Probability and Statistics (MATH-365)

Data Types and Variables in R

Dr. Aamir Alaud Din

September 17, 2025

Why Data Types

- Why are there data types in a computer programming language?
- Data Types in human communication
- Data Types in R
  - Logical
  - Numeric
  - Integer
  - Complex
  - Character
  - Raw

Logical

- TRUE, FALSE
- Decision making
- No binary operations
- Operators are applicable

```
x <- TRUE
print(class(x))
```

- Output

```
"logical"
```

Numeric

- Math operations
- Measurement

```
x <- 12.3
y <- 55
print(class(x))
print(class(y))
```

- Output

```
"numeric"
"numeric"
```

Integer

- Math operations
- Counting

```
x <- 5L
print(class(x))
```

- Output

```
"integer"
```

Complex

- Math operations
- Imaginary numbers dealing

```
x <- 2 + 5i
print(class(x))
```

- Output

```
"complex"
```

Character

- No math operations
- Text processing

```
x <- "Aamir"
y <- "TRUE"
z <- "23.5"
print(class(x))
print(class(y))
print(class(z))
```

- Output

```
"character"
"character"
"character"
```

Raw

- No math operations
- ASCII codes or hexademical notation

```
x <- CharToRaw("Hello")
print(x)
print(class(x))
```

- Output

```
48 65 6c 6c 6f
"raw"
```

R-Objects

- R variables are not declared as some data type but they are assigned with R-Objects.
- Then data type of R becomes the R-Object.
- Type of R-objects are as follows.
  - Vectors
  - Lists
  - Matrices
  - Arrays
  - Factors
  - Data Frames

Vectors

- Vectors in r are initialized with `c()` function.

- Elements of vectors can be any data types we discussed above.
- Mixed data types can also be elements of vectors.

```
names <- c('Aayesha', 'Hashir', 'Fatimah', 'Shaheer')
print(names)
ages <- c(14, 12, 9, 7, 'Ages')
print(ages)
```

- Output

```
"Aayesha" "Hashir"  "Fatimah" "Shaheer"
"14"   "12"   "9"    "7"    "Ages"
```

- Vector elements can be called by their indices.
- In above example, the variable `names` has indices starting from 1 for `Aayesha` through 4 for `Shaheer`.

```
n1 <- names[1]
n2 <- names[1:3]
print(n1)
print(n2)
```

- Output

```
"Aayesha"
"Aayesha" "Hashir"  "Fatimah"
```

- What will be the output of the following r statement?

```
ages <- c(14, 12, 9, 7, 'Ages')
print(ages[4] + 7)
```

- The following statement will work.

```
ages <- c(14, 12, 9, 7, 'Ages')
print(as.numeric(ages[4]) + 7)
```

- Output

```
14
```

Lists

- A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
lst1 <- list(c(1, 3, 5), 9.5, sin)
print(lst1)
print(lst1[3])
```

- Output

```
[[1]]
[1] 1 3 5

[[2]]
[1] 9.5

[[3]]
function (x)  .Primitive("sin")

[[1]]
function (x)  .Primitive("sin")
```

Matrices

- A matrix is a two-dimensional rectangular data set.
- It can be created using a vector input to the matrix function.
- Indices are assigned column wise.

```
M <- matrix(c('a', 'b', 'c', 'd', 'e', 'f'), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
print(M[2,])
print(M[,2])
print(M[2,][2])
```

- Output

```
     [,1] [,2] [,3]
[1,] "a"  "b"  "c"
[2,] "d"  "e"  "f"
[1] "d" "e" "f"
[1] "b" "e"
[1] "e"
```

Arrays

- While matrices are confined to two dimensions, arrays can be of any number of dimensions.

- The array function takes a dim attribute which creates the required number of dimension.
- In the below example we create an array with two elements which are 3x3 matrices each.

```
arr <- array(c('red', 'green'), dim = c(3, 3, 2))
print(arr)
print(arr[,,2])
```

- Output

```
, , 1

     [,1]    [,2]    [,3]
[1,] "red"   "green" "red"
[2,] "green" "red"   "green"
[3,] "red"   "green" "red"

, , 2

     [,1]    [,2]    [,3]
[1,] "green" "red"   "green"
[2,] "red"   "green" "red"
[3,] "green" "red"   "green"

     [,1]    [,2]    [,3]
[1,] "green" "red"   "green"
[2,] "red"   "green" "red"
[3,] "green" "red"   "green"
```

Factors

- Factors are the r-objects which are created using a vector.
- It stores the vector along with the distinct values of the elements in the vector as labels.
- The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector.
- They are useful in statistical modeling.
- Factors are created using the `factor()` function.
- The `nlevels` functions gives the count of levels.

```
apple_colors <- c('green', 'green', 'yellow', 'red', 'red', 'red',
'green')

apple_factors <- factor(apple_colors)

print(apple_factors)
print(nlevels(apple_factor))
```

- When we execute the above code, it produces the following result.

```
 [1] green  green  yellow red    red    red    green
Levels: green red yellow
[1] 3
```

Data Frames

- Data frames are tabular data objects.
- Unlike a matrix in data frame each column can contain different modes of data.
- The first column can be numeric while the second column can be character and third column can be logical.
- It is a list of vectors of equal length.
- Data Frames are created using the `data.frame()` function.

```
BMI <-  data.frame(
   gender = c("Male", "Male","Female"),
   height = c(152, 171.5, 165),
   weight = c(81,93, 78),
   Age = c(42,38,26)
)
print(BMI)
```

- When we execute the above code, it produces the following result.

