Lecture 2. Introduction to R (Getting Started)

R and Data Visualization

BIG2006, Hanyang University, Fall 2022

A first R session

```
x <- c(1,2,4)
x[2:3]
```

```
## [1] 2 4
```

- $<-$ or $=$: standard assignment operator
- No fixed type associated with variables
- c : concatenate
- x[2:3] (subsetting) : subvector of x consisting of elements 2 through 3, which are 2 and 3 here

Note: Any number is also considered to be a one-element vector.

```
x <- c(1,2,4)
q <- c(x,x,8)
q
```

```
## [1] 1 2 4 1 2 4 8
```

- Set q to (1, 2, 4, 1, 2, 4, 8) including the duplicates
- q : to print the vector to the screen, simply type its name

Note: If you type any variable name (or any expression) while in interactive mode, R will print out the value of that variable (or expression).

Summary statistics

- Mean, standard deviation, and other summary statistics of our data set

```
c(mean(x),sd(x))
```

```
## [1] 2.333333 1.527525
```

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.500   2.000   2.333   3.000   4.000
```

Comments

- Save the computed mean in a variable
- Use # to write comments

```
y <- mean(x); y1 <- c(mean(x),sd(x))
y # print out y
```

```
## [1] 2.333333
```

```
y1
```

```
## [1] 2.333333 1.527525
```

Note: Comments are valuable for documenting program code and help you to remember what you were doing.

Internal data sets

```
data()
```

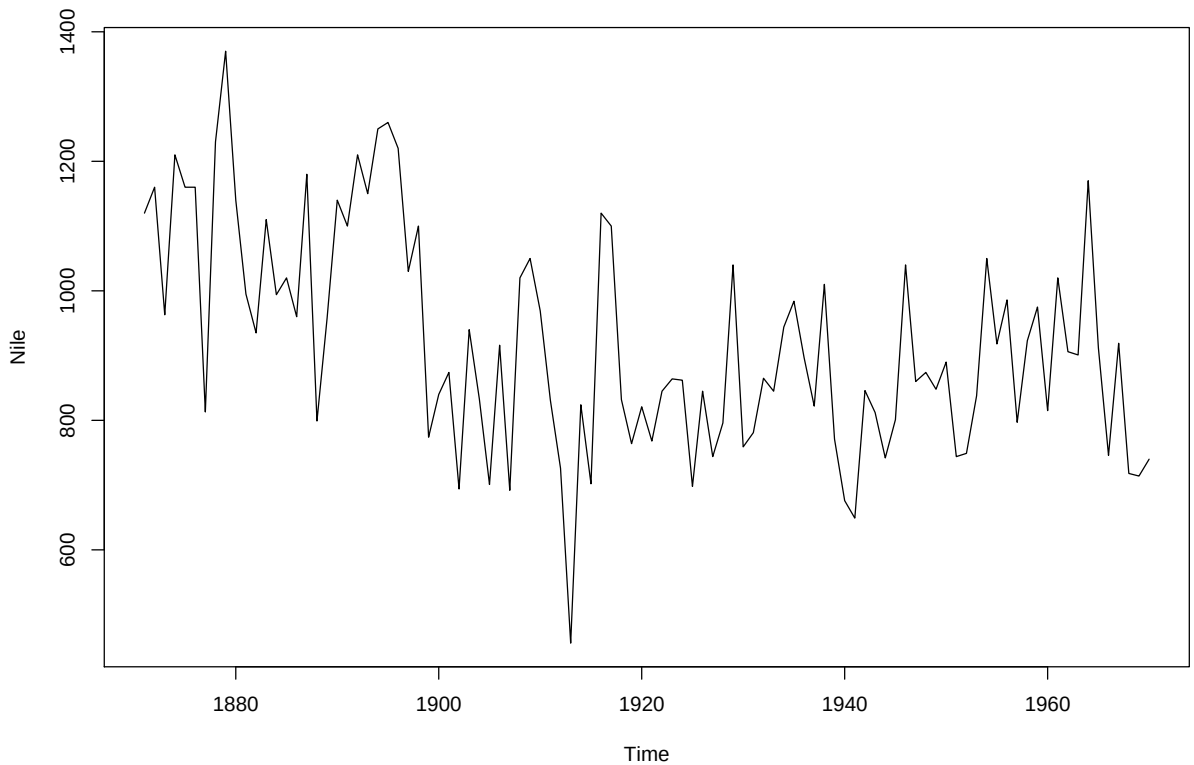
```
Data sets in package 'datasets':

