



You Ready? An Introduction to R for Researchers

Eunnara Cho, PhD

(Research Biologist, Computational Toxicology Research Group,
Exposure and Biomonitoring Division)

December 17, 2025

Hopefully by the end of this workshop, you will:

- Become familiar with the main features of RStudio
- Know how to upload your dataset
- Know how to install and use R packages and functions
- Learn basic data manipulation and operations in R
- Learn to generate basic figures using ggplot2
- Learn how to save your results

Why use R?

- Designed specifically for statistics and data analysis and visualization
 - Thousands of freely available functions and packages for data analysis and visualization
- Can avoid manual errors by automating repetitive tasks with code
- Create customized, professional figures for presentations and publications

R Programming Language

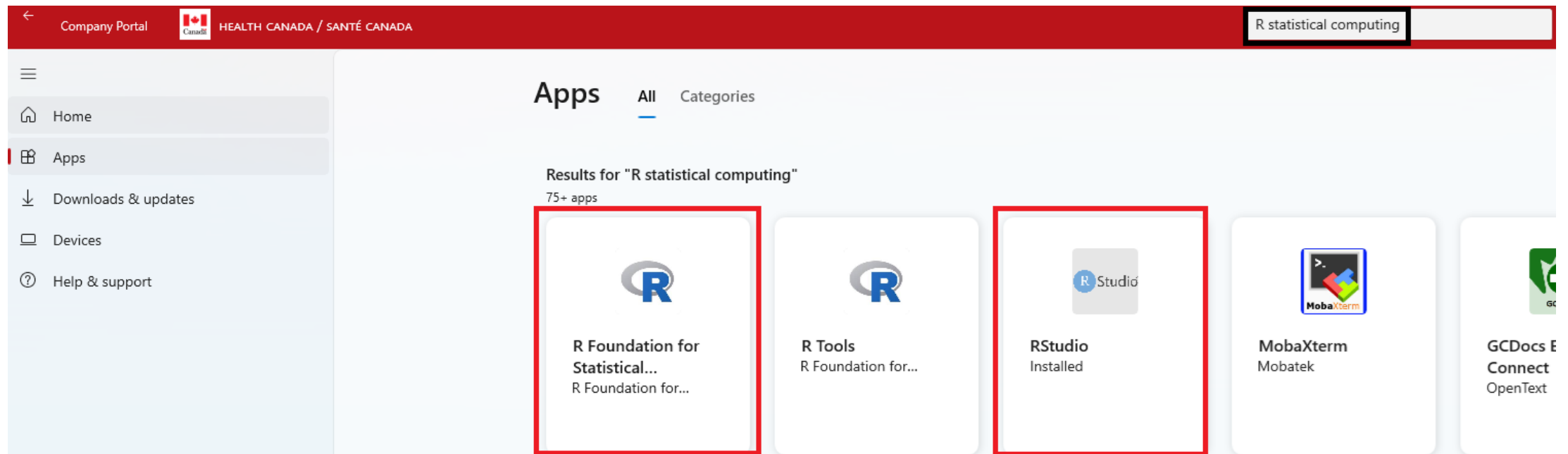
- R is an object-oriented language
 - Everything in R is an object
- An object is something that contain data
 - Different types of data: numbers, characters, functions, symbols, plots, and mathematical and logical expressions
- All R code manipulates objects
- Objects may have attributes, such as names, dimensions, and class

RStudio

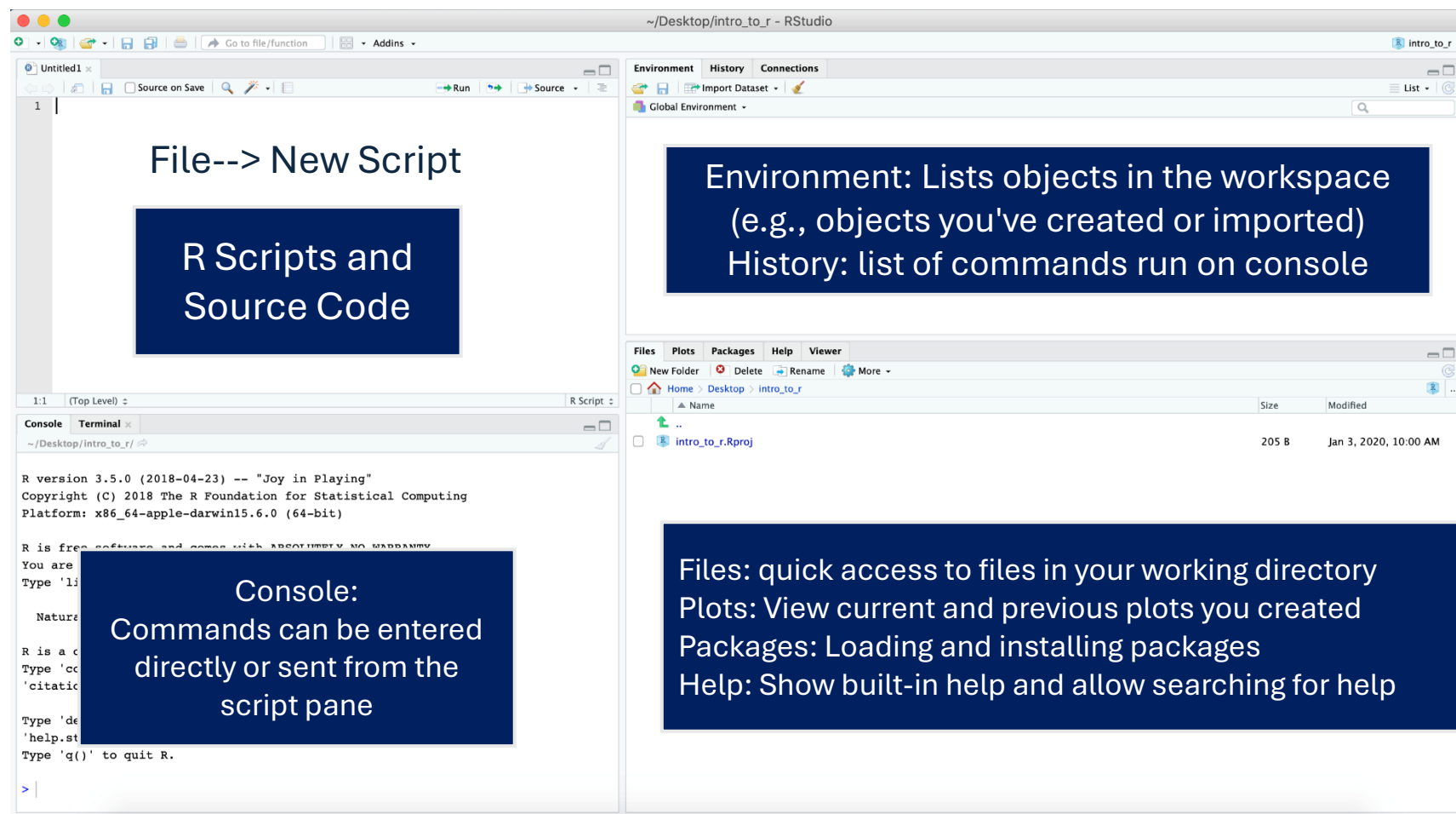
- An Integrated Development Environment (IDE)
 - A platform for programming in R
 - Write, edit, and run your R code
 - Install and use packages and functions
- In RStudio, you can load, manipulate (data wrangling), analyze, and visualize your data

