Statistical Learning in Marketing

EBM214A05.2022-2023.1

FREE EDITION*

SUMMARY OF EVERYTHING FROM WEEK 1

LECTURES, CODE EXAMPLES, TIPS & STEP-BY-STEP ASSIGNMENT 1 GUIDE.

Note: This course has no readings.

Enhanced with a dynamic table of contents.

For the full version Google rug mads madlad

100%

OF THE PROFIT FROM THIS SUMMARY IS DONATED TO THE FOLLOWING NGOs:

SIAN | LGBT Asylum Support | alzheimer nederland | DUTCH CANCER SOCIETY | VOEDSELBANKEN.NL Groningen & regionaal distributiecentrum | Aids Fonds | SOAIDS | STOP AIDS NOW!

This summary helps you study & people in need.
Please do not share it for free, but ask your
friends to buy it and do a good deed.
MADS MADLAD

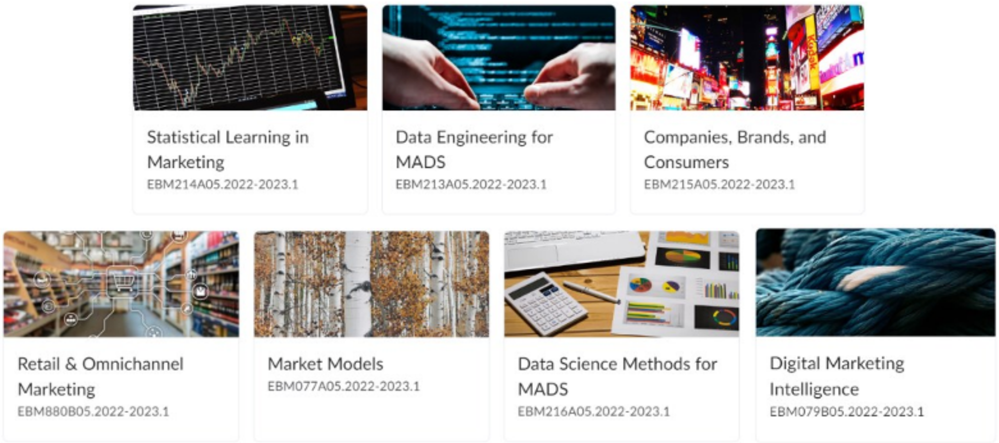
Note from MADS MADLAD:

Thank you for checking out my free summary. When I was writing these I sometimes struggled with this program, but there were no summaries available. This is why I decided to write something that is truly complete with a lot of effort put into it.

It helped me and my friends get good grades, but I always had you in mind, the future reader. When necessary, I always went the extra mile to make my summaries, more readable, organized and complete.

If you feel like it, leave me a review of how the course is going using this summary, it will make my day to hear your feedback.

Check out my other extensive summaries for other MADS courses:



Statistical Learning in Marketing
EBM214A05.2022-2023.1

Data Engineering for MADS
EBM213A05.2022-2023.1

Companies, Brands, and Consumers
EBM215A05.2022-2023.1

Retail & Omnichannel Marketing
EBM880B05.2022-2023.1

Market Models
EBM077A05.2022-2023.1

Data Science Methods for MADS
EBM216A05.2022-2023.1

Digital Marketing Intelligence
EBM079B05.2022-2023.1

Contact info:

If you need help or have an inquiry, contact me: https://www.georgedreemer.com

Connect with me on LinkedIn: https://www.linkedin.com/in/georgedreemer/

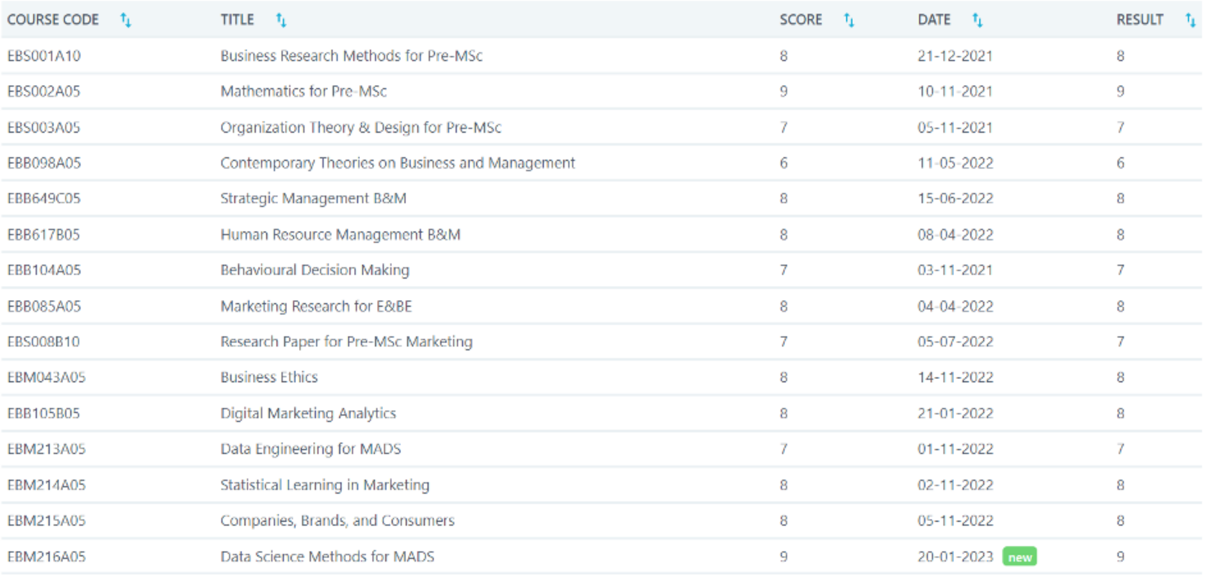
Donations:

By no means am I looking for fellow students to send me money! But if you feel like sending me some ETH or BTC, you can do so here:
--> ETH: 0x123e086c6808459e7fC6Ac7F64a77dBA1dDe0149
--> BTC: bc1qgwzc82vph5v8rmzef4ywechjf85772n7m2e22g

MADS MADLAD

wishes you good luck & perseverance.

Grades Testimony:



COURSE CODE ⇅	TITLE ⇅	SCORE ⇅	DATE ⇅	RESULT ⇅
EBS001A10	Business Research Methods for Pre-MSc	8	21-12-2021	8
EBS002A05	Mathematics for Pre-MSc	9	10-11-2021	9
EBS003A05	Organization Theory & Design for Pre-MSc	7	05-11-2021	7
EBB098A05	Contemporary Theories on Business and Management	6	11-05-2022	6
EBB649C05	Strategic Management B&M	8	15-06-2022	8
EBB617B05	Human Resource Management B&M	8	08-04-2022	8
EBB104A05	Behavioural Decision Making	7	03-11-2021	7
EBB085A05	Marketing Research for E&BE	8	04-04-2022	8
EBS008B10	Research Paper for Pre-MSc Marketing	7	05-07-2022	7
EBM043A05	Business Ethics	8	14-11-2022	8
EBB105B05	Digital Marketing Analytics	8	21-01-2022	8
EBM213A05	Data Engineering for MADS	7	01-11-2022	7
EBM214A05	Statistical Learning in Marketing	8	02-11-2022	8
EBM215A05	Companies, Brands, and Consumers	8	05-11-2022	8
EBM216A05	Data Science Methods for MADS	9	20-01-2023 new	9

Table of Contents

Week 1 (Lecture 1)

Lecture 1 – Intro to R + Intermediate R

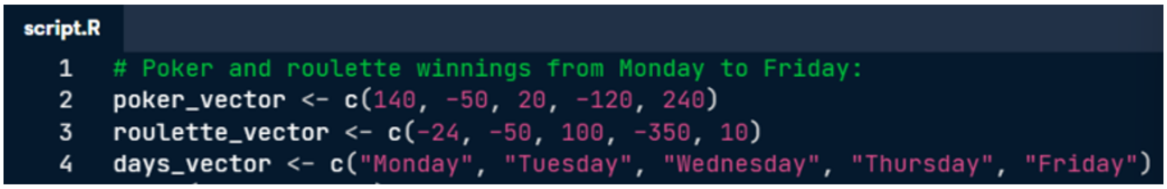
Introduction to R: Variables, Vectors, Matrices, Factors, Data Frames and Lists.

