The Basics of R

Tadao Hoshino （星野匡郎）

Econometrics II: ver. 2019 Spring Semester

Introduction to R

What is **R**?

- **R** is free and open source computer software for statistical analysis.
- **R** has no license limitations. You can install and run it anytime and anywhere.
- One of the greatest advantages of **R** is that users of **R** can distribute their own packages (extensions) through **R** server, called CRAN,[1] and we can implement the state-of-the-art statistical tools quite easily.
- For now, **R** and **Python** are two of the most popular programming languages for statistical analysis.[2]

[1] Comprehensive R Archive Network

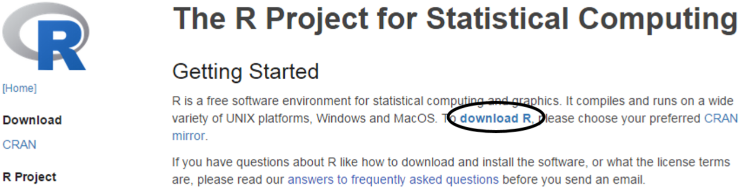
[2] **Python** is a general purpose programming language which can be used for a wide variety of applications.

How to install **R** on Windows

- Go to the website of **R** "The R Project for Statistical Computing", where its URL is

  `https://www.r-project.org`

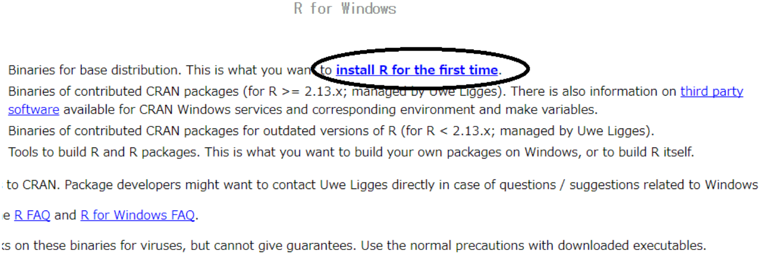
- Click on the link download R.



- Then, you will be asked to select which server you want download from. Choose "Japan - Tokyo".
- Click on the link Download R for Windows.

How to install R on Windows

- Click on the link **Install R for the first time**.



- Click on the link **Download R X.X.X for Windows**, where X.X.X gives the version of **R**.

- Then, the installer will be downloaded as "R-X.X.X-win.exe".

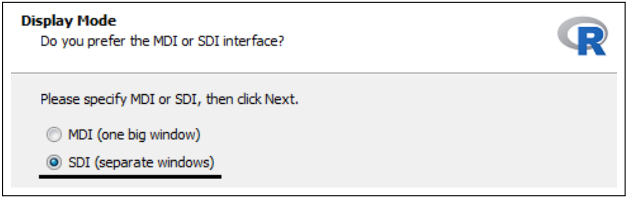
How to install **R** on Windows

- Double-click the installer in the download folder to launch the installer.
- Click "Next" several times.
- At the page that says "Startup options", choose

  Yes (customized startup)

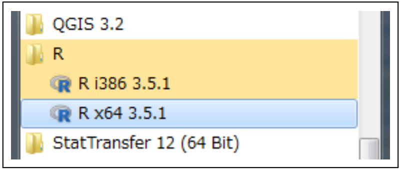
  The default choice is "No (accept defaults)".
- The next page says "Display Mode". Choose

  SDI (separate windows)



How to install **R** on Windows

- The other options can be left as defaults. Click "Next" several times until the installation is finished.
- To start **R**, click on the Windows button on the bottom left of your screen, click "All Programs", and select "R x64 X.X.X".
  - If your computer is 32-bit, select "R i386 X.X.X".
  - The 64 bit version of **R** can handle larger memory than the 32-bit version. There is no functional difference between them.

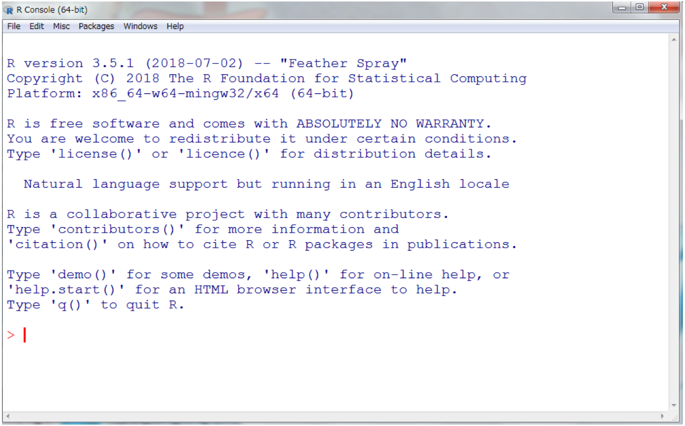


R-Studio

- **R-Studio** is a software that provides a more efficient and user-friendly programming environment for using **R**.
- It includes a code editor, debugging, and visualization tools.
- Although it is not mandatory, I would recommend using **R-Studio** for beginners.

