Introduction to R

Shunsuke Tsuda

EC114 Introduction to Quantitative Economics
2026 Autumn



Contents

Today (Lab 1)

- Features of R, an open-source programming language
- Install R and RStudio (on your laptop)
  - Once you've installed R and RStudio, you can also work on R programming at home
  - At Lab 1 on November 10 (Week 6, IT Lab L, Albert Sloman Library (LIB.G.5.0)), please feel free to bring your own laptop, although the computers in the lab should also have R and RStudio available
- Basic mathematical operations and calculations in R
- Work with data in R

Next session (Lab 2)

- Descriptive statistics
- Data visualization

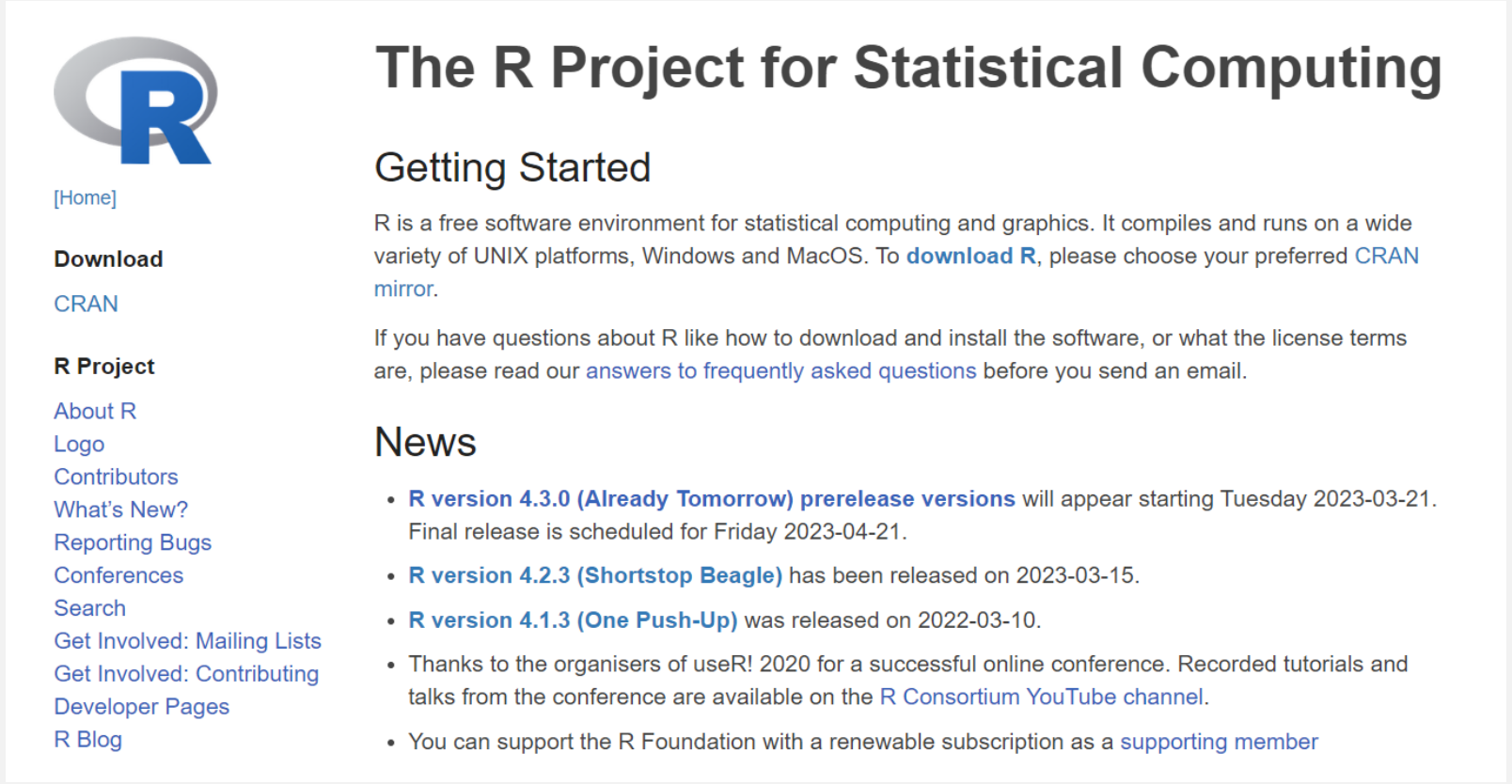
I acknowledge Tomohiro Hara for sharing his lecture notes, on which these slides are partly based.

Why Use R for Data Analysis?

- Because it’s free
  - Excel, used in general society, and Stata, often used by economists, are paid software
- Many methods used in economic analysis are already available as functions
  - By using packages, you can execute processes without writing detailed programs yourself
- You can save past commands, allowing you to reproduce past processes
  - ”Reproducibility” is very important in scientific research

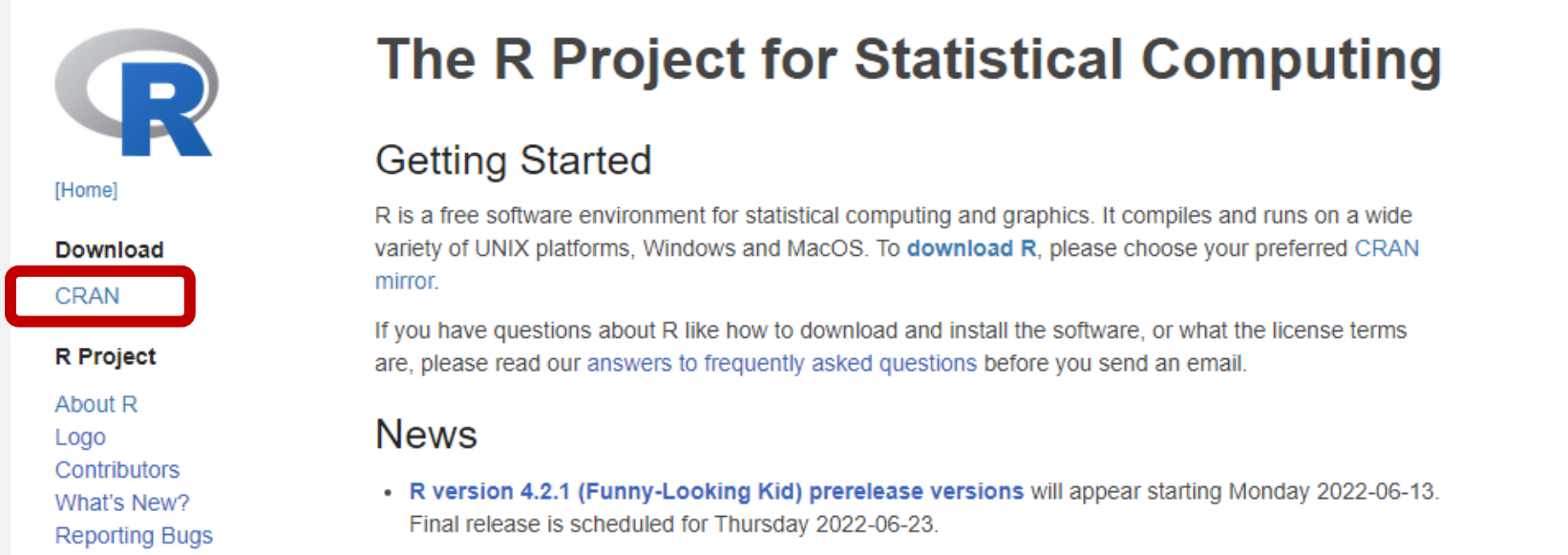
Let’s Install R!

