

Graduate Student Lunch & Learn

Rafael Della Coletta
02.17.2022



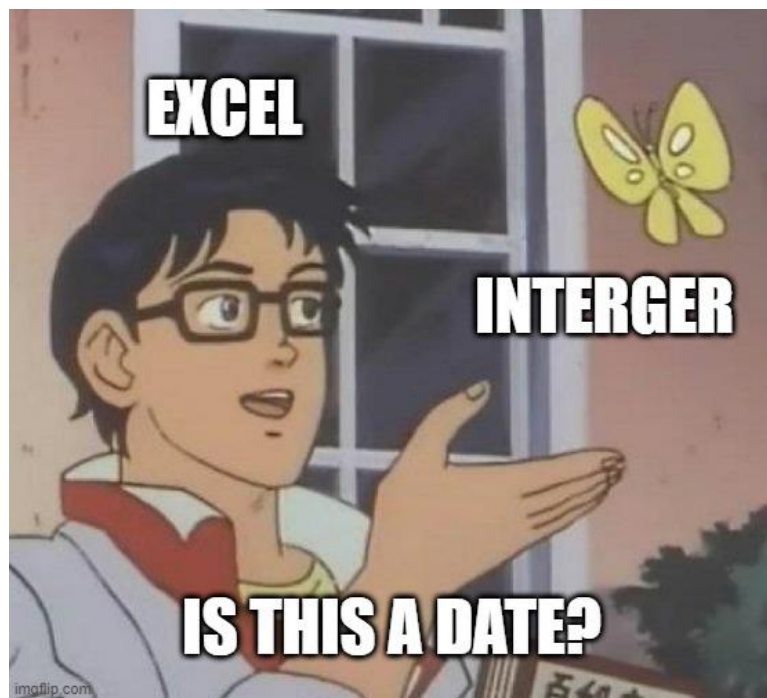
What I wish I knew when started grad school...

(with memes)

1 Excel sucks

If statistics programs/languages were cars...



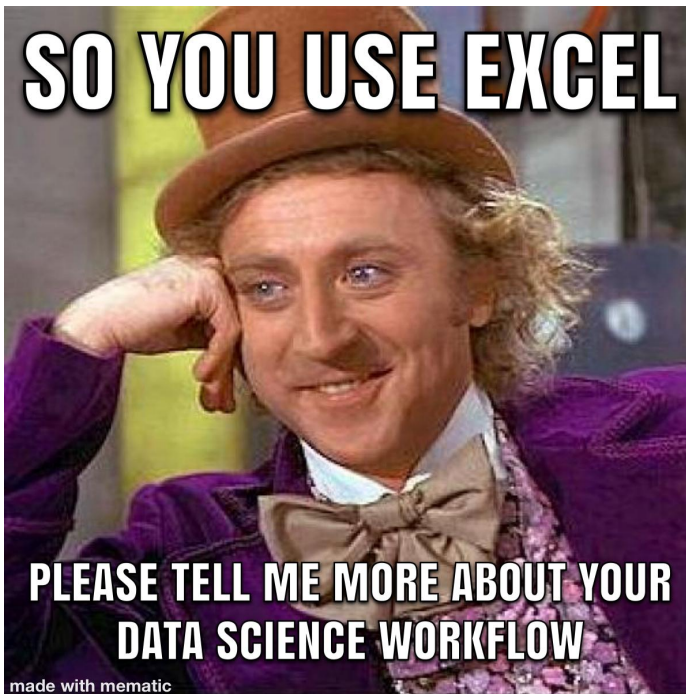


EXCEL

INTERGER

IS THIS A DATE?

imgflip.com



SO YOU USE EXCEL

**PLEASE TELL ME MORE ABOUT YOUR
DATA SCIENCE WORKFLOW**

made with mematic

2 Learning to code is hard, but worth it



Matt Dancho (Business Scie... · 9/27/21 ...