Getting started – Download R and RStudio

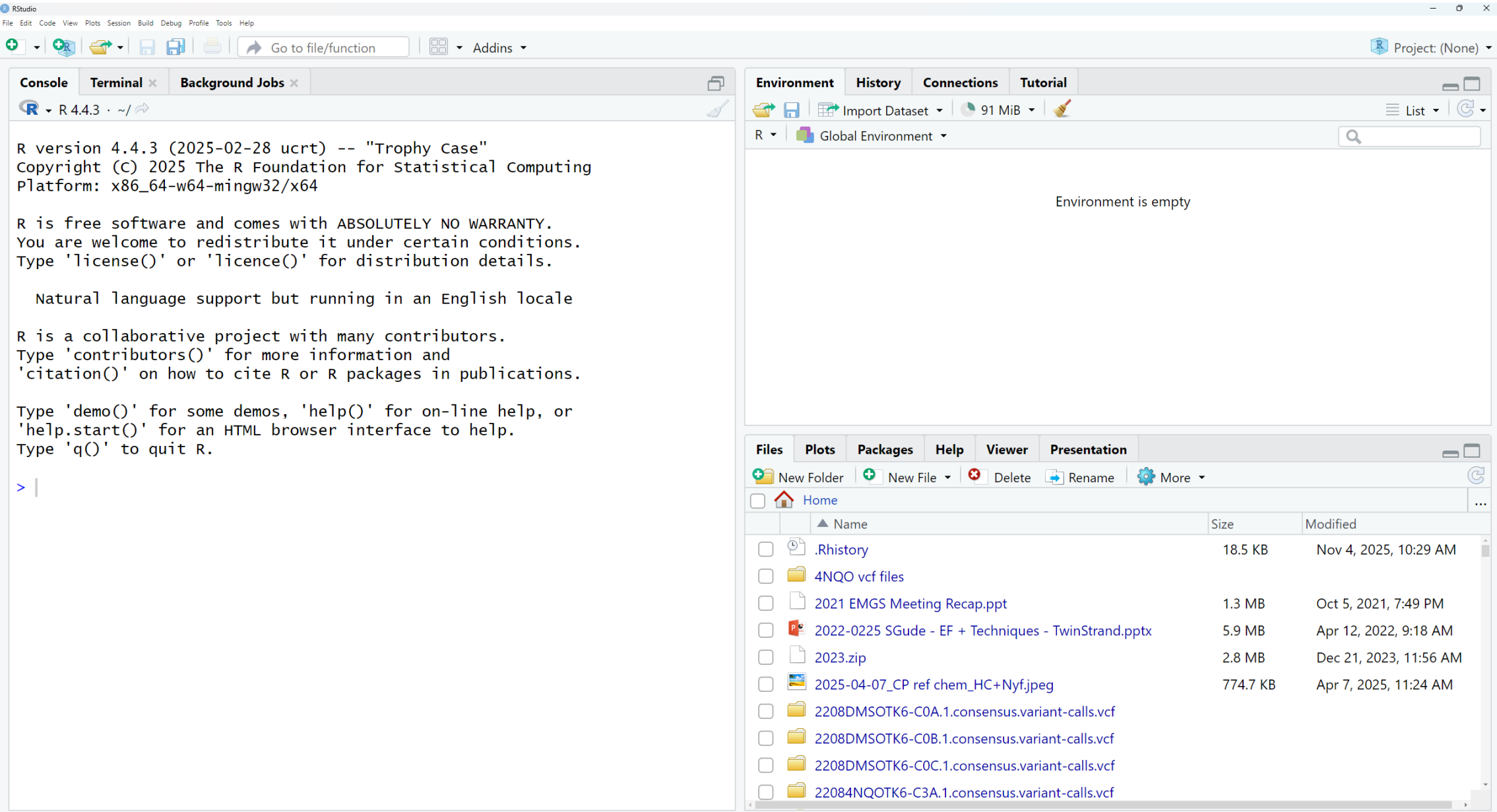
- Search “R statistical computing” on the Company Portal
- Install “R Foundation for Statical Computing” and “RStudio”



A Guided Tour of RStudio

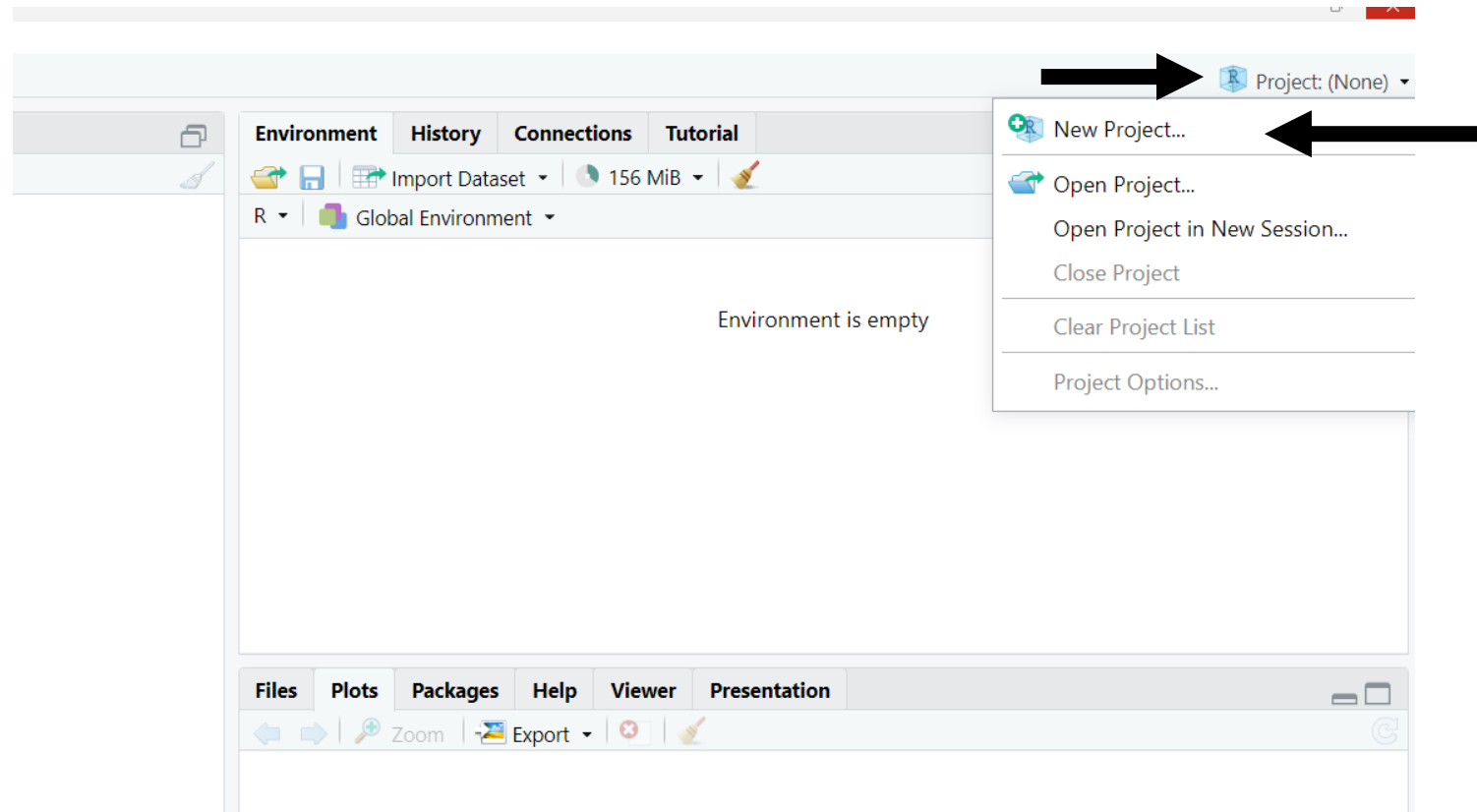


RStudio Interface



Before you start, create a new project in RStudio

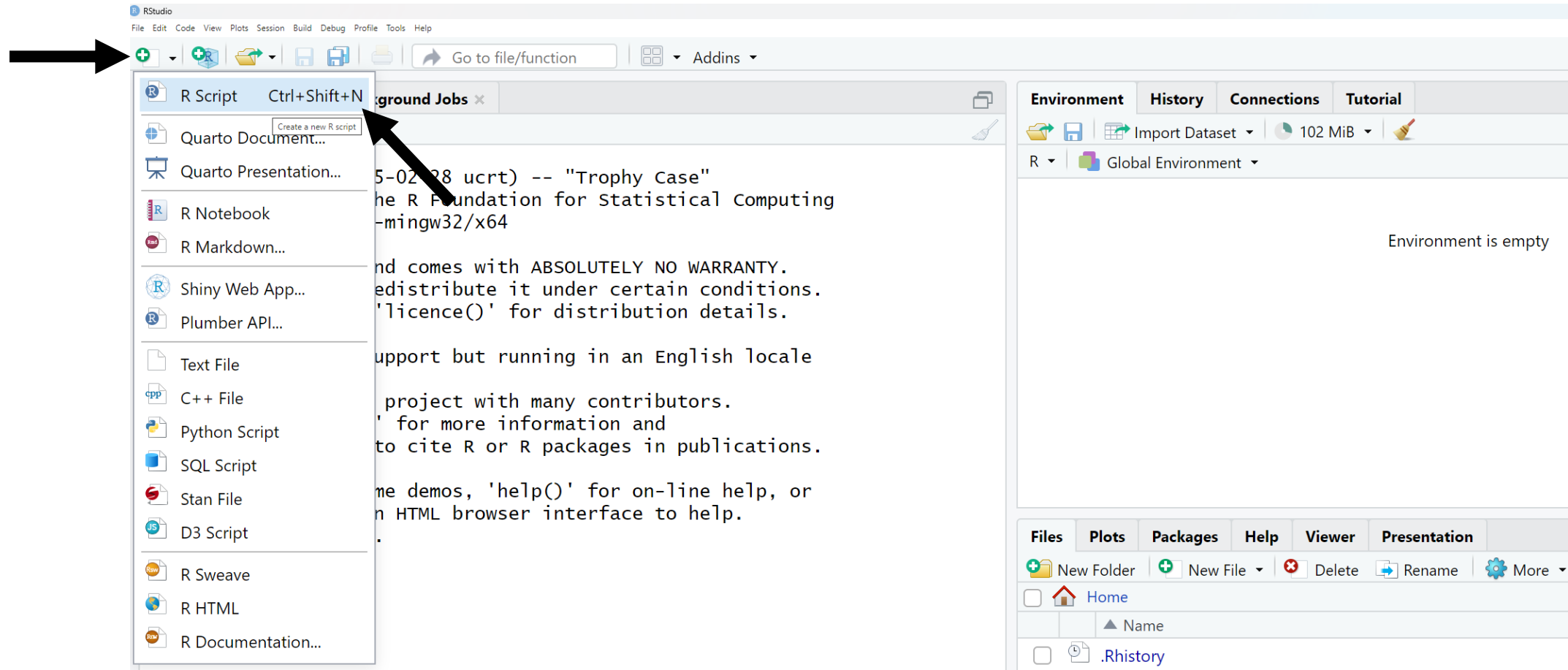
Under the “Project” tab at the top right corner of RStudio