- Search for “R statistics” or access https://www.r-project.org/



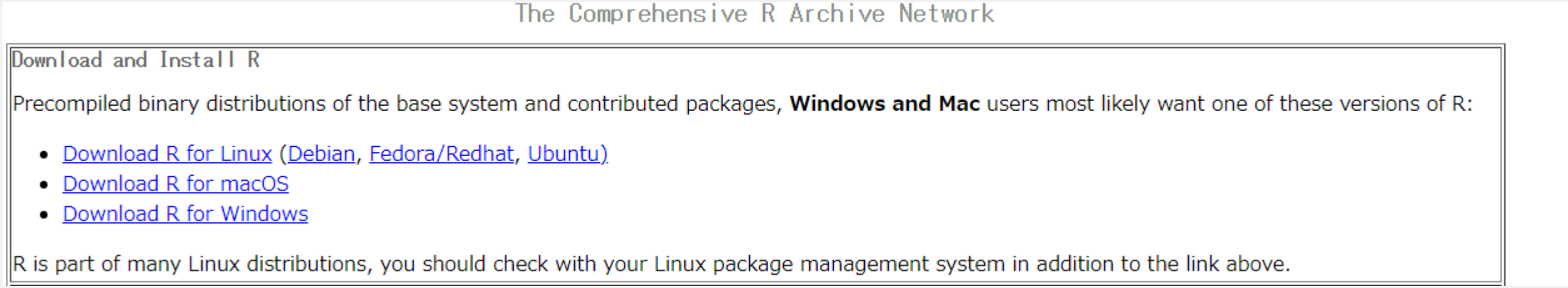
R is stored on multiple mirror sites

- Click CRAN (Comprehensive R Archive Network)
- You can download from any mirror site, but it is typically fast and recommended to download from institutions that are geographically close (UK in our case)



Download according to your OS

- Mac users click "R for macOS", Windows users click "R for Windows"
- Save the file to "Downloads" (For Windows: "R-4.2.3-win.exe", for Mac: "R-4.2.3.pkg")
- Double-click the downloaded exe or pkg file
- If a message like "Do you want to allow this app to make changes to your device?" appears, select "Yes”



Proceeding with Setup Options

- Language to use is English
- In “Select Components”, choose “User installation”
- In “Startup options”, choose “No (accept defaults)”
- Check "Create a desktop shortcut
- Check all items under the "Registry entries" section
- For Mac, you will need to enter your PC's username and password
- Indeed, you can continue to click “Next” with the default setup options that appear

What is RStudio?

- An application for R programming
- Statistical analysis is possible with R alone without RStudio, but RStudio has an abundance of convenient packages, making work easier

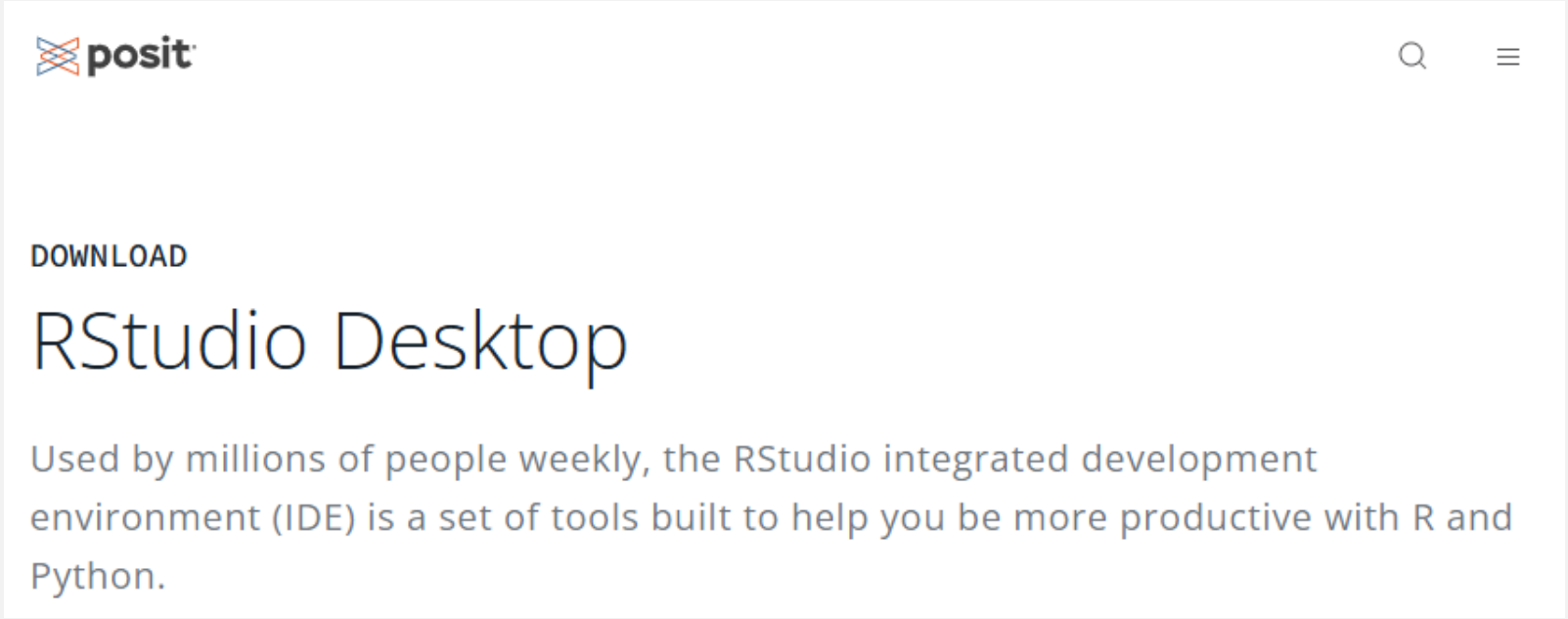
- R can also be used with other applications
  - E.g.1 Visual Studio Code (VS Code): Besides R, multiple languages like Python, Julia, and C/C+ can be used
  - E.g.2 Google Colaboratory: Online coding is possible with a Google account. It's mainly for machine learning with Python, etc., but R is also available





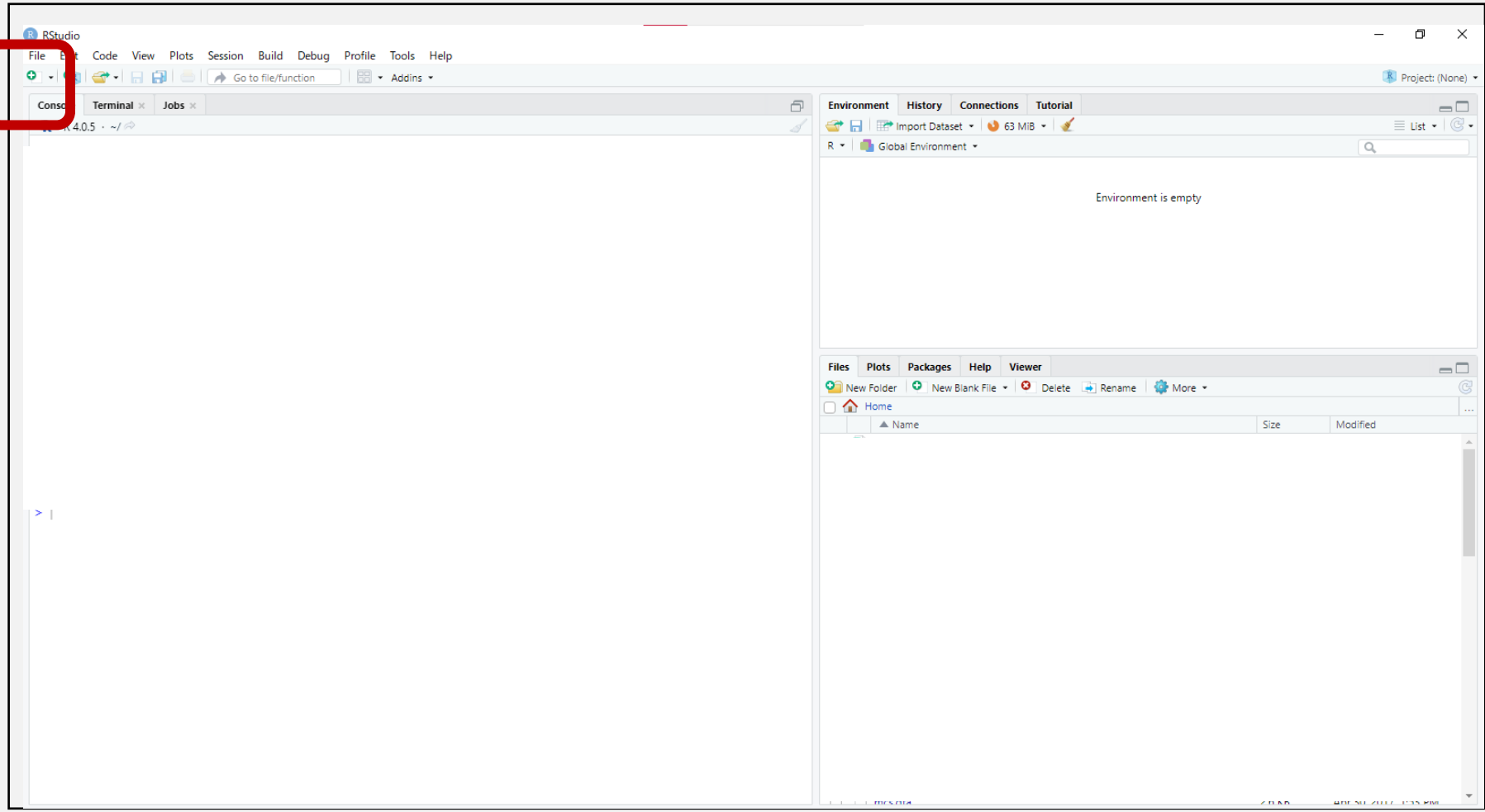

Let’s install RStudio!

