Basic Skills in R – Part 1

VN-Biostat Pre-Workshop

Basic Skills in R – Part 1

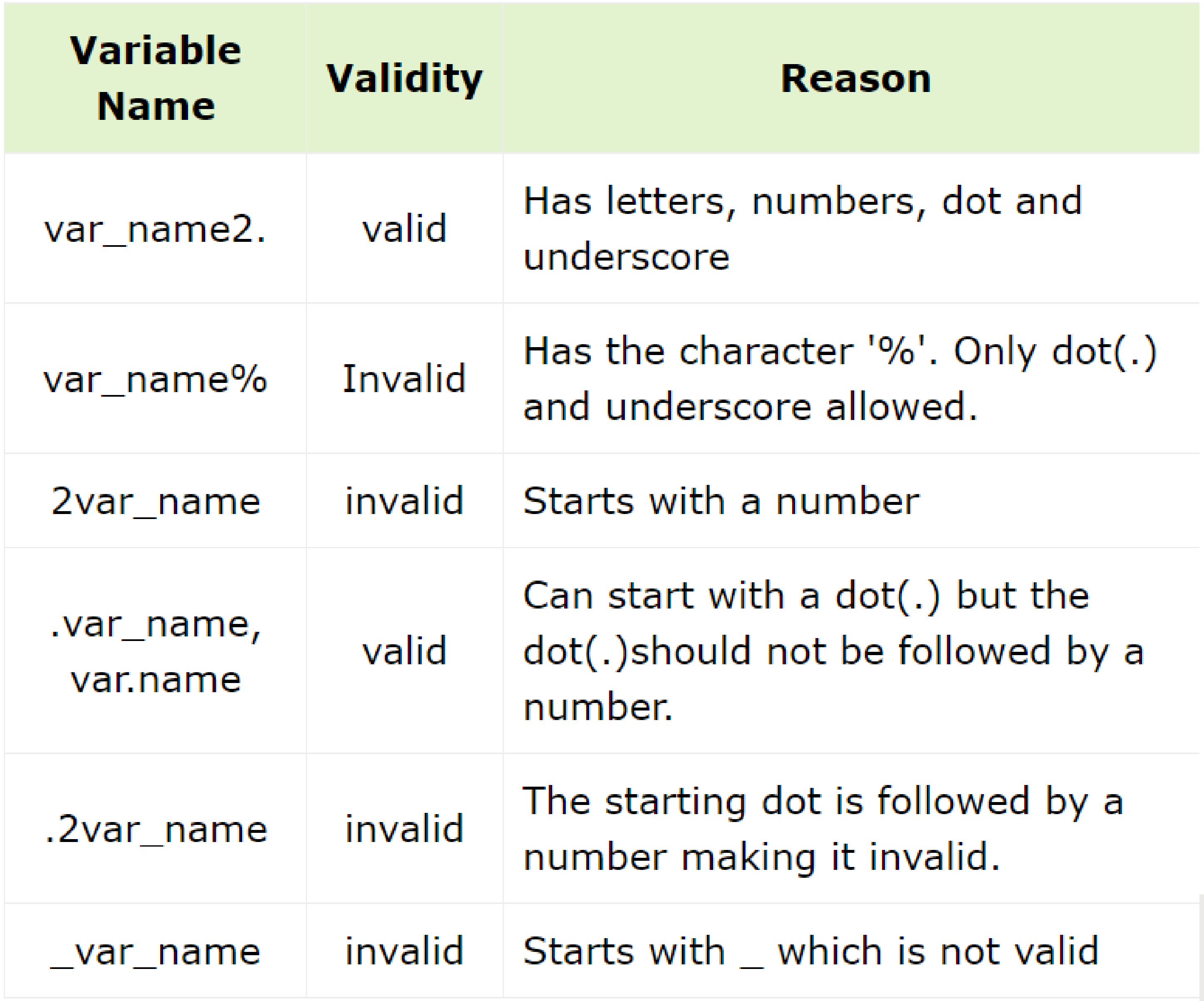
- R syntax
- Variables and Data Types
- Vectors, Matrices and Data Frames
- Data objects in R
    - Scalars
    - Vectors
    - Factors
    - Lists Matrices
    - Arrays
    - Data frames
- Online helps

R syntax

- R syntax follows a set of rules for writing code, including conventions for variable names and comments.
- Understanding and following these rules is essential for writing clear and readable code.

Variable Names

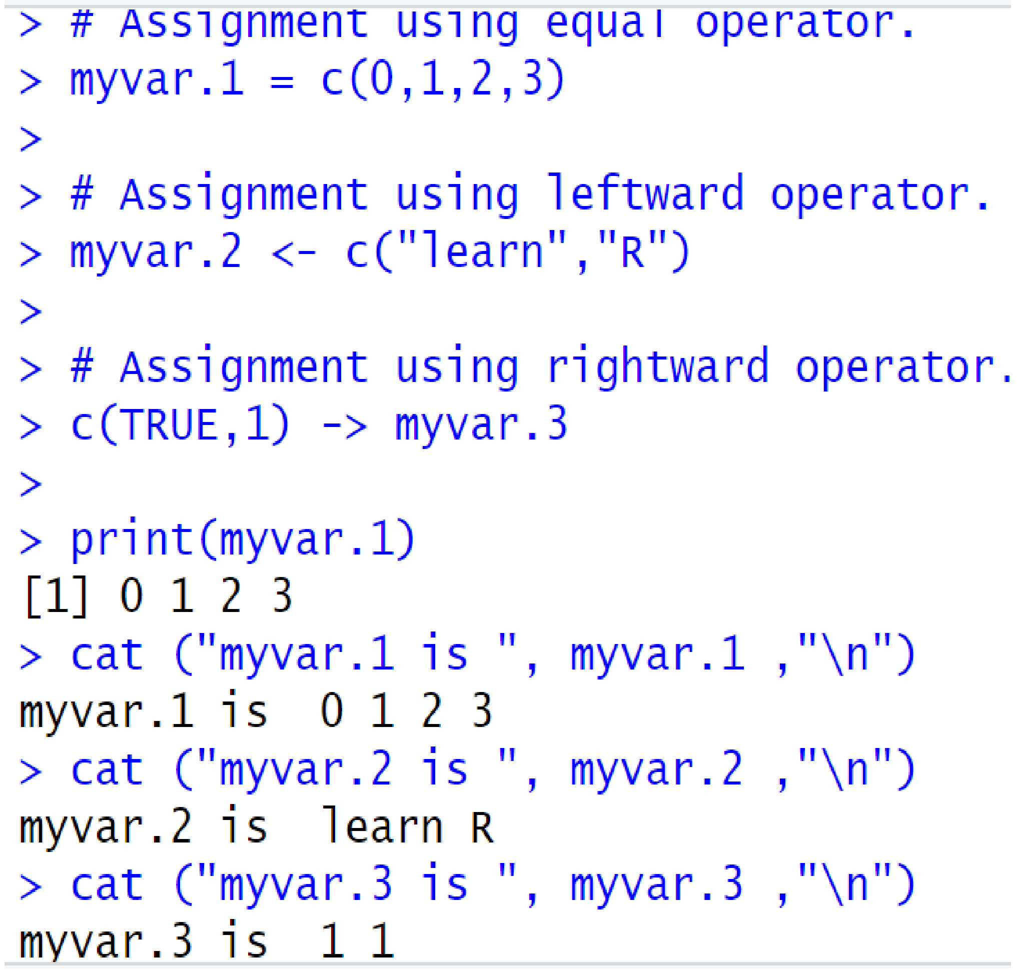
- A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.



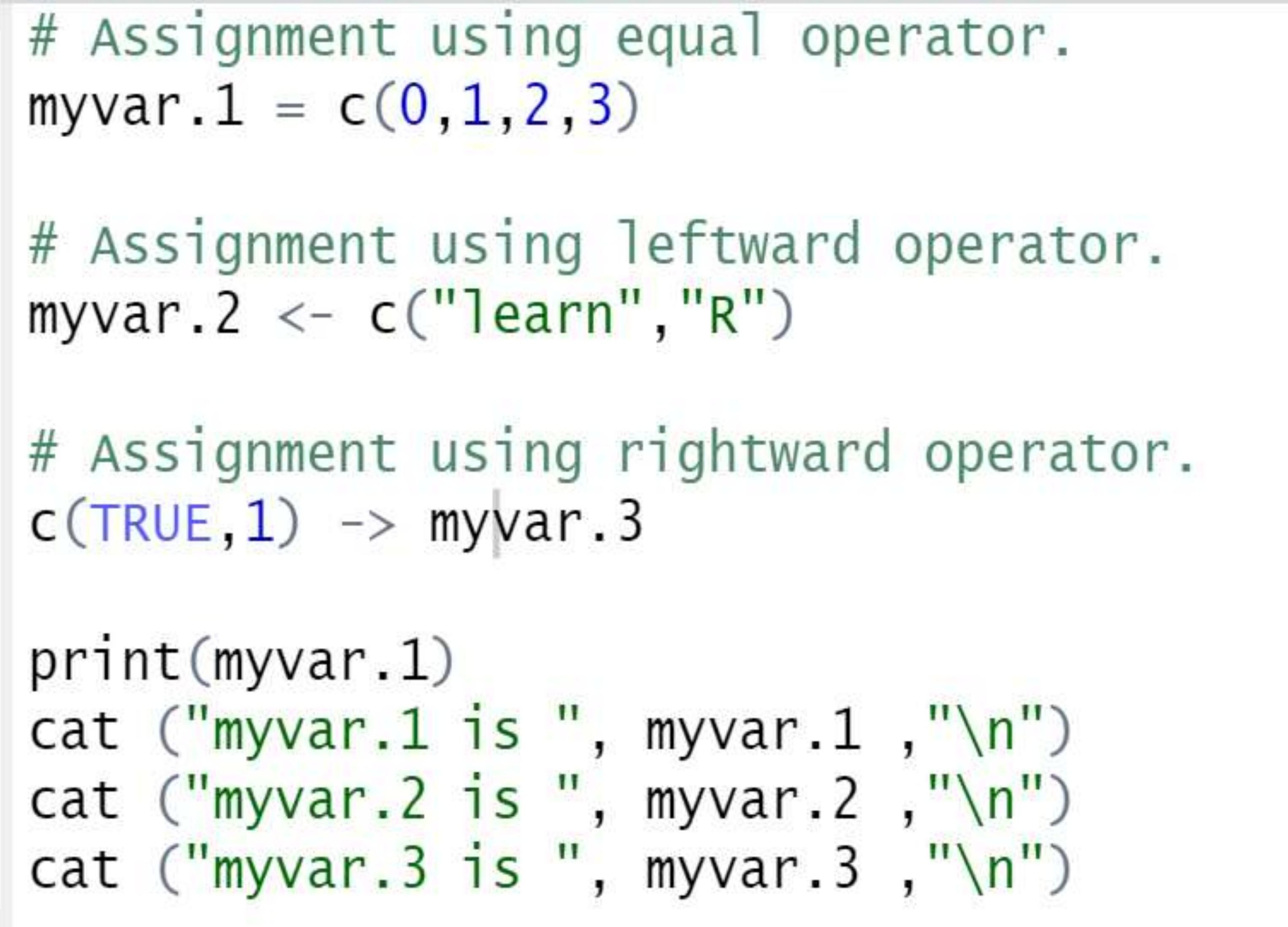
Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name, var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.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.



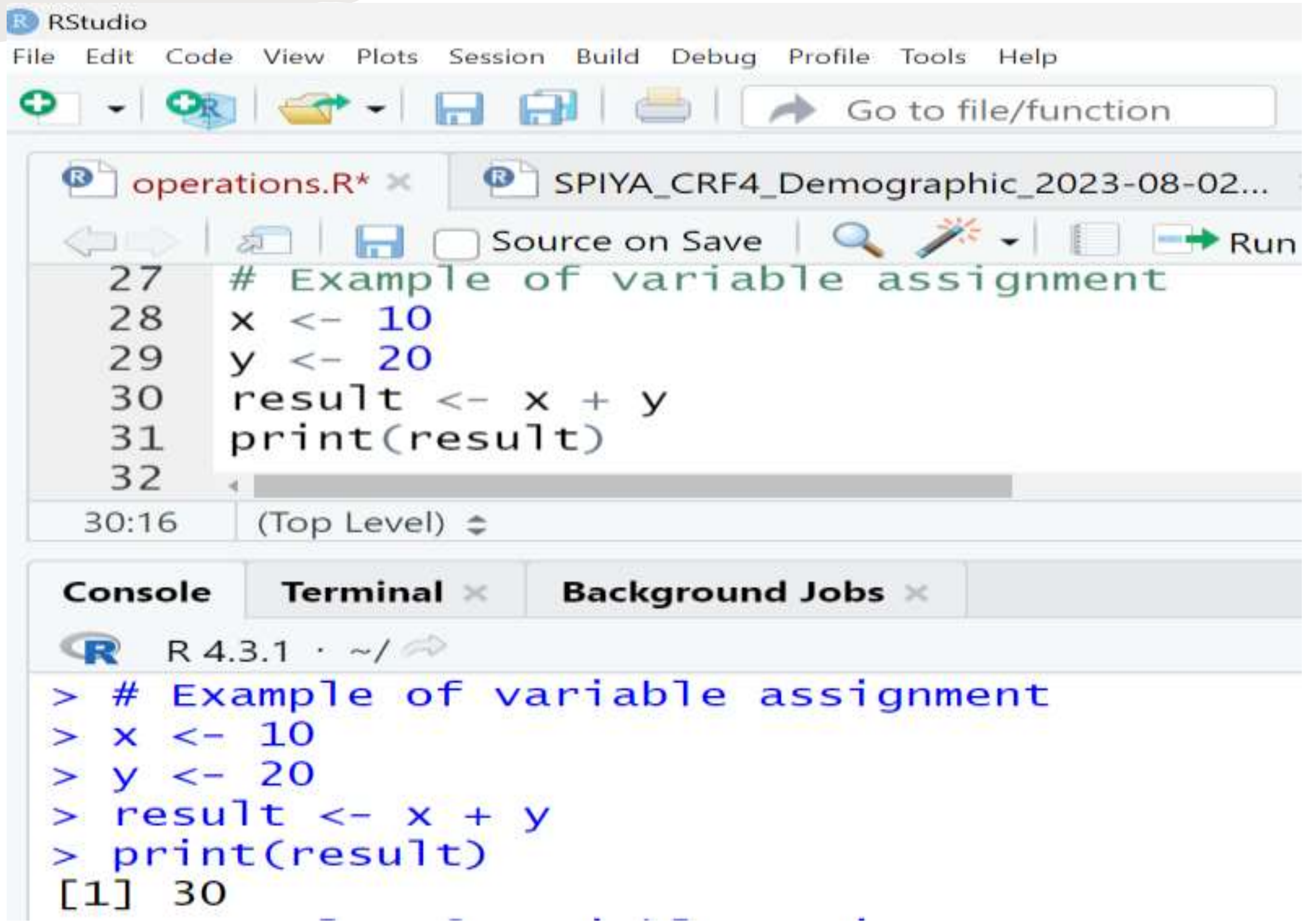
```
> # Assignment using equal operator.
> myvar.1 = c(0,1,2,3)
>
> # Assignment using leftward operator.
> myvar.2 <- c("learn","R")
>
> # Assignment using rightward operator.
> c(TRUE,1) -> myvar.3
>
> print(myvar.1)
[1] 0 1 2 3
> cat ("myvar.1 is ", myvar.1 ,"\n")
myvar.1 is  0 1 2 3
> cat ("myvar.2 is ", myvar.2 ,"\n")
myvar.2 is  learn R
> cat ("myvar.3 is ", myvar.3 ,"\n")
myvar.3 is  1 1
```



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

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

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

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

Variables and Data Types

- Variables in R are used to store and manipulate data.
- Variables are nothing but reserved memory locations to store values.
- R supports various data types including numeric, character, logical, and factor.