The Basics of **R**

- When you start **R**, the window that first appears is called the **R** console.
- You can type or paste commands here. The console window also displays the results of the commands and error reports (if any).



The Basics of **R**

Basic arithmetic operations

- Addition: "+", subtraction: "−", multiplication: "$\ast$" and division: "/".
- For example, type `2 + 3` in the console, and press Enter:

```
Type 'demo()' for some demos, 'help()'
'help.start()' for an HTML browser inte
Type 'q()' to quit R.

> 2 + 3
[1] 5
> |
```

- The command you have run is shown in red text, and the result is in blue.

The Basics of R

- If you want to write more than one command in a single line, you can use a semicolon `;` as a command delimiter.
- Anything following a hash (sharp) sign # is ignored and it is not processed by R. Thus, this can be used to include comments.

```
> 2 + 3 # addition
[1] 5
> 2 - 3 # subtraction
[1] -1
> 2*3; 2/3 # multiplication and division
[1] 6
[1] 0.6666667
>
```

- Power function: `X^a`, square root: `sqrt()`, natural log: `log()`, exponential function: `exp()`.
- Trigonometric functions: `sin()`, `cos()`, `tan()`.
- These operations can be combined in one expression.

```
> 7^2
[1] 49
> sqrt(7)
[1] 2.645751
> log(5)
[1] 1.609438
> exp(sin(1) + cos(1))
[1] 3.981957
> 
```

- If you want to clear the console window, press [Ctrl] and [L] at the same time.

- If you want to assign the number "a" to the variable "X", you can use

  X <- a.

- The assign symbol consists of two separate characters < and −, "less than" and "minus" with no space between them.
  - You are not limited to just storing a single number. Virtually any type of **R** object (vector, matrix, data frames, functions, texts, etc) can be stored in a single object.

```
> X <- (2 + sqrt(5))*(exp(0.5) + exp(-0.5))
> X
[1] 9.5534
> X^3
[1] 871.9146
> A <- "Hoshino" # Texts must be enclosed in " ".
> A
[1] "Hoshino"
> A + 1 # You cannot add a number to a text.
Error in A + 1 : non-numeric argument to binary operator
```

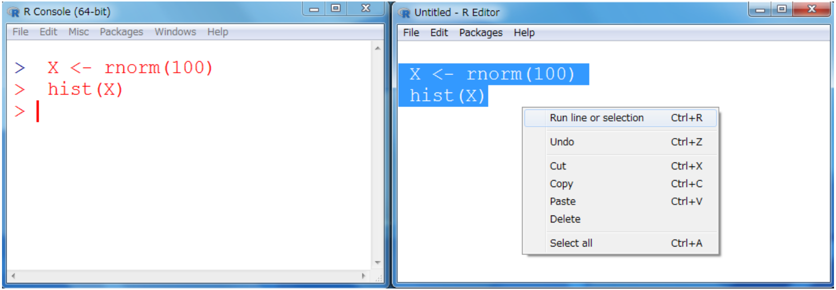
Script files

- When long and involved calculations are needed, typing each command directly into the console is inconvenient and error-prone.
- Besides, once the console window is closed all the commands you have executed will disappear.
- Script files are text files that contain a list of commands to execute. You can execute the commands directly from the script file all at once.
- To create a new script file, click on "File" in the menu bar and then click "New Script".

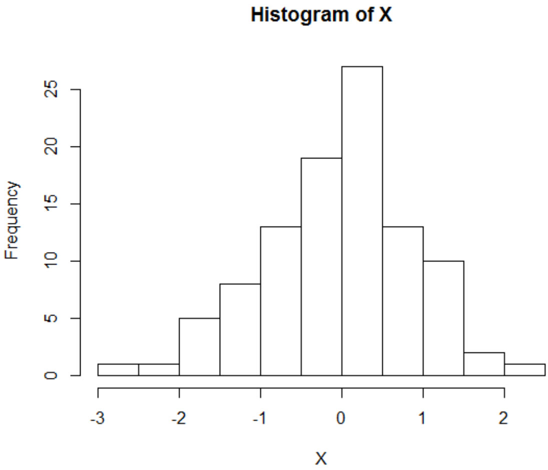
*You do not have to save the whole **R** workspace. Just remember to save the script file.