- Search for “RStudio” or access https://www.rstudio.com/ or https://posit.co/download/rstudio-desktop/
- Again, download the one that matches your OS (Supports both Windows and Mac)

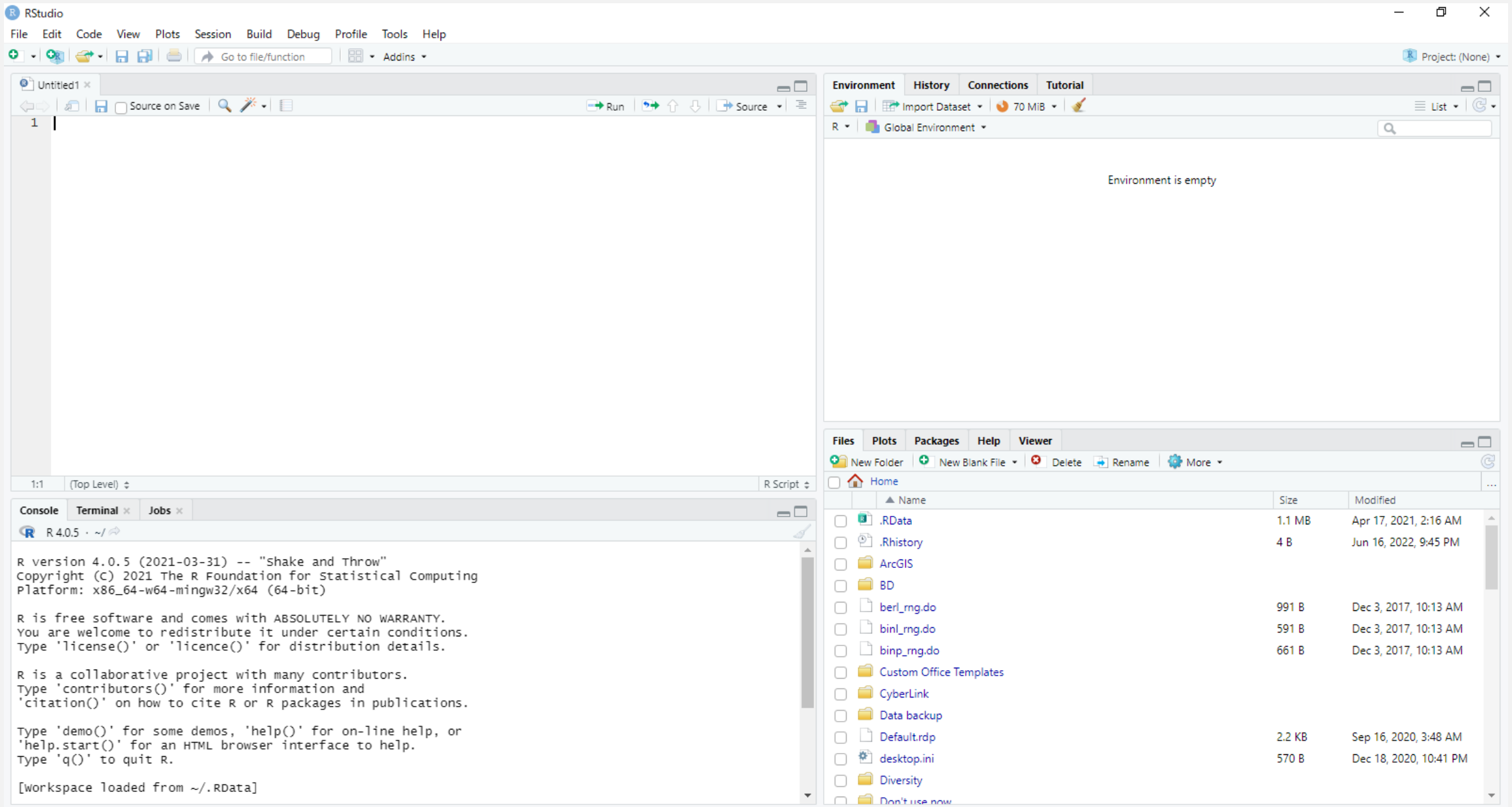


Start using RStudio

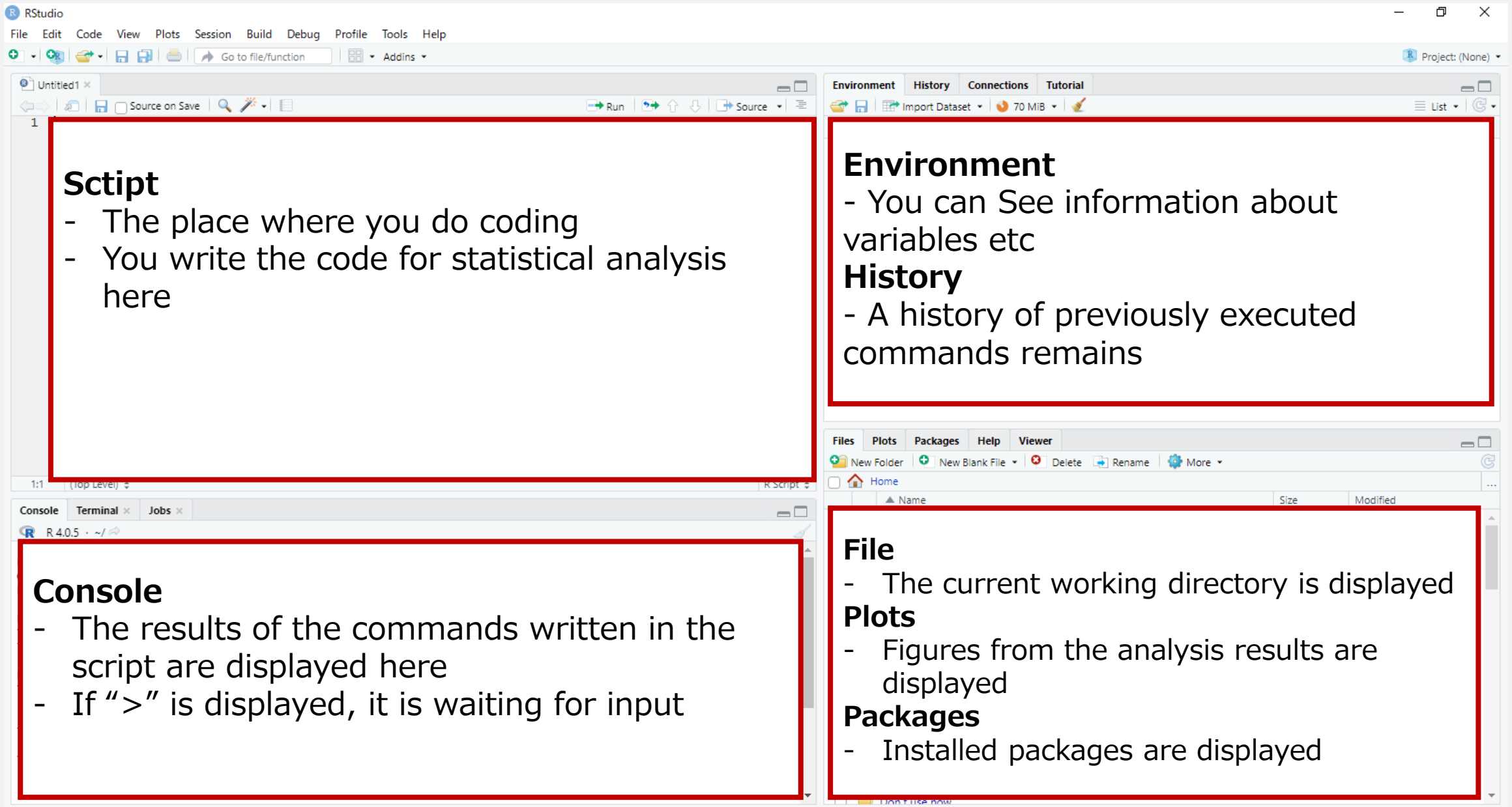
- Start RStudio (Search for it if it's not on your desktop)
- Click the "+" in the upper left and select "R Script"
  → The page should become four-paned



What can we do in each window in RStudio?