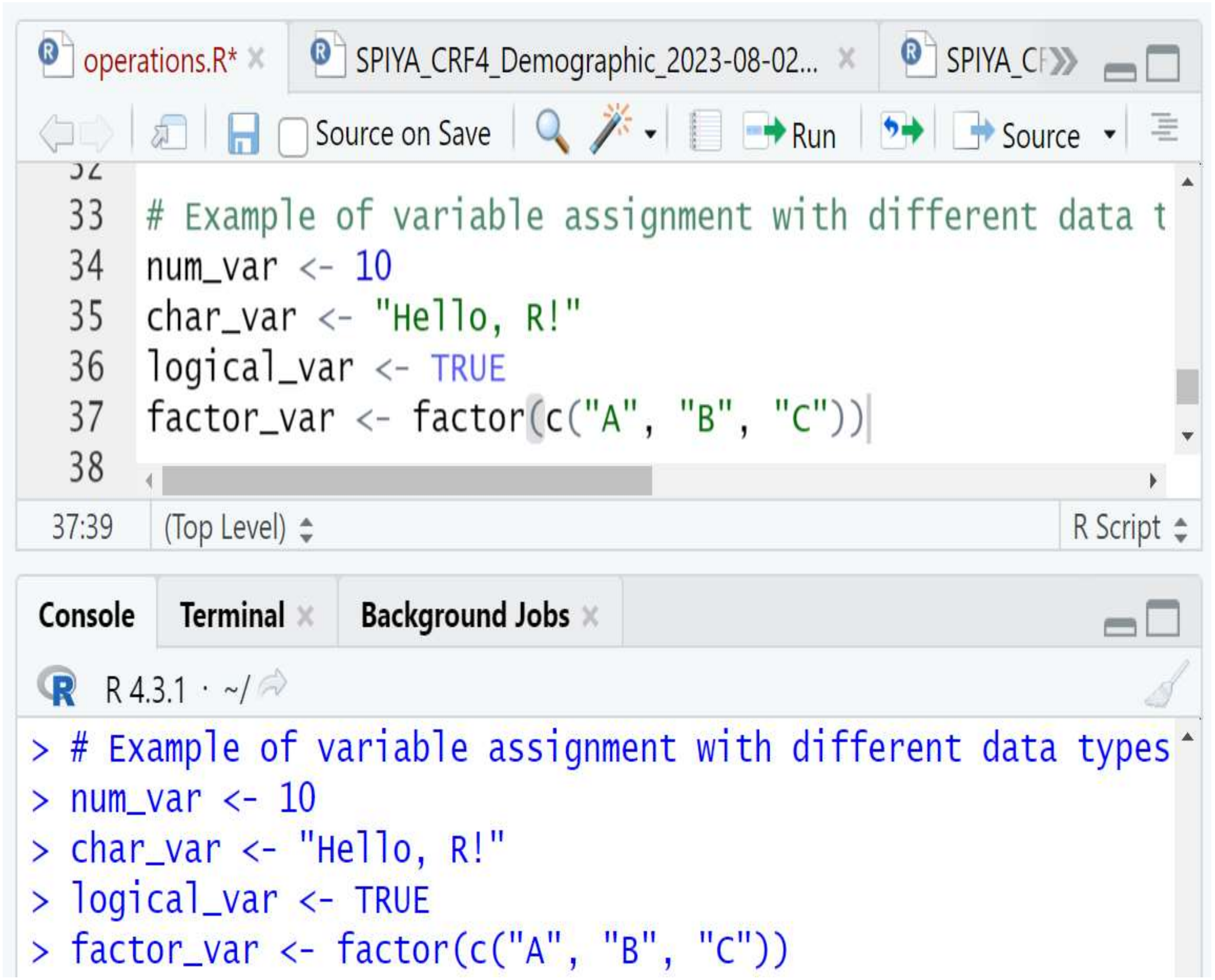


```
# Example of variable assignment
x <- 10
y <- 20
result <- x + y
print(result)
```

```
> # Example of variable assignment
> x <- 10
> y <- 20
> result <- x + y
> print(result)
[1] 30
```

Variables and Data Types

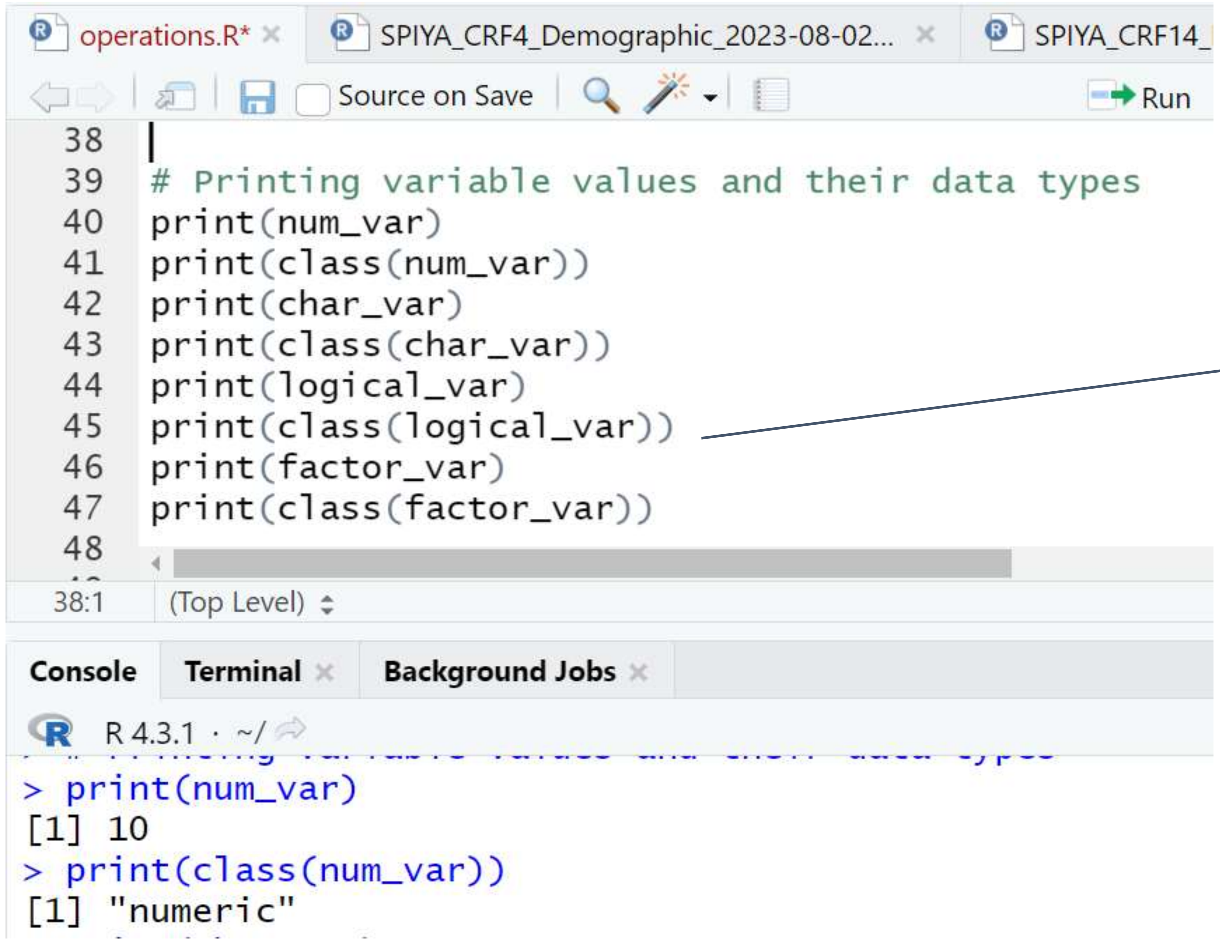
- Variables can be assigned values of different data types, and type conversion is possible.



```r
# Example of variable assignment with different data t
num_var <- 10
char_var <- "Hello, R!"
logical_var <- TRUE
factor_var <- factor(c("A", "B", "C"))
```

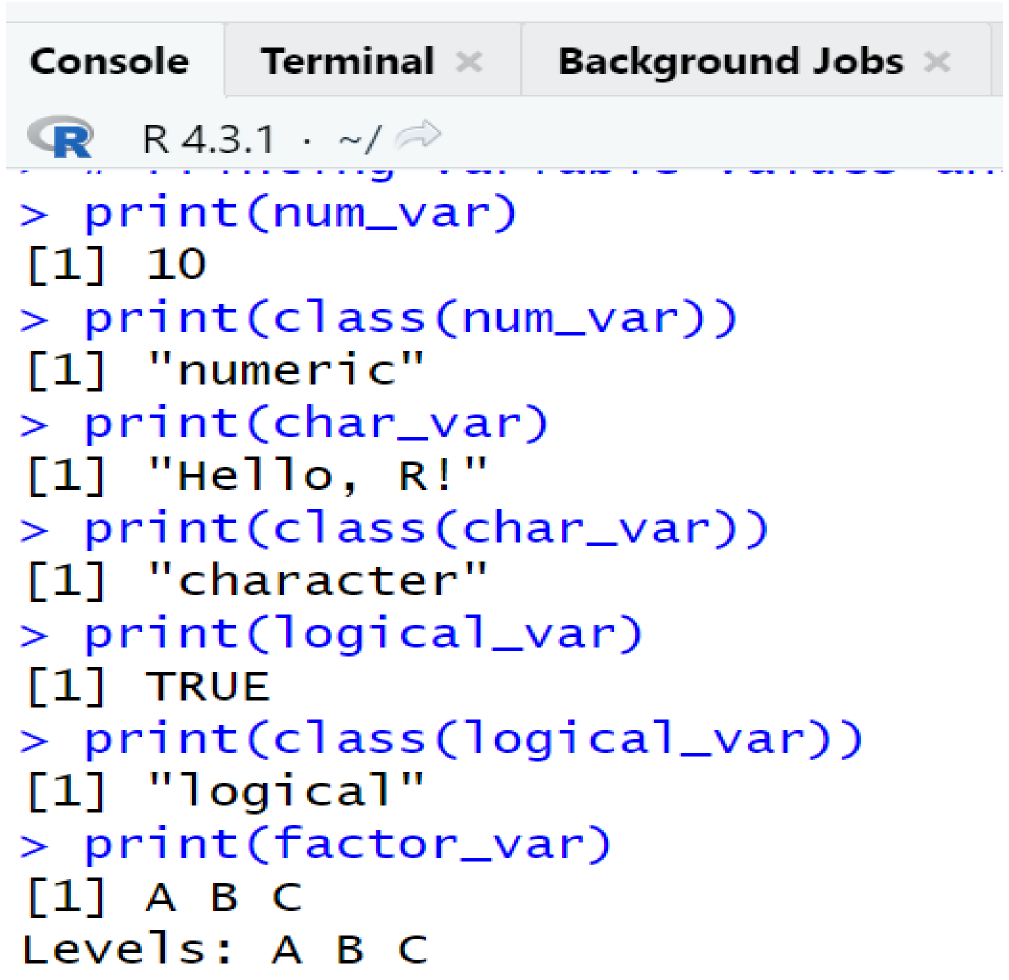
```
> # Example of variable assignment with different data types
> num_var <- 10
> char_var <- "Hello, R!"
> logical_var <- TRUE
> factor_var <- factor(c("A", "B", "C"))
```

Variables and Data Types



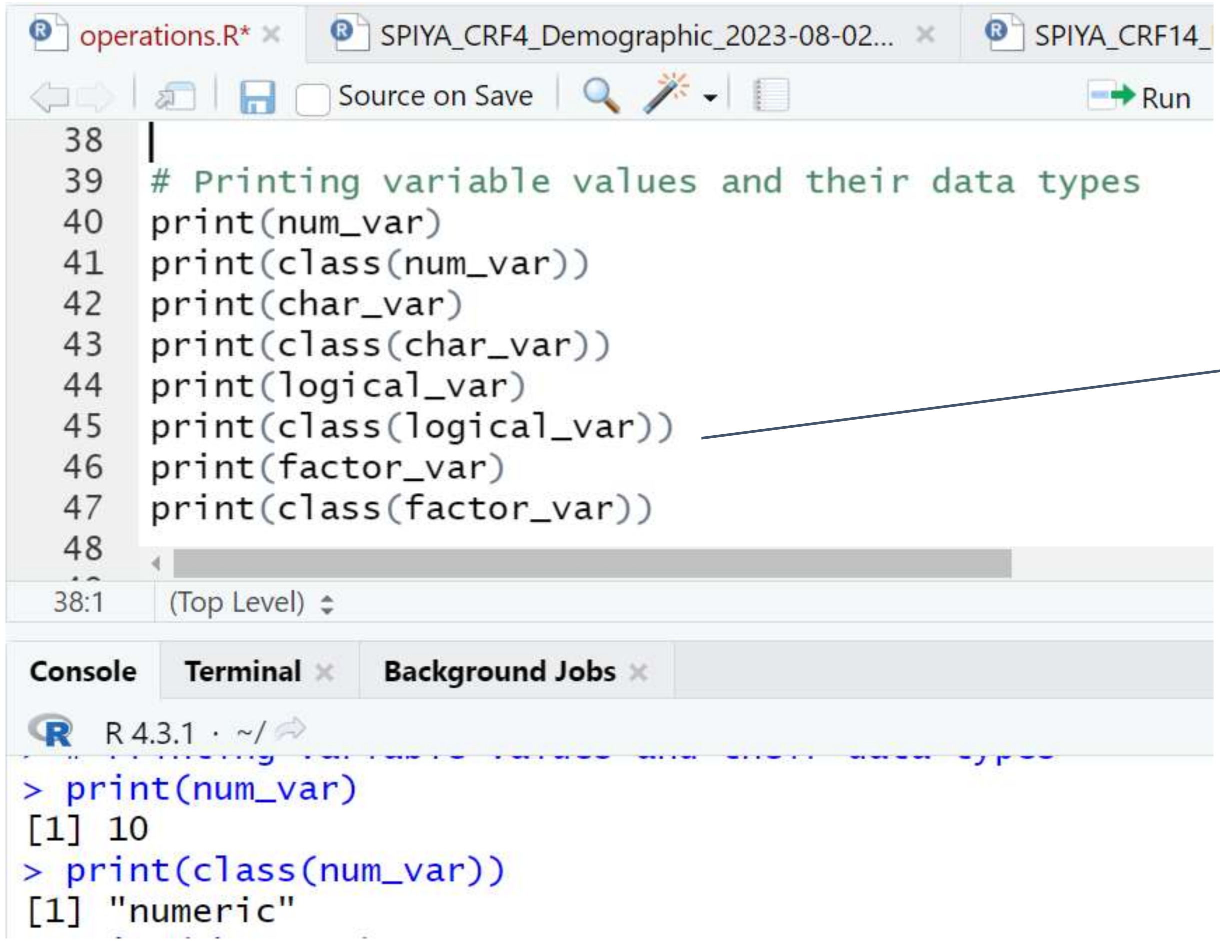
```r

# Printing variable values and their data types
print(num_var)
print(class(num_var))
print(char_var)
print(class(char_var))
print(logical_var)
print(class(logical_var))
print(factor_var)
print(class(factor_var))