Script files

- After typing the commands in the script file, select the part you want to execute.
- Right click and choose "Run Line or Selection" or press [Ctrl] and [R] at the same time.



- `rnorm(100)`: draw 100 random numbers from $N(0,1)$.
- `hist(X)`: make a histogram of X (a new window will pop up).


Histogram of X
Frequency
0
5
10
15
20
25
-3
-2
-1
0
1
2
X

Script files

- To save the script file, click on "File" in the menu bar in the script editor, and then click "Save as". (The file extension is "r".)
- Note that double-clicking the script file does not open **R** window and the script itself. To open the saved script file, launch **R** first, and choose "Open script" in the "File" in the menu bar.
- If you want to open the script file only, you can use a text editor like Notepad.

Programming Exercise 1: Dice Roll Simulation

Dice Roll Simulation

- To simulate dice rolls in **R**, we can use a function called `sample()`. The usage of this function is:

```
sample(1:6, n, replace = T)
```

  - This function draws n random numbers from the vector specified in the first argument.
  - The notation "`1:6`" means the vector[3] of integers $(1, 2, ..., 6)$.
  - "`replace = T`" is required when the size of sample is larger than the length of the first argument. (Without this option, each number cannot be drawn more than once.)

[3] A vector is a sequence of numbers.

- For example, if you want to simulate 100 dice rolls, execute the following code:

```
sample(1:6, 100, replace = T)
```

```
> Dice100 <- sample(1:6, 100, replace = T)
> Dice100
  [1] 4 2 6 3 4 2 5 2 6 1 1 2 4 3 6 5 1 4 6
 [20] 3 1 3 1 1 3 6 4 6 3 3 3 6 3 5 2 4 4 5
 [39] 4 4 5 1 2 1 4 5 3 3 5 4 2 1 1 4 2 2 1
 [58] 3 4 5 5 2 3 6 1 1 3 5 2 2 6 2 2 5 1 6
 [77] 6 2 1 3 6 3 2 1 2 4 4 5 4 1 3 4 6 5 6
 [96] 4 2 1 6 3
>
```

NOTE: Your results may be different from mine. To fix the simulation results, you need to set the "random seed" before you start generating random numbers.

Dice Roll Simulation

- Compute the sample average of `Dice100` by `mean(Dice100)`:

```
> Dice100
  [1] 4 2 6 3 4 2 5 2 6 1 1 2 4 3 6 5 1 4 6
 [20] 3 1 3 1 1 3 6 4 6 3 3 3 6 3 5 2 4 4 5
 [39] 4 4 5 1 2 1 4 5 3 3 5 4 2 1 1 4 2 2 1
 [58] 3 4 5 5 2 3 6 1 1 3 5 2 2 6 2 2 5 1 6
 [77] 6 2 1 3 6 3 2 1 2 4 4 5 4 1 3 4 6 5 6
 [96] 4 2 1 6 3
> mean(Dice100)
[1] 3.35
>
```

Dice Roll Simulation

- Similarly, create `Dice20000` (20,000 dice rolls):

```
Dice20000 <- sample(1:6, 20000, replace = T).
```

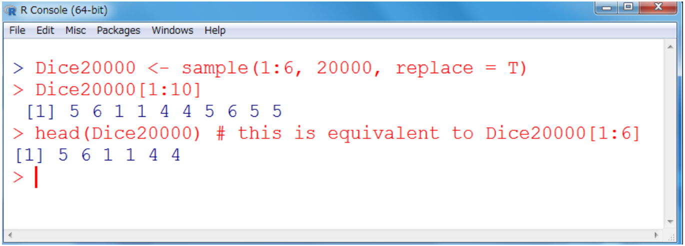
- Now, 20000 random numbers are stored in the object `Dice20000`. If you directly type "`Dice20000`" into the console, your console will be flooded.
- Let's just display the first several elements of `Dice20000`.

Dice Roll Simulation

- For example, if you want to look over the first 10 results,

  `Dice20000[1:10]`

  In **R**, the square brackets `[ ]` are used to identify a subset of the elements' indices.
- Using a function `head()` is also convenient. This function displays the first 6 elements of the data.