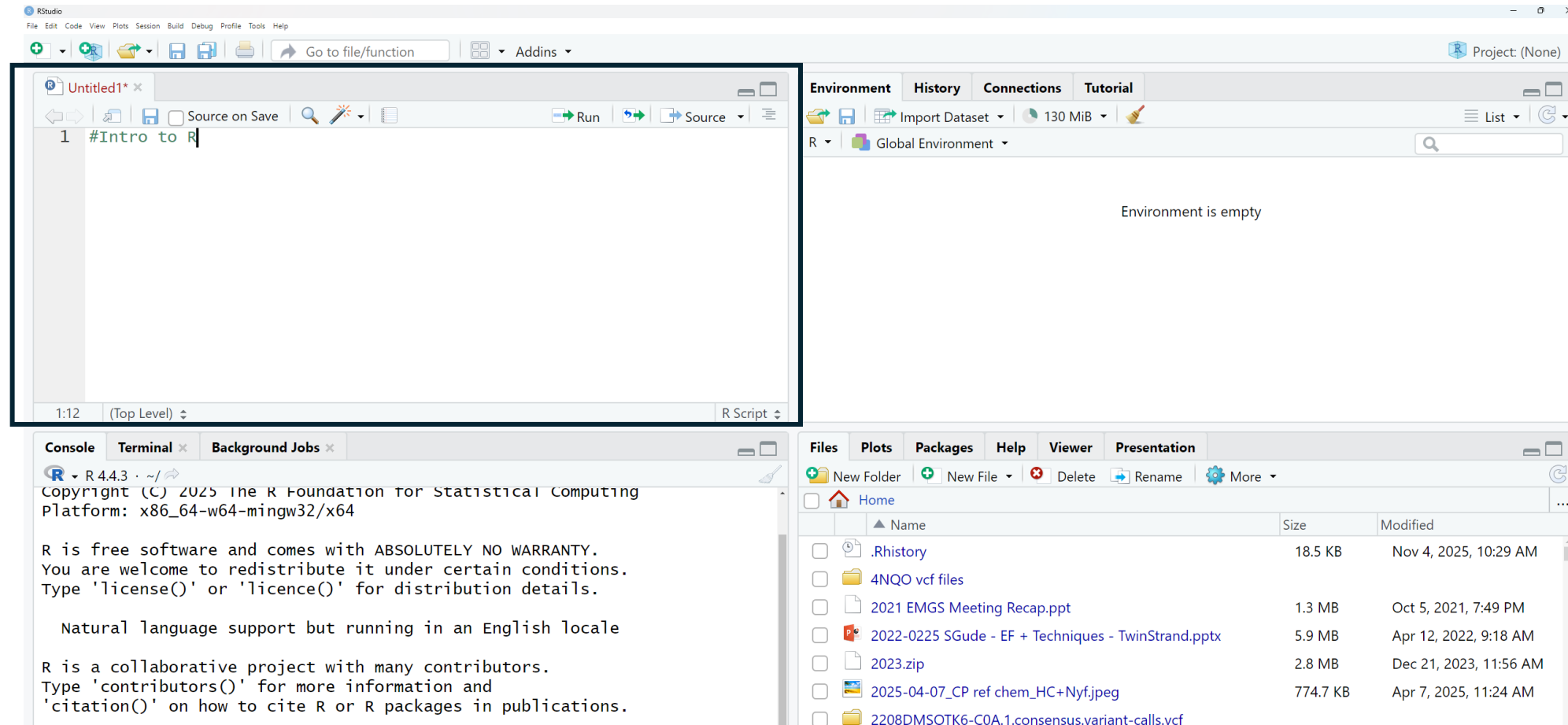
- A project is a self-contained workspace/folder that keeps all your scripts, data, and output
 - Your working directory

Create a new R Script

At the top left corner of RStudio



Create a new R Script



Introducing R Commands

- R is an interpreted language
 - Code are executed line by line
- Accessed through a command-line interpreter
 - This requires the user knowledge of commands and their parameters, and the syntax of the language
- Upon starting, there is a “>” in the console
 - R is prompting you to type something, so this is called a prompt
- The commands that you type into the console are called expressions

Creating and naming an object

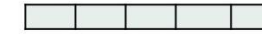
- To create an object, the assignment operator (`<-`) is used
- `<-` assigns value to a name:
 - `object_name <- 125`
 - `object.name <- 357`
 - `ObjectName <- "Object"`
- Names are case sensitive
- Never use space or special characters in object names
 - Including mathematical operators
- Names should not start with a number

Objects in R

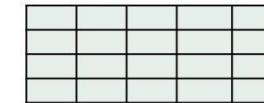
- Numeric
 - `my_number <- 5`
- Character
 - `my_name <- "Eunnara"`
- Vector (simple list of values)
 - `even_numbers <- c(2, 4, 6, 8)`
 - `my_vowels <- c("a", "e", "i", "o")`
- Data frame (data tables)
- Plots (images)
- Functions

Variables	Example
integer	100
numeric	0.05
character	"hello"
logical	TRUE
factor	"Green"

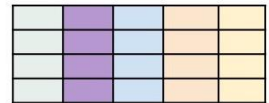
Vector



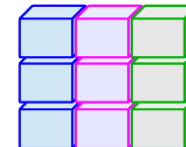
Matrix



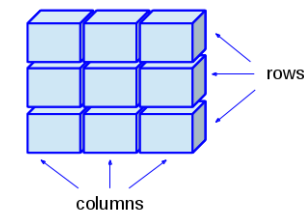
Data frame



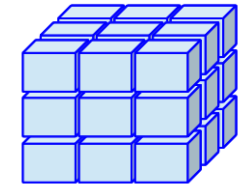
Vector

Data Frame
(Table)

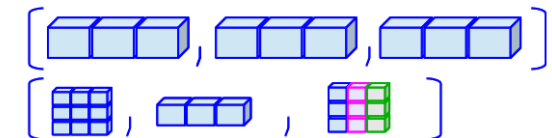
Matrix



Array

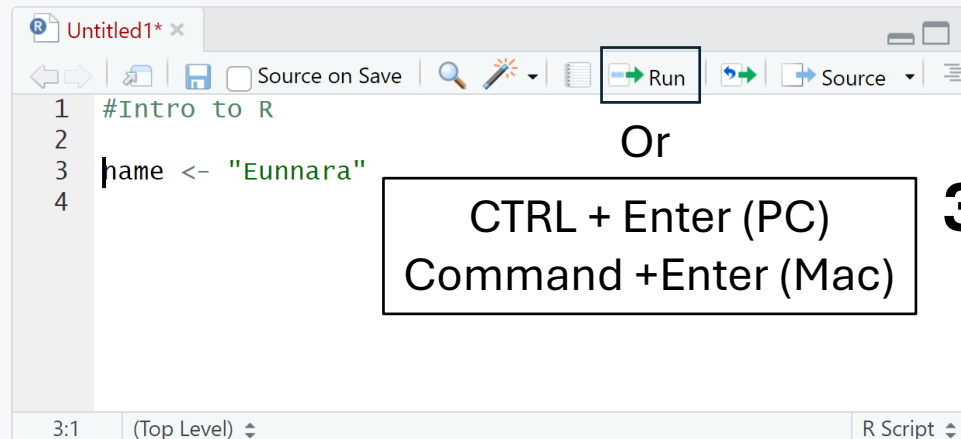


Lists



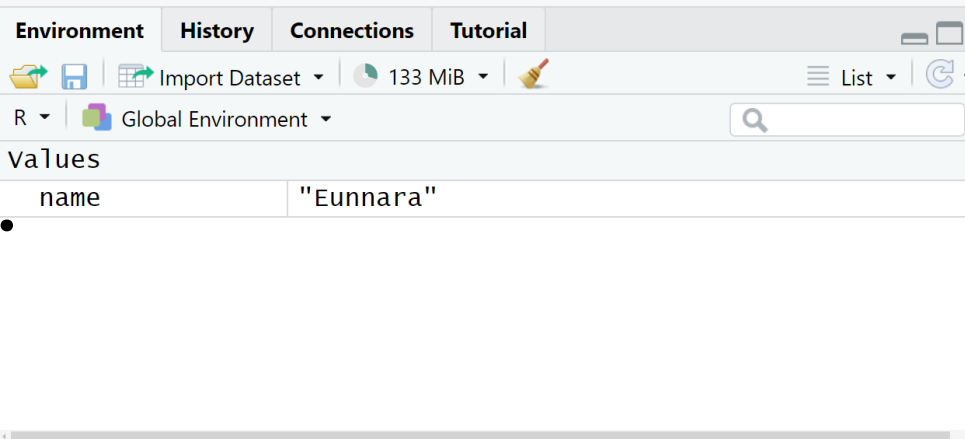
Write and execute your commands

1.