What can we do in each window in RStudio?



Sctipt
- The place where you do coding
- You write the code for statistical analysis here

Environment
- You can See information about variables etc

History
- A history of previously executed commands remains

Console
- The results of the commands written in the script are displayed here
- If “>” is displayed, it is waiting for input

File
- The current working directory is displayed

Plots
- Figures from the analysis results are displayed

Packages
- Installed packages are displayed

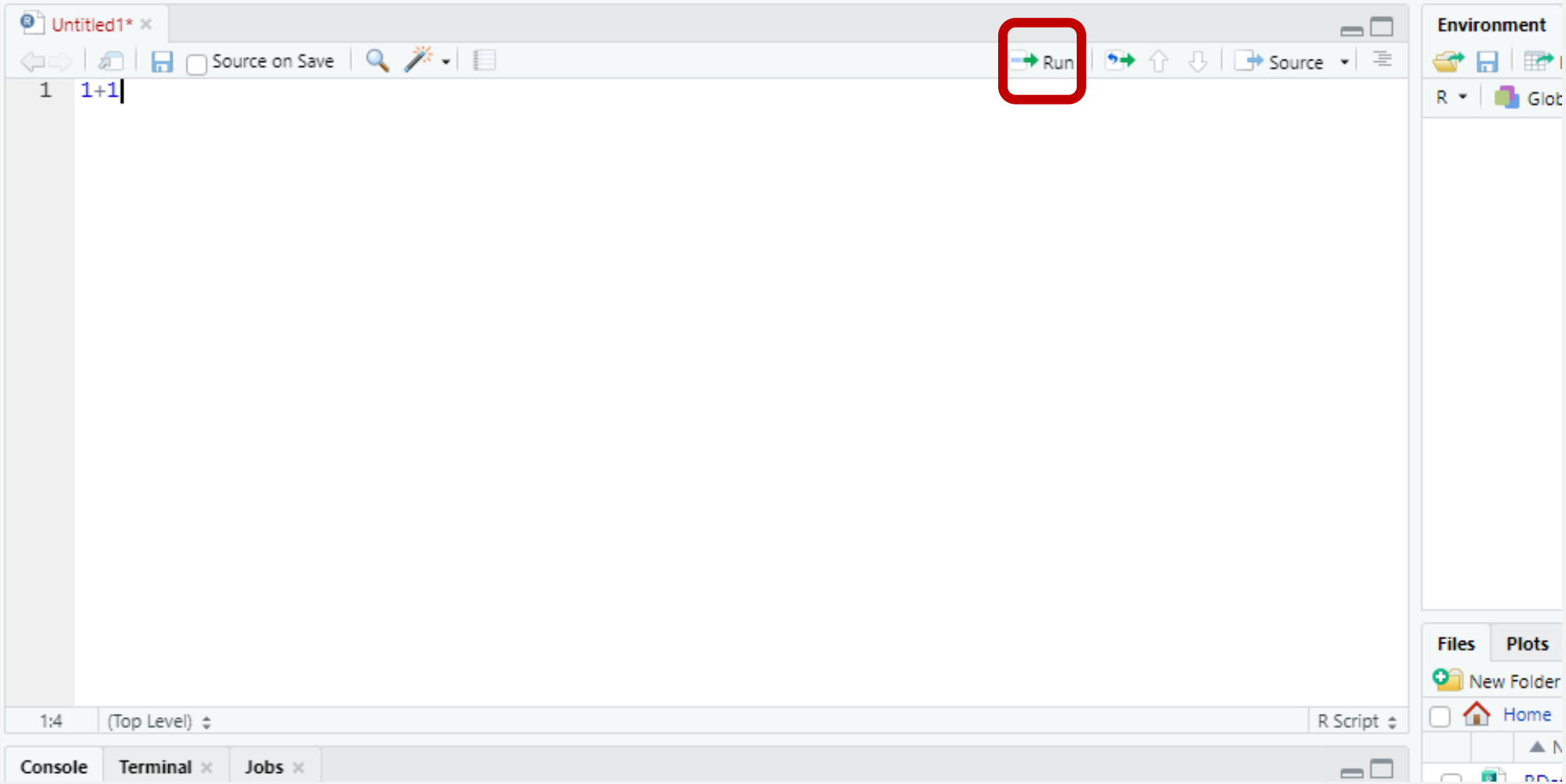
The layout can also be changed

- The layout may differ depending on settings
- From Tools in the manu bar (top of page), Global Options → Panel Layout
  - A 2x2 menu will be displayed, so choose what to display in each slot
- In Global Options, you can also change the font, size, screen color (dark mode), etc

Let’s try some simple calculations!

- Type the following formula into the script and click "Run"

$$1+1$$



Basic Operations (Let's try a few)

- `-` : Subtraction (e.g. `5-2`)
- `*` : Multiplication (e.g. `5*2`)
- `/` : Division (e.g. `5/2`)
- `^` : Exponentiation (e.g. `5^2`)
- `sqrt()` : Square root (e.g. `sqrt(9)`)
- `log()` : Natural logarithm (e.g. `log(9)`)
- `log(x,y)` : Natural logarithm of x with base y (e.g. `log(9)`)
- `exp()` : Exponential function (e.g. `exp(9)`)
- `floor()` : Round down (e.g. `floor(9.5)`)
- `ceiling()` : Round up (e.g. `ceiling(9.4)`)
- `round()` : Round (e.g. `round(9.5)`)

Exercise 1: Calculate the following expressions

1) $5 \times (2 - 1)$

2) The number $5\sqrt{3} - 2$ rounded

3) e^{20}

4) $Y = AL^{0.3}K^{0.7}$ with $A = 2, L = 10, K = 50$

Exercise 1: Answers

1) $5 \times (2 - 1)$

```
5*(2-1)= 5
```

2) The number $5\sqrt{3} - 2$ rounded

```
round(5*sqrt(3)-2)= 7
```

3) e^{20}

```
exp(20)= 485165195
```

4) $Y = AL^{0.3}K^{0.7}$ with $A = 2, L = 10, K = 50$

```
2*(10^0.3)*(50^0.7)= 61.70339
```

Tip: It's better to write it in one go without splitting the function as much as possible

Console & Script

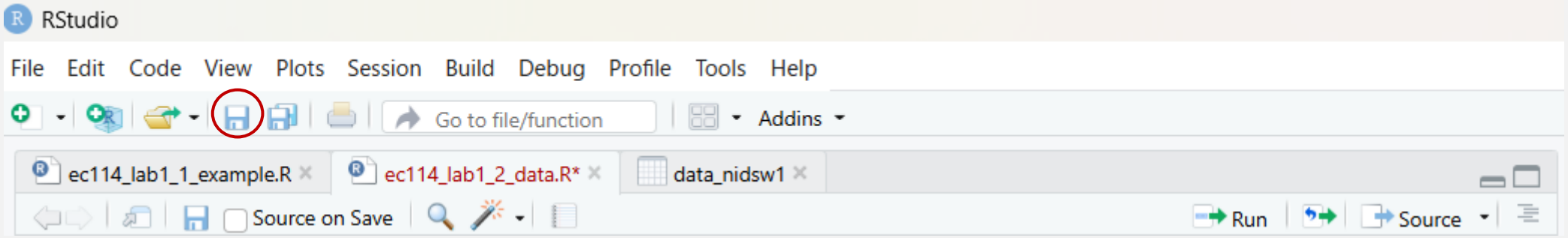
- R commands can be directly entered and executed in the Console
- However, it is better to write them in an R script and create a file with the ”.R”extension

Why?