```

```
> print(num_var)
[1] 10
> print(class(num_var))
[1] "numeric"
```



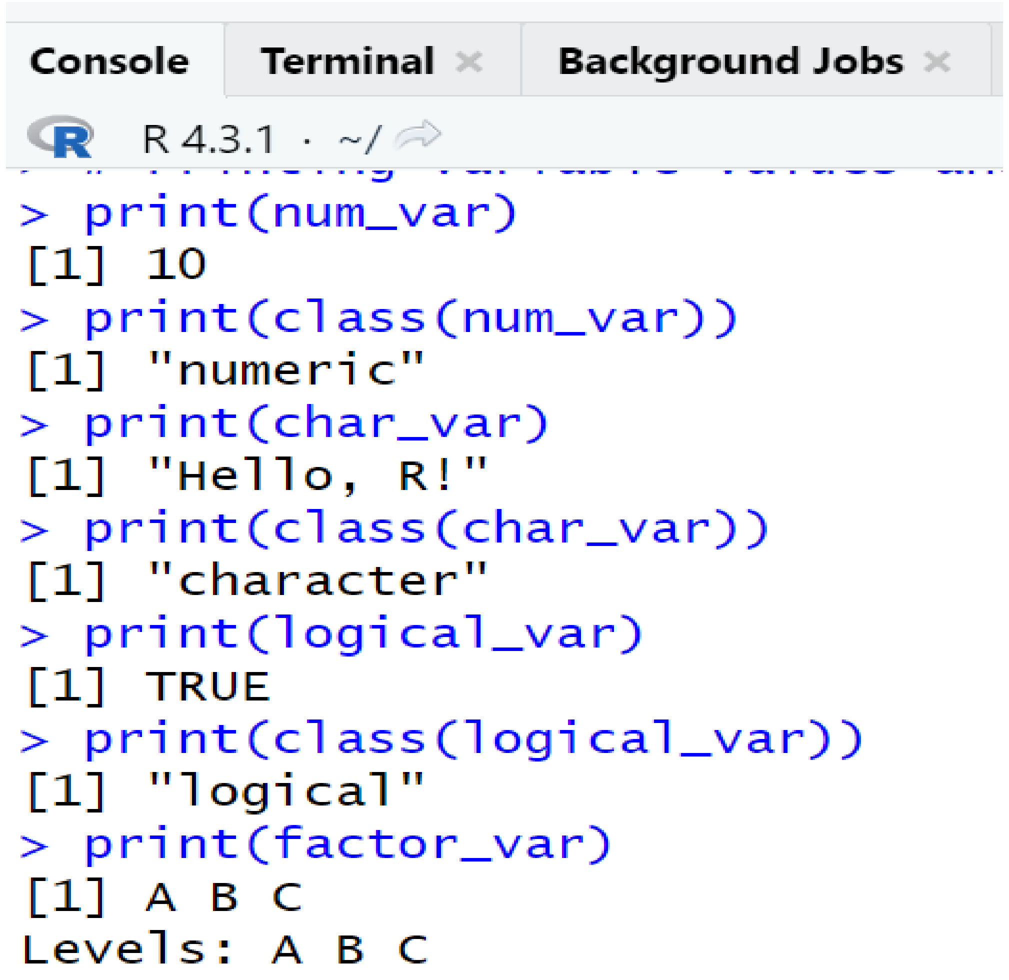
```
> print(num_var)
[1] 10
> print(class(num_var))
[1] "numeric"
> print(char_var)
[1] "Hello, R!"
> print(class(char_var))
[1] "character"
> print(logical_var)
[1] TRUE
> print(class(logical_var))
[1] "logical"
> print(factor_var)
[1] A B C
Levels: A B C
```

Variables and Data Types



```r

# Printing variable values and their data types
print(num_var)
print(class(num_var))
print(char_var)
print(class(char_var))
print(logical_var)
print(class(logical_var))
print(factor_var)
print(class(factor_var))

```

```
> print(num_var)
[1] 10
> print(class(num_var))
[1] "numeric"
```



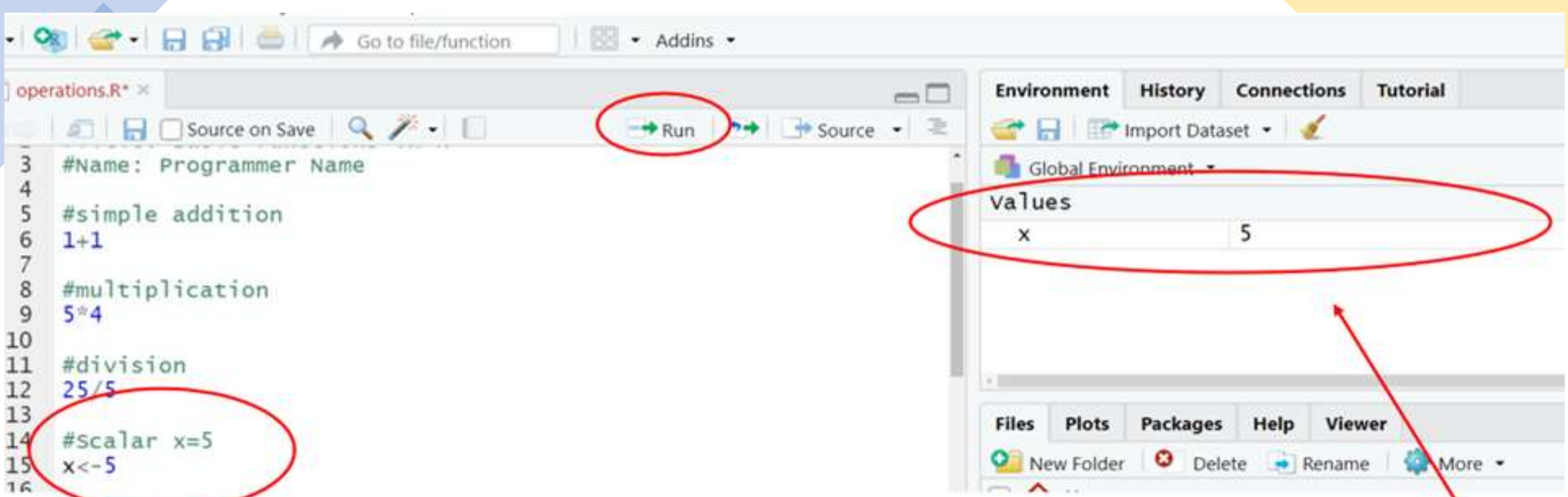
```
> print(num_var)
[1] 10
> print(class(num_var))
[1] "numeric"
> print(char_var)
[1] "Hello, R!"
> print(class(char_var))
[1] "character"
> print(logical_var)
[1] TRUE
> print(class(logical_var))
[1] "logical"
> print(factor_var)
[1] A B C
Levels: A B C
```

Data objects in R

- Scalars
- Vectors
- Factors
- Lists
- Data frames

Scalar

- Scalars are the **most simple** data type. They are object with one character or numeric value.
- **For example: x=5, w='Sunday'**
  - **x and w are the scalars**
- Recall the symbols "<-" used to create data objects which is known as the "assignment operator".
- The operationalization of scalars in R is shown on the next slide



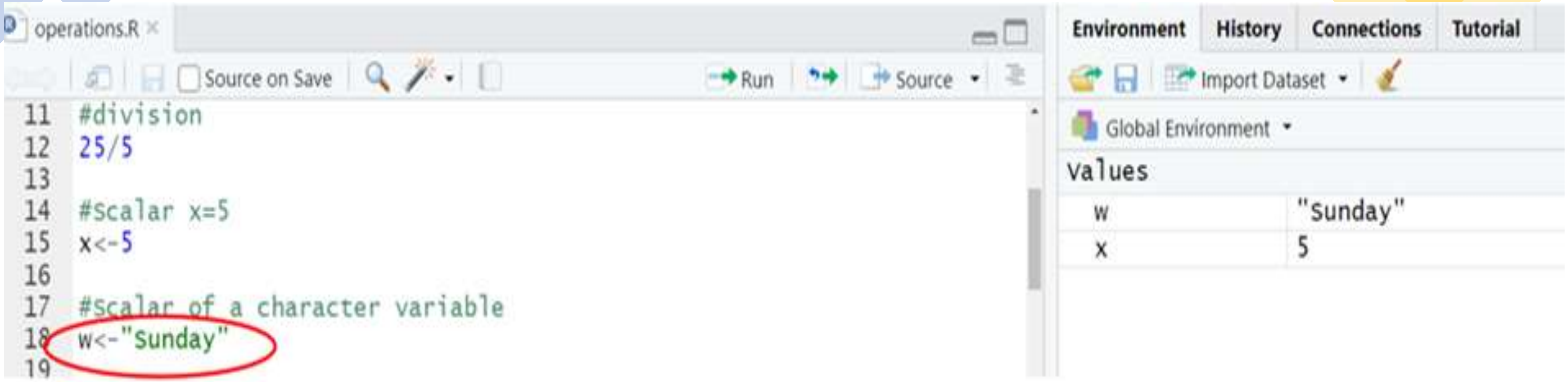
```
#Name: Programmer Name

#simple addition
1+1

#multiplication
5*4

#division
25/5

#Scalar x=5
x<-5
```

```
Values
  x                 5
```

Type the code x<-5, highlight the code, then select run

*Note: if it is a single line of code, you could also place your cursor on the line and then select run

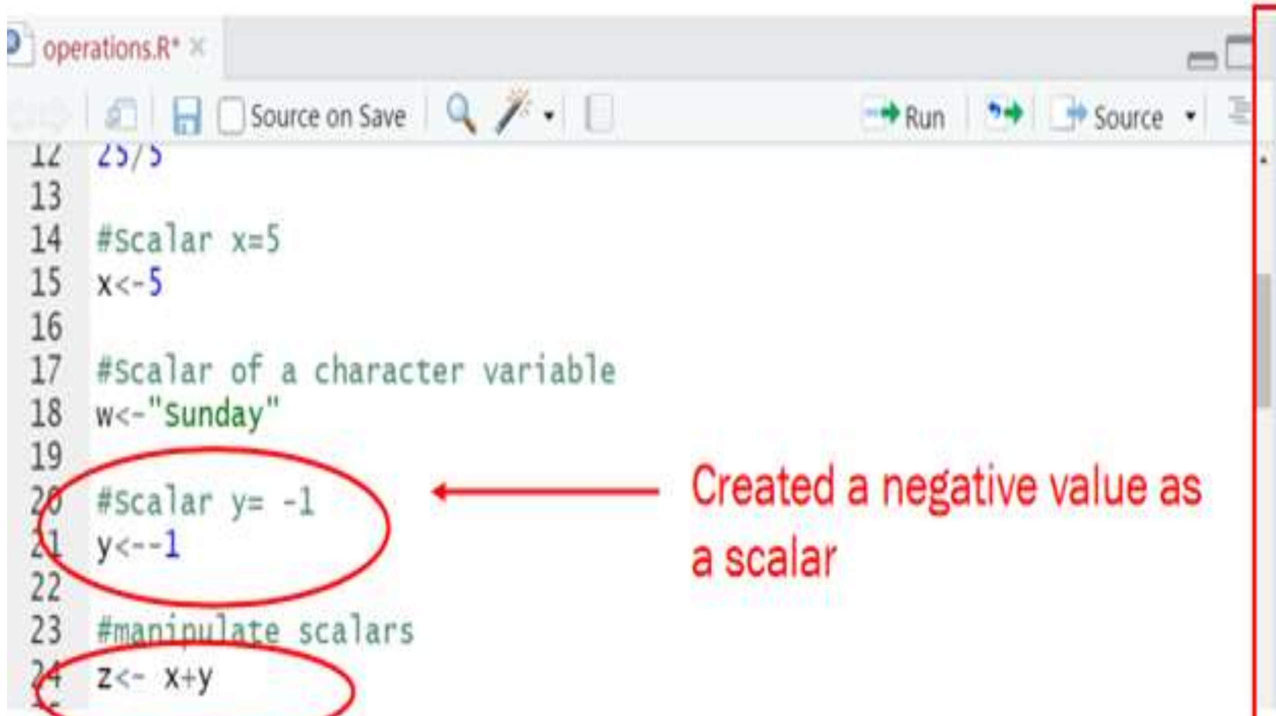
You will see the newly assigned value in the environment and history window



```
#division
25/5

#Scalar x=5
x<-5

#Scalar of a character variable
w<-"Sunday"
```

```
Values
  w     "Sunday"
  x     5
```

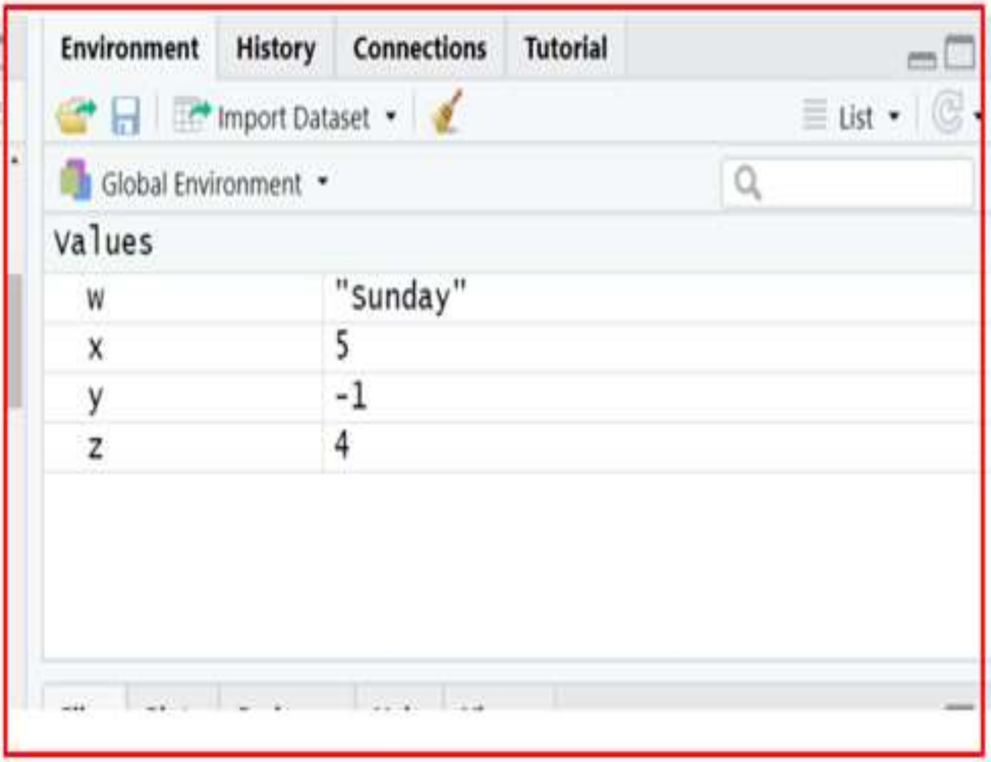
Although you could use double quotations or single quotations to create a character vector, be sure to be consistent and preferably stick to the use of double quotations.



```
25/5

#Scalar x=5
x<-5

#Scalar of a character variable
w<-"Sunday"

#Scalar y= -1
y<--1

#manipulate scalars
z<- x+y
```