AirPassengers                Monthly Airline Passenger Numbers 1949-1960
BJsales                      Sales Data with Leading Indicator
BJsales.lead (BJsales)       Sales Data with Leading Indicator
BOD                          Biochemical Oxygen Demand
CO2                          Carbon Dioxide Uptake in Grass Plants
ChickWeight                  Weight versus age of chicks on different diets
DNase                        Elisa assay of DNase
EuStockMarkets               Daily Closing Prices of Major European Stock Indices, 1991-1998
Formaldehyde                 Determination of Formaldehyde
HairEyeColor                 Hair and Eye Color of Statistics Students
Harman23.cor                 Harman Example 2.3
Harman74.cor                 Harman Example 7.4
Indometh                     Pharmacokinetics of Indomethacin
InsectSprays                 Effectiveness of Insect Sprays
JohnsonJohnson               Quarterly Earnings per Johnson & Johnson Share
LakeHuron                    Level of Lake Huron 1875-1972
LifeCycleSavings             Intercountry Life-Cycle Savings Data
Loblolly                     Growth of Loblolly pine trees
Nile                         Flow of the River Nile
Orange                       Growth of Orange Trees
OrchardSprays                Potency of Orchard Sprays
PlantGrowth                  Results from an Experiment on Plant Growth
Puromycin                    Reaction Velocity of an Enzymatic Reaction
```

```r
# Nile data set (including the flow of the Nile River)
c(mean(Nile),sd(Nile))
```

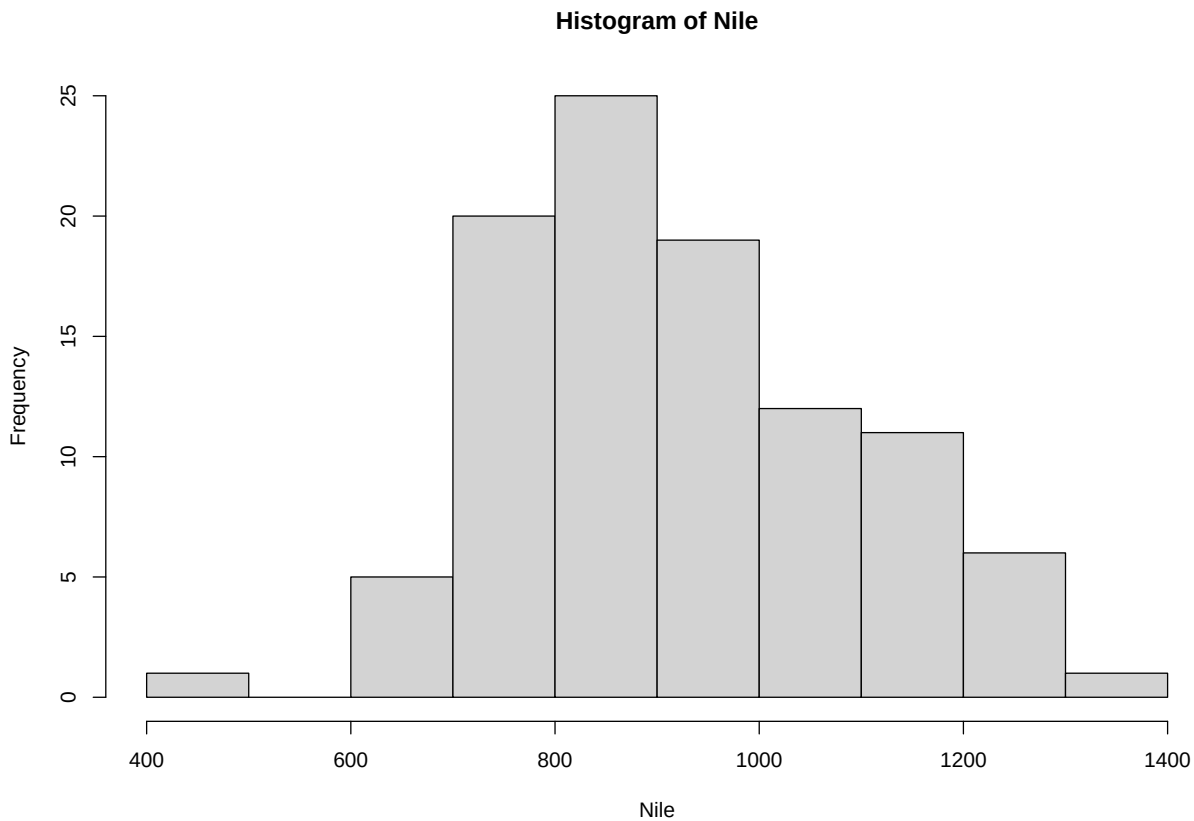
```
## [1] 919.3500 169.2275
```