```
  gender height weight Age
1   Male  152.0     81  42
2   Male  171.5     93  38
3 Female  165.0     78  26
```

- Another environmental science related example is the survival of four aquatic species in seawater under different arsenic concentrations is given below.

```
survival <- data.frame(
  species = c('Tuna', 'Prawns', 'Lobster', 'Crab'),
  quantity = c(50, 50, 50, 50),
  av_weight = c(125, 0.040, 0.95, 1.25),
  threshold_as = c(18, 24, 30 ,36),
  weight_18 = c(125, 0.04, 0.95, 1.25),
  weight_24 = c(120, 0.04, 0.92, 1.20),
  weight_30 = c(85, 0.02, 0.59, 0.89),
  weight_36 = c(40, 0.005, 0.32, 0.33),
  survive_18 = c(50, 50, 50, 50),
  sruvive_24 = c(45, 40, 46, 48),
  survive_30 = c(30, 20, 35, 38),
  survive_36 = c(18, 0, 24, 24)
)
print(survival)
print(survival$species)
print(survival$av_weight)
print(survival$weight_30)
print(survival$quantity)
print(survival$survive_30)
```

- The output is as shown below.

```
  species quantity av_weight threshold_as weight_18 weight_24 weight_30 weight_36
survive_18 sruvive_24 survive_30 survive_36
1    Tuna       50    125.00           18    125.00    120.00     85.00    40.000
50         45         30         18
2  Prawns       50      0.04           24      0.04      0.04      0.02     0.005
50         40         20          0
3 Lobster       50      0.95           30      0.95      0.92      0.59     0.320
50         46         35         24
4    Crab       50      1.25           36      1.25      1.20      0.89     0.330
50         48         38         24
[1] "Tuna"    "Prawns"  "Lobster" "Crab"
[1] 125.00   0.04   0.95   1.25
[1] 50 50 50 50
[1] 30 20 35 38
```

R Variables

- A variable provides us with named storage that our programs can manipulate.
- A variable in R can store an atomic vector, group of atomic vectors or a combination of many R objects.
- A valid variable name consists of letters, numbers and the dot or underline characters.
- The variable name starts with a letter or the dot not followed by a number.

Variable Name	Validity	Reason
var_name2.	Valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed
2var_name	Invalid	Starts with a number
.var_name	Valid	Can start with a dot(.) but the dot(.)should not be followed by a number
var.name	Valid	The dot(.) can be anywhere and no starting number
.2var_name	Invalid	The starting dot is followed by a number making it invalid
_var_name	Invalid	Starts with _ which is not valid

Variable Assignment

- The variables can be assigned values using leftward, rightward and equal to operator.
- The values of the variables can be printed using `print()` or `cat()` function.
- The `cat()` function combines multiple items into a continuous print output.

```
# Assignment using equal operator.
var.1 = c(0,1,2,3)

# Assignment using leftward operator.
var.2 <- c("learn","R")

# Assignment using rightward operator.
c(TRUE,1) -> var.3

print(var.1)
cat ("var.1 is ", var.1 ,"\n")
cat ("var.2 is ", var.2 ,"\n")
cat ("var.3 is ", var.3 ,"\n")
```

- When we execute the above code, it produces the following result.

```
[1] 0 1 2 3
var.1 is  0 1 2 3
var.2 is  learn R
var.3 is  1 1
```

- **Note** – The vector `c(TRUE,1)` has a mix of logical and numeric class.
- So logical class is coerced to numeric class making TRUE as 1.

Data Type of a Variable

- In R, a variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it.
- So R is called a dynamically typed language, which means that we can change a variables data type of the same variable again and again when using it in a program.

```
var_x <- "Hello"
cat("The class of var_x is ",class(var_x),"\n")

var_x <- 34.5
cat("  Now the class of var_x is ",class(var_x),"\n")

var_x <- 27L
cat("   Next the class of var_x becomes ",class(var_x),"\n")
```

- The output is shown below.

```
The class of var_x is  character
   Now the class of var_x is  numeric
      Next the class of var_x becomes  integer
```

Finding Variables

- To know all the variables currently available in the workspace we use the `ls()` function.
- Also the `ls()` function can use patterns to match the variable names.

```
print(ls())
```

- The execution leads to following results.

```
[1] "my var"     "my_new_var" "my_var"     "var.1"
[5] "var.2"      "var.3"      "var.name"   "var_name2."
[9] "var_x"      "varname"
```

- **Note** – It is a sample output depending on what variables are declared in your environment.
- The `ls()` function can use patterns to match the variable names.

```
# List the variables starting with the pattern "var".
print(ls(pattern = "var"))
```

- The output is as follows.

```
[1] "my var"     "my_new_var" "my_var"     "var.1"
[5] "var.2"      "var.3"      "var.name"   "var_name2."
[9] "var_x"      "varname"
```

- The variables starting with dot(.) are hidden, they can be listed using `"all.names = TRUE"` argument to `ls()` function.

```
print(ls(all.name = TRUE))
```

- The output is as follows.

```
[1] ".cars"        ".Random.seed" ".var_name"    ".varname"     ".varname2"
[6] "my var"       "my_new_var"   "my_var"       "var.1"        "var.2"
[11]"var.3"        "var.name"     "var_name2."   "var_x"
```

Deleting Variables

- Variables can be deleted by using the `rm()` function.
- Below we delete the variable var.3.
- On printing the value of the variable error is thrown.

```
rm(var.3)
print(var.3)
```

- When we execute the above code, it produces the following result.

```
 [1] "var.3"
Error in print(var.3) : object 'var.3' not found
```

- All the variables can be deleted by using the `rm()` and `ls()` function together.

```
rm(list = ls())
print(ls())
```

- The output is shown below.

```
character(0)
```