- If you input directly into the Console, it is difficult to reproduce past operations
- With an R Script, all operations are recorded, and you can reproduce the same result anytime
- Choose the area to run, then press the "Run" button or Control+Enter
- Or, Control+A+Enter to run the entire code
- Easy to share operations with other people

Saving the Source Code File

- Save the source code written in the Script in the code folder
  - To be able to reproduce the same analysis in the future by re-running the same code
  - Reproducibility is a very important item in data analysis
  - Again, it is desirable that anyone running the code gets the same result
- You can save by pressing the floppy disk button in the editor (or just by pressing Control + S). Do it often!



You can also create your own functions

■ As an example, define the function $y = 3x$ as `example1`

```
example1 <- function(x){
y <- 3*x
return(y)
}
```

- Write the name of the function to be defined first
- The content enclosed in {} is the body of the function
- The 2nd line is the actual formula
- The result is printed by `return(y)`

■ After that

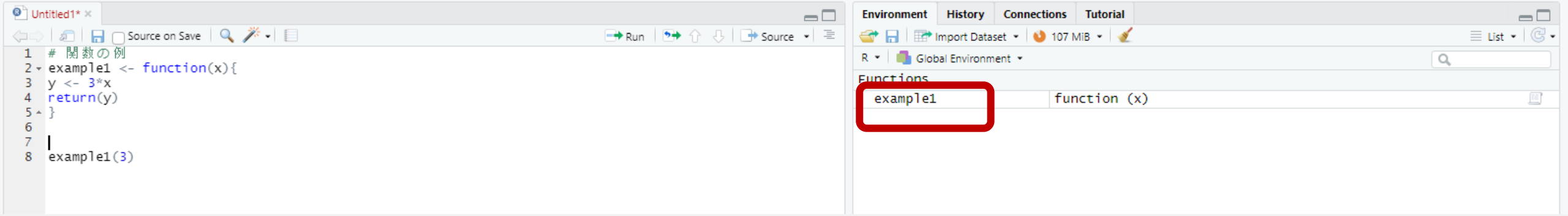
```
example1(3)
```

etc., inputting a value of x will output the value of y

■ The items specified within the function's () are called arguments

Custom functions can be checked

- The created function is saved in the Environment



- `rm(example1)` can delete the function from Environment
- When are custom functions useful?

Functions with 2 variables can also be defined

```
example2 <- function(x,y){
z <- x*y
Return(z)
}
Example2(2,3)
```

- Similarly, functions with n variables can also be defined
- The return value can also be displayed in an easy-to-understand way
  - Change the `return(z)` in the `example2` function above to `Cat("product of",x,"and",y,"are",z)`
  - `cat()` is a function that displays variable values and text
  - Words enclosed in `" "` are recognized as text

Exercise2

Create the following functions and calculate by substituting the actual values

1) $z = x^2 + y^2,\ x = 2, y = 3$

2) $U = \log(x) + \log(3y)\,,\ x = 4, y = 15$

3) $Y = AL^{0.3}K^{0.7}, A = 2, L = 10, K = 50$

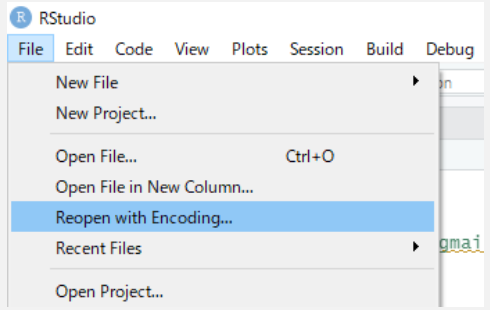
Commenting Out

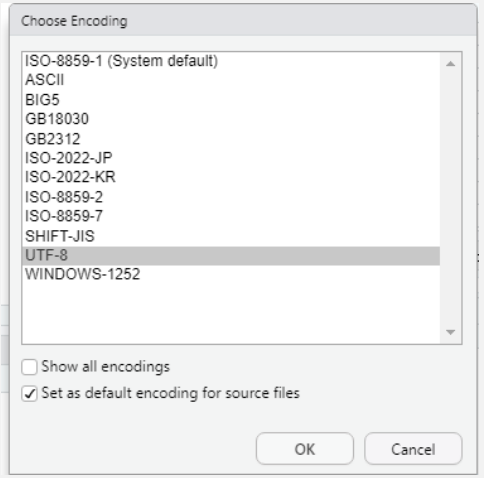
- You can add comments to make the written code easier to understand (this is called commenting out)
- Comments/text written after # are not calculated



Encoding

- Sometimes text information (like text variables in data or commenting out) cause garbled text:
  - It is good to use UTF-8, which is a global standard
  - It is good to set the default to UTF-8
  - If the text is garbled when you read data, try opening the file with a different encoding method
- In the upper left, "File" -> "Reopen with Encoding..." -> Select "UTF-8" and check "Set as default encoding for source files"





Use Packages to Shorten Code

- R has various packages created by researchers and others in addition to pre-installed packages
  - There are countless packages, so it is important to gather information by yourself
    1. Search for "xxx in R" (where xxx is the action you want to implement)
    2. Search from the CRAN package list (Enter the keyword you want to search with Ctrl+f)
  - Check for the existence of a package before trying difficult (or seemingly difficult) actions yourself
  - When installing a package from CRAN:
    `install.packages("[package_name]", dependencies = TRUE)`
- Or, type the package installation command directly into the Console
  `install.packages("[package_name]")`
- After installing the package, you need to load it with `library([package_name])` in the script

When you don't know how to use a package or function

- `help([function_name])`
  → How to use the function `[function_name]`
- `args([function_name])`
  → Display the arguments of the function `[function_name]`
- `example([function_name])`
  → Display usage examples of the function `[function_name]`
- `help(package=[package_name])`
  → How to use the package `[package_name]`
- Of course, searching online is also fine

Saving and Loading Data

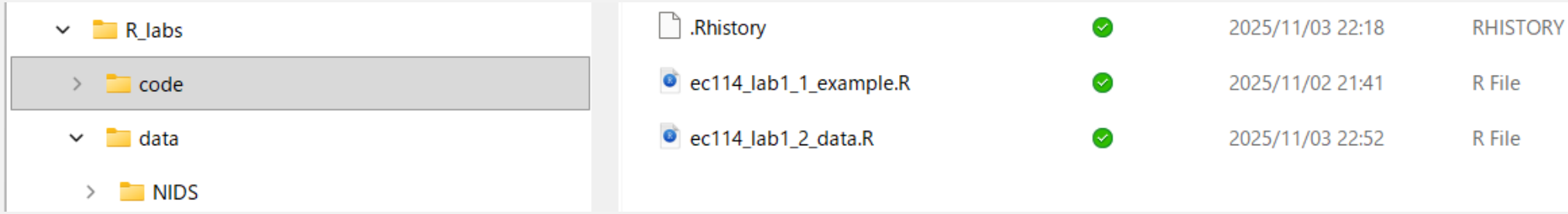
- Load a CSV format file
- Download ”nids_w1_for_educ.csv” into the “data/NIDS” folder
- The data this time is processed for practice, so do **NOT** use it for other classes or reports
  - The data can also be downloaded from the NIDS website
  - If you want to analyze the data further by yourself, please download the data from NIDS and use it
  - Data processing requires more skill and judgement than you might imagine

Data Used: NIDS (National Income Dynamics Study)

- Household-level data from South Africa
  - A nationally representative sample of the country's population
  - 7,300 households and 28,000 individuals
  - Panel data: Surveyed the same households and individuals 5 times between 2008 and 2017
  - Surveyed various indicators via interviews, such as poverty, household well-being, household composition, health, and labor market participation
  - Many variables are publicly available
  - (Some variables that could lead to personal identification—such as detailed location, date of birth, and detailed occupation—can only viewed and analyzed by researchers on-site)

Setting Directories: Only relative positions matter!