Variables: the most basic element, it is one single value.

- **Different data types:**
  - **Numeric:** values are numbers or decimals
  - **Integer:** values are number but NOT decimals
  - **Character:** text strings
  - **Factor:** special variable which is a string, but can be only a limited number of strings (e.g., Low, Medium, High). Usually used for categorical data.
  - **Logical:** only two values TRUE or FALSE

Note from MADS Madlad: *This is a great source to see these data types with examples and further explanations - https://statsandr.com/blog/data-types-in-r/*

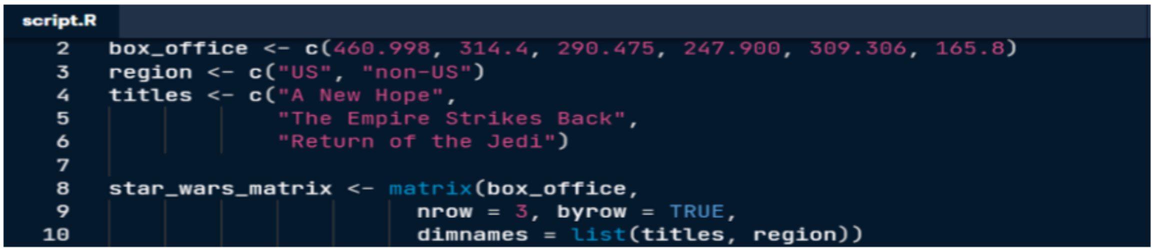
Vectors: one-dimensional array that can hold multiple data of the same data type (e.g., only strings or numeric).



script.R

```r
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
```

Matrices: two-dimensional collection with a fixed number of rows and columns of elements of the same data type.

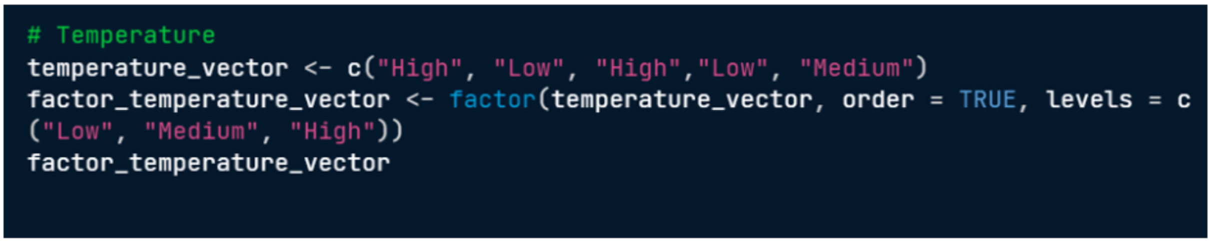


script.R

```r
box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306, 165.8)
region <- c("US", "non-US")
titles <- c("A New Hope",
            "The Empire Strikes Back",
            "Return of the Jedi")

star_wars_matrix <- matrix(box_office,
                           nrow = 3, byrow = TRUE,
                           dimnames = list(titles, region))
```

Factors: specific types of variables that is categorical in nature.

- Values are like labels (e.g. Rural, Urban…)
- Only a limited number of categories

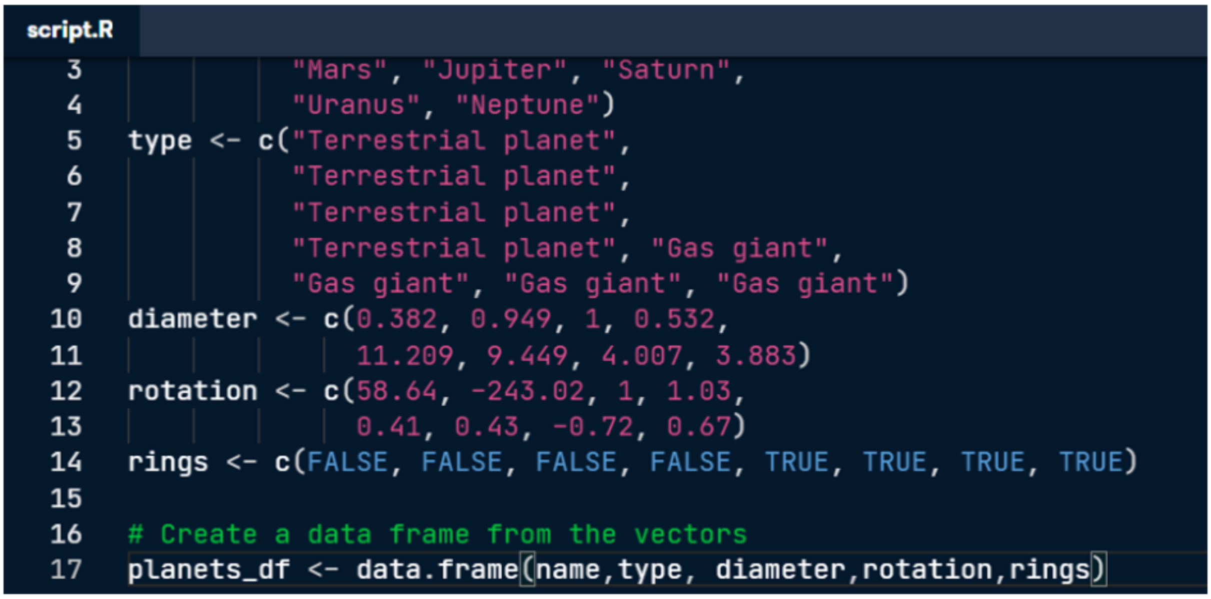


```r
# Temperature
temperature_vector <- c("High", "Low", "High","Low", "Medium")
factor_temperature_vector <- factor(temperature_vector, order = TRUE, levels = c
("Low", "Medium", "High"))
factor_temperature_vector
```

Data Frames (DF): multi-dimensional array with different data types.

When inspecting a df there are some handy basic functions:

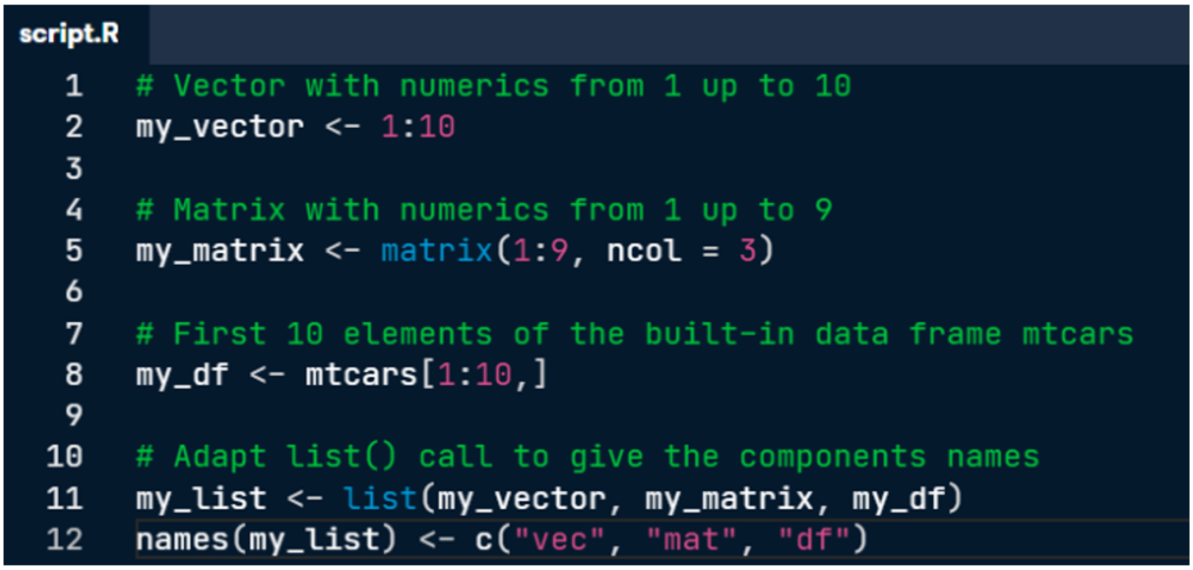
- head() – prints the header and first few rows of your data frame
- tail() – prints the header and last few rows of the data frame
- str() – tells you about the structure of you df, namely:
    - total number of observations
    - total number of variables
    - full list of the variable names
    - data type of each variable
    - the first observartion/row



```r
          "Mars", "Jupiter", "Saturn",
          "Uranus", "Neptune")
type <- c("Terrestrial planet",
          "Terrestrial planet",
          "Terrestrial planet",
          "Terrestrial planet", "Gas giant",
          "Gas giant", "Gas giant", "Gas giant")
diameter <- c(0.382, 0.949, 1, 0.532,
              11.209, 9.449, 4.007, 3.883)
rotation <- c(58.64, -243.02, 1, 1.03,
              0.41, 0.43, -0.72, 0.67)
rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)

# Create a data frame from the vectors
planets_df <- data.frame(name,type, diameter,rotation,rings)
```

***Lists*:** a list of items with different length, type, characteristics.

You can put: vectors, matrices and even data frames in a list!

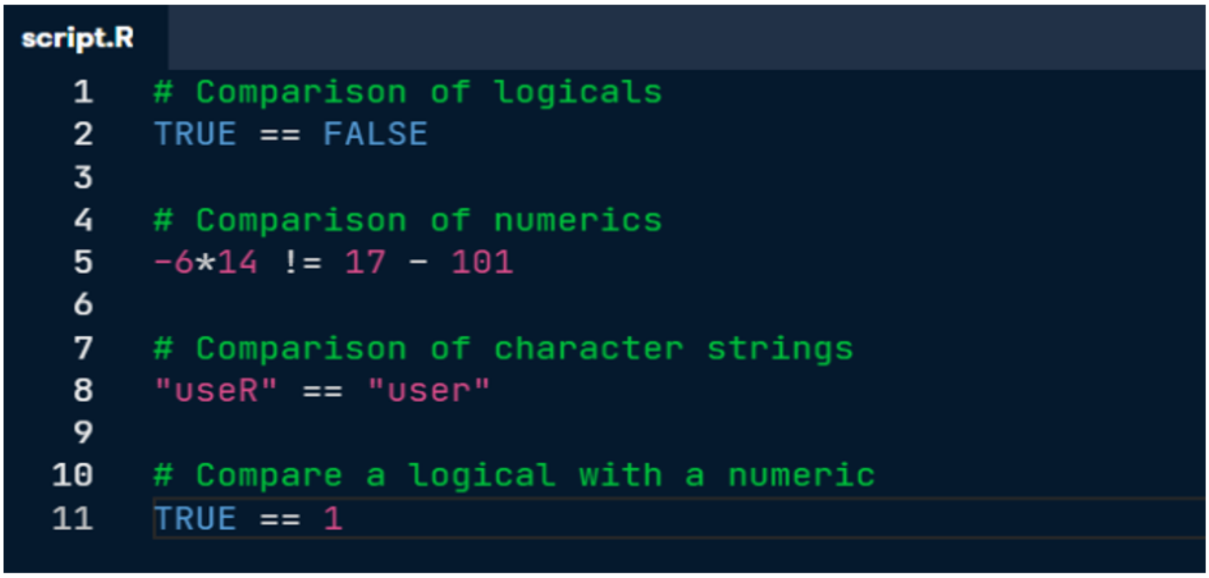


```r
# Vector with numerics from 1 up to 10
my_vector <- 1:10

# Matrix with numerics from 1 up to 9
my_matrix <- matrix(1:9, ncol = 3)

# First 10 elements of the built-in data frame mtcars
my_df <- mtcars[1:10,]

# Adapt list() call to give the components names
my_list <- list(my_vector, my_matrix, my_df)
names(my_list) <- c("vec", "mat", "df")
```

Intermediate R: Conditionals, Control flow, Loops, Functions, Apply family, Utilities.

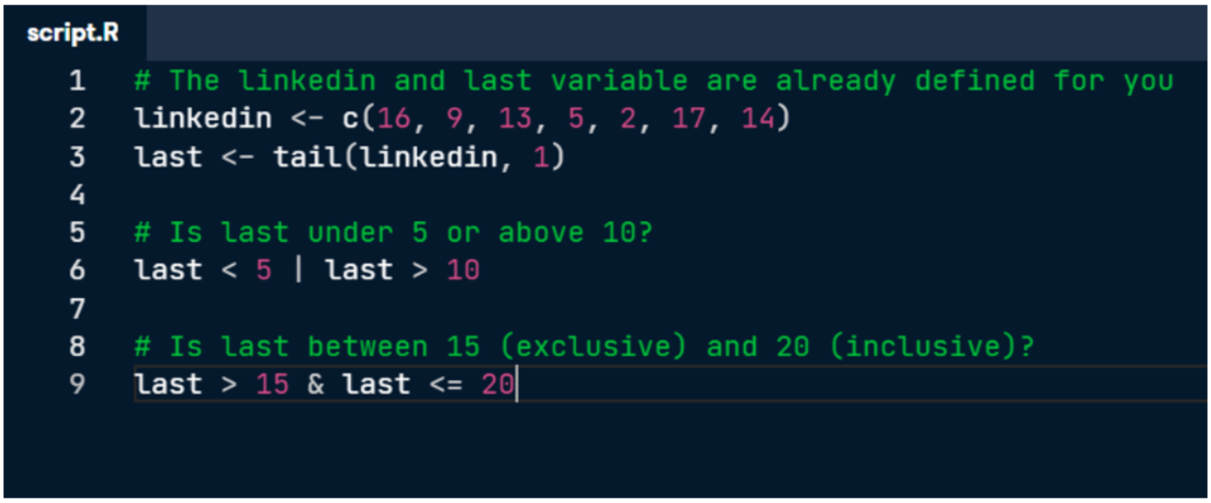
Conditionals and Control Flow

- *Relational operators*: comparing R objects
  - ○ Equality **==**
  - ○ Greater/Less Than **>** / **<**
  - ○ Not Equal **!=**