Even seasoned useRs get smacked sometimes. Never give up. [#rstats](#)



3 Programming is 90% googling



Bonnie @The_GreatBonnie · 1/23/22



Are you a developer or you can't relate.



4 Best code is working code



Dorsa Amir
@DorsaAmir



When your code is a mess but it somehow still works.



R basics: syntax, data structures, loops, etc.

(no memes)



≠



File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins R 4.1.2

basics.R

```
1 # using your first function
2 hello <- "yet another hello world"
3 print(hello)
4
5 # modifying variables
6 genes_A <- 100
7 genes_B <- 50
8 total_genes <- genes_A + genes_B
9 print(total_genes)
10
```

Environment History Connections Tutorial

Import Dataset 162 MiB

R Global Environment

Values

genes_A	100
genes_B	50
hello	"yet another hello world"
total_genes	150

Files Plots Packages Help Viewer

New Folder Upload Delete Rename More

Cloud > project > scripts

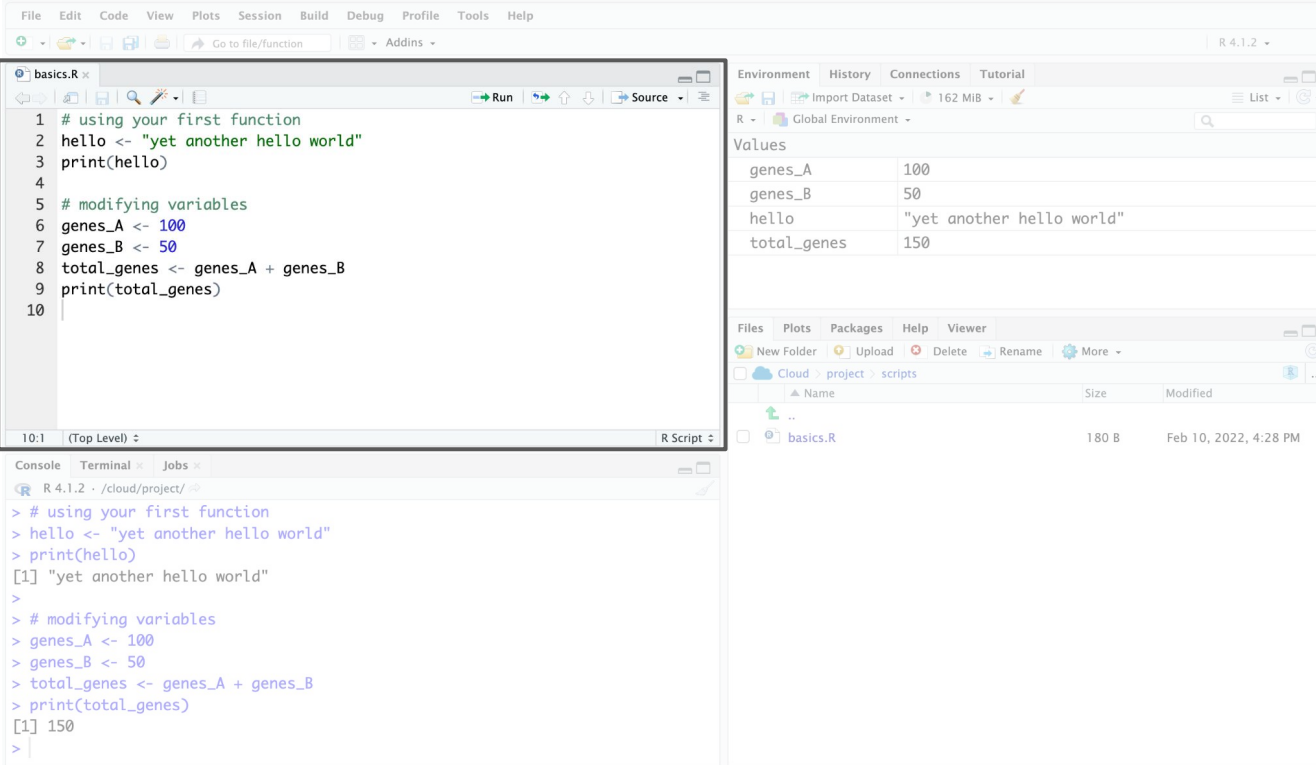
Name	Size	Modified
..		
basics.R	180 B	Feb 10, 2022, 4:28 PM