- A directory is like an “address” that indicates the file’s save location
- Now we have directories for storing code and data
- To import the data by running the code, only the relative position matters (wherever they are on your computer):



- Then, you can import the csv data using a relative path:

```
17  data_nidsw1 = read.csv(file='../data/NIDS/nids_w1_for_educ.csv')
```

- `../` The code navigates to the directory immediately above the current directory = the location of the code itself (if you open the code by clicking the file in this folder and run it)
- `../../../` Likewise, the code navigates to three steps above

Use a Relative Path for Portability

- Code and data must be migrated to other machines in various situations:
  - Buy a new computer
  - In a more powerful server at the university
  - Run a code on your collaborator's computer
- **Be "portable": Code should work in any machines without any changes**
- Tips for improving portability:
  - Fix rules for the directory structure and be consistent with it throughout a project and among collaborators
  - Write code that does not depend on a machine-specific path: For this purpose, we wrote a **relative path**
  - Then, any computer can run the code without any changes in the code, as long as the directory structure is fixed

Avoid Using an Absolute Path, which is Not Portable

- A bad example with an absolute path that is machine-specific:

```
69  # set working directory
70  setwd("C:/Users/Tomohiro/Dropbox/TeachingMaterials/MicroDataAn/2022/Data/NIDS/w1")
```

- Suppose Tomohiro and I are working on a joint project
- Tomohiro sets the working directory using an absolute path on his computer
- However, if I need to run the same R script, I have to change it to the absolute path on my own computer..
- This is **NOT portable**!

Instead Use a Relative Path for Portability

- Type the package installation command directly into the Console `install.packages("this.path")`
- After installing the package, you need to load it with `library(this.path)` in the script
- You can then use this package to set your working directory to the location of your script: `setwd(this.dir())`
- You can then load the data using a relative path (i.e. the location of the data relative to the current working directory):

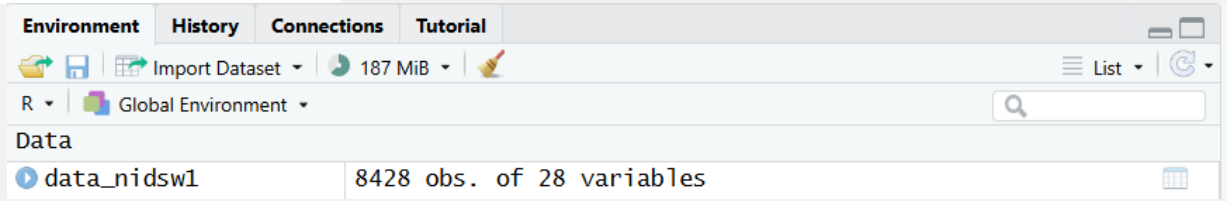
```
 8  library("this.path")
 9  setwd(this.dir()) # Use a relative path, which makes the entire project "portable"
10  getwd()           # You can check the current path
11  ##### Inputs
12  data_nidsw1  = read.csv(file='../data/NIDS/nids_w1_for_educ.csv', header=T)
```

Reading Data

`[data name]=read.csv(file='[file name]', header=T)`:
Load data

- `[data name]` is the data name displayed in the Environment after reading
  - In this example, it is `data_nidsw1`
- `[file name]` is the name of the saved file
  - This time, it is `nids_w1_for_educ.csv`
- `header=T` is when the first line of the CSV file contains variable names (`T` can also be `True`)
- `header=F` means the data starts from the first line

```
16 data_nidsw1 = read.csv(file='../data/NIDS/nids_w1_for_educ.csv', header=T)
17
```



- If no error is displayed in the console and the data name appears in the Environment, the loading was successful

Data Wrangling: Using Libraries

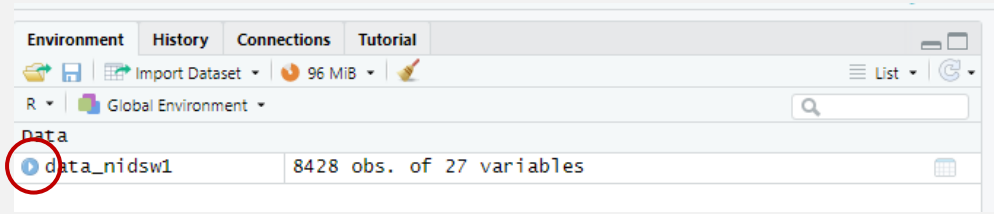
- R has "standardly implemented functions (`mean`,`var`, etc) and more advanced methods that are not standardly implemented
- This time, we will use a package called `dplyr` for data wrangling
  - `library("[package_name]")` → `library("dplyr")`
  - If you get an error message `"there is no package called 'dplyr'"`, the package is not installed, so installation is necessary

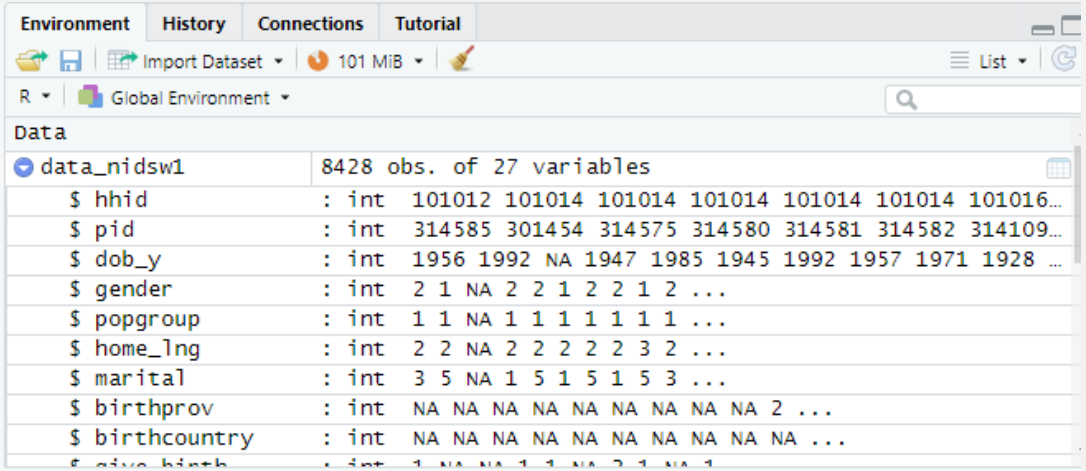
Data Wrangling: Installing Packages

- For methods that are not installed, you need to install packages
- Once a package is installed, there is no need to install it again
  - To display a list of packages installed in R, type `library()` in the console
  - To display a list of packages that are currently loaded, , type `search()` in the console
- Type the package installation command directly into the Console `install.packages("dplyr")`
- After installing the package, you need to load it with `library(dplyr)` in the script
- It is ok to ignore a message "`The following objects are masked from…`"

Checking the Data

- Click the triangle button next to the data in the Environment window to see a list of variables and some of the data content
  - 8428obs. of 28 variables