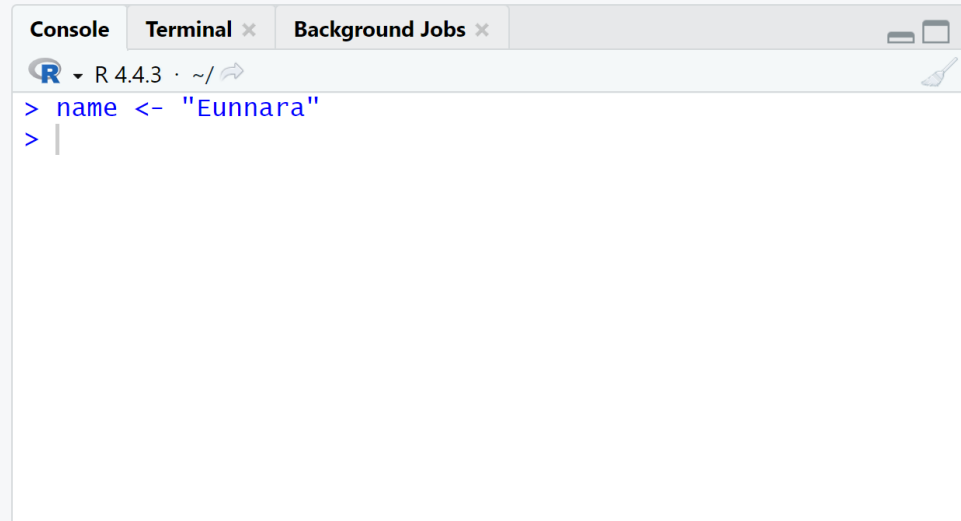


Or

3.



2.



Environment | History | Connections | Tutorial

R | Global Environment

values

name	"Eunnara"

Files | Plots | Packages | Help | Viewer | Presentation

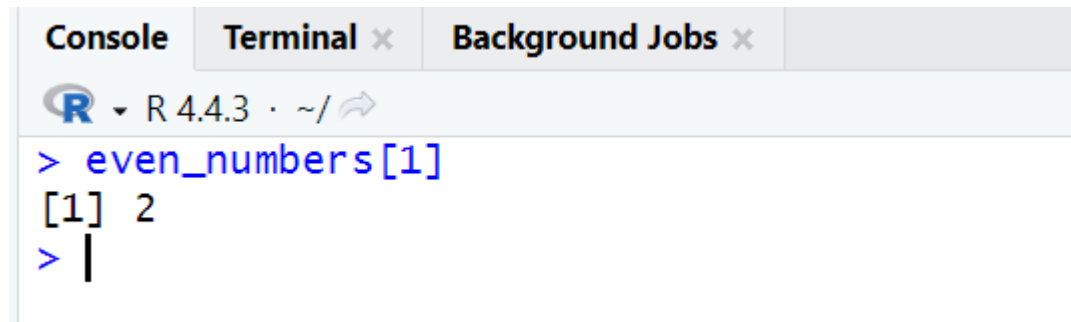
Folder | File | Delete | Rename

Home

Name	Size	Modified
.Rhistory	18.5 KB	Nov 4, 2025, 10:29 AM
4NQO vcf files		
2021 EMGS Meeting Recap.ppt	1.3 MB	Oct 5, 2021, 7:49 PM
2022-0225 SGude - EF + Techniques - ...	5.9 MB	Apr 12, 2022, 9:18 AM
2023.zip	2.8 MB	Dec 21, 2023, 11:56 AM
2025-04-07_CP ref chem_HC+Nyf.jpeg	774.7 KB	Apr 7, 2025, 11:24 AM
2208DMSOTK6-C0A.1.consensus.varia...		
2208DMSOTK6-C0B.1.consensus.varia...		

Indexing data in R

- **Numbering starts from 1**
 - Some programming languages start from 0
- Consider the vector:
 `even_numbers <- c(2, 4, 6, 8)`
 - To access the first number in the vector (2),

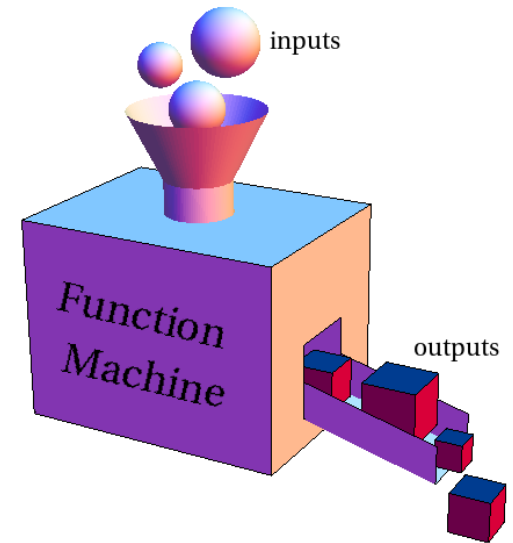
A screenshot of an R console window. The window has three tabs: 'Console', 'Terminal', and 'Background Jobs'. The 'Console' tab is active. The R logo and version 'R 4.4.3' are visible. The command prompt shows the command `even_numbers[1]` being entered, followed by the output `[1] 2`.

```
R 4.4.3 · ~/
> even_numbers[1]
[1] 2
> |
```

- Similarly, for a table/data frame:
 `dataframe[row, column]`

Functions in R

- Set of R statements that are executed in a specific order to process input arguments to produce an output
 - Output can be made into an object
 - Output can be numeric, character, vector, data table, plot
- Reusable for different input
- Hundreds of thousands of premade functions available in R
- Users can write their own functions too
- `function_name(input1, input2,....)`
 - Different functions require a different number of input arguments



Examples of functions

Category	Function
Mathematical Functions	<code>abs()</code> , <code>sqrt()</code> , <code>round()</code> , <code>exp()</code> , <code>log()</code> , <code>cos()</code> , <code>sin()</code> , <code>tan()</code>
Statistical Functions	<code>mean()</code> , <code>median()</code> , <code>cor()</code> , <code>var()</code> , <code>aov()</code>
Data Manipulation Functions	<code>unique()</code> , <code>subset()</code> , <code>aggregate()</code> , <code>order()</code>
File Input/Output Functions	<code>read.csv()</code> , <code>write.csv()</code> , <code>read.table()</code> , <code>write.table()</code>

<https://www.geeksforgeeks.org/r-language/functions-in-r-programming/>

Console Terminal x Background Jobs x

R 4.4.3 · ~/

```
> sqr|
```

sqrt {base}

sqrt(x)

Miscellaneous Mathematical Functions

abs(x) computes the absolute value of x, sqrt(x) computes the (principal) square root of x, `\sqrt{x}`.

The naming follows the standard for computer languages such as C or Fortran.