Console Terminal Jobs

R 4.1.2 · /cloud/project/

```
> # using your first function
> hello <- "yet another hello world"
> print(hello)
[1] "yet another hello world"
>
> # modifying variables
> genes_A <- 100
> genes_B <- 50
> total_genes <- genes_A + genes_B
> print(total_genes)
[1] 150
>
```


Write your code here



The screenshot displays the RStudio interface with the following components:

- Script Editor:** Contains R code for a function and variable manipulation. An arrow points from the text "Write your code here" to this pane.
- Console:** Shows the execution output of the code in the script editor.
- Environment Pane:** Displays the current values of variables in the workspace.

Script Editor Code:

```
1 # using your first function
2 hello <- "yet another hello world"
3 print(hello)
4
5 # modifying variables
6 genes_A <- 100
7 genes_B <- 50
8 total_genes <- genes_A + genes_B
9 print(total_genes)
10
```

Console Output:

```
> # using your first function
> hello <- "yet another hello world"
> print(hello)
[1] "yet another hello world"
>
> # modifying variables
> genes_A <- 100
> genes_B <- 50
> total_genes <- genes_A + genes_B
> print(total_genes)
[1] 150
>
```

Environment Pane Values:

Variable	Value
genes_A	100
genes_B	50
hello	"yet another hello world"
total_genes	150

Files Pane: Shows a file named `basics.R` with a size of 180 B, last modified on Feb 10, 2022, at 4:28 PM.

The screenshot displays the RStudio interface with three main panels:

- Source Editor:** Contains an R script named `basics.R` with the following code:

```
1 # using your first function
2 hello <- "yet another hello world"
3 print(hello)
4
5 # modifying variables
6 genes_A <- 100
7 genes_B <- 50
8 total_genes <- genes_A + genes_B
9 print(total_genes)
10
```
- Environment Panel:** Shows the current environment with the following values:

Variable	Value
<code>genes_A</code>	<code>100</code>
<code>genes_B</code>	<code>50</code>
<code>hello</code>	<code>"yet another hello world"</code>
<code>total_genes</code>	<code>150</code>
- Console Panel:** Shows the execution output of the script:

```
> # using your first function
> hello <- "yet another hello world"
> print(hello)
[1] "yet another hello world"
>
> # modifying variables
> genes_A <- 100
> genes_B <- 50
> total_genes <- genes_A + genes_B
> print(total_genes)
[1] 150
>
```

An arrow points from the text "R does its things here" to the console panel.

R does its things here

Keep track of all your objects here

The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for a function and variable manipulation.
- Environment Pane:** Shows the current environment with the following objects:

Object	Value
genes_A	100
genes_B	50
hello	"yet another hello world"
total_genes	150
- Console:** Shows the execution output of the code in the source editor.
- Files Pane:** Displays the project structure, including a file named 'basics.R'.

Source Editor Code:

```
1 # using your first function
2 hello <- "yet another hello world"
3 print(hello)
4
5 # modifying variables
6 genes_A <- 100
7 genes_B <- 50
8 total_genes <- genes_A + genes_B
9 print(total_genes)
10
```

Console Output:

```
> # using your first function
> hello <- "yet another hello world"
> print(hello)
[1] "yet another hello world"
>
> # modifying variables
> genes_A <- 100
> genes_B <- 50
> total_genes <- genes_A + genes_B
> print(total_genes)
[1] 150
>
```

The image shows the RStudio IDE interface. The main editor window displays a script named 'basics.R' with the following code:

```
1 # using your first function
2 hello <- "yet another hello world"
3 print(hello)
4
5 # modifying variables
6 genes_A <- 100
7 genes_B <- 50
8 total_genes <- genes_A + genes_B
9 print(total_genes)
10
```