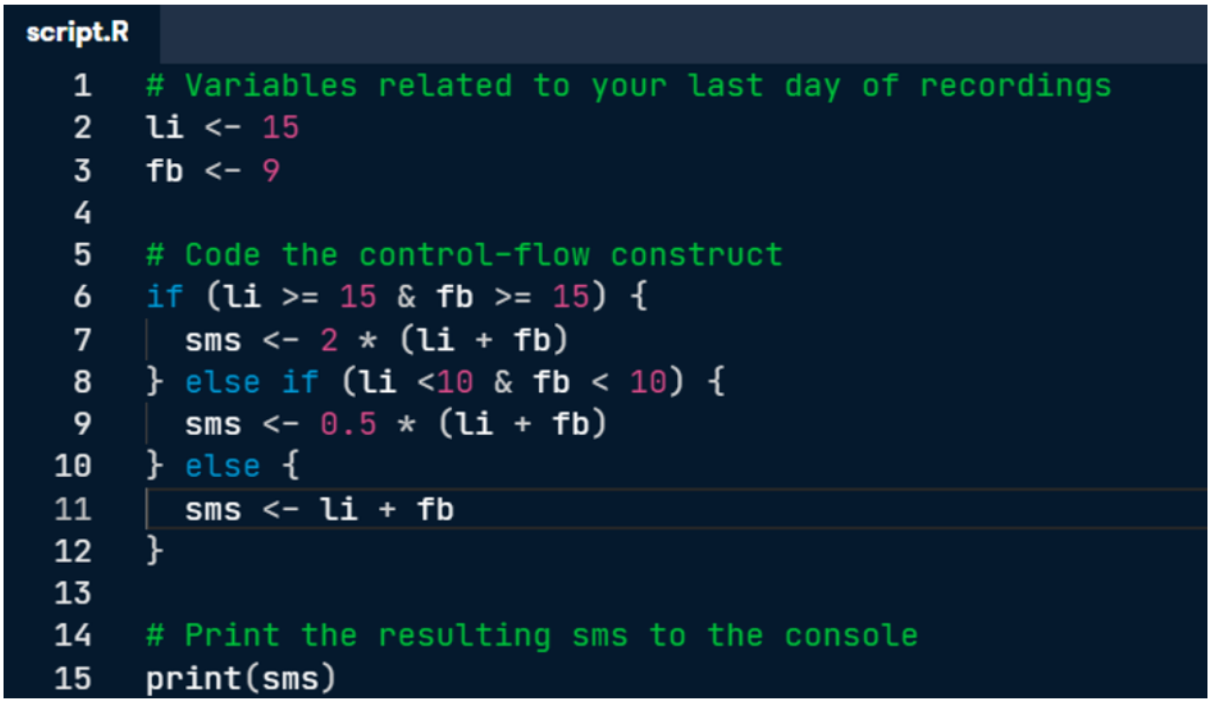
```r
# Comparison of logicals
TRUE == FALSE

# Comparison of numerics
-6*14 != 17 - 101

# Comparison of character strings
"useR" == "user"

# Compare a logical with a numeric
TRUE == 1
```

- *Logical operators*:
  - AND **&**
  - OR **|**



```r
# The linkedin and last variable are already defined for you
linkedin <- c(16, 9, 13, 5, 2, 17, 14)
last <- tail(linkedin, 1)

# Is last under 5 or above 10?
last < 5 | last > 10

# Is last between 15 (exclusive) and 20 (inclusive)?
last > 15 & last <= 20
```

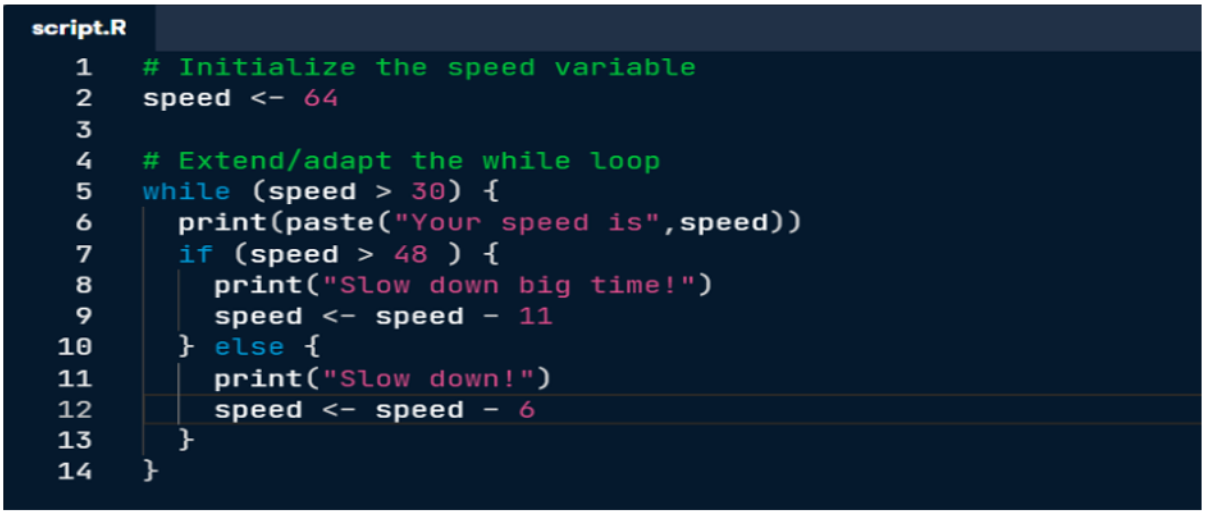
- *Conditional Statements*:
  - **if**
  - **else if**
  - **else**



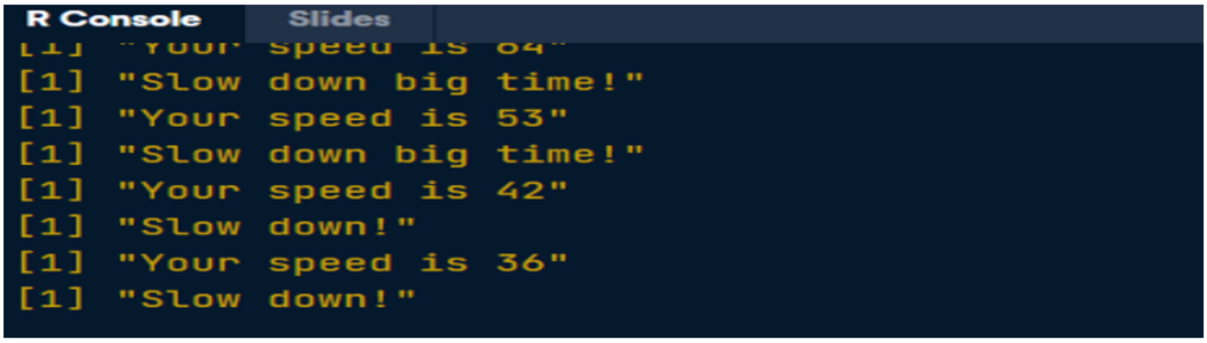
```r
# Variables related to your last day of recordings
li <- 15
fb <- 9

# Code the control-flow construct
if (li >= 15 & fb >= 15) {
  sms <- 2 * (li + fb)
} else if (li <10 & fb < 10) {
  sms <- 0.5 * (li + fb)
} else {
  sms <- li + fb
}

# Print the resulting sms to the console
print(sms)
```

Loops

- *While loops***:** look like repeated if statements

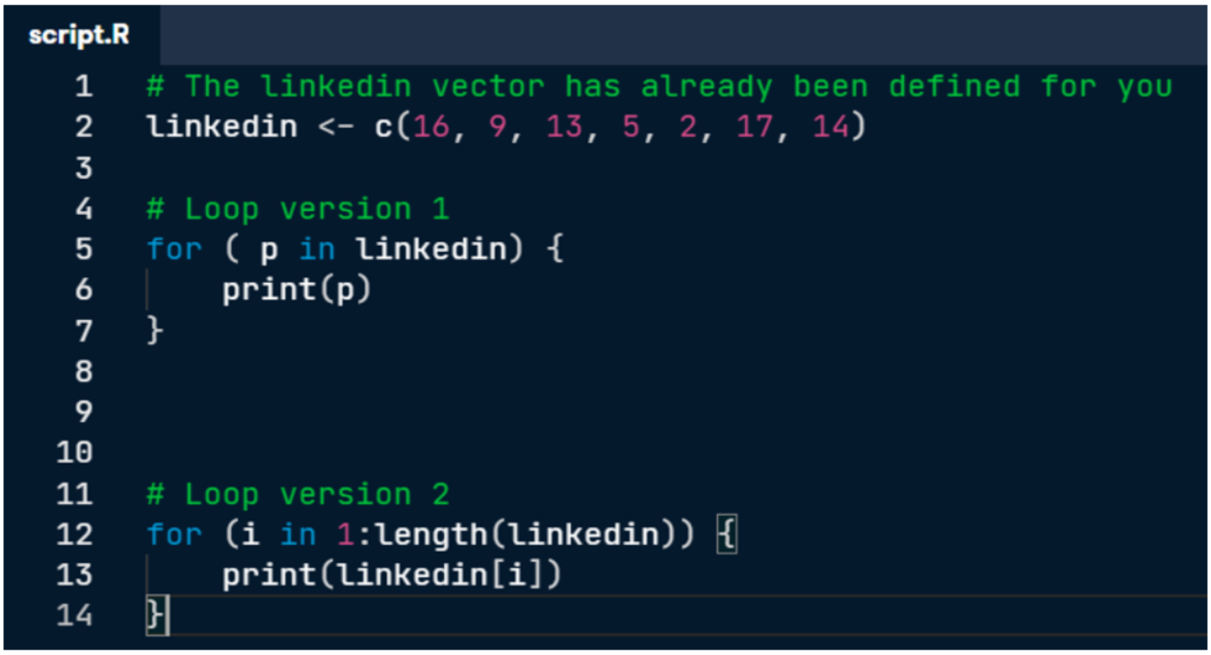


```r
# Initialize the speed variable
speed <- 64

# Extend/adapt the while loop
while (speed > 30) {
  print(paste("Your speed is",speed))
  if (speed > 48 ) {
    print("Slow down big time!")
    speed <- speed - 11
  } else {
    print("Slow down!")
    speed <- speed - 6
  }
}
```



```
[1] "Your speed is 64"
[1] "Slow down big time!"
[1] "Your speed is 53"
[1] "Slow down big time!"
[1] "Your speed is 42"
[1] "Slow down!"
[1] "Your speed is 36"
[1] "Slow down!"
```