Press F1 for additional help

Console Terminal x Background Jobs x

R 4.4.3 · ~/

```
> sqrt(25)
[1] 5
> sqrt_25 <- sqrt(25)
> sqrt_25
[1] 5
```

← Executing this command calculates the square root of 25, like a calculator

← You can save the output of sqrt() by creating an object and assigning its value as the square root of 5

Project: (None)

Environment History Connections Tutorial

Import Dataset 177 MiB

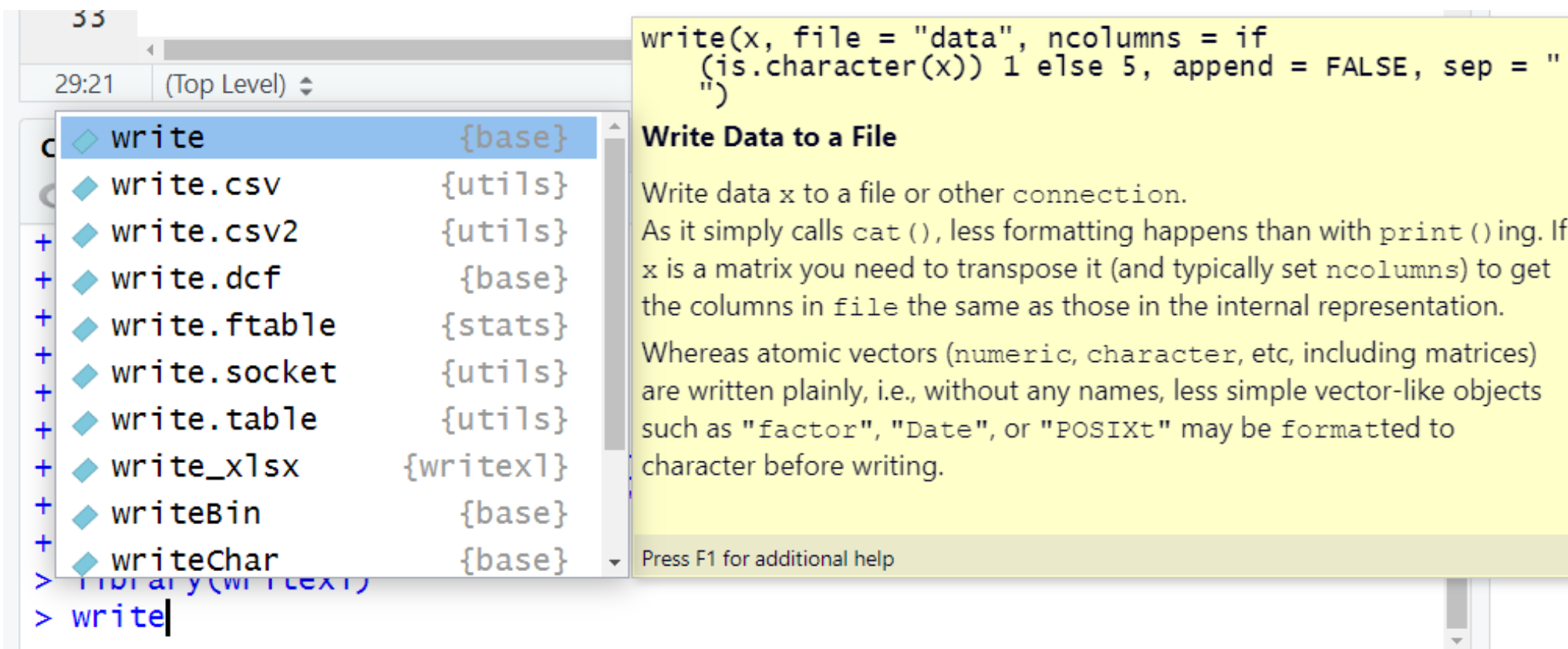
R Global Environment

Values

sqrt_25	5
---------	---

↑ Objects created will show up in the “Environment” tab

Required input arguments



The screenshot shows the RStudio interface. On the left, a file explorer shows a file named '33'. Below it, a console window displays the command `> write`. A dropdown menu is open, listing various write functions: `write` (base), `write.csv` (utils), `write.csv2` (utils), `write.dcf` (base), `write.ftable` (stats), `write.socket` (utils), `write.table` (utils), `write_xlsx` (writexl), `writeBin` (base), and `writeChar` (base). The `write` function is selected. On the right, a documentation panel for the `write` function is displayed. It shows the function signature: `write(x, file = "data", ncolumns = if (is.character(x)) 1 else 5, append = FALSE, sep = "`. Below the signature, the title 'Write Data to a File' is followed by a description: 'Write data x to a file or other connection. As it simply calls `cat()`, less formatting happens than with `print()`ing. If x is a matrix you need to transpose it (and typically set `ncolumns`) to get the columns in file the same as those in the internal representation. Whereas atomic vectors (numeric, character, etc, including matrices) are written plainly, i.e., without any names, less simple vector-like objects such as "factor", "Date", or "POSIXt" may be formatted to character before writing.' At the bottom of the panel, it says 'Press F1 for additional help'.

```
write(x, file = "data", ncolumns = if
      (is.character(x)) 1 else 5, append = FALSE, sep = "
      ")
```

Write Data to a File

Write data x to a file or other connection.

As it simply calls `cat()`, less formatting happens than with `print()`ing. If x is a matrix you need to transpose it (and typically set `ncolumns`) to get the columns in file the same as those in the internal representation.

Whereas atomic vectors (numeric, character, etc, including matrices) are written plainly, i.e., without any names, less simple vector-like objects such as "factor", "Date", or "POSIXt" may be formatted to character before writing.

Press F1 for additional help

The Help tab shows you how to use a function



The screenshot displays the RStudio interface. On the left, the 'Console' tab is active, showing the command `> ?read.csv` entered. A text box with a black border and white background is overlaid on the console, containing the instruction: **To open the help page, execute: `?function_of_interest`**. On the right, the 'Help' tab is active, displaying the usage information for the `read.csv` function. The 'Usage' section shows the syntax for `read.table` and `read.csv` functions, with various arguments and their default values.

To open the help page, execute:
`?function_of_interest`

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"",  
           dec = ".", numerals = c("allow.loss", "warn.loss"),  
           row.names, col.names, as.is = !stringsAsFactors,  
           na.strings = "NA", colClasses = NA, nrows = -1,  
           skip = 0, check.names = TRUE, fill = !blank.line,  
           strip.white = FALSE, blank.lines.skip = TRUE,  
           comment.char = "#",  
           allowEscapes = FALSE, flush = FALSE,  
           stringsAsFactors = FALSE,  
           fileEncoding = "", encoding = "unknown", text, s)
```

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",  
         dec = ".", fill = TRUE, comment.char = "", ...)
```

AI is also a great resource

What's on your mind today?

+ how do I use read.csv to open my file in R?



To use `read.csv()` in R, you just need to provide the path to your file and (optionally) set a few arguments depending on how your data is formatted.

Basic usage

r

```
data <- read.csv("path/to/your/file.csv")
```

 Copy code

What's on the agenda today?

+ Which R function can I use to merge two dataframes together?



To merge two data frames in R, the most common built-in function is:

✓ `merge()`

```
r
merged_df <- merge(df1, df2, by = "id")
```

Copy code

You can also specify:

- `by.x` / `by.y` (if the key columns have different names)
- `all = TRUE` (full join)
- `all.x = TRUE` (left join)
- `all.y = TRUE` (right join)

[No Title]

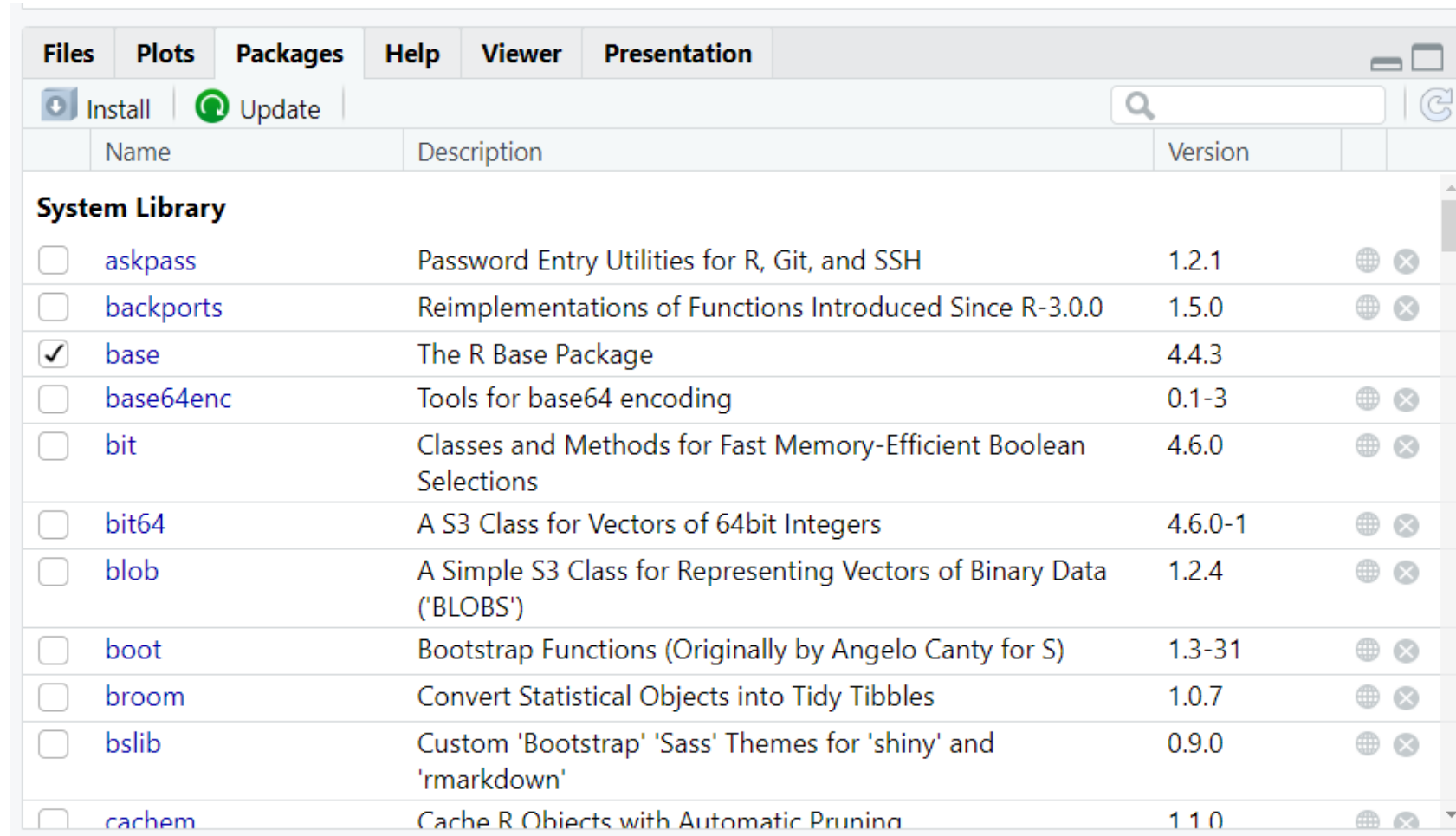
👉 Tidyverse alternatives (from dplyr):