The Environment pane on the right shows the current state of the workspace:

Values	
genes_A	100
genes_B	50
hello	"yet another hello world"
total_genes	150

The Files pane at the bottom right shows the project structure:

Name	Size	Modified
..		
basics.R	180 B	Feb 10, 2022, 4:28 PM

The Console pane at the bottom left shows the output of the script:

```
> # using your first function
> hello <- "yet another hello world"
> print(hello)
[1] "yet another hello world"
>
> # modifying variables
> genes_A <- 100
> genes_B <- 50
> total_genes <- genes_A + genes_B
> print(total_genes)
[1] 150
>
```

An arrow points from the text 'See where your files are' to the Files pane.

See where your files are

Basic syntax

```
# using your first function  
hello <- "yet another hello world"  
print(hello)
```

Basic syntax

Comment (aka your best friend)



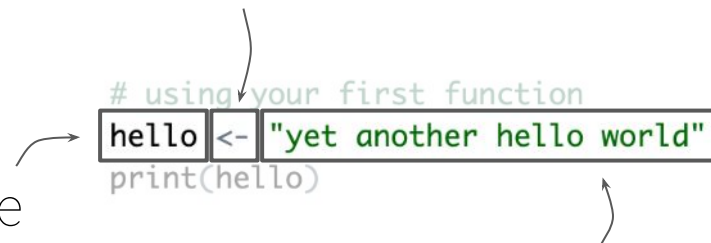
```
# using your first function  
hello <- "yet another hello world"  
print(hello)
```

Basic syntax

Assignment operator

Object / Variable

```
# using your first function  
hello <- "yet another hello world"  
print(hello)
```

A diagram illustrating the basic syntax of an assignment in R. The code snippet is: `# using your first function` (comment), `hello <- "yet another hello world"` (assignment), and `print(hello)` (print statement). The word `hello` is enclosed in a box and labeled "Object / Variable" with an arrow. The `<-` operator is labeled "Assignment operator" with an arrow. The string `"yet another hello world"` is also enclosed in a box and labeled "Some data:" with an arrow.

Some data:

- **Character / string** → "hello world!"
- **Integer** → 30
- **Numeric** → 2.5
- **Logical** → TRUE or FALSE

Basic syntax

```
# using your first function  
hello <- "yet another hello world"  
print(hello)
```

Function



Style guide

<https://style.tidyverse.org/>

Best practices on how to name variables and write clean code!

Variable and function names should use only lowercase letters, numbers, and `_`. Use underscores (`_`) (so called snake case) to separate words within a name.

```
# Good
day_one
day_1
```

```
# Bad
DayOne
dayone
```

Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful (this is not easy!).

```
# Good
day_one
```

```
# Bad
first_day_of_the_month
djm1
```

Operators

Arithmetic

+ addition

- subtraction

***** multiplication

/ division

^ or ****** exponentiation

x %% y integer-divide x by y and
return the remainder

x %/ y integer division

Operators

Logical

<

less than

<=

less than or equal to

>

greater than

>=

greater than or equal to

==

exactly equal to

!=

not equal to

!x

not x

x | y

x or y

x & y

x and y

Operations

Operations

Be careful! Certain operations are **specific** to certain data types



Operations

`class(10) → numeric`

`class(10) → character`

`as.numeric(10) → 10`

Operations

$$\boxed{10} == \boxed{10} \rightarrow \boxed{\text{TRUE}}$$

$$\boxed{B} != \boxed{B} \rightarrow \boxed{\text{FALSE}}$$

$$\boxed{\text{TRUE}} + \boxed{\text{TRUE}} \rightarrow \boxed{2}$$

$$\boxed{\text{FALSE}} + \boxed{\text{FALSE}} \rightarrow \boxed{0}$$

Functions

Functions

`print(my_variable)`



num

`?round`



Rounds the values in its first argument to the specified number of decimal places (default 0)