- The sample average of `Dice20000` is roughly equal to 3.5.

```
> Dice20000 <- sample(1:6, 20000, replace = T)
> Dice20000[1:10]
 [1] 5 6 1 1 4 4 5 6 5 5
> head(Dice20000) # this is equivalent to Dice20000[1:6]
[1] 5 6 1 1 4 4
> mean(Dice20000)
[1] 3.48935
> 
```

- Recall that the expected value of a dice roll is $E(\text{dice roll}) = 3.5$.
- From the above results, we can see that the sample average with $n = 20000$ is more close to 3.5 than the case with $n = 100$.
- This fact is known as the law of large numbers: as the sample size increases, the sample average converges to its population mean.

Dice Roll Simulation

- It is informative to visualize how the sample average converges to its mean.
- We create a function whose input is the sample size and the output is the corresponding sample average.
- We can use `function(){}` to create an original function.

```
Dice <- function(n){
    Dicerolls <- sample(1:6, n, replace = T)
    mean(Dicerolls)
    }
```

The parameter of the function is specified in the round brackets `()`, and the commands you want to run within the curly brackets `{}`.

- Then, if you specify any number for n, the function `Dice()` returns the sample average of dice rolls over the n trials.

R Console (64-bit)

File Edit Misc Packages Windows Help

```
> Dice <- function(n){
+ Dicerolls <- sample(1:6, n, replace = T)
+ mean(Dicerolls)
+ }
>
> Dice(5); Dice(123); Dice(999)
[1] 3
[1] 3.804878
[1] 3.401401
>
```

- Next, using the above created function `Dice()`, we repeat the dice rolling experiment for different n's: $n = 1, \ldots, N$.
- Such calculation can be performed using a so-called for-loop.
- The basic syntax of for-loop is as follows:

```
for(i in sequence){
    statement
    }
```

Here, `sequence` is a vector, where `i` takes on each of its value during the loop. In each iteration, `statement` is executed. The iteration stops when `i` reaches to the final element of `sequence`.

Dice Roll Simulation

- First, set any number for N, and create a blank vector of length N (vector of zeros).

```
N <- 1000
R <- numeric(N)
```

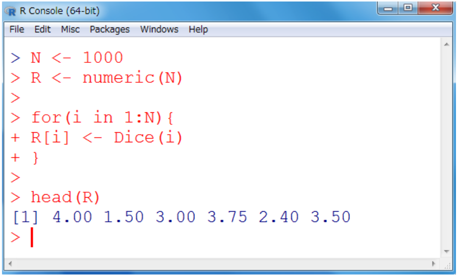
  The function `numeric()` creates a vector of the specified length with each element equal to 0.

- For-loop:

```
for(i in 1:N){
    R[i] <- Dice(i)
    }
```

Dice Roll Simulation

The i-th element of `R` corresponds to the average of the dice rolls over i realizations.




```
> N <- 1000
> R <- numeric(N)
>
> for(i in 1:N){
+ R[i] <- Dice(i)
+ }
>
> head(R)
[1] 4.00 1.50 3.00 3.75 2.40 3.50
>
```

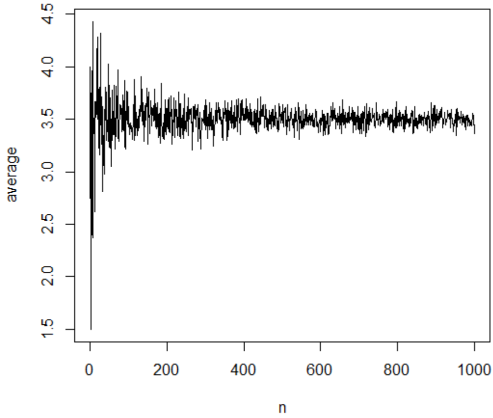
- The final step is to draw a graph with the x-axis being n and the y-axis being `Dice(`n`)`.
- We can use `plot()` function.[4]

```
plot(1:N, R, xlab = "n", ylab = "average", type = "l")
```

- Here, `type = "l"` (the small letter of L) is an option required when drawing a line plot (graph).
- Then a new window (R Graphics) will pop up.

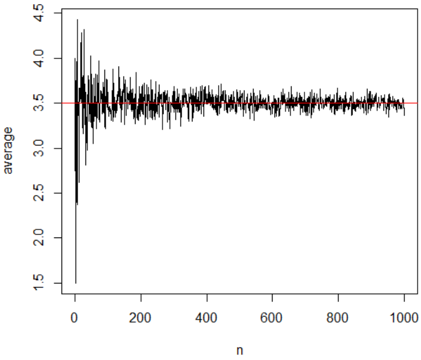
[4] `plot()` is a general purpose plotting function, which can produce a wide variety of scatterplots and graphs.