- `left_join(df1, df2, by = "id")`
- `right_join(df1, df2, by = "id")`





















Packages

- Sets of functions that are related to each other
- Some packages are included with R and loaded by default
- Some need to be installed and loaded to use the functions:
 - Function to install packages: **install.packages("package_name")**
 - Function to load packages: **library(package_name)**
- Some packages rely on functions from other packages
 - These other packages are called "dependencies"

The Packages tab shows pre-installed base packages



The screenshot shows the RStudio interface with the 'Packages' tab selected. The 'System Library' section lists several pre-installed packages. The 'base' package is checked, indicating it is installed. Other packages like 'askpass', 'backports', 'base64enc', 'bit', 'bit64', 'blob', 'boot', 'broom', 'bslib', and 'cachem' are listed with their descriptions and versions. The interface includes tabs for Files, Plots, Packages, Help, Viewer, and Presentation, along with buttons for Install and Update.

	Name	Description	Version	
System Library				
<input type="checkbox"/>	askpass	Password Entry Utilities for R, Git, and SSH	1.2.1	 
<input type="checkbox"/>	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.5.0	 
<input checked="" type="checkbox"/>	base	The R Base Package	4.4.3	
<input type="checkbox"/>	base64enc	Tools for base64 encoding	0.1-3	 
<input type="checkbox"/>	bit	Classes and Methods for Fast Memory-Efficient Boolean Selections	4.6.0	 
<input type="checkbox"/>	bit64	A S3 Class for Vectors of 64bit Integers	4.6.0-1	 
<input type="checkbox"/>	blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.2.4	 
<input type="checkbox"/>	boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-31	 
<input type="checkbox"/>	broom	Convert Statistical Objects into Tidy Tibbles	1.0.7	 
<input type="checkbox"/>	bslib	Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'	0.9.0	 
<input type="checkbox"/>	cachem	Cache R Objects with Automatic Pruning	1.1.0	 

Useful packages

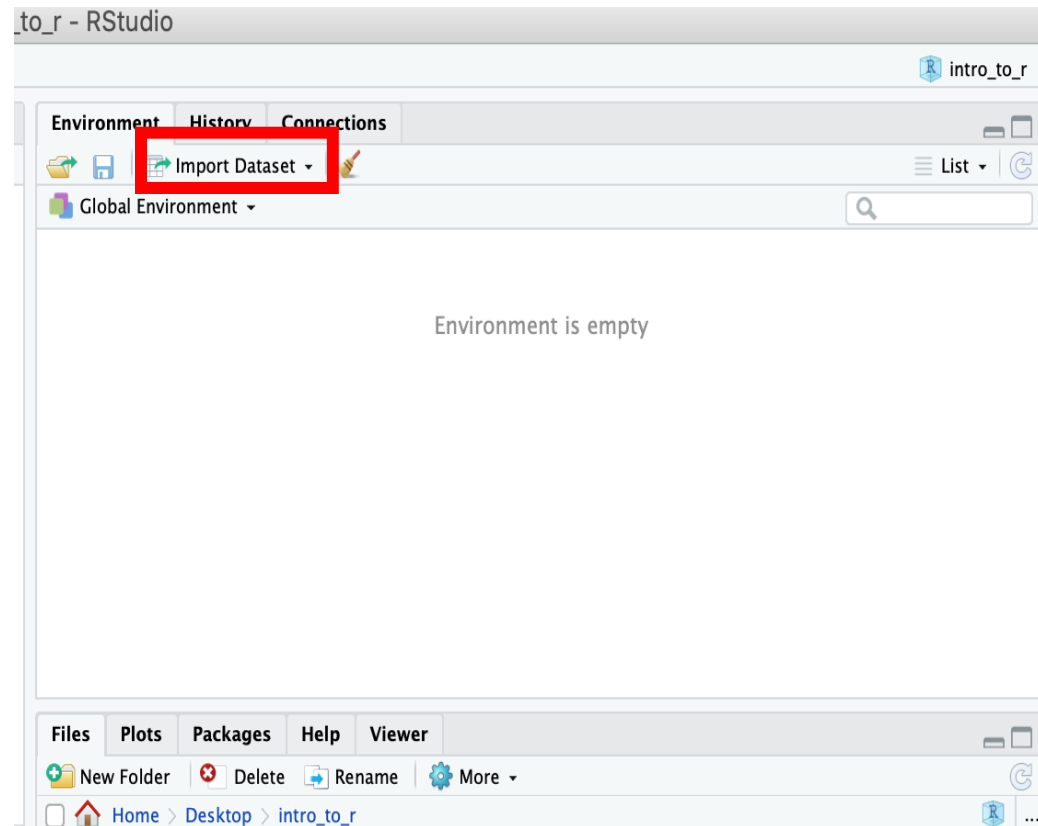
- Tidyverse
 - Collection of R packages for data manipulation and visualization
 - dplyr – manipulate data in tables
 - tidyr – reshaping data
 - readr – import datasets (csv, tsv)
 - stringr – manipulate text in data
 - ggplot2 – data visualization
- readxl and writexl
 - For importing and exporting .xls and .xlsx files

Load your data on RStudio

- You can import common file formats like .xlsx, .txt, .csv, and .tsv
- However, **plain text files (.csv, .tsv) are recommended**
 - Can be opened on any computer
 - Transferable across programming languages
 - No special software required
 - Less chance of corrupted files
- .xls or .xlsx files require Excel, a proprietary software

How to load your data on RStudio

1) Manually import your data into your R environment using the “Import Dataset” button



2) Use functions to import your data

- **`your_data <- read.csv(path_to_file)`**
- **`your_data <- read.table(path_to_file)`**
- **`your_data <- read_xlsx(path_to_file)`**

Example:

```
your_data <- read.csv("C:/Users/EUCHO/OneDrive - HC-SC  
PHAC-ASPC/Documents/sample_data.csv")
```

- Backslashes (\) in the pathway need to be replaced with **forward slashes (/)**

Data frames

- Similar to data tables in Excel spreadsheets
 - Very commonly used data structure in R
- Columns contain variables
 - Different columns can have different types of data (e.g. numeric, character, logical)
- Rows contain observations for each variable
- All columns must have the same length

Animal	Chemical	Dose	Weight	Vehicle
Mouse	ChemA	0	28	SolventA
Mouse	ChemA	10	27	SolventA
Mouse	ChemA	20	25	SolventA
Mouse	ChemA	40	23	SolventA
Mouse	ChemB	0	28	SolventB
Mouse	ChemB	10	26	SolventB
Mouse	ChemB	20	24	SolventB
Mouse	ChemB	40	22	SolventB
Mouse	ChemC	0	28	SolventA
Mouse	ChemC	10	25	SolventA
Mouse	ChemC	20	22	SolventA
Mouse	ChemC	40	17	SolventA

Data visualization: ggplot2 package

- Package for developing graphics
- Plots are built by combining layers of data, aesthetics, and geometry:
 - Data – your dataset (data frame)
 - Aesthetics – define x- and y- axis, other properties like colour
 - Geometry – type of plot (point, line, bar)
- Additional layers can be added to customize your plots
 - Labels
 - Legends
 - Trend lines etc.