- *For loops***:** iterate over all elements in a sequence.
  - **Two ways:** normal and indexed:

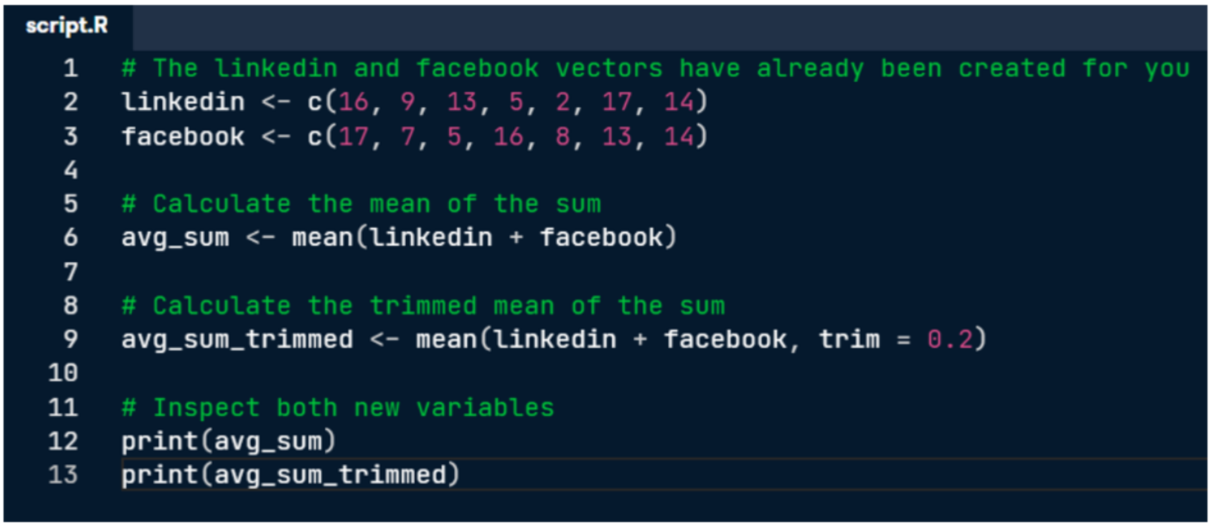


```r
# The linkedin vector has already been defined for you
linkedin <- c(16, 9, 13, 5, 2, 17, 14)

# Loop version 1
for ( p in linkedin) {
    print(p)
}



# Loop version 2
for (i in 1:length(linkedin)) {
    print(linkedin[i])
}
```

Functions: a defined operation that does something with the input you give it.

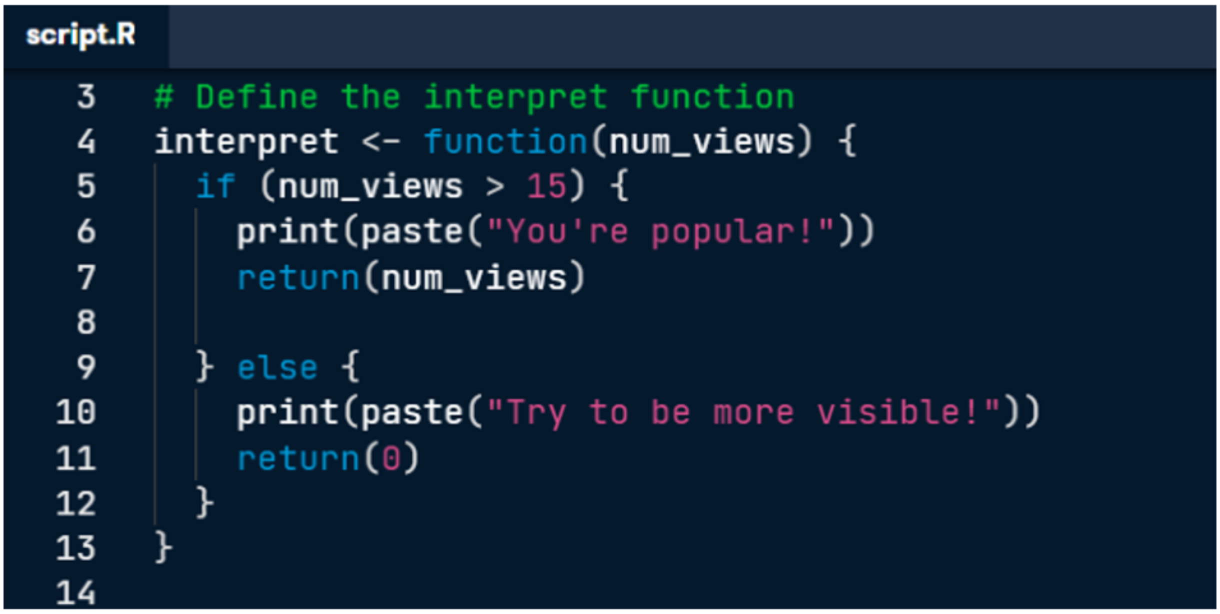
R offers both in-built functions and the ability to define our own functions.

- Pre-defined functions example: **mean()**

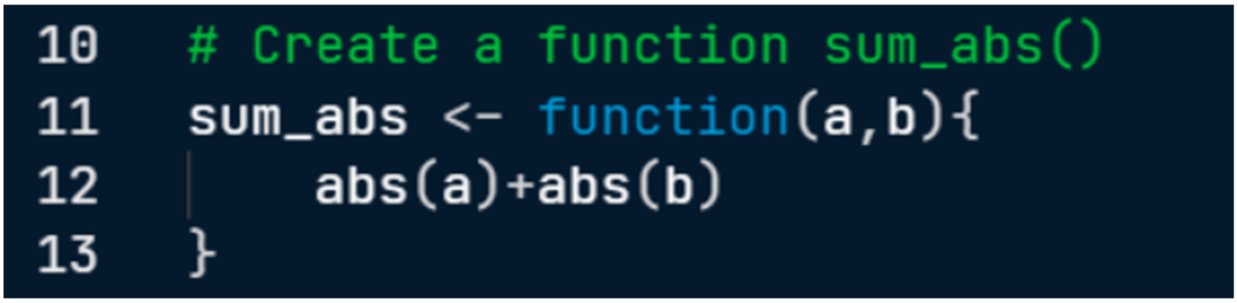


```r
# The linkedin and facebook vectors have already been created for you
linkedin <- c(16, 9, 13, 5, 2, 17, 14)
facebook <- c(17, 7, 5, 16, 8, 13, 14)

# Calculate the mean of the sum
avg_sum <- mean(linkedin + facebook)

# Calculate the trimmed mean of the sum
avg_sum_trimmed <- mean(linkedin + facebook, trim = 0.2)

# Inspect both new variables
print(avg_sum)
print(avg_sum_trimmed)
```