```r
plot(Nile)
```



- Plot a histogram of the data (bare-bones simple)
- All kind of optional bells and whistles for plotting

```
hist(Nile)
```



Quit R

- Quit R by calling the q() function
- Pressing CTRL-D in Linux or CMD-D on a Mac

```
q()
```

Introduction to Functions

- The heart of R programming consists of writing functions.
- A function is a **group of instructions** that takes inputs, uses them to compute other values, and returns a result.

Simple example: counting the odd numbers

```r
# counts the number of odd integers in x
oddcount <- function(x) {
  k <- 0 # assign 0 to k
  for (n in x) {
    if (n %% 2 == 1) k <- k+1
    # %% is the modulo operator
  }
  return(k)
}

oddcount(c(1,1.5,3,7,10,11,12,15))
```

```
## [1] 5
```

- Guess what happens with the following code:

```
k <- 0; x <- c(1,1.5,3,7,10,11,12,15)
# 1
for (n in x){
  if (n %% 2 == 1) k <- k+1
}
```

1. Set `n` to `x[1]`, and then tests that value for being odd or not
2. If the value is odd, the count variable `k` is incremented.
3. Then `n` is set to `x[2]`, tested for being odd or not, and so on.

- The following code would also work (unless `x` have length 0).

```
k <- 0; x <- c(1,1.5,3,7,10,11,12,15)
# 2
for (i in 1:length(x)){
  # length(x): the number of elements in x
  if (x[i] %% 2 == 1) k <- k+1
}
```

Note: One of the major themes of R programming is to avoid loops if possible; if not, keep loops simple.

The formulation 1 do not need to resort to using the `length(x)` function and array indexing.

```r
return(k) # or simple k
```

```
## [1] 5
```

- The function returns the computed value of `k` to the code that called it.

Formal or actual argument

In programming terminology,

- `x` is the **formal argument** (or **formal parameter**) of the function `oddcount()`.

  Here, `x` is just a placeholder in the function defined

- $\texttt{c}(1, 1.5, 3, 7, 10, 11, 12, 15)$ in the first function call is the **actual argument**.

  $\texttt{c}(1, 1.5, 3, 7, 10, 11, 12, 15)$ is the value actually used in the computation

Variable scope (local and global)

- A variable that is visible only within a function body is said to be **local** to that function.

  In `oddcount()`, `k` and `n` are local variables because they disappear after the function returns.

```
oddcount(c(1,5,2,100,3,99))
```

```
## [1] 4
```

```
> n
Error: object 'n' not found
```

Note: The formal parameters in an R function are local variables.

- Variable created outside function are **global** and are available within functions as well.

```
f <- function(x) return(x+y)
```

```
> f(5)
Error in f(5) : object 'y' not found
```

```
y <- 3
f(5)
```

```
## [1] 8
```

Here y is global variable.

Defualt arguments

```
> g <- function(x, y=2, z=T){ ... }
```

- `y` will be initialized to 2 if we do not specify `y` in the call.
- `z` will have the default value `TRUE`.

```
> g(12, z=FALSE)
```

- The value 12 is for `x`, and we accept the default value of 2 for `y`.
- But, we override the default for `z`, setting it to `FALSE`.
- R has a *Boolean* type; has the logical values `TRUE` and `FALSE`.

Preview of data structures

- R has a variety of data structures.
- The **vector** type is really the heart of R.

Scalars (or individual numbers)

- Do not really exist in R
- Individual numbers are actually one-element vectors.