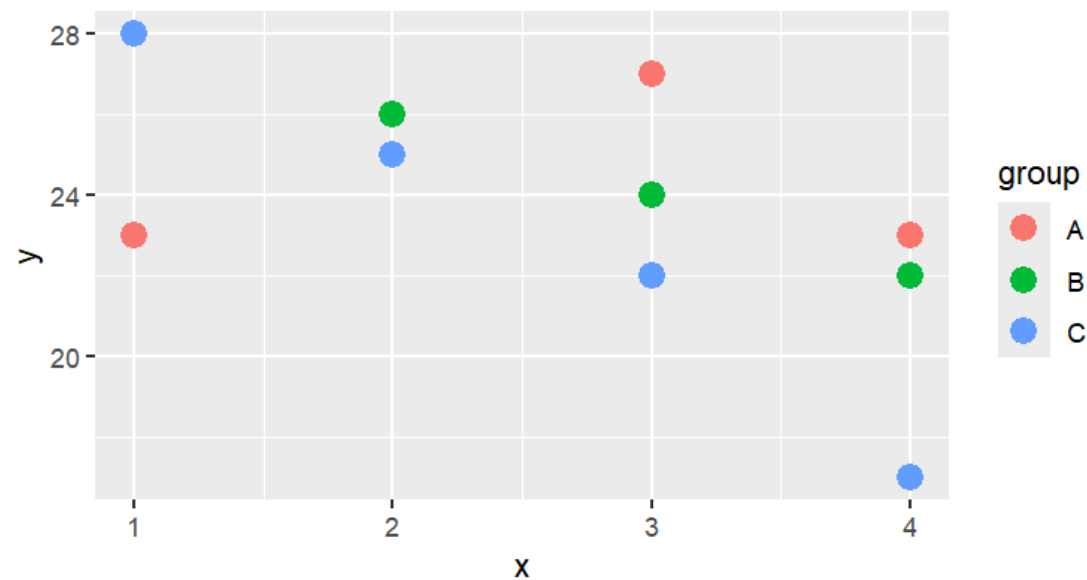
Example ggplot2 code

```
ggplot(your_data, aes(x = x, y = y, color = group)) +  
  geom_point(size = 4)
```

your_data

group	x	y
A	1	23
A	2	25
A	3	27
A	4	23
B	1	28
B	2	26
B	3	24
B	4	22
C	1	28
C	2	25
C	3	22
C	4	17

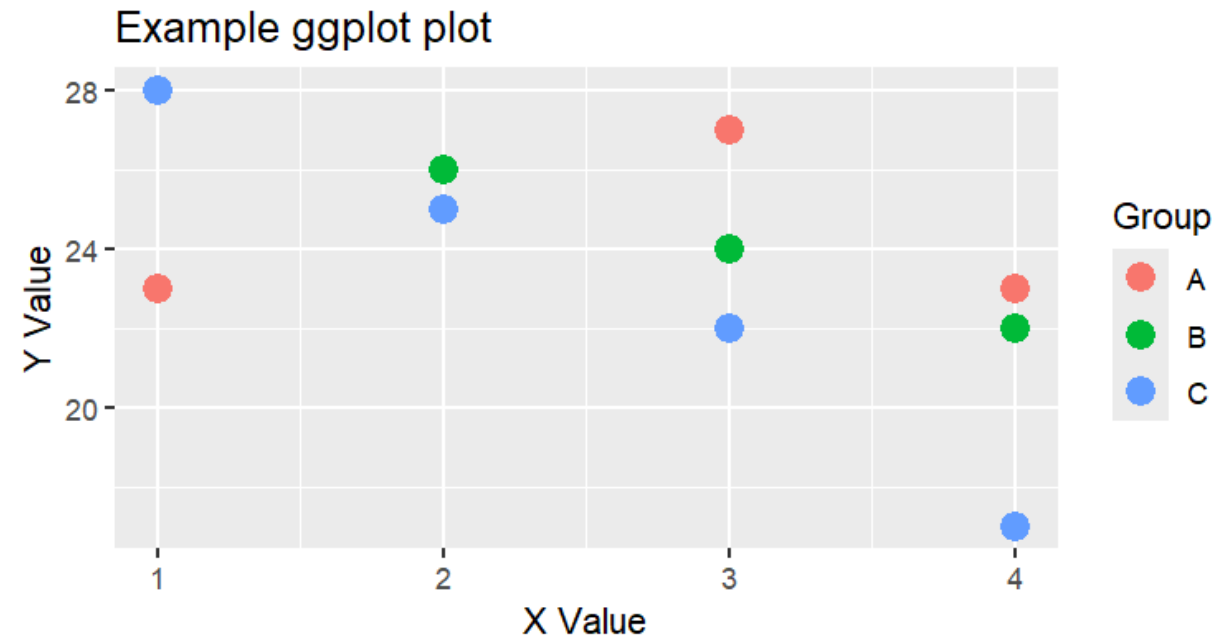
ggplot output



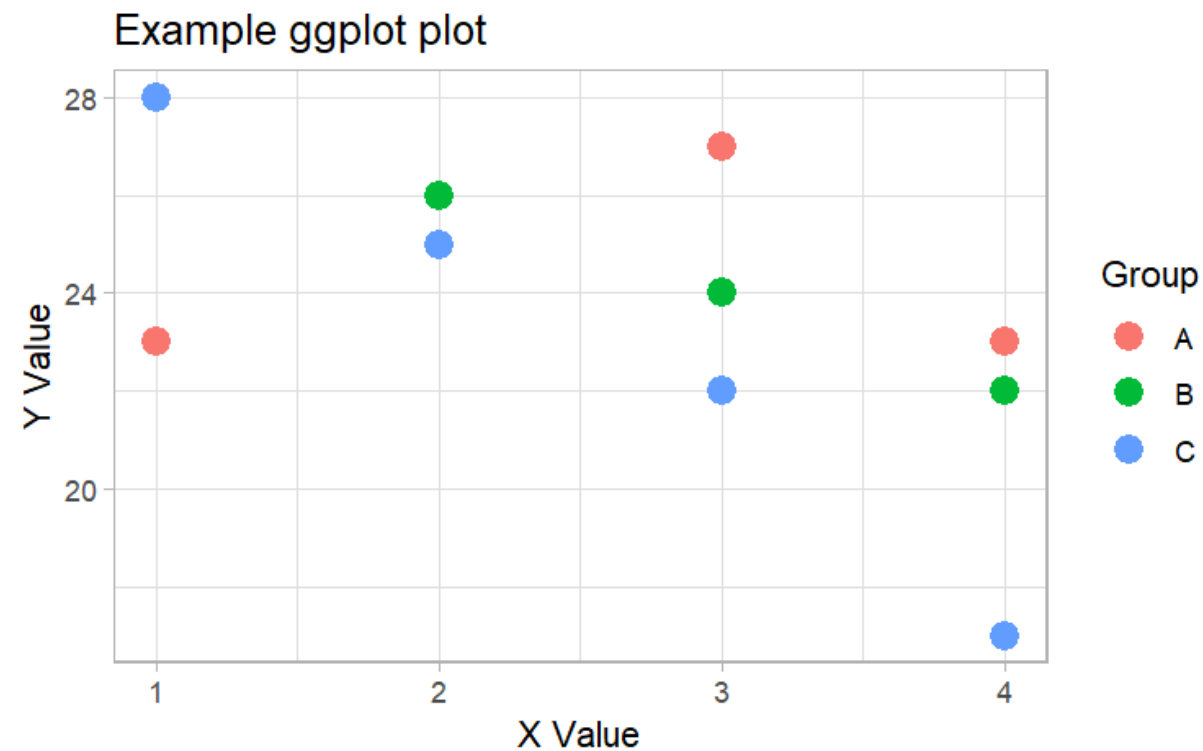

```
ggplot(your_data, aes(x = x, y = y, color = group)) +
```

```
  geom_point(size = 4) +
```

```
  labs(  
    title = "Example ggplot plot",  
    x = "X Value",  
    y = "Y Value",  
    color = "Group"  
  )
```



```
ggplot(your_data, aes(x = x, y = y, color = group)) +  
  geom_point(size = 4) +  
  labs(  
    title = "Example ggplot plot",  
    x = "X Value",  
    y = "Y Value",  
    color = "Group"  
  ) +  
  theme_light()
```



```
ggplot(your_data, aes(x = x, y = y, color = group)) +
```