- Self-made function examples:



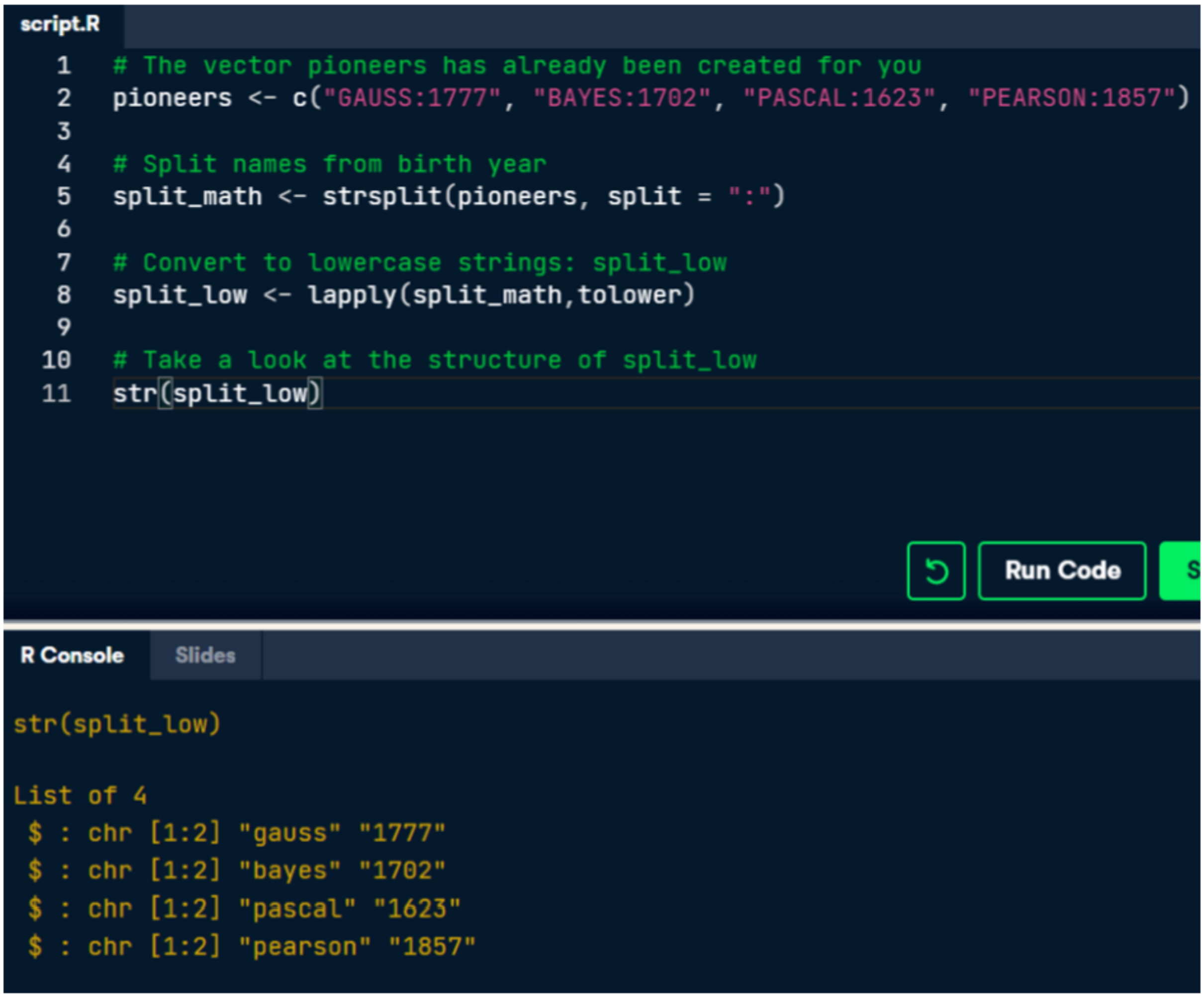
```r
# Define the interpret function
interpret <- function(num_views) {
  if (num_views > 15) {
    print(paste("You're popular!"))
    return(num_views)

  } else {
    print(paste("Try to be more visible!"))
    return(0)
  }
}

```



```r
# Create a function sum_abs()
sum_abs <- function(a,b){
     abs(a)+abs(b)
}
```

***Apply family*:**

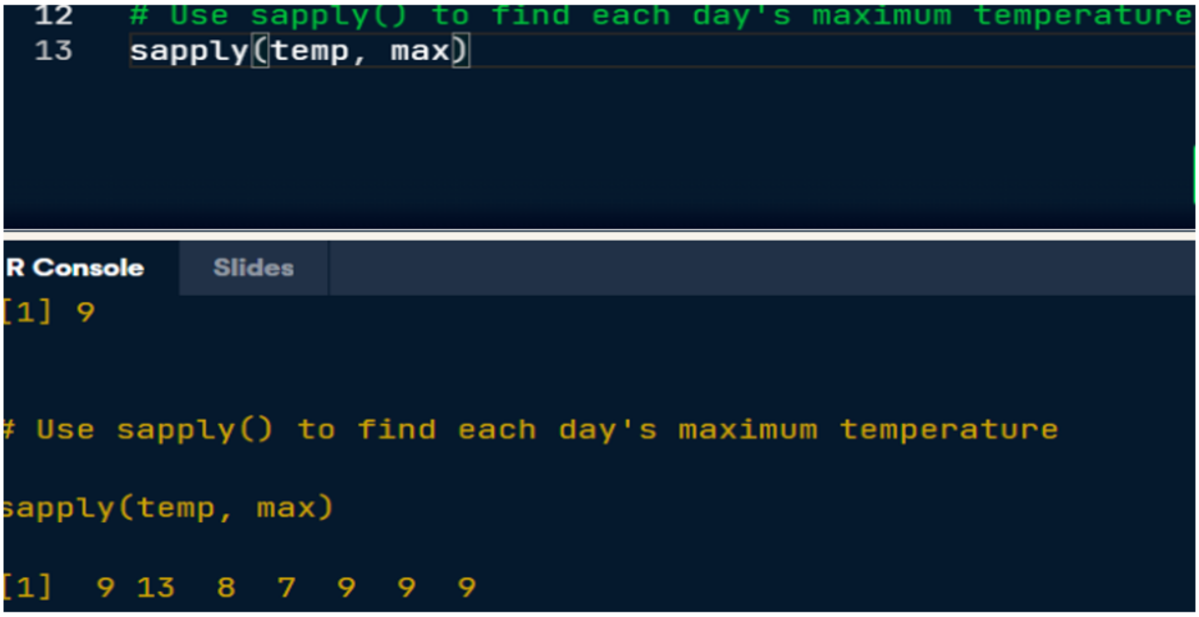
- *Lapply*: apply a function to a specified set of data, and return a list.



```r
# The vector pioneers has already been created for you
pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")

# Split names from birth year
split_math <- strsplit(pioneers, split = ":")

# Convert to lowercase strings: split_low
split_low <- lapply(split_math,tolower)

# Take a look at the structure of split_low
str(split_low)
```

```
str(split_low)

List of 4
 $ : chr [1:2] "gauss" "1777"
 $ : chr [1:2] "bayes" "1702"
 $ : chr [1:2] "pascal" "1623"
 $ : chr [1:2] "pearson" "1857"
```