Created a negative value as a scalar

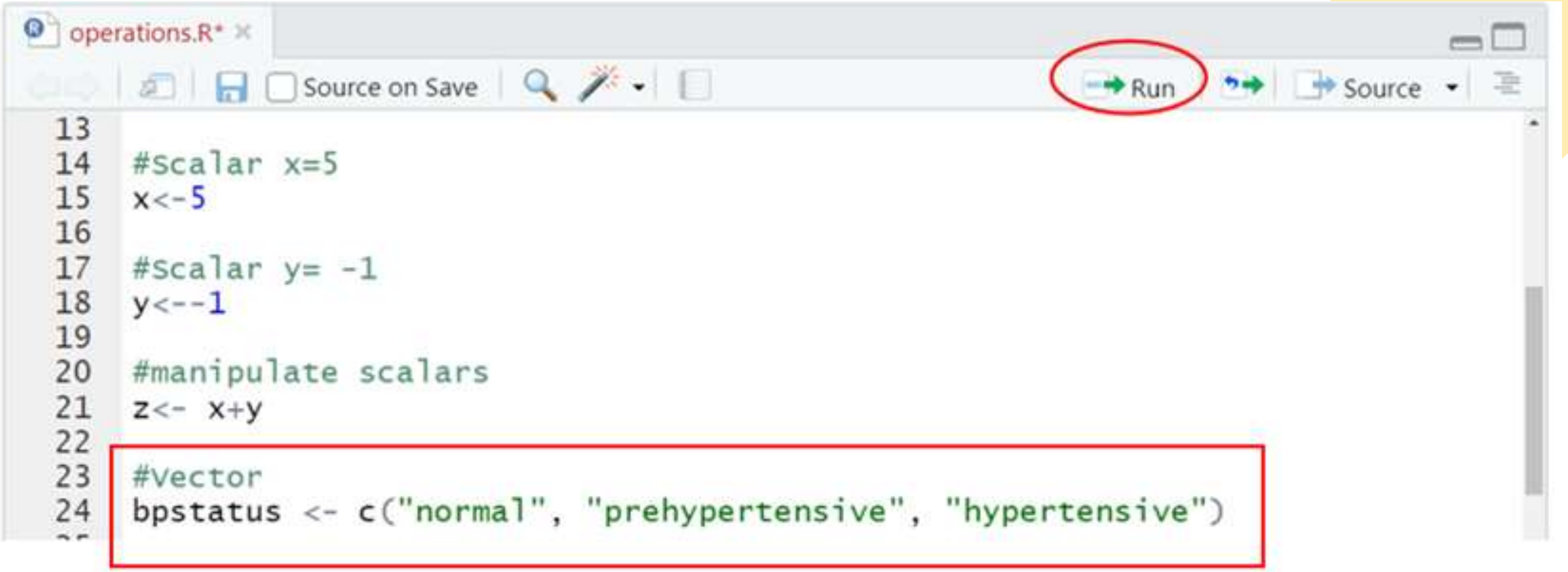
Summing two scalars to create a new scalar called “z”



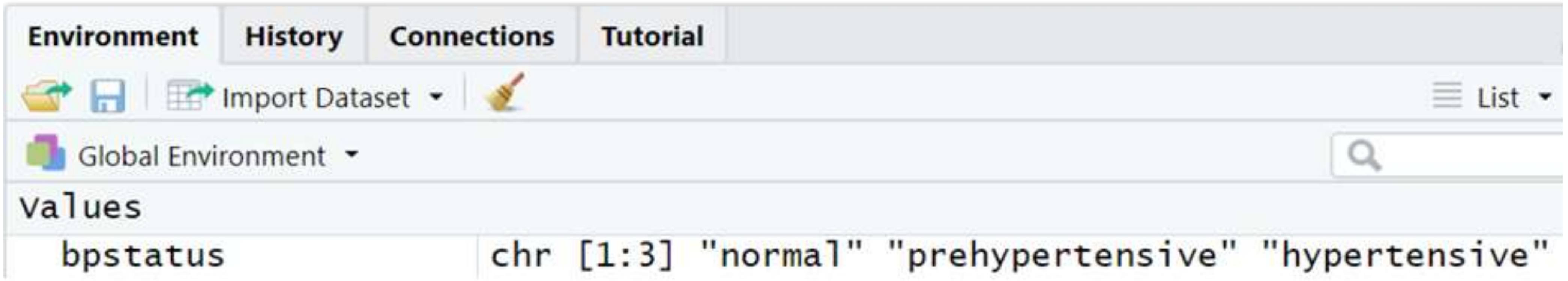
```
Values
  w       "Sunday"
  x       5
  y       -1
  z       4
```

Vectors

- **Most common data objects used in R**
- **Variables are generally stored as vectors**
- **Sequence of data elements of the same type:**
    - **Numerical vectors**
        - **1, 3, 5, 6**
    - **Character vectors**
        - **“normal” “prehypertensive” “hypertensive”**
    - **And more…**
- **Create a vector using “c()” function (shown on the next slide)**

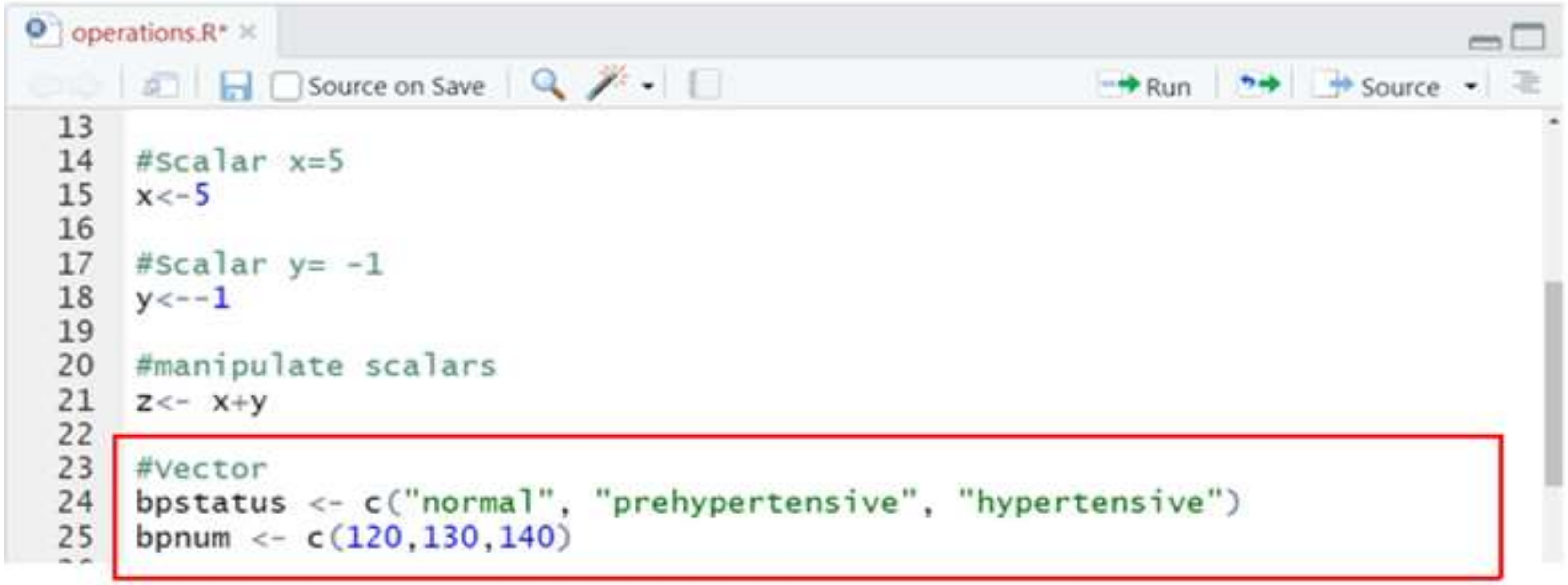


```
#Scalar x=5
x<-5

#Scalar y= -1
y<--1

#manipulate scalars
z<- x+y

#Vector
bpstatus <- c("normal", "prehypertensive", "hypertensive")
```

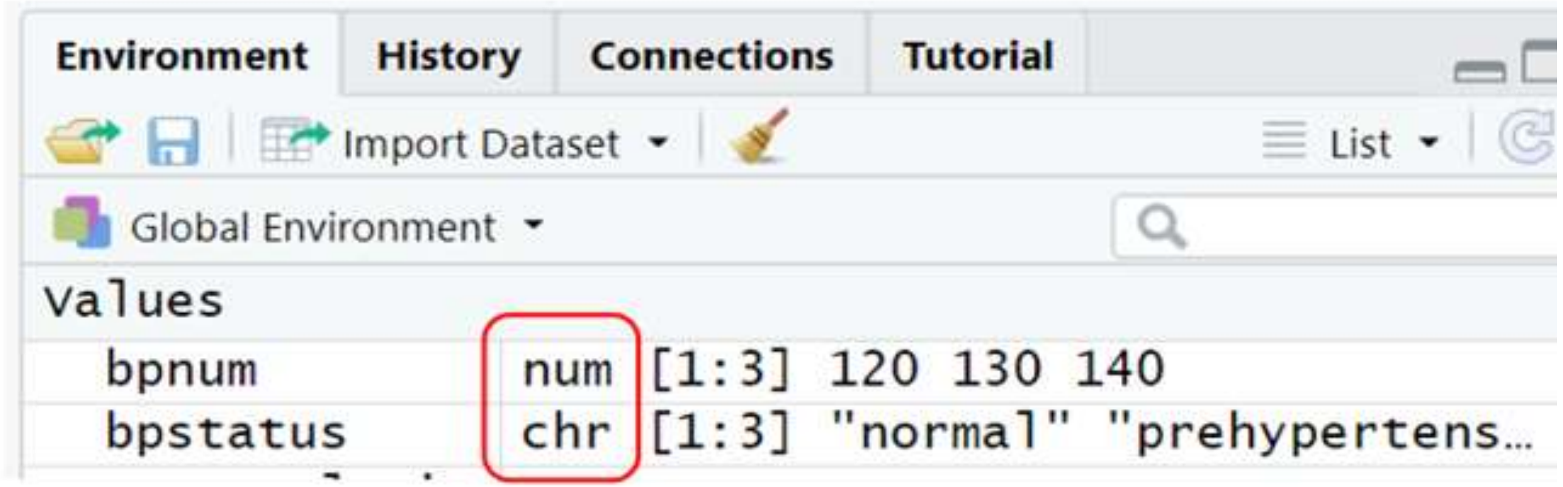


```
Values
  bpstatus          chr [1:3] "normal" "prehypertensive" "hypertensive"
```

Notice that in your environment and history window, you will see your newly created character vector designated as 'chr'



```
#Scalar x=5
x<-5

#Scalar y= -1
y<--1

#manipulate scalars
z<- x+y

#Vector
bpstatus <- c("normal", "prehypertensive", "hypertensive")
bpnum <- c(120,130,140)
```

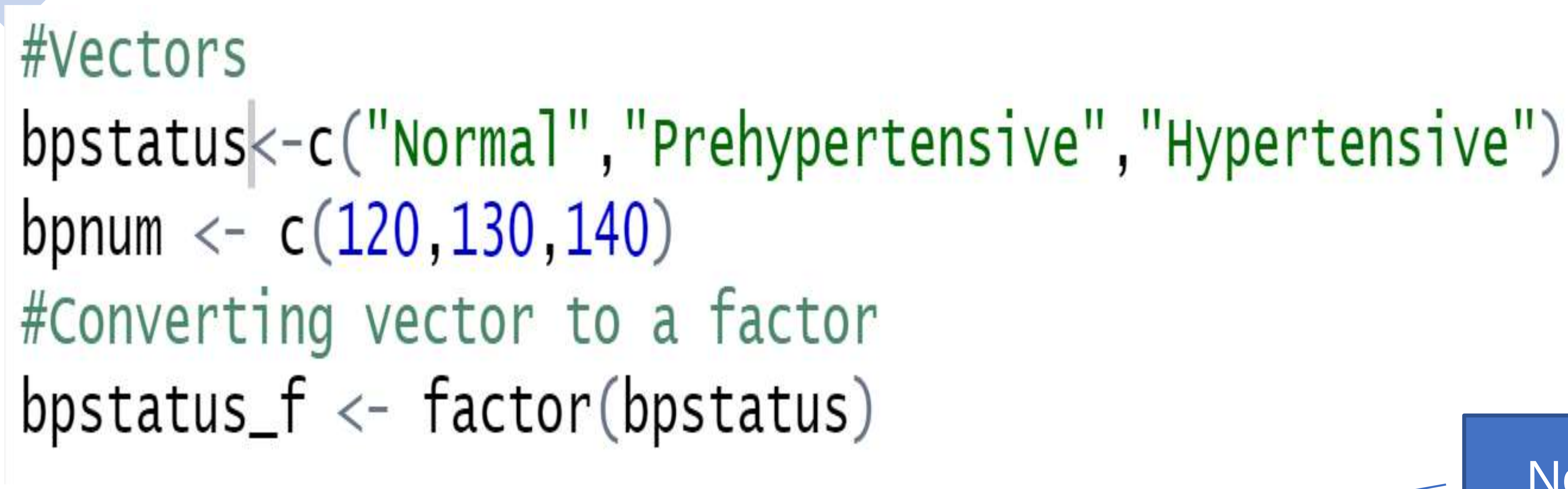


```
Values
  bpnum        num [1:3] 120 130 140
  bpstatus     chr [1:3] "normal" "prehypertens…
```

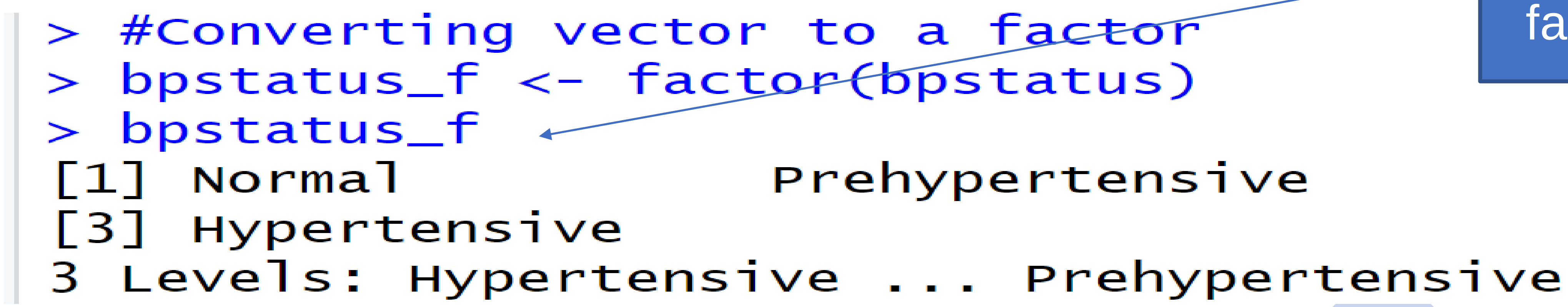
'num' stands for numerical vector and 'chr' stands for character vector

Factors

- Factors are vectors that are categorized.
- This has a special utility for example, in modeling or when constructing frequency tables
- The assigned labels (values) could be character, numeric, or Boolean (“AND”, “OR” and “NOT”)
- Creating a factor in R options (as shown on the next slide)



```r
#Vectors
bpstatus<-c("Normal","Prehypertensive","Hypertensive")
bpnum <- c(120,130,140)
#Converting vector to a factor
bpstatus_f <- factor(bpstatus)
```



```
> #Converting vector to a factor
> bpstatus_f <- factor(bpstatus)
> bpstatus_f
[1] Normal          Prehypertensive
[3] Hypertensive
3 Levels: Hypertensive ... Prehypertensive
```

New created factor variable

Lists

Simply put, lists are a collection of items in a particular order

- Can accommodate heterogenous elements

Lists can be useful for organizing information.

