables, rows to observations (different jargon).
- Look like a matrix and offer similar subsetting methods, but are not matrices.
- Internally, a data frame is a list with a series of objects of the same length.
- The list-elements (variables) are most often vectors of potentially different types, however, more complex objects such as lists or matrices (for specific variables) are possible.
- Variables (columns) are always named, the names are no longer optional.
- Observations (rows) also always have names, by default they are set to "1", "2", … but can be changed.

Ref:

- Hadley Wickham’s Advanced R book
- Roger Peng’s R Programming for Data Science book
- DataCamp’s Intermediate R course
- Coursera’s R Programming course
- Data Carpentry (http://datacarpentry.org/), data camp, data quest, Kaggle
- Harvard Chan Bioinformatics Core (HBC) under the open access terms of the Creative Commons Attribution license (CC BY 4.0),
- The Book of R: A First Course in Programming and Statistics by Tilman M. Davies
- UC Business Analytics R Programming Guide and R_bootcamp