```
x <- 8
x
```

```
## [1] 8
```

Note: The [1] here signifies that the following row of numbers begins with element 1 of a vector. R was indeed treating x as a vector, albeit a vector with just one element.

Character Strings

- Single-element vectors of mode character

```
x <- c(9,8,2022)
c(length(x),mode(x))
```

```
## [1] "3"       "numeric"
```

```
z <- c("HYU","Stat","29 88")
c(length(z),mode(z))
```

```
## [1] "3"         "character"
```

- R has various string-manipulation functions.
- Many deal with putting strings together or taking them apart, such as the two shown here:

```
# concatenate the strings
u <- paste("spatial","statistic","s")
u
```

```
## [1] "spatial statistic s"
```

```
# split the string according to blanks
v <- strsplit(u," ")
v
```

```
## [[1]]
## [1] "spatial"   "statistic" "s"
```

Matrices

- An R **matrix** corresponds to the mathematical concept of the same name: a rectangular array of numbers.

```
m <- rbind(c(9,4),c(1,5))
m
```

```
##      [,1] [,2]
## [1,]    9    4
## [2,]    1    5
```

```
m %*% c(1,2)
```

```
##      [,1]
## [1,]   17
## [2,]   11
```

- An useful feature of R is that you can extract submatrices froma matrix, much as you extract subvectors from vectors.

```
m[1,] # row 1
```

```
## [1] 9 4
```

```
m[,2] # column 2
```

```
## [1] 4 5
```

Note: Matrices are indexed using double subscripting, much as in C/C++, although subscripts start at 1 instead of 0.

Lists

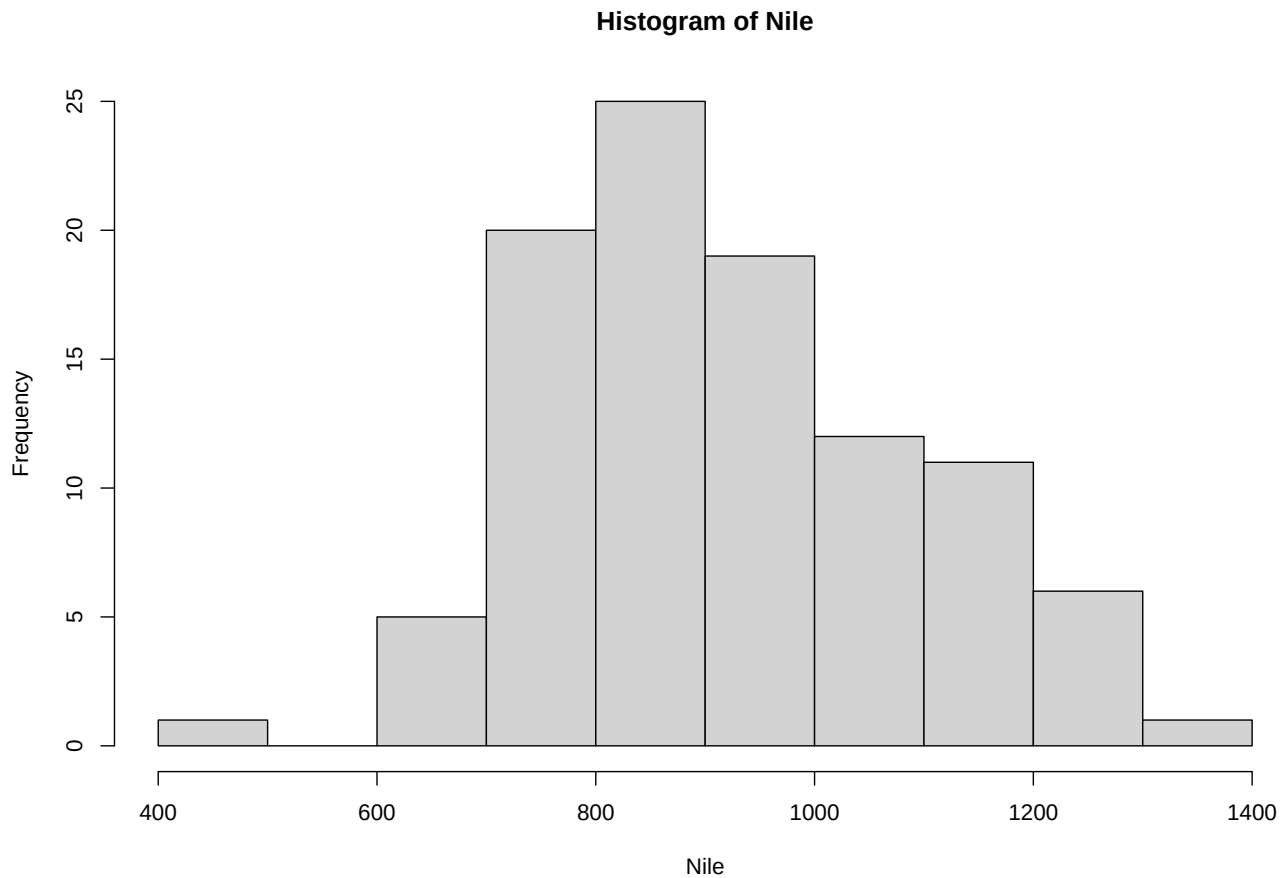
- An R **list** is a container for values, but its contents can be items of different data types.
- List elements are accessed using two-part names, which are indicated with the dollar sign $ in R.