Unlike vectors, they can contain different modes/types of data.

- List of names: [Jill, Bill, Sally]
- List of numbers: [1, 5, 96]
- List made up of names and numbers: [“Jill”, 5, 6.8, “B″]

How to think of a list: ‘a general container’

- Movie: each movie has a cast, crew, budget, script, etc.
- List: can also contain multiple data frames (ie., datasets)

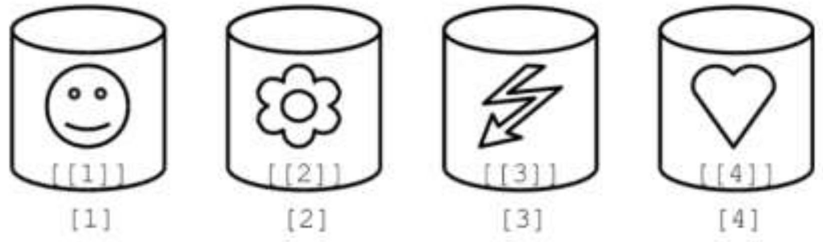


Figure 3.1: Schematic representation of a list of length four

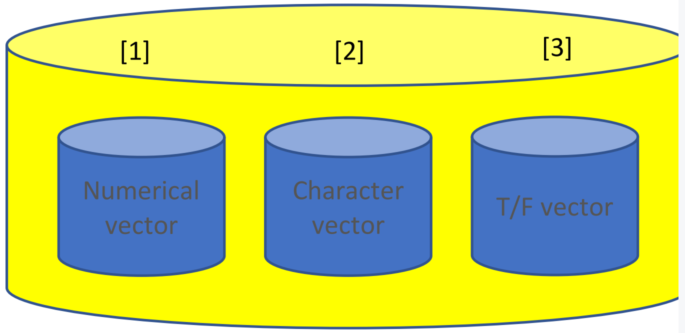
https://bookdown.org/medepi/phds/working-with-lists-and-data-frames.html

Lists in R

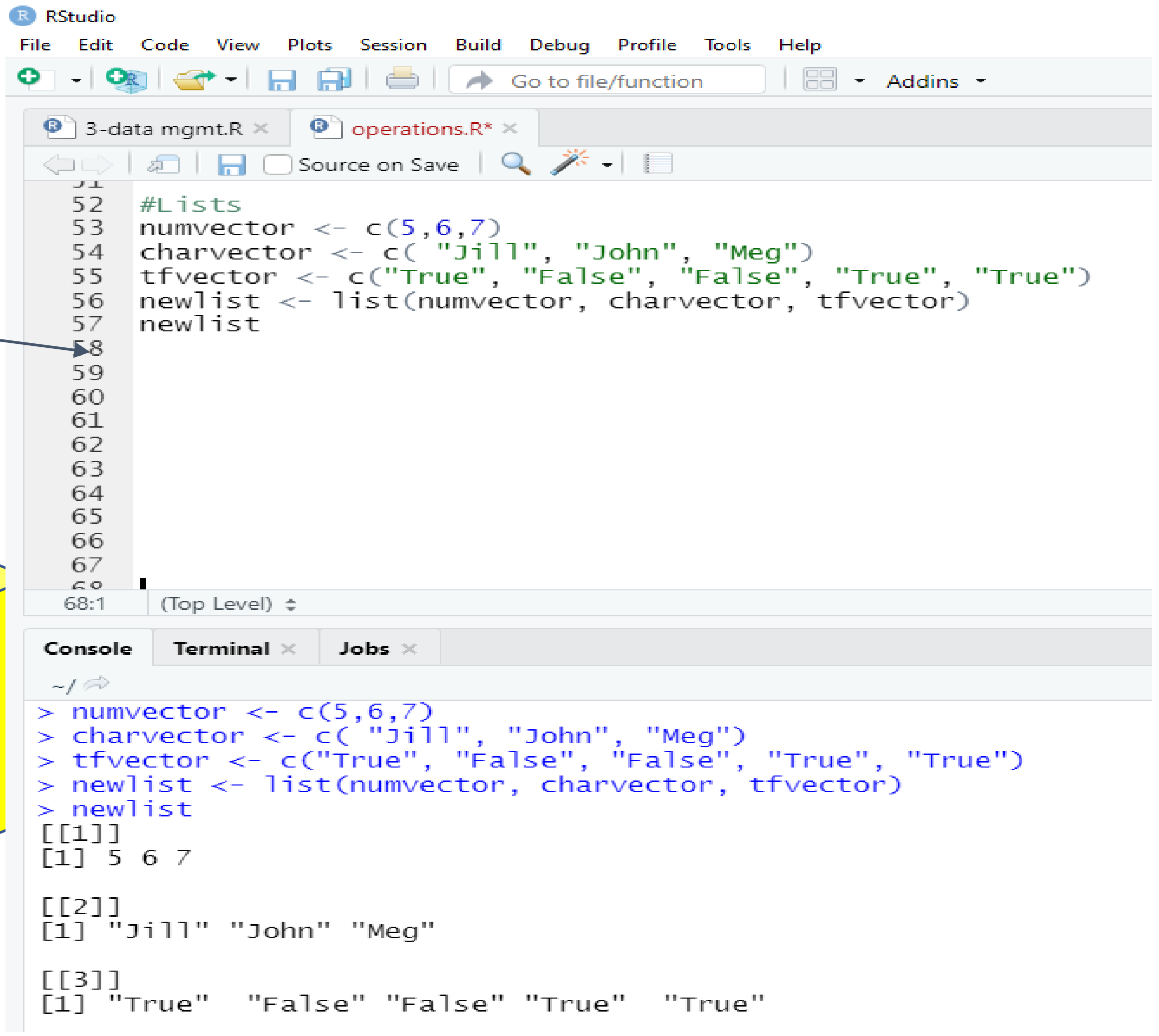
Here, we are creating a list to store 3 vectors with information we are interested in.

1) Numerical vector
2) Character vector
3) Vector with true false information

We first create the vectors and then combine them into a list.

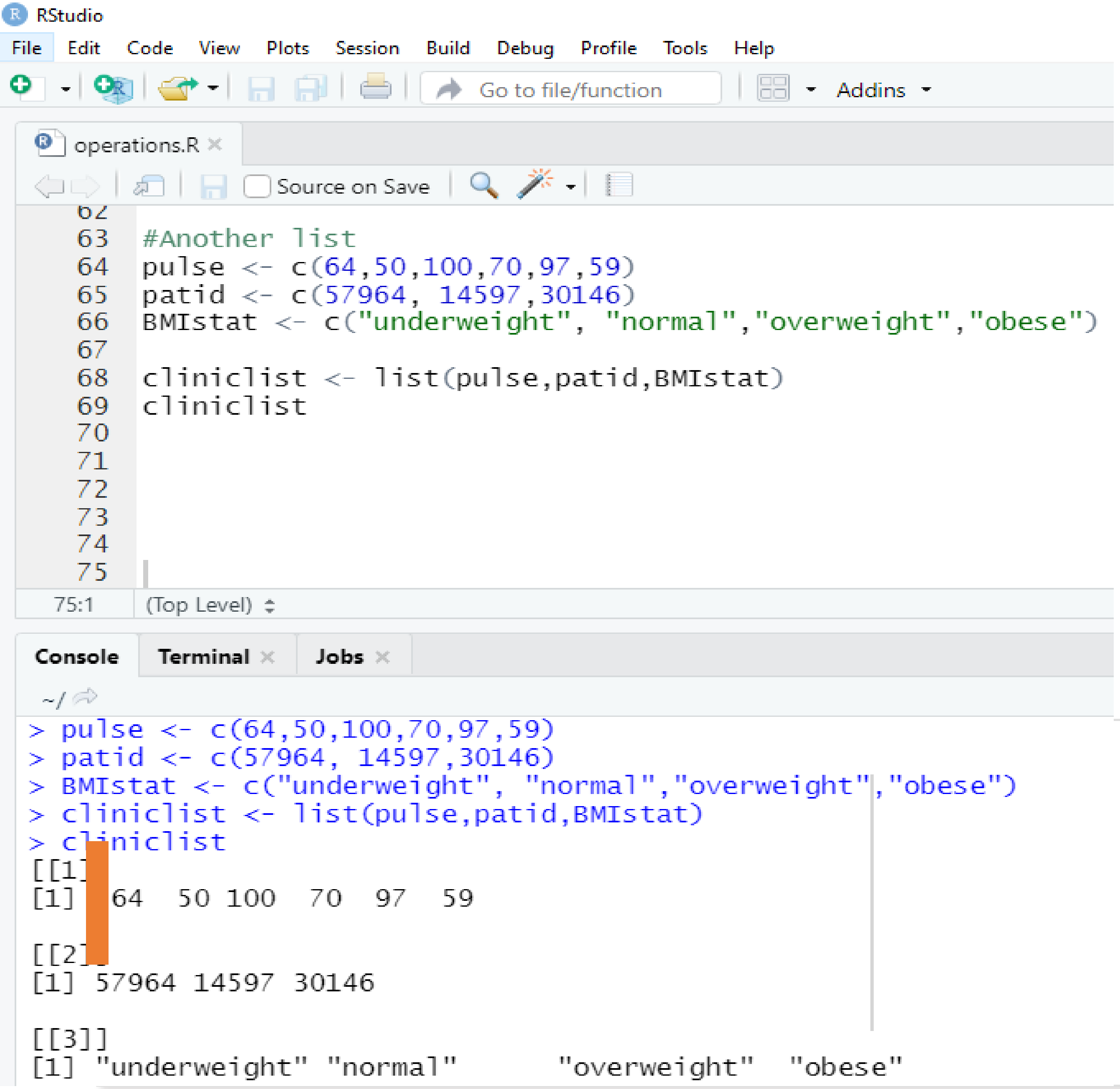


Yellow container represents list



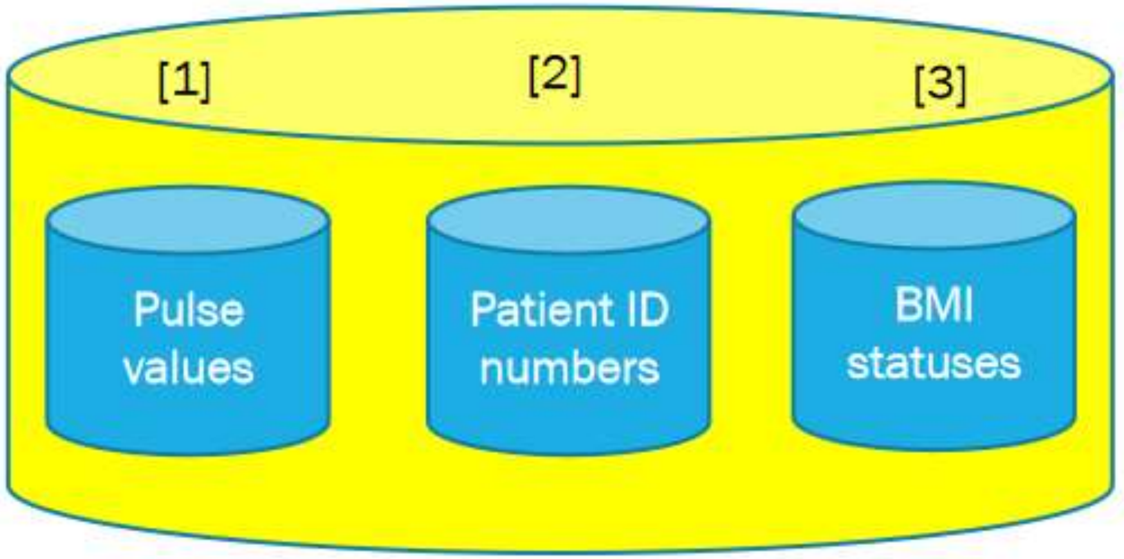
```
#Lists
numvector <- c(5,6,7)
charvector <- c( "Jill", "John", "Meg")
tfvector <- c("True", "False", "False", "True", "True")
newlist <- list(numvector, charvector, tfvector)
newlist
```

```
> numvector <- c(5,6,7)
> charvector <- c( "Jill", "John", "Meg")
> tfvector <- c("True", "False", "False", "True", "True")
> newlist <- list(numvector, charvector, tfvector)
> newlist
[[1]]
[1] 5 6 7

[[2]]
[1] "Jill" "John" "Meg"

[[3]]
[1] "True"  "False" "False" "True"  "True"
```



```r
#Another list
pulse <- c(64,50,100,70,97,59)
patid <- c(57964, 14597,30146)
BMIstat <- c("underweight", "normal","overweight","obese")

cliniclist <- list(pulse,patid,BMIstat)
cliniclist
```

```
> pulse <- c(64,50,100,70,97,59)
> patid <- c(57964, 14597,30146)
> BMIstat <- c("underweight", "normal","overweight","obese")
> cliniclist <- list(pulse,patid,BMIstat)
> cliniclist
[[1]]
[1]  64  50 100  70  97  59

[[2]]
[1] 57964 14597 30146

[[3]]
[1] "underweight" "normal"      "overweight"  "obese"
```



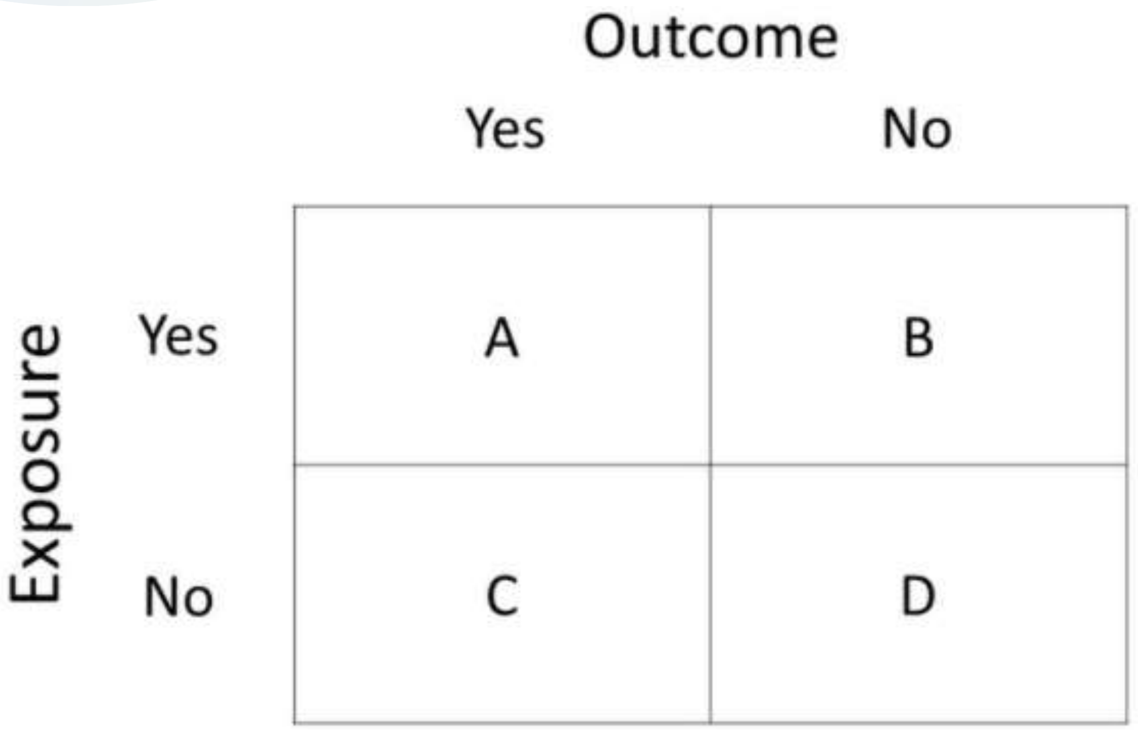
Yellow container represents list

- Here, we are creating a list to store 3 vectors with information we are interested in:

1) Pulse values
2) Patient ID numbers
3) BMI statuses (underweight, normal weight, overweight, obese)

- We first create the vectors and then combine them into a list.