Dice Roll Simulation



Add a red horizontal line at y $= 3.5$:

```
abline(h = 3.5, col = "red")
```

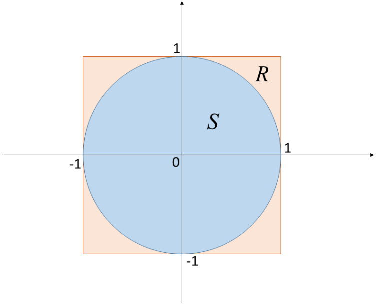


Programming Exercise 2: Area Calculation by Simulation

Area Calculation by Simulation

- In this exercise, we create a program that calculates the area of a circle with a radius of 1. (Of course, the area is equal to $\pi \approx 3.14$.)
- Consider a square with vertices at {(1,1), (-1,1),(-1,-1), (1,-1)}, and let its area be R.
- Trivially, the area of this circle, say S, can be obtained by

$$S = R \times \underbrace{\text{the ratio of } S \text{ to } R}_{S/R}.$$



Area Calculation by Simulation

- Clearly, $R = 4$. We only need to known the ratio of S to R (S/R).
- The idea is simple:
  1. Draw a pair of random numbers (x, y) from Uniform$[-1, 1]^2$ many times: $\{(x_i, y_i) : i = 1, ..., N\}$.
  2. Among all N pairs of random numbers, calculate the proportion of the pairs that fall into S:[5]

$$P_N = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{(x_i, y_i) \in S\},$$

     where $\mathbf{1}\{\}$ is an indicator function which takes 1 when the condition is true and 0 otherwise.
  3. Finally, calculate $S_N = 4 \times P_N$, and S_N is an approximate value of S.

[5] This method of calculating an approximate probability, i.e., by generating a set of random numbers and observing the proportion that meet the condition, is called the Monte Carlo simulation. Monte Carlo is the name of a district in Monaco that is famous for gambling and casinos.

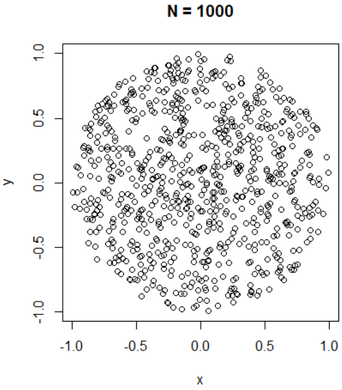
The above procedure can be coded as follows:

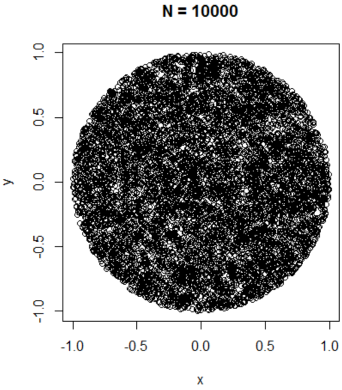
```
N <- 1000
x <- runif(N, -1, 1)
y <- runif(N, -1, 1)

inS <- ifelse(x^2 + y^2 < 1, 1, 0)
PN  <- sum(inS)/N
SN  <- 4*PN
print(SN)
```

- `runif(n,a,b)`: draw n random numbers from Uniform$[a, b]$.
- `ifelse(a,b,c)`: if `a` is true, it returns `b`, and if not, returns `c`.

Scatterplots for (x, y) located inside the circle S:

```
plot(x[inS == 1], y[inS == 1], xlab = "x", ylab = "y", main = "N = 1000")
```





Area Calculation by Simulation

```
>  N <- 1000
>  x <- runif(N, -1, 1)
>  y <- runif(N, -1, 1)
>
>  inS <- ifelse(x^2 + y^2 < 1, 1, 0)
>  PN  <- sum(inS)/N
>  SN  <- 4*PN
>  print(SN)
[1] 3.108
>  N <- 10000
>  x <- runif(N, -1, 1)
>  y <- runif(N, -1, 1)
>
>  inS <- ifelse(x^2 + y^2 < 1, 1, 0)
>  PN  <- sum(inS)/N
>  SN  <- 4*PN
>  print(SN)
[1] 3.1384
```

- As the number of simulations increases, the approximation becomes more precise.
- This fact is also due to the law of large numbers.