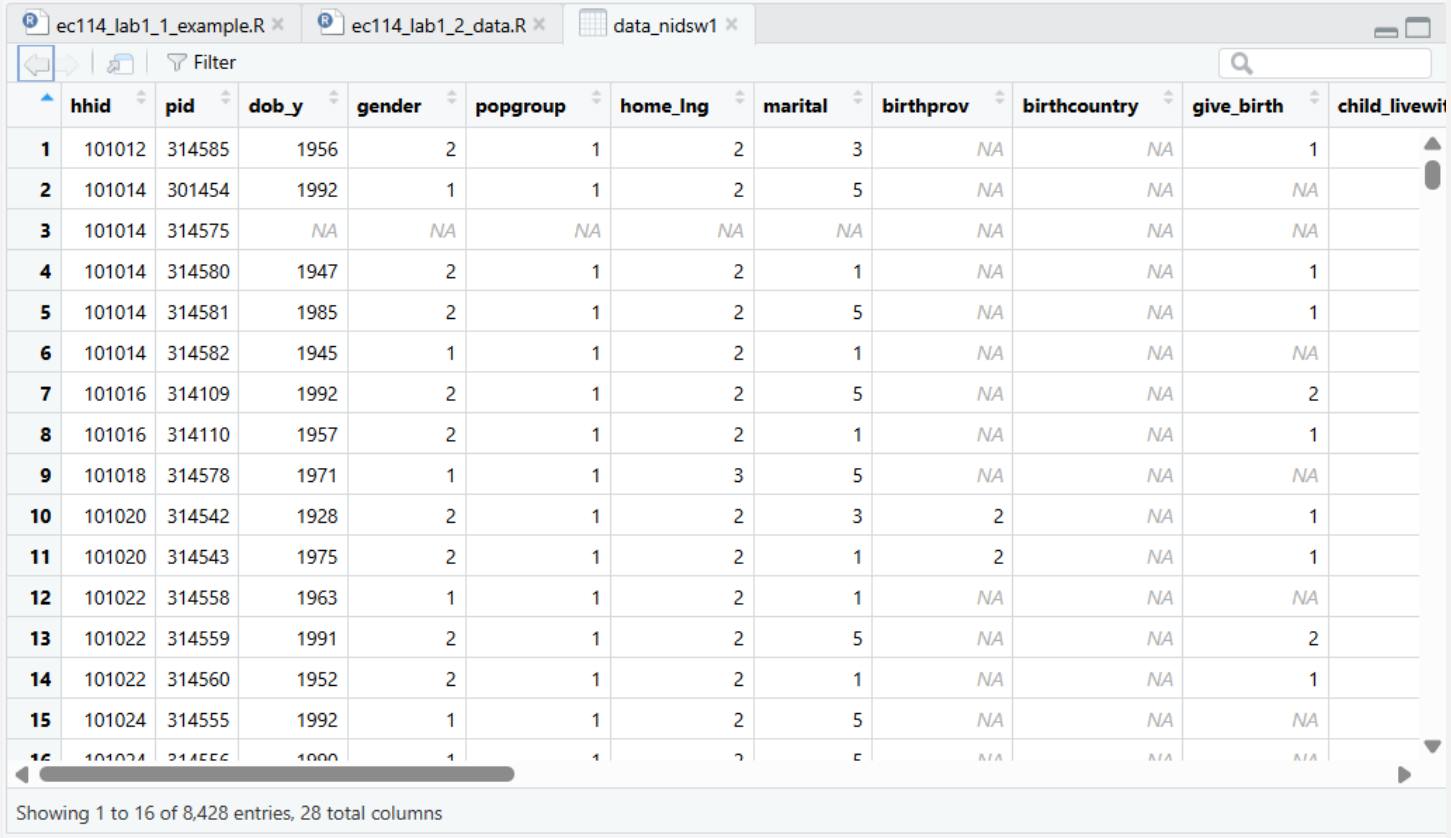




  - You can also check the first 6 rows of data with `head(data_nidsw1)`
  - It is important to first check if the data has been loaded as intended and expected

Checking the Entire Data

- `View(data_nidsw1)`: By typing this, a new tab will open in the Source window, and you can check the entire data (you can double-click the data in Environment)
- Missing data is displayed as `N/A` (abbreviation for Not Available)


ec114_lab1_1_example.R | ec114_lab1_2_data.R | data_nidsw1

Filter

	hhid	pid	dob_y	gender	popgroup	home_lng	marital	birthprov	birthcountry	give_birth	child_livewit
1	101012	314585	1956	2	1	2	3	NA	NA	1	
2	101014	301454	1992	1	1	2	5	NA	NA	NA	
3	101014	314575	NA	NA	NA	NA	NA	NA	NA	NA	
4	101014	314580	1947	2	1	2	1	NA	NA	1	
5	101014	314581	1985	2	1	2	5	NA	NA	1	
6	101014	314582	1945	1	1	2	1	NA	NA	NA	
7	101016	314109	1992	2	1	2	5	NA	NA	2	
8	101016	314110	1957	2	1	2	1	NA	NA	1	
9	101018	314578	1971	1	1	3	5	NA	NA	NA	
10	101020	314542	1928	2	1	2	3	2	NA	1	
11	101020	314543	1975	2	1	2	1	2	NA	1	
12	101022	314558	1963	1	1	2	1	NA	NA	NA	
13	101022	314559	1991	2	1	2	5	NA	NA	2	
14	101022	314560	1952	2	1	2	1	NA	NA	1	
15	101024	314555	1992	1	1	2	5	NA	NA	NA	

Showing 1 to 16 of 8,428 entries, 28 total columns

Checking Variables

- Microdata usually comes with not only the data itself (.csv) but also its "manual":
  - Variable list (codebook): A list of variable definitions
  - Questionnaire: The questions asked during the actual survey
  - Technical paper: A technical note summarizing the sample design (random sampling method, etc.)
- For example, in this data..
  - `gender` a variable taking values 1 and 2, but it's impossible to tell which is male and which is female from the csv file alone
  - `educ` is a variable indicating years of education, but it can take negative values like -5 or -3
    - -5 indicates Not applicable, -3 indicates Missing, but this is not clear from the data itself
    - Analyzing these as-is will lead to strange results!
  - This data is a partially processed version of the original NIDS data
- Checking and wrangling data before data analysis is very important

Checking Variables (cont'd)

- If the variable does not take many possible values, you can create a table to output a list of possible values and their observation counts

```
table(data_nidsw1$gender)
```

```
> table(data_nidsw1$gender)

   1    2
3121 4661
```

Data Wrangling: Generating New Variables

- As an example, generate `log_income` as the logarithm of the variable `income`

```
data_nidsw1 <- mutate(data_nidsw1,
log_income=log(income))
```

- Inside `mutate()`..
  - Use `data_nidsw1` as the data
  - Generate a new variable `log_income=log(income)`

Data Wrangling: Generating Variables with ifelse (1)

- You can create a variable that meets specific conditions by using the `ifelse` function within `Mutate`
- For example, generate a variable that is "1 if White, 0 otherwise" from the `popgroup` variable
  - `popgroup` is a variable representing the sample's race
  - `African=1, Coloured=2, Asian/Indian=3, White=4` (see codebook)

```
data_nidsw1 <- mutate(data_nidsw1, white=ifelse(popgroup == 4, 1, 0))
```

  - The arguments in `ifelse()` are 1) condition, 2) action if condition is met, 3) action if condition is not met
  - Use a double equals sign == to indicate a specific condition
  - In this case, when `popgroup` is 4, `white` is 1, otherwise 0

Data Wrangling: Generating Variables with ifelse (2)

- If the data is a categorical variable (`African==1`, `Coloured==2`, `etc`) that categorizes attributes into values, you can check if the variable was created as intended by making a simple table

```
with(data_nidsw1, table(popgroup, white))
```

- A table like the one on the right will appear in the console

- The combination of `popgroup==1` (`African`) and `white==1` is 0
- The combination of `popgroup==4` (`white`) and `white==0` is also 0, so it is as intended

popgroup	white 0	1
-3	19	0
1	6131	0
2	1102	0
3	95	0
4	0	435

Data Wrangling: Generating Variables with ifelse (3)

- You can add multiple conditions by putting another `ifelse` in the 3rd argument
-
```
data_nidsw1 <- mutate(data_nidsw1,
white_or_coloured=ifelse(popgroup == 4, 1,
ifelse(popgroup == 2, 1, 0)))
```
- When the first condition is not met (in this example, when not `White (popgroup==4)`), it proceeds to the second `ifelse`
- The second `ifelse` is 1 if `Coloured(popgroup==2)`, otherwise 0
- In principle, you can keep adding as many `ifelse`s as you want

Data Wrangling: Generating Variables with ifelse (4)

- The process from the previous page can also be done with the following code

```
data_nidsw1 <- mutate(data_nidsw1,
white_or_coloured_rev=ifelse(popgroup == 4, 1, 0))
```

```
data_nidsw1 <- mutate(data_nidsw1,
white_or_coloured_rev=ifelse(popgroup == 2, 1,
white_or_coloured_rev))
```

- The first block of code is: "if `popgroup == 4`, then 1, otherwise 0“
- The second block of code is: "if `popgroup == 2`, then 1, otherwise keep as `white_or_coloured_rev`“