Matrices



		Outcome	
		Yes	No
Exposure	Yes	A	B
	No	C	D

- A matrix is a collection of elements of the same data type such as numeric, character, etc. arranged into *n* number of rows and *n* number of columns. For example:
  - A 2x2 table is an example of a matrix with two rows and two columns
- Putting your data into a matrix/table format is an efficient way to analyze data in R.

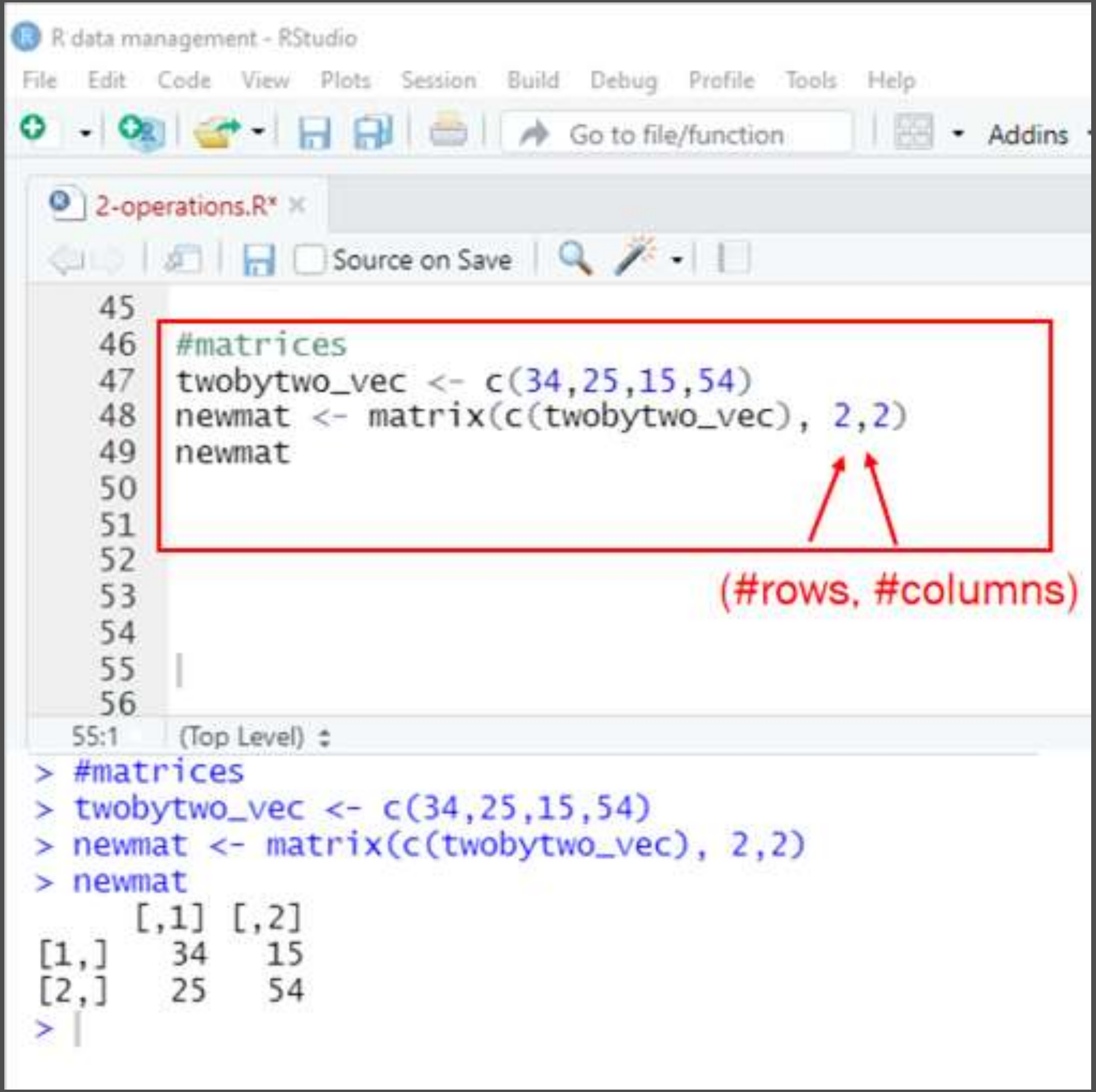
Matrices in R

	Disease	No Disease
Exposed	34	15
Non-exposed	25	54

First, we make one vector entering the information from our 2x2 table.

Next, we are creating a 2x2 matrix using the matrix function. We will name our matrix 'newmat' using the two previous vectors you created. After you input your values, specify the (#rows, #columns).

You will see the matrix created here in the console window.

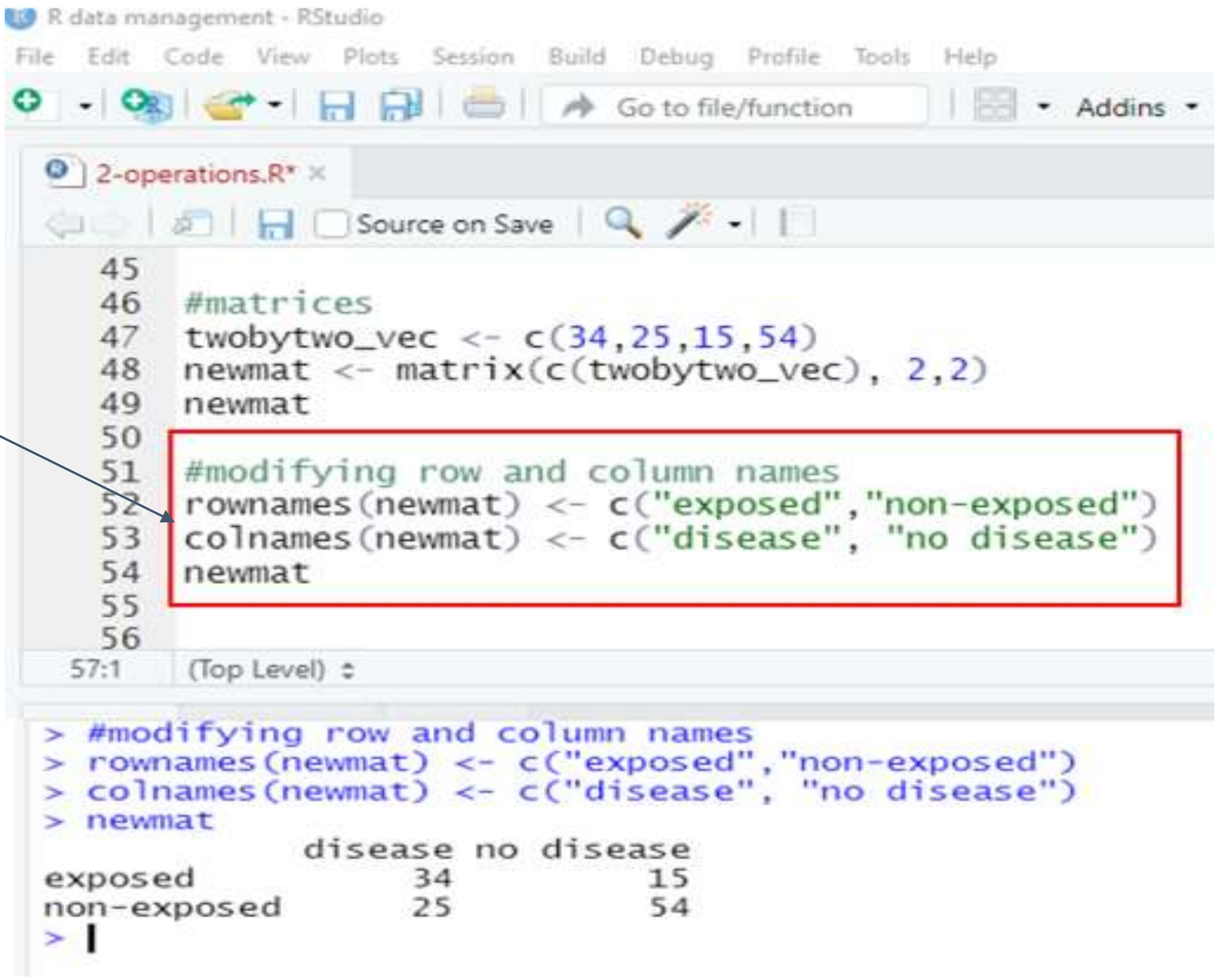

R data management - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
2-operations.R*
Source on Save
55:1
(Top Level)
(#rows, #columns)

```
#matrices
twobytwo_vec <- c(34,25,15,54)
newmat <- matrix(c(twobytwo_vec), 2,2)
newmat
```

```
> #matrices
> twobytwo_vec <- c(34,25,15,54)
> newmat <- matrix(c(twobytwo_vec), 2,2)
> newmat
     [,1] [,2]
[1,]   34   15
[2,]   25   54
>
```

Matrices in R

If you want to change the row names and column names use the 'rownames' and 'colnames' command on the 'newmat' matrix


R data management - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
2-operations.R*
Source on Save
57:1
(Top Level)

```
#matrices
twobytwo_vec <- c(34,25,15,54)
newmat <- matrix(c(twobytwo_vec), 2,2)
newmat

#modifying row and column names
rownames(newmat) <- c("exposed","non-exposed")
colnames(newmat) <- c("disease", "no disease")
newmat
```

```
> #modifying row and column names
> rownames(newmat) <- c("exposed","non-exposed")
> colnames(newmat) <- c("disease", "no disease")
> newmat
            disease no disease
exposed          34         15
non-exposed      25         54
> 
```

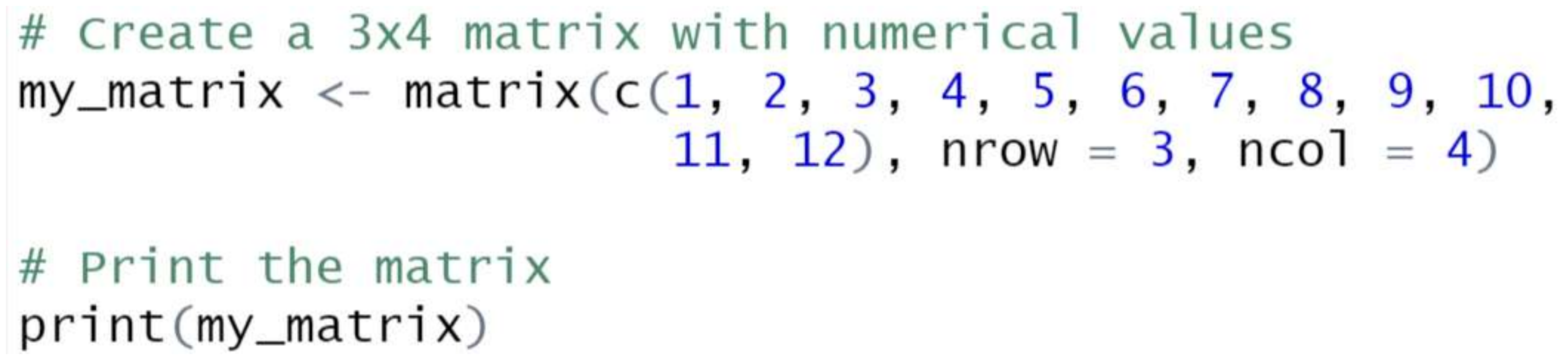
You will see that the rows and columns have been re-named

Exercises

1. Create a 3x4 matrix containing numerical values and display the result.
2. Create a 5x2 matrix containing character variables and display the result.

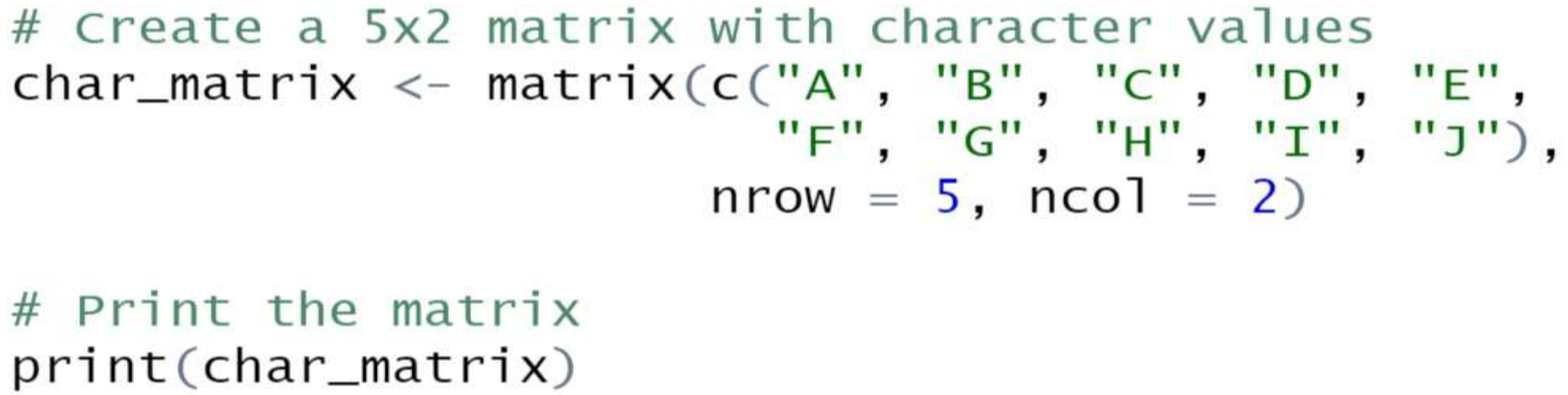
Solutions

1.



```r
# Create a 3x4 matrix with numerical values
my_matrix <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                      11, 12), nrow = 3, ncol = 4)

# Print the matrix
print(my_matrix)
```

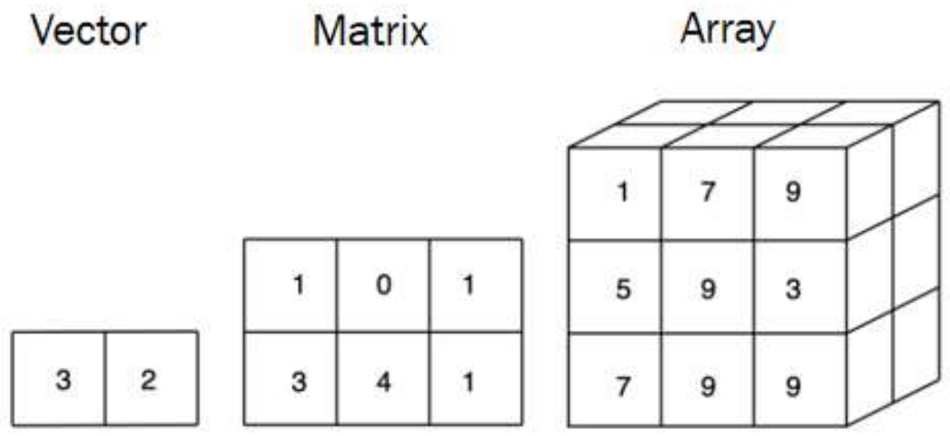
2.