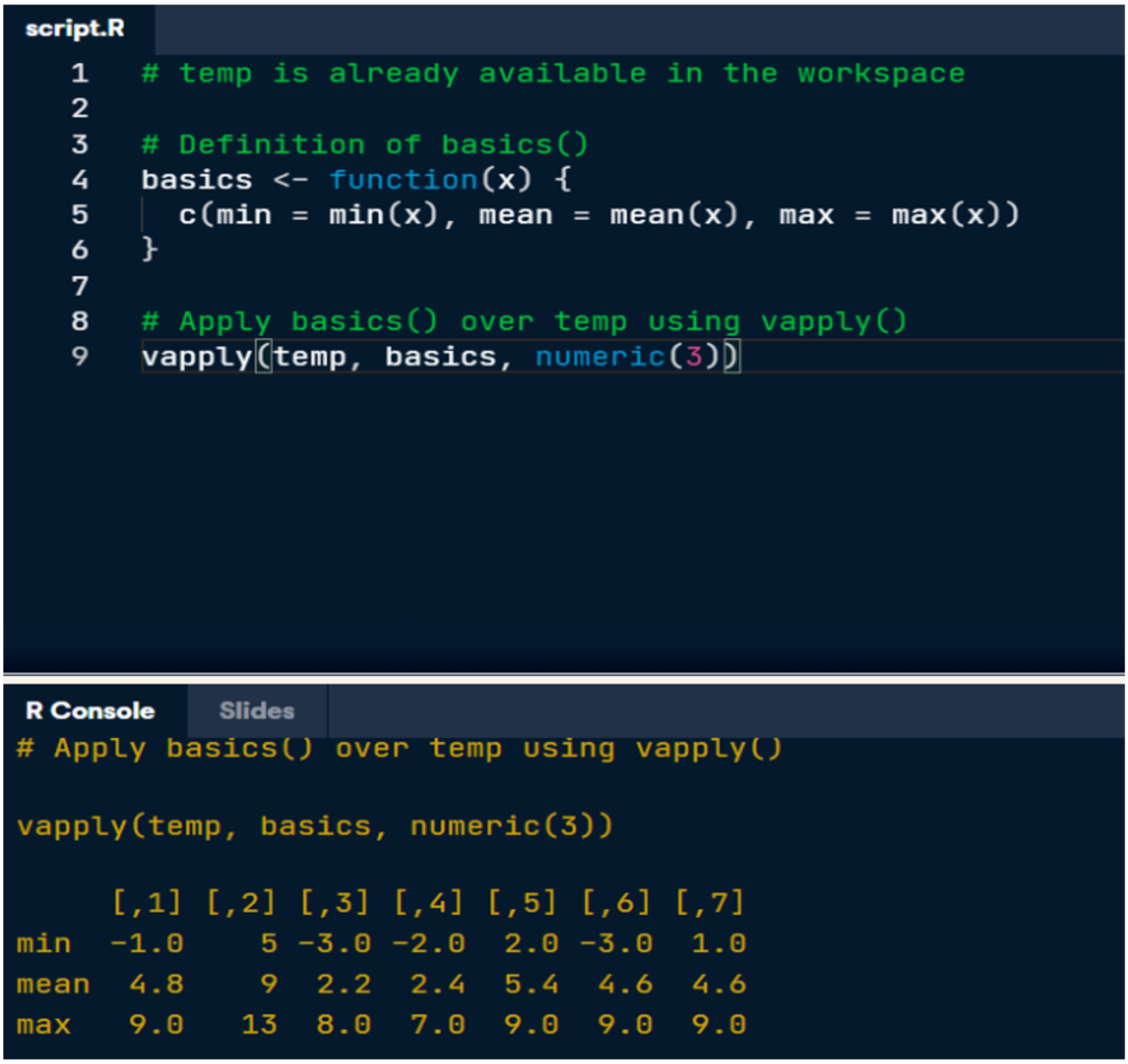
- *Sapply*: apply a function to a specified set of data, and return a vector.



```r
# Use sapply() to find each day's maximum temperature
sapply(temp, max)
```

```
[1] 9


# Use sapply() to find each day's maximum temperature

sapply(temp, max)

[1]  9 13  8  7  9  9  9
```

- *Vapply*: more robust version of Sapply, where you have more control over the ouput.



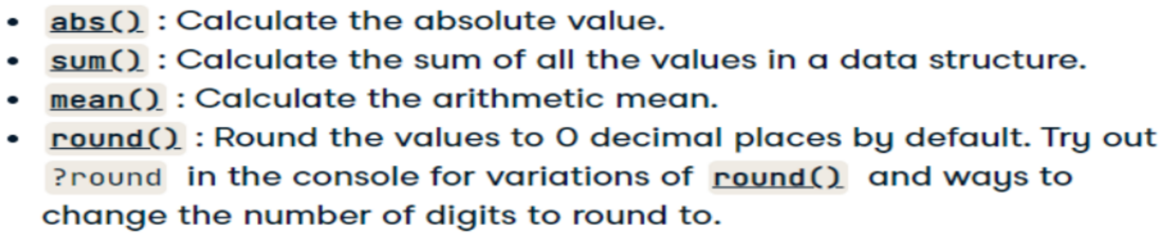
```r
# temp is already available in the workspace

# Definition of basics()
basics <- function(x) {
  c(min = min(x), mean = mean(x), max = max(x))
}

# Apply basics() over temp using vapply()
vapply(temp, basics, numeric(3))
```

```
# Apply basics() over temp using vapply()

vapply(temp, basics, numeric(3))

     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
min  -1.0    5 -3.0 -2.0  2.0 -3.0  1.0
mean  4.8    9  2.2  2.4  5.4  4.6  4.6
max   9.0   13  8.0  7.0  9.0  9.0  9.0
```

Utilities

R contains a large number of useful utilities and functions to do basic work on your data. You can also add external libraries with specialized utilities.