Data Wrangling: Handling Missing Values

- In R data, missing values are represented as NA
- In some original data, values that could not be collected in the questionnaire survey are sometimes replaced with other values -> Must check the codebook
- In this data as well, for example, `popgroup` has -3 for missing values -> want to change to NA

```
data_nidsw1 <- mutate(data_nidsw1,
popgroup=na_if(popgroup,-3))
```

- Use the `na_if` function with `mutate`
  - Argument 1 is the target variable, Argument 2 is the value entered in place of the missing value

Data Wrangling: Renaming Variables

- Use `rename` when the variable name is hard to understand
- In this data, the variable `dob_y` represents the birth year, but it is unclear
  - Want to change it to the name `birth_year`

```
data_nidsw1 <- rename(data_nidsw1, birth_year = dob_y)
```

Data Wrangling: Extracting Only a Subset of Data

- You might want to extract only a subset of data for checking or analysis purposes
- You can extract a subset of data using `filter`
  - Generate a new data frame and store the extracted data in it
- To extract only the data for White (`white==1`)

```
data_nidsw1_white <- filter(data_nidsw1, white==1)
```

- `data_nidsw1_white` is the new data frame name
- Argument 1 is the data frame to use, Argument 2 is the condition (this time, `white==1`)
  - You can add multiple conditions by writing `&` for "and" and by writing `|` for "or"

Data Wrangling: Other Main Useful Operations

- Extract only certain variables → `select`
  - E.g.1 `new_data1<-select(data_nidsw1, hhid, pid, gender)`
  - E.g.2 `new_data2<-select(data_nidsw1, -hhid, -pid)`
    - In this case, variables **other than** those marked with – are extracted
- Sorting the data based on variable values → `arrange`
  - E.g.3 (Ascending): `new_data3<-arrange(data_nidsw1, birth_year)`
  - E.g.4 (Descending): `new_data4<-arrange(data_nidsw1, desc(birth_year))`
  - E.g.5 You can also sort the data based on two variables: `new_data5<-arrange(data_nidsw1, birth_year, desc(pid))`

Exercise 3: Perform the following operations

1) Create a variable named "`female`" that is 1 if female, 0 if male, and check if the variable was created (hint: the `gender` variable is 1 for male, 2 for female)

```
data_nidsw1 <- mutate(data_nidsw1, female=ifelse(gender==2, 1, 0))
with(data_nidsw1, table(gender,female))
```

2)`Income` has -3 entered as a missing value, so change it to `NA`

```
data_nidsw1 <- mutate(data_nidsw1, income=na_if(income,-3))
```

3) `Income` is in South African Rand. Convert it to GBP and create a new variable (`income_GBP`) (hint: 1 Rand ≒ 0.05 GBP)

```
data_nidsw1 <- mutate(data_nidsw1, income_GBP=income*0.05)
```

4) Create a data frame of only White female

```
data_nidsw1_female <- filter(data_nidsw1, white==1 & female==1)
```

Exercise 3: Perform the following operations

1) Create a variable named "`female`" that is 1 if female, 0 if male, and check if the variable was created (hint: the `gender` variable is 1 for male, 2 for female)

```
data_nidsw1 <- mutate(data_nidsw1, female=ifelse(gender==2, 1, 0))
with(data_nidsw1, table(gender,female))
```

2) `Income` has -3 entered as a missing value, so change it to `NA`

```
data_nidsw1 <- mutate(data_nidsw1, income=na_if(income,-3))
```

3) `Income` is in South African Rand. Convert it to GBP and create a new variable (`income_GBP`) (hint: 1 Rand ≒ 0.05 GBP)

```
data_nidsw1 <- mutate(data_nidsw1, income_GBP=income*0.05)
```

4) Create a data frame of only White female

```
data_nidsw1_female <- filter(data_nidsw1, white==1 & female==1)
```

4') Similarly, create a data frame of only African or Coloured

Saving Data

- Save the data after wrangling is finished
- Use the `readr::write_rds()` package

```
readr::write_rds(data_nidsw1,
file="../data/cleaned/nids_w1_for_educ_cleaned.Rds")
```

- The first argument is the data frame to save, the second argument is the file name to save as
  - `.Rds` is R's data format
  - Other extensions like `.csv` can also be used if you want to read it in other software (in this case, use `readr::write_csv()`)
- **Absolutely do NOT save it with the same name as the original data file**
  - If you delete/update the original data file itself, you will not be able to reproduce the analysis you performed

A Tip for Clearer Code Writing

- Rather than writing the output data file name when saving like:

```
82  readr::write_rds(data_nidsw1, file="../data/cleaned/nids_w1_for_educ_cleaned.Rds")
```

- Clearer to specify inputs and outputs at the beginning:

```
 9  ##### Inputs
10  data_nidsw1   = read.csv(file='../data/NIDS/nids_w1_for_educ.csv', header=T)
11
12  ##### Outputs
13  data_cleaned = "../data/cleaned/nids_w1_for_educ_cleaned.Rds"
```

- Then, when you save at the end, you can just write

```
84  readr::write_rds(data_nidsw1, file=data_cleaned)
```

- **Code also has a documentation purpose:**
  - You can immediately see inputs and outputs of the code
  - Document *minimally* by commenting out

Code as Self-Documentation

- You will see the importance of clear code writing as the project scale enlarges (with many inputs and outputs, many lines, many functions, many packages, etc)..

RStudio Source Editor

generate_location_figures.R

Source on Save | Run | Source

```r
### Last modified: SO 20250703

rm(list = ls(all.names = TRUE)) #will clear all objects includes hidden objects.
gc() #free up memrory and report the memory usage.

# Load packages
if (!require("pacman")){install.packages("pacman")}
# remove.packages("osmdata")
pacman::p_load(tidyverse, haven, sf, osmdata, ggmap, terra, tidyterra)

raw_data <- "../../../data/targeting/raw_cleaned/targeting_cleaned/"
raw_data_others <- "../../../data/targeting/raw/"
for_analysis <- "../../../data/targeting/for_analysis/"
output_figure <- "../../../drafts_outputs/figures/targeting/"
output_figure_general <- "../../../drafts_outputs/figures/"
external_data <- "../../../data/external_data/"

# Input files
data_input_raw_cleaned <- paste0(raw_data, "targeting_raw_combined_cleaned_since202501.dta")
link_data_path <- paste0(for_analysis, "targeting_cleaned_long_link_level_data_since20250128.dta")
image_input <- paste0(external_data, "satellite_imagery/satellite_imagery.tif")
image_input_Nairobi <- paste0(external_data, "satellite_imagery/satellite_imagery_Nairobi.tif")
gadm_ken_3_path <- paste0(raw_data_others, "gadm41_KEN_shp/gadm41_KEN_3.shp")
gadm_ken_2_path <- paste0(raw_data_others, "gadm41_KEN_shp/gadm41_KEN_2.shp")
manual_Nairobi_map_nei <- paste0(external_data, "boundaries/boundaries_neiborhood.shp")
manual_Nairobi_map_commu <- paste0(external_data, "boundaries/boundaries_community.shp")
bbox_Nairobi <- paste0(external_data, "satellite_imagery/Nairobi_box.shp")
bbox_study_area <- paste0(external_data, "satellite_imagery/study_area.shp")

# Load data
data <- read_dta(data_input_raw_cleaned)
```

Toward Productive Project Management (Advanced)

If you are interested in doing empirical research by yourself, the following resources are useful to write better code and organize data effectively:

- Gentzkow & Shapiro (2014) “Code and Data for the Social Sciences: A Practitioner’s Guide” (LINK)
  - This is a must-read as a starting point

- Tsuda (2024) “Software Engineering for Social Scientists” (LINK)
  - Tips for improving **portability**, **clarity and maintainability**, **accuracy**, **efficiency**, and **reproducibility** of research

References for R Programming

- The R Project for Statistical Computing (LINK)
- Hands-On Programming with R (LINK)
- Stack overflow (LINK)
- R as GIS for Economists (LINK) if you are interested in spatial data and analysis

But before spending a lot of time with books/manuals..

- Whenever you get an error, you can do a google search for it. Most problems can be easily resolved by this.
- You can also seek help from AI (e.g. ChatGPT, Claude, Google Gemini)