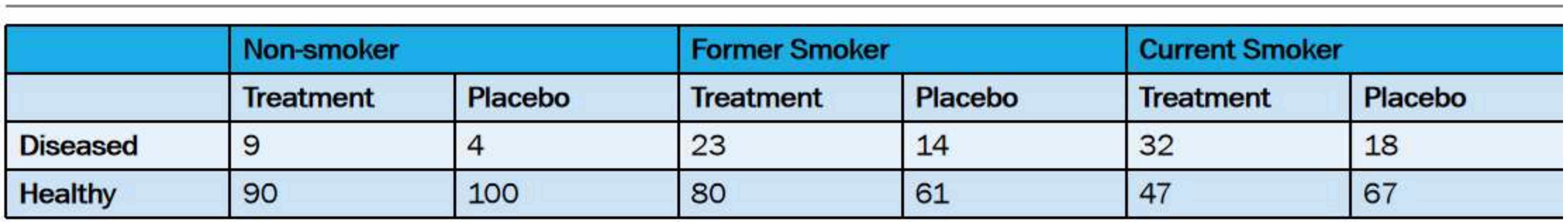


```r
# Create a 5x2 matrix with character values
char_matrix <- matrix(c("A", "B", "C", "D", "E",
                        "F", "G", "H", "I", "J"),
                      nrow = 5, ncol = 2)

# Print the matrix
print(char_matrix)
```

Arrays

- An array is a generalization of matrices to n-dimensions.
- It may be easier to think of arrays as stratified tables. (example on next slide)





	Non-smoker		Former Smoker		Current Smoker	
	Treatment	Placebo	Treatment	Placebo	Treatment	Placebo
Diseased	9	4	23	14	32	18
Healthy	90	100	80	61	47	67

This is an example of a three dimensional array:

Outcome status vs. treatment status vs. Smoking status

Arrays: practical example

We will now learn how to create the array in R (next slide)

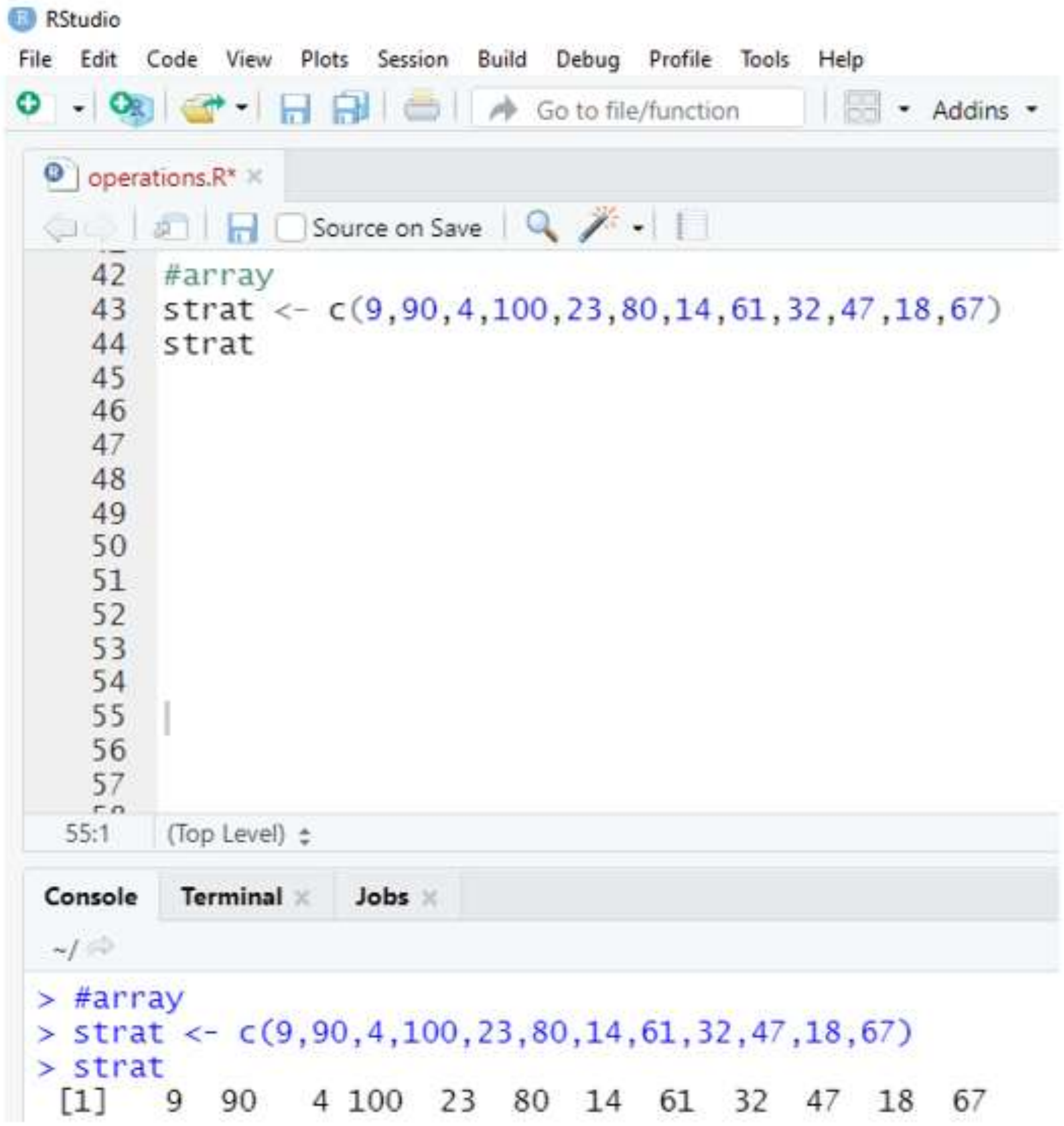
	Non-smoker		Former Smoker		Current Smoker	
	Treatment	Placebo	Treatment	Placebo	Treatment	Placebo
Diseased	9	4	23	14	32	18
Healthy	90	100	80	61	47	67

Enter the data by columns

Start by making a numerical vector arbitrarily named 'strat'

*for matrices you have to input information column-wise and since this is a collection of 3 matrices, you have to follow the same format

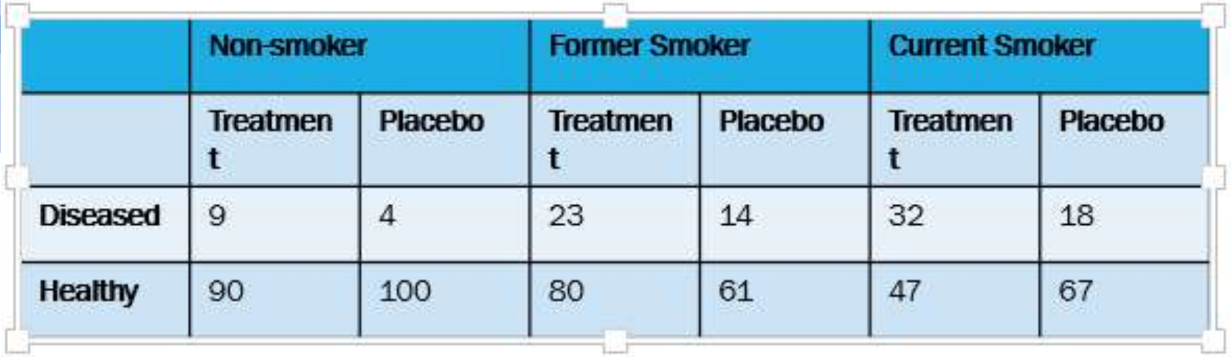
Strat vector is printed here



```
#array
strat <- c(9,90,4,100,23,80,14,61,32,47,18,67)
strat
```

```
> #array
> strat <- c(9,90,4,100,23,80,14,61,32,47,18,67)
> strat
 [1]   9  90   4 100  23  80  14  61  32  47  18  67
```

VN-Biostat Pre-Workshop

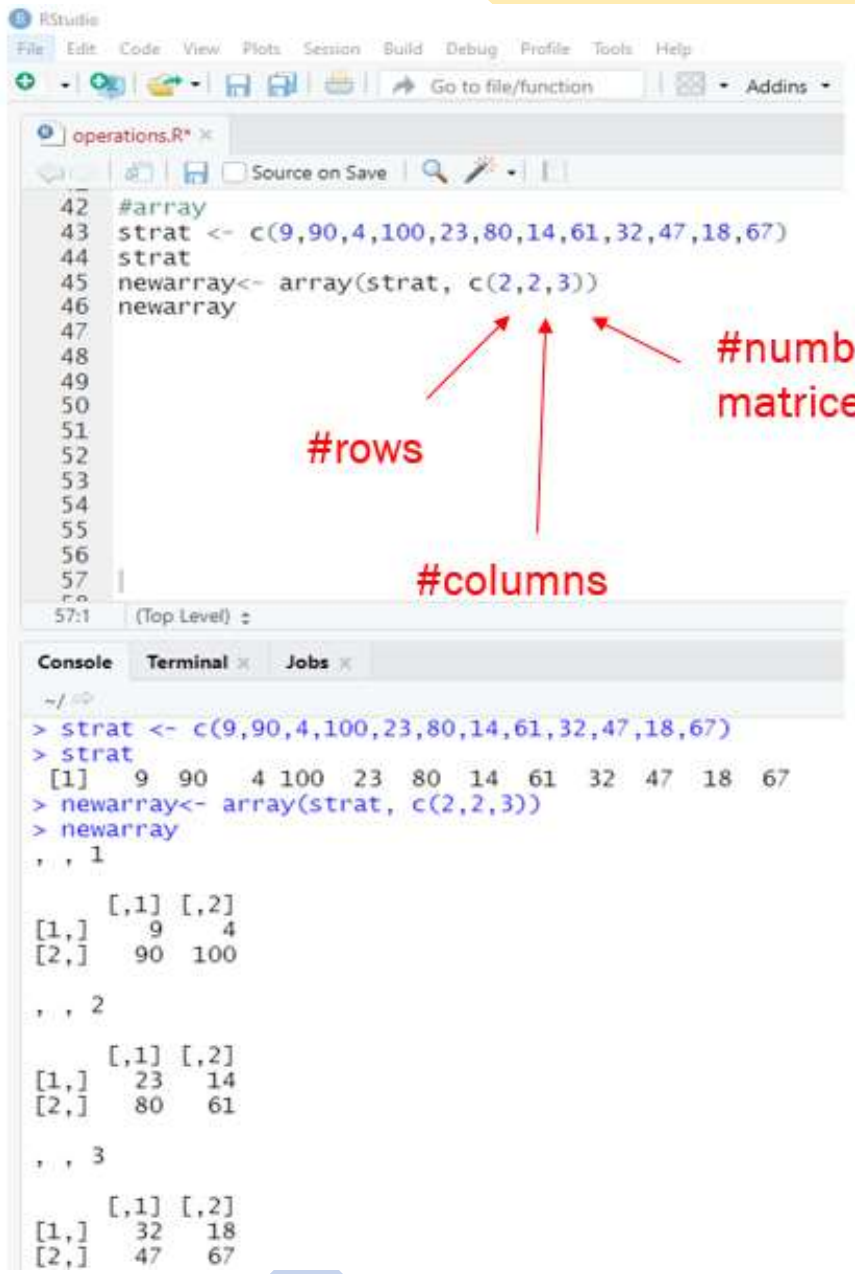


	Non-smoker		Former Smoker		Current Smoker	
	Treatmen t	Placebo	Treatmen t	Placebo	Treatmen t	Placebo
Diseased	9	4	23	14	32	18
Healthy	90	100	80	61	47	67

Use the "array" command to create a three-dimensional array.

We are creating an array called 'newarray' by taking the information in the strat

You will see your tables down here



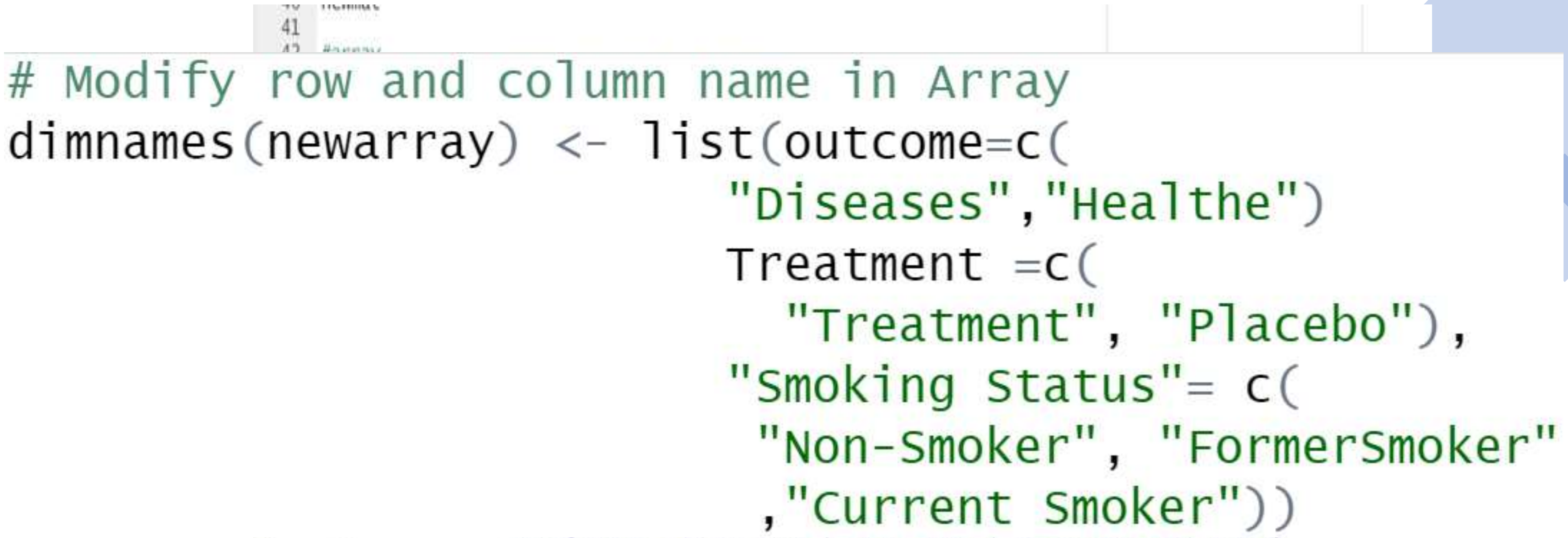
```
#array
strat <- c(9,90,4,100,23,80,14,61,32,47,18,67)
strat
newarray<- array(strat, c(2,2,3))
newarray
```

```
> strat <- c(9,90,4,100,23,80,14,61,32,47,18,67)
> strat
 [1]   9  90   4 100  23  80  14  61  32  47  18  67
> newarray<- array(strat, c(2,2,3))
> newarray
, , 1

     [,1] [,2]
[1,]    9    4
[2,]   90  100

, , 2

     [,1] [,2]
[1,]   23   14
[2,]   80   61

, , 3

     [,1] [,2]
[1,]   32   18
[2,]   47   67
```

We have to modify the names of the dimensions. We start with rows. We will add a header “Outcome” in which we will include two categories: diseased or healthy. We add another header called “treatment” containing “treatment” or “placebo”

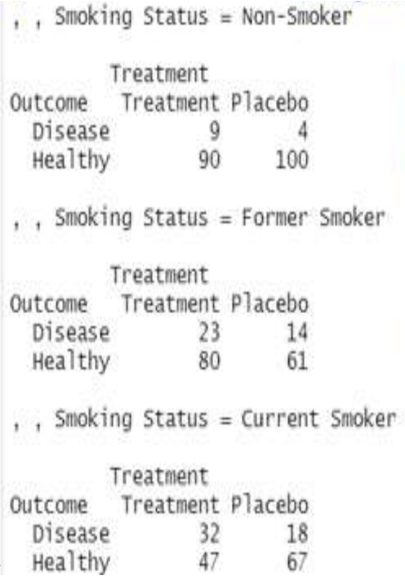
Then we add a comma, and stratify by [3] matrices for smoking status: none, former, current.