- **Mathematical utilities:**

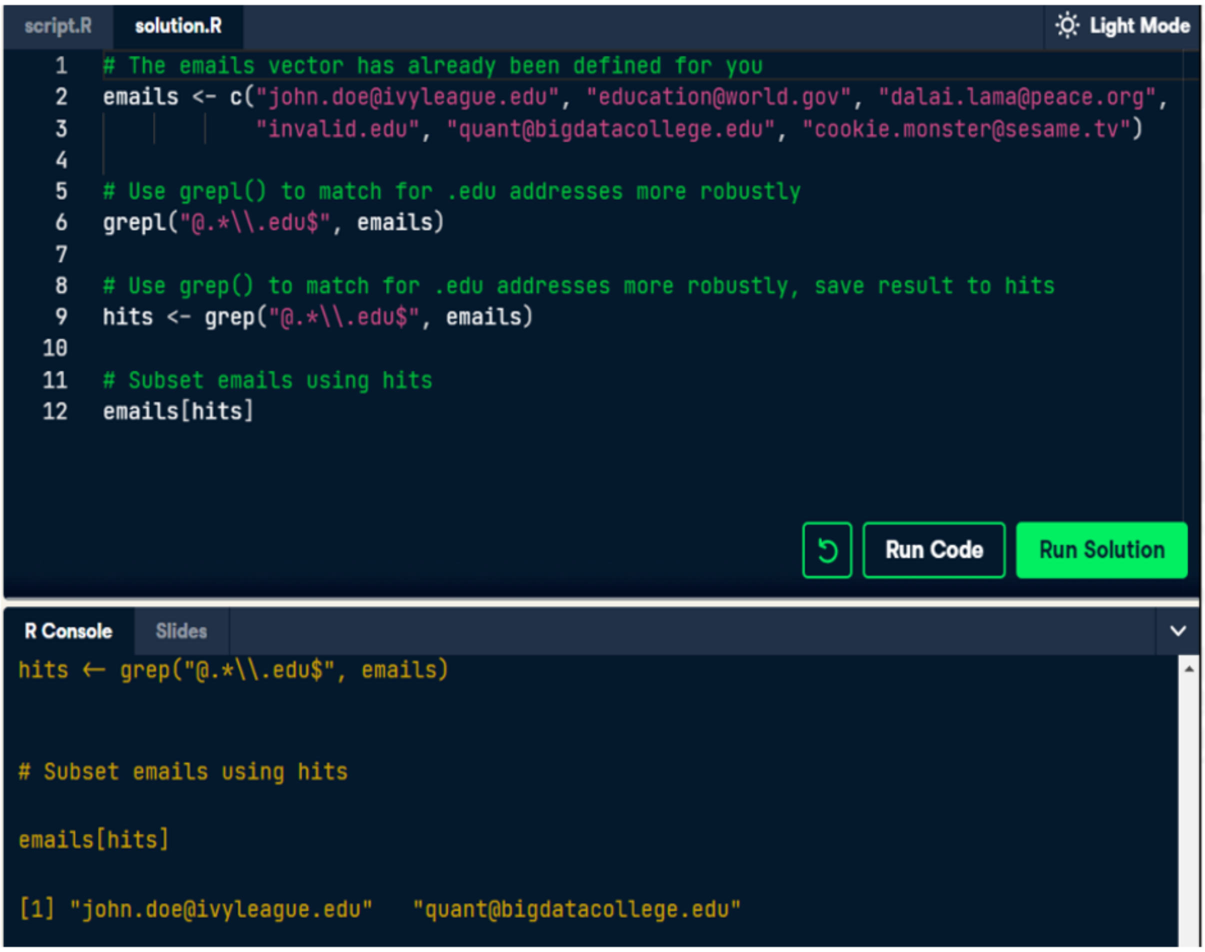


- `abs()` : Calculate the absolute value.
- `sum()` : Calculate the sum of all the values in a data structure.
- `mean()` : Calculate the arithmetic mean.
- `round()` : Round the values to 0 decimal places by default. Try out `?round` in the console for variations of `round()` and ways to change the number of digits to round to.

- *Data utilities*:

- `seq()` : Generate sequences, by specifying the `from` , `to` , and `by` arguments.
- `rep()` : Replicate elements of vectors and lists.
- `sort()` : Sort a vector in ascending order. Works on numerics, but also on character strings and logicals.
- `rev()` : Reverse the elements in a data structures for which reversal is defined.
- `str()` : Display the structure of any R object.
- `append()` : Merge vectors or lists.
- `is.*()` : Check for the class of an R object.
- `as.*()` : Convert an R object from one class to another.
- `unlist()` : Flatten (possibly embedded) lists to produce a vector.

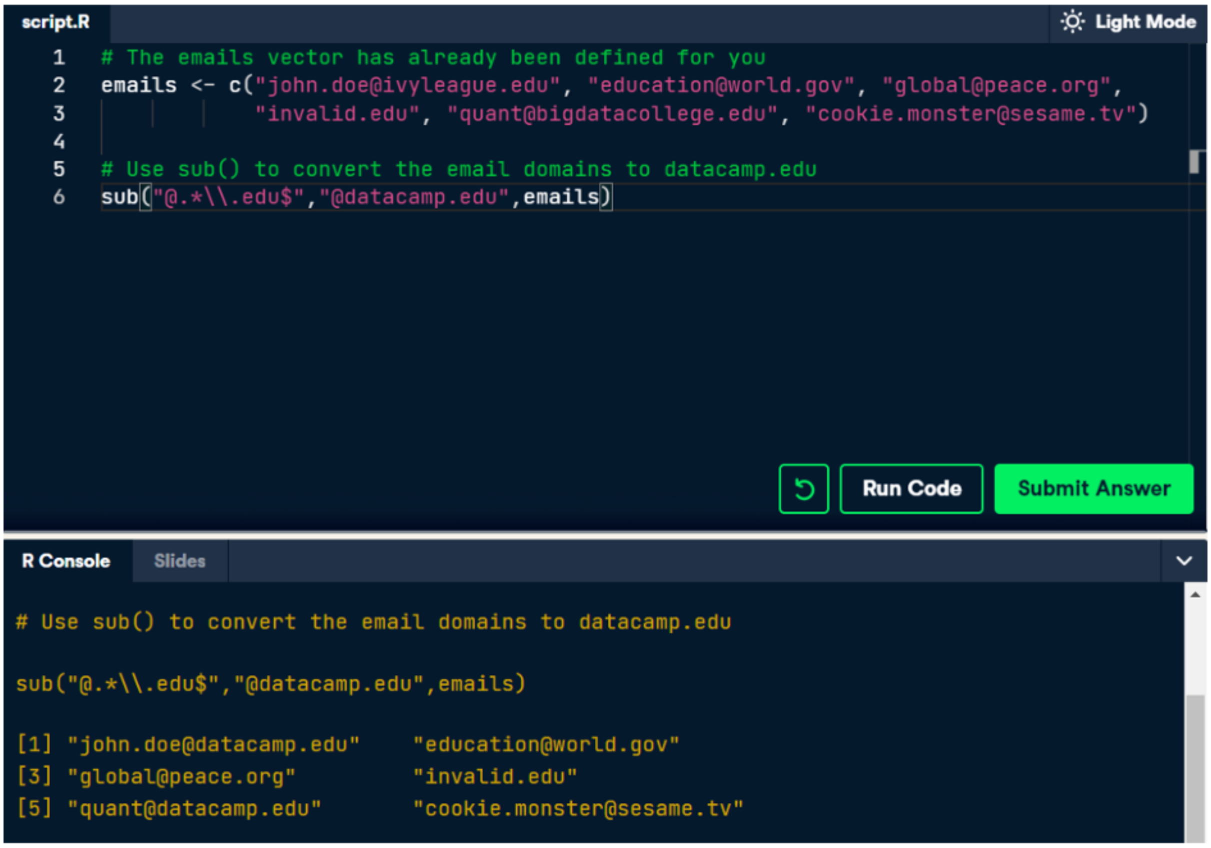
○ **Grepl and grep:** R's Regex - check whether a regular expression could be matched with a character vector.



```r
# The emails vector has already been defined for you
emails <- c("john.doe@ivyleague.edu", "education@world.gov", "dalai.lama@peace.org",
            "invalid.edu", "quant@bigdatacollege.edu", "cookie.monster@sesame.tv")

# Use grepl() to match for .edu addresses more robustly
grepl("@.*\\.edu$", emails)

# Use grep() to match for .edu addresses more robustly, save result to hits
hits <- grep("@.*\\.edu$", emails)

# Subset emails using hits
emails[hits]
```

```
hits <- grep("@.*\\.edu$", emails)


# Subset emails using hits

emails[hits]

[1] "john.doe@ivyleague.edu"   "quant@bigdatacollege.edu"
```

- **Sub and gsub:** takes it one step further from grep/grepl, by both matching and then replacing the match with what you assigned.



```r
# The emails vector has already been defined for you
emails <- c("john.doe@ivyleague.edu", "education@world.gov", "global@peace.org",
            "invalid.edu", "quant@bigdatacollege.edu", "cookie.monster@sesame.tv")

# Use sub() to convert the email domains to datacamp.edu
sub("@.*\\.edu$","@datacamp.edu",emails)
```

```
# Use sub() to convert the email domains to datacamp.edu

sub("@.*\\.edu$","@datacamp.edu",emails)

[1] "john.doe@datacamp.edu"    "education@world.gov"
[3] "global@peace.org"         "invalid.edu"
[5] "quant@datacamp.edu"       "cookie.monster@sesame.tv"
```

- *Importing Data in R* /w Readr package
    - **Read_csv**
    - **Read_tsv**
    - **Read_delim**

Note from MADS Madlad: *When you have done the Datacamp courses Intro to R and Intermediate R yourself, you will get a better feel for all of this than just by reading it. It's also way more fun!*

Good luck and congrats on finishing your first week of the MADS program.