```
x <- list(u=2022, v="HYU")
x
```

```
## $u
## [1] 2022
##
## $v
## [1] "HYU"
```

- A common use of lists is to combine multiple values into a single package that can be returned by a function.

```
hn <- hist(Nile)
```



```r
print(hn)
```

```
## $breaks
##  [1]  400  500  600  700  800  900 1000 1100 1200 1300 1400
##
## $counts
##  [1]  1  0  5 20 25 19 12 11  6  1
##
## $density
##  [1] 0.0001 0.0000 0.0005 0.0020 0.0025 0.0019 0.0012 0.0011 0.0006
##
## $mids
##  [1]  450  550  650  750  850  950 1050 1150 1250 1350
##
## $xname
## [1] "Nile"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

```r
str(hn)
```

```
## List of 6
##  $ breaks  : int [1:11] 400 500 600 700 800 900 1000 1100 1200 1300
##  $ counts  : int [1:10] 1 0 5 20 25 19 12 11 6 1
##  $ density : num [1:10] 0.0001 0 0.0005 0.002 0.0025 0.0019 0.0012 0
##  $ mids    : num [1:10] 450 550 650 750 850 950 1050 1150 1250 1350
##  $ xname   : chr "Nile"
##  $ equidist: logi TRUE
##  - attr(*, "class")= chr "histogram"
```

Note: Here `str` stands for **structure**. This function shows the internal structure of any R object, not just lists.

Data Frames

- A **data frame** in R is a list, with each component of the list being a vector corresponding to a column in our "matrix" of data.

```
d <- data.frame(list(departments=c("Mathematics","Statistics",
                "Computer Science and Engineering")),
                codes=c("MAT","STS","CSE"), num_facs=c(20,5,30))
d
```

```
##                         departments codes num_facs
## 1                       Mathematics   MAT       20
## 2                        Statistics   STS        5
## 3 Computer Science and Engineering   CSE       30
```

```
d$codes
```

```
## [1] "MAT" "STS" "CSE"
```

Classes

- R is an object-oriented language.
- **Objects** are instances of **classes**.
- Classes are a bit more abstract than the data types, and we will **skip** the concept.

Getting Help

A plethora of resources are available to help you learn more about R.

These include several facilities within R itself and, of course, on the Web.

The help() Fucntion

- To get online help, invoke `help()`, e.g., to get information on the `rep()` or `seq()` function, type this:

```
help(rep) # or ?repeat
help(seq) # or ?seq
```

Note: Special characters and some reserved words must be quoted when used with the `help()` function.

```
?"for"
?"<"
```

The example() Function

- Each of the help entries comes with examples.
- One nice feature of R is that the `example()` function will actually run those examples for you.

```
> example(seq)
```

- The `seq()` function generates various kinds of numeric sequences in arithmetic progression.
- A series of sample graphs for the `persp()` function can be displayed.

```
> example(persp)
```

The help.search() Function

- Use the function `help.search()` to do a Google-style search through R's documentation.

```
help.search("multivariate normal")
```

```
??"multivariate normal"
```

- See that the function `mvrnorm()` will do the job, and it is in the package `MASS`.

Help on the Internet

There are many excellent resources on R on the Internet.

- The R Project's own manuals are available from the R home page: http:/www.r-project.org/.
- The RSeek search engine: http://www.rseek.org/.

Reference

- Matloff, N. The Art of R Programming: A Tour of Statistical Software Design. No Starch Press. Chapter 1.