*Notice that we are using a “list” command.

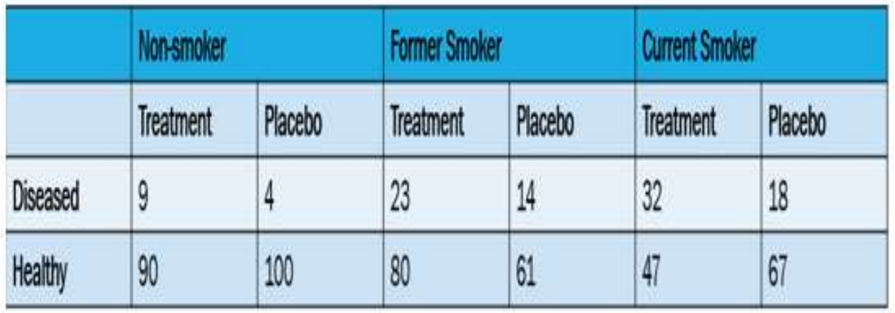


```
# Modify row and column name in Array
dimnames(newarray) <- list(outcome=c(
                         "Diseases","Healthe")
                         Treatment =c(
                           "Treatment", "Placebo"),
                         "Smoking Status"= c(
                          "Non-Smoker", "FormerSmoker"
                          ,"Current Smoker"))
```

Output is shown here.



```
, , Smoking Status = Non-Smoker

         Treatment
Outcome   Treatment Placebo
  Disease         9       4
  Healthy        90     100

, , Smoking Status = Former Smoker

         Treatment
Outcome   Treatment Placebo
  Disease        23      14
  Healthy        80      61

, , Smoking Status = Current Smoker

         Treatment
Outcome   Treatment Placebo
  Disease        32      18
  Healthy        47      67
```

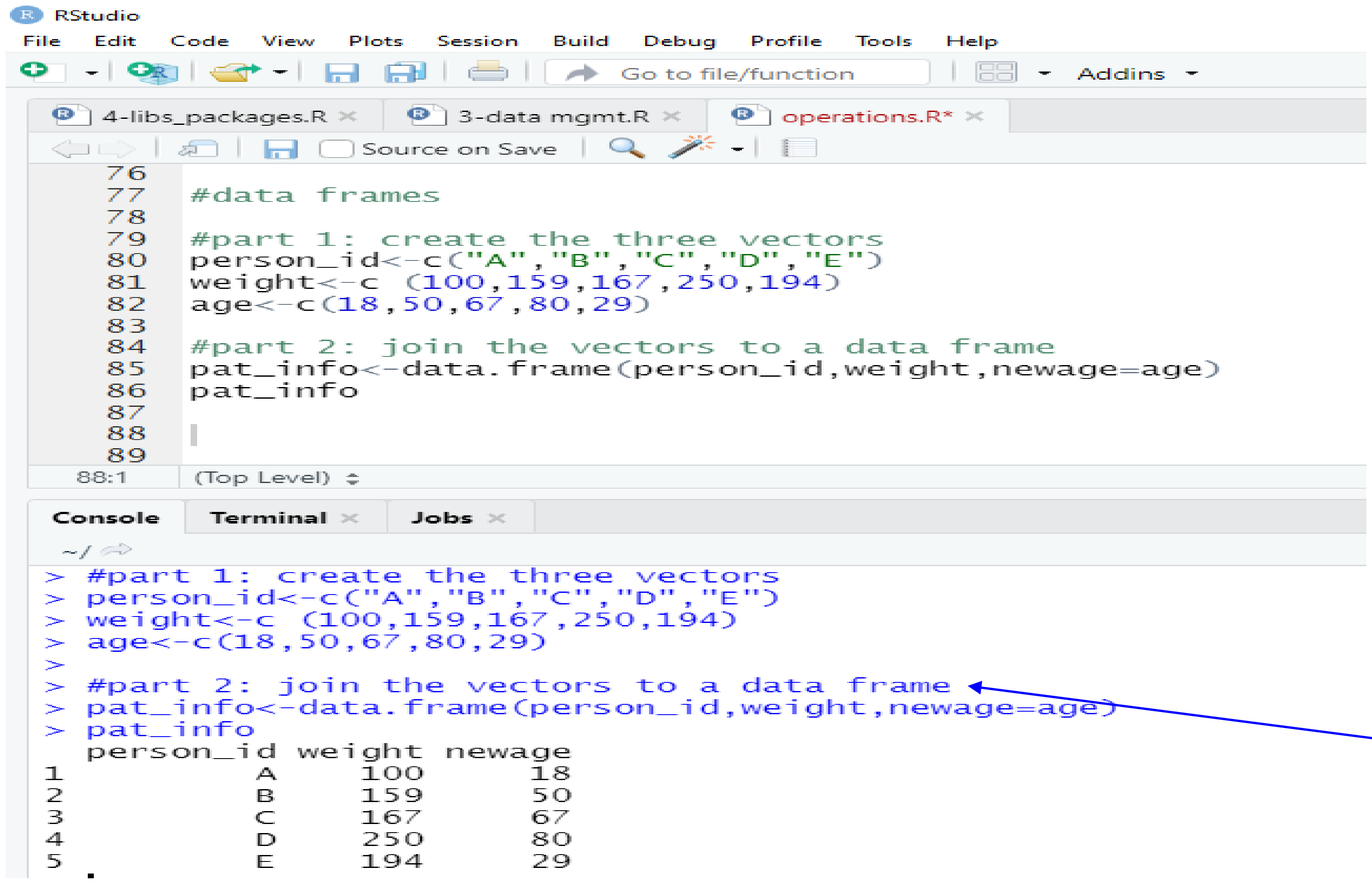


	Non-smoker		Former Smoker		Current Smoker	
	Treatment	Placebo	Treatment	Placebo	Treatment	Placebo
Diseased	9	4	23	14	32	18
Healthy	90	100	80	61	47	67

DATA FRAMES

- Data frames are versatile data objects in R and can be thought of as spreadsheets.
- Each column of a data frame is a vector with its own data elements
- Example on next slide

DATA FRAMES



```
#data frames

#part 1: create the three vectors
person_id<-c("A","B","C","D","E")
weight<-c (100,159,167,250,194)
age<-c(18,50,67,80,29)

#part 2: join the vectors to a data frame
pat_info<-data.frame(person_id,weight,newage=age)
pat_info
```

```
> #part 1: create the three vectors
> person_id<-c("A","B","C","D","E")
> weight<-c (100,159,167,250,194)
> age<-c(18,50,67,80,29)
>
> #part 2: join the vectors to a data frame
> pat_info<-data.frame(person_id,weight,newage=age)
> pat_info
  person_id weight newage
1         A    100     18
2         B    159     50
3         C    167     67
4         D    250     80
5         E    194     29
```

Part 1: Creating 3 vectors called: person_id, weight, and age

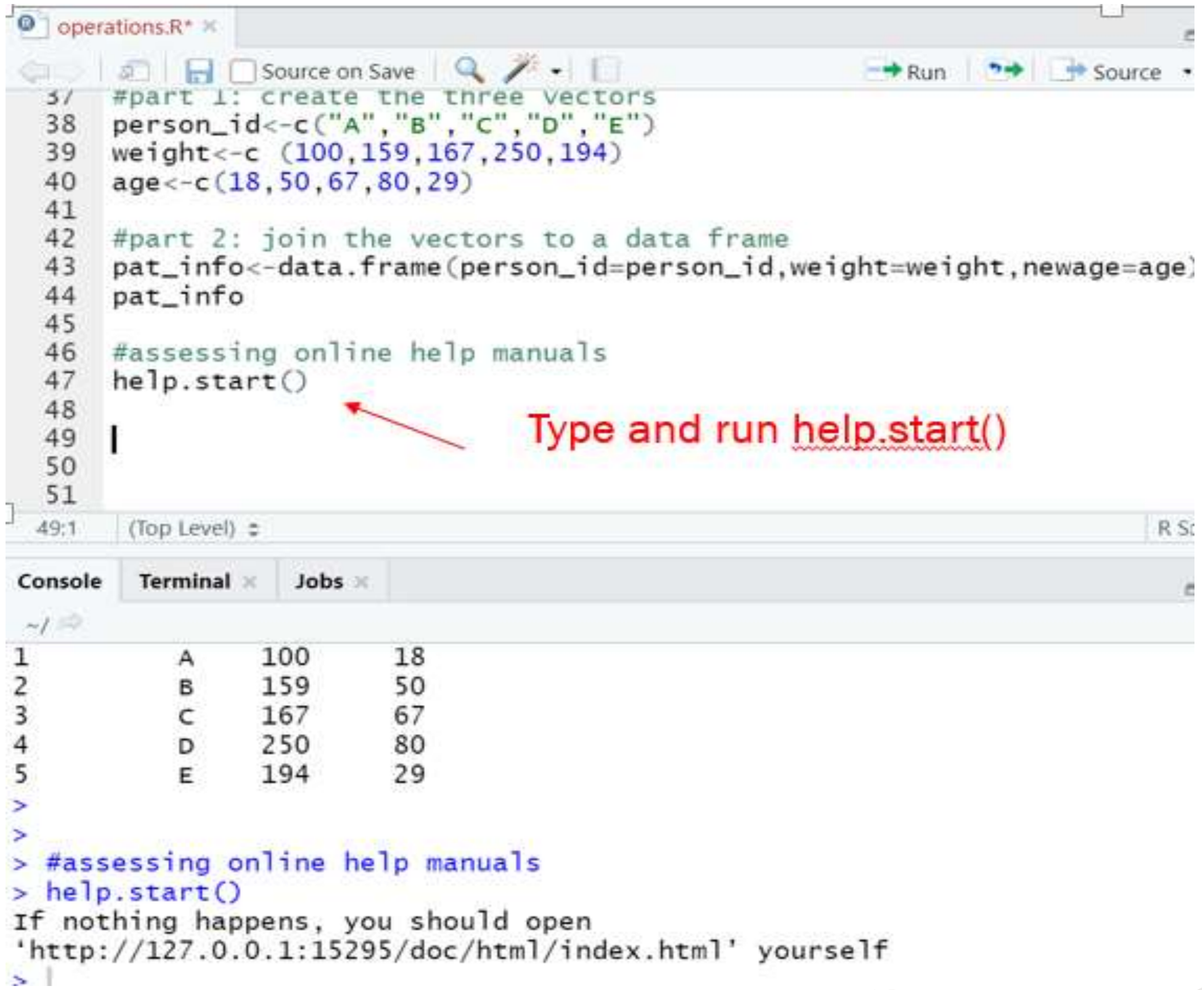
Part 2: creating a dataframe named pat_info containing all previously created vectors using the 'data.frame' function

You can choose to rename your columns in the new data frame. For instance, I'm creating a column called newage that will contain all data elements from the age vector created in part 1.

Type pat_info to view your dataframe in the console window.

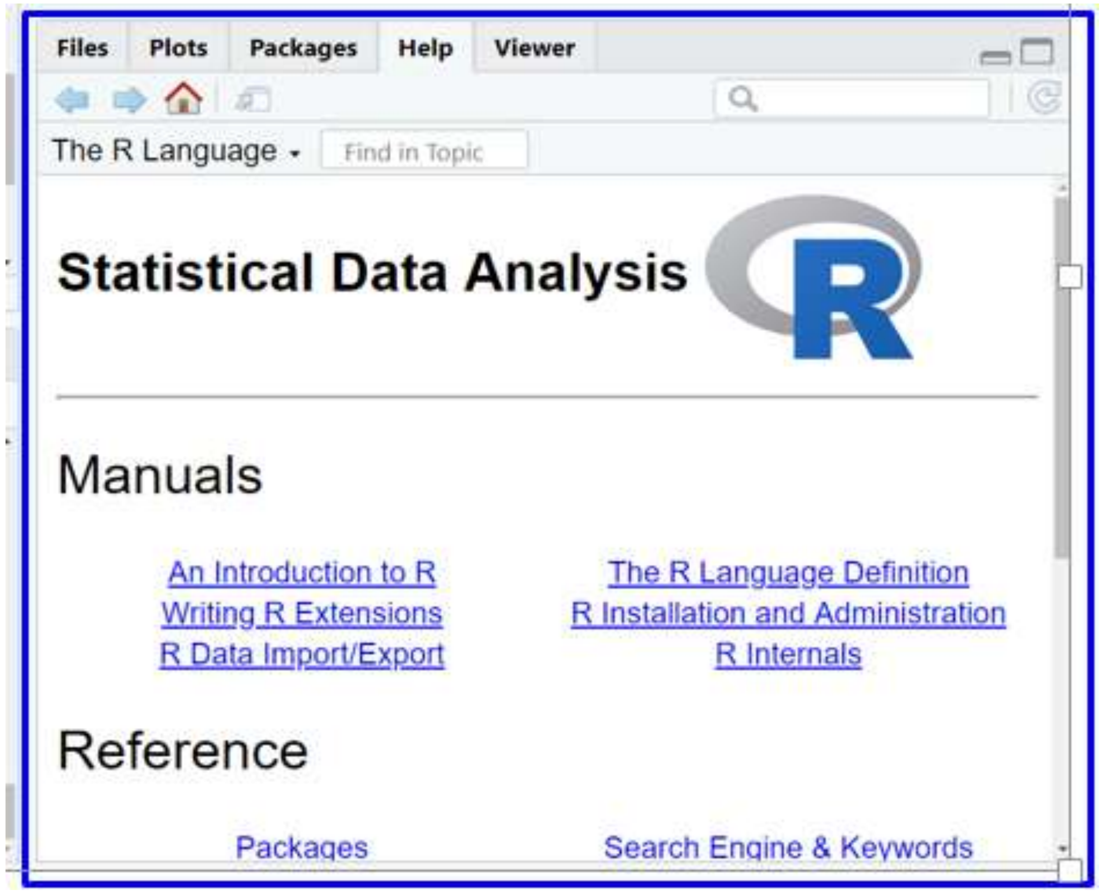
Assessing online help manuals

help.start()



```
#part 1: create the three vectors
person_id<-c("A","B","C","D","E")
weight<-c (100,159,167,250,194)
age<-c(18,50,67,80,29)

#part 2: join the vectors to a data frame
pat_info<-data.frame(person_id=person_id,weight=weight,newage=age)
pat_info

#assessing online help manuals
help.start()
```

```
1          A     100       18
2          B     159       50
3          C     167       67
4          D     250       80
5          E     194       29
>
>
> #assessing online help manuals
> help.start()
If nothing happens, you should open
'http://127.0.0.1:15295/doc/html/index.html' yourself
>
```



You will notice the help window pop up in the bottom right corner

Lecture summary

- R syntax
- Variables and Data Types
- Vectors, Matrices and Data Frames
- Data objects in R
    - Scalars
    - Vectors
    - Factors
    - Lists
    - Matrices
    - Arrays
    - Data frames
- Online helps

References

- https://bookdown.org/medepi/phds/working-with-vectores-matrices-and-arrays.html#understanding-arrays

Thank You!