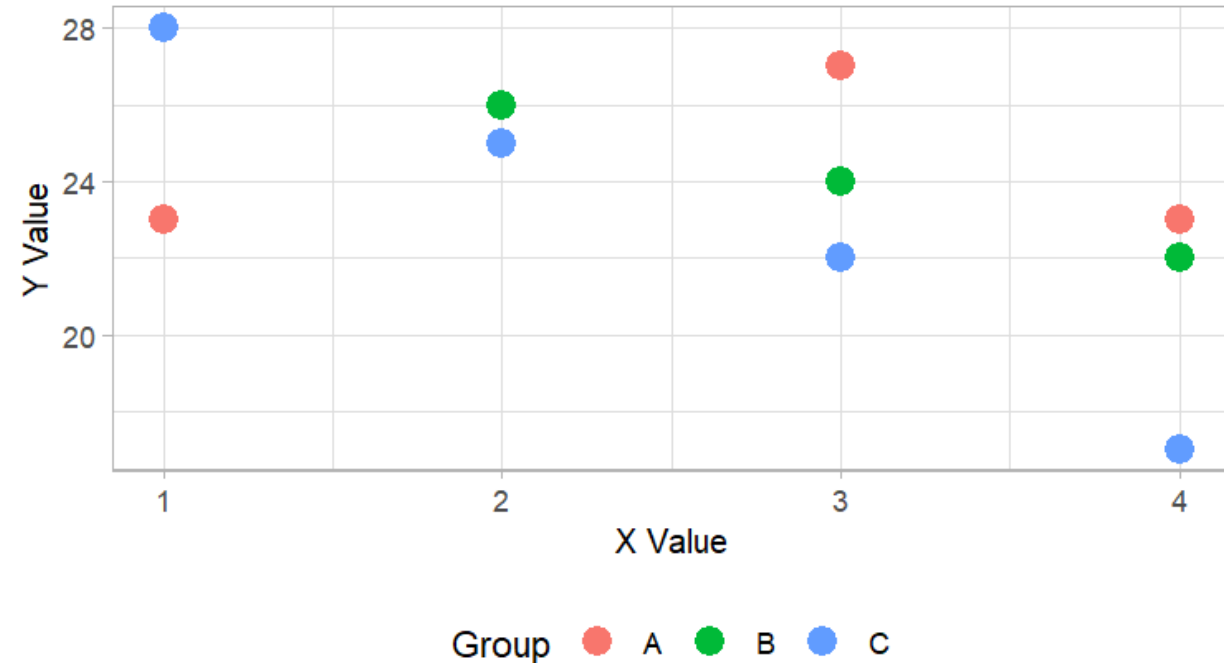
```
  geom_point(size = 4) +
```

```
  labs(
    title = "Example ggplot plot",
    x = "X Value",
    y = "Y Value",
    color = "Group"
  ) +
```

```
  theme_light() +
```

```
  theme(plot.title = element_text(face = "bold",
                                    size = 15),
        axis.title.x = element_text(size = 10),
        axis.title.y = element_text(size = 10),
        legend.position = "bottom"
  )
```

Example ggplot plot



Functions for data reshaping and manipulation

Functions in the tidyverse package can perform similar operations as the Sort & Filter option in Excel:

- `select()`
- `filter()`
- `group_by()`
- `arrange()`
- and many more

Selecting columns: select(data_frame, columns)

```
data_2 <- select(your_data, group, x)
```

your_data

group	x	y
A	1	23
A	2	25
A	3	27
A	4	23
B	1	28
B	2	26
B	3	24
B	4	22
C	1	28
C	2	25
C	3	22
C	4	17



data_2

group	x
A	1
A	2
A	3
A	4
B	1
B	2
B	3
B	4
C	1
C	2
C	3
C	4

Selecting columns: select(data_frame, columns)

```
data_2 <- select(your_data, Chemical:Weight)
```

your_data

Animal	Chemical	Dose	Weight	Vehicle
Mouse	ChemA	0	28	SolventA
Mouse	ChemA	10	27	SolventA
Mouse	ChemA	20	25	SolventA
Mouse	ChemA	40	23	SolventA
Mouse	ChemB	0	28	SolventB
Mouse	ChemB	10	26	SolventB
Mouse	ChemB	20	24	SolventB
Mouse	ChemB	40	22	SolventB
Mouse	ChemC	0	28	SolventA
Mouse	ChemC	10	25	SolventA
Mouse	ChemC	20	22	SolventA
Mouse	ChemC	40	17	SolventA



data_2

Chemical	Dose	Weight
ChemA	0	28
ChemA	10	27
ChemA	20	25
ChemA	40	23
ChemB	0	28
ChemB	10	26
ChemB	20	24
ChemB	40	22
ChemC	0	28
ChemC	10	25
ChemC	20	22
ChemC	40	17

Columns can also be specified by numbers

```
data_2 <- select(your_data, 2:4)
```

your_data

Animal	Chemical	Dose	Weight	Vehicle
Mouse	ChemA	0	28	SolventA
Mouse	ChemA	10	27	SolventA
Mouse	ChemA	20	25	SolventA
Mouse	ChemA	40	23	SolventA
Mouse	ChemB	0	28	SolventB
Mouse	ChemB	10	26	SolventB
Mouse	ChemB	20	24	SolventB
Mouse	ChemB	40	22	SolventB
Mouse	ChemC	0	28	SolventA
Mouse	ChemC	10	25	SolventA
Mouse	ChemC	20	22	SolventA
Mouse	ChemC	40	17	SolventA



data_2

Chemical	Dose	Weight
ChemA	0	28
ChemA	10	27
ChemA	20	25
ChemA	40	23
ChemB	0	28
ChemB	10	26
ChemB	20	24
ChemB	40	22
ChemC	0	28
ChemC	10	25
ChemC	20	22
ChemC	40	17

Filtering rows: filter(data_frame, filtering criteria)

- To set the filtering criteria, **operators** can be used:

For comparing two values:

Operator	Meaning	Example
==	equal to	5 == 5 → TRUE
!=	not equal to	5 != 3 → TRUE
<	less than	2 < 3 → TRUE
>	greater than	5 > 8 → FALSE
<=	less than or equal	3 <= 3 → TRUE
>=	greater than or equal	4 >= 5 → FALSE

For combining logical values:

Operator	Meaning	Example
&	AND	TRUE & FALSE → FALSE
 	OR	
!	NOT	!TRUE → FALSE

- Other than operators, functions that return TRUE/FALSE may be used to filter:
 - is.na(column)
 - grepl("word", column)
 - starts_with("word")

Filtering rows: filter(data_frame, filtering criteria)

```
data_2 <- filter(your_data, group == "A")
```

Values in the Group column are characters and, thus, need to be specified using “” in the filtering criteria

your_data

group	x	y
A	1	23
A	2	25
A	3	27
A	4	23
B	1	28
B	2	26
B	3	24
B	4	22
C	1	28
C	2	25
C	3	22
C	4	17



data_2

group	x	y
A	1	23
A	2	25
A	3	27
A	4	23

More than one filtering criteria

your_data

group	x	y
A	1	23
A	2	25
A	3	27
A	4	23
B	1	28
B	2	26
B	3	24
B	4	22
C	1	28
C	2	25
C	3	22
C	4	17

group == "A" & y < 25



x == 1 | y > 25



group == "B" &
(y < 25 | x == 1)



group != "B" &
(y > 25 | x == 1)



data_2

group	x	y
A	1	23
A	4	23

group	x	y
A	1	23
A	3	27
B	1	28
B	2	26
C	1	28

group	x	y
B	1	28
B	3	24
B	4	22

group	x	y
A	1	23
A	3	27
C	1	28

Combining multiple functions with pipes (%>%)

- To chain a series of dplyr functions together, use the pipe operator %>%
- The pipe operator will pass the result of a function on to the next function
- Instead of:
 data_2 <- filter(data_1, x > 15)
 data_3 <- select(data_2, group, y)
- Below will result in the same dataframe data_3:
 data_3 <- data_1 %>%
 filter(x > 15) %>%
 select(group, y)

Grouping and summarizing

- You can perform an operation on a group of rows at once using **group_by(column with the grouping variable)** and **summarise()**:

```
data_2 <- your_data %>%  
  group_by(Dose) %>%  
  summarise(mean_weight = mean(Weight))
```

Animal	Chemical	Dose	Weight
1	ChemA	0	28
2	ChemA	10	27
3	ChemA	20	25
4	ChemA	40	23
5	ChemA	0	28
6	ChemA	10	26
7	ChemA	20	24
8	ChemA	40	22
9	ChemA	0	28
10	ChemA	10	25
11	ChemA	20	22
12	ChemA	40	17



Dose	mean_weight
0	28.0000
10	26.0000
20	23.6667
40	20.6667

Sorting data: `arrange(column to sort by)`

- Ascending order:

```
your_data %>%  
  arrange(group)
```

- Descending order:

```
your_data %>%  
  arrange(desc(group))
```

- If you want to sort by the grouping variable first, use **`.by_group=TRUE`**:

```
your_data %>%  
  group_by(Chemical) %>%  
  arrange(Dose, .by_group=TRUE)
```

Saving your output in your working directory

- `write.csv(data_2, file = "file_name.csv")`
- `write.table(data_2, file = "file_name.txt")`
- `write_xlsx(data_2, file = "file_name.xlsx")`

- `ggsave(filename = "plot_name.jpeg", plot = your_plot, height = x, width = y)`

- You can also enter the complete file path in the file name if you want to save your file somewhere else other the working directory

Resources

- Dr. Kristin Eccles' Introduction to R Github repository:
 - https://github.com/kristineccles/Introduction_to_R
- ggplot2 Cheatsheet
 - <https://rstudio.github.io/cheatsheets/html/data-visualization.html>
- dplyr Cheatsheet
 - <https://rstudio.github.io/cheatsheets/html/data-transformation.html>
- ChatGPT