`round(my_variable)`



num

Functions

```
my_data <- read.table(file = "filename.txt", sep = "\t")
```

Function name



Arguments

*type **?read.table** for more arguments and their default values

Data structures

Data structures

```
my_vector <- c(1, 55, 2.5, 333)
```



```
my_vector <- c("C", "O", "R", "N")
```



Data structures

```
my_matrix <- matrix(
```

1	55	2.5	333
0.2	0	11	2
0	9.9	2.5	758
1	5	2	90


)


Data structures

```
my_dataframe <- data.frame(
```

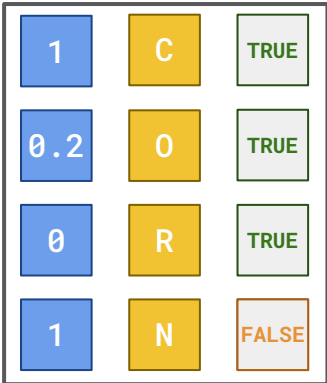
1	C	TRUE	333
0.2	0	TRUE	2
0	R	TRUE	758
1	N	FALSE	90

Data structures

numbers = 

best_plant = 

my_list <- **list**(

data = )

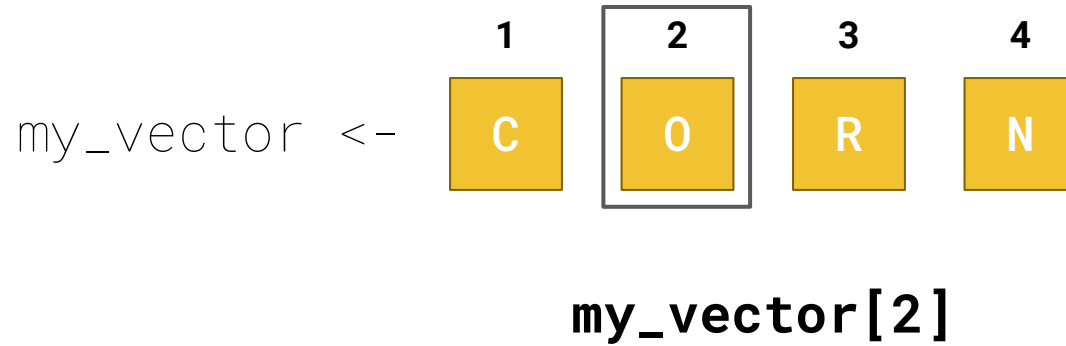
Subsetting

Subsetting

my_vector <-

1	2	3	4
C	O	R	N

Subsetting



Subsetting

`my_vector <-`

1	2	3	4
C	O	R	N

`my_vector[1:3]`

Subsetting

`my_vector <-`

1	2	3	4
C	O	R	N

`my_vector[c(1,4)]`

Subsetting

`my_df <-`

	1	2	3	4
1	1	C	TRUE	333
2	0.2	0	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

Subsetting

`my_df <-`

	1	2	3	4
1	1	C	TRUE	333
2	0.2	0	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

`my_df[,]`

Subsetting

`my_df <-`

	1	2	3	4
1	1	C	TRUE	333
2	0.2	0	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

`my_df[2,3]`

Subsetting

`my_df <-`

	1	2	3	4
1	1	C	TRUE	333
2	0.2	0	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

`my_df[4,]`

Subsetting

`my_df <-`

	1	2	3	4
1	1	C	TRUE	333
2	0.2	0	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

`my_df[, c(1, 3)]`

Subsetting

my_df <-

	1	2	3	4
1	1	C	TRUE	333
2	0.2	0	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

```
my_df[my_df[, 1] < 0.5, c(1,3)]
```

*also check **subset()** and **grep()** functions

Subsetting

```
df_with_header <-
```

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	O	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

Subsetting

```
df_with_header <-
```

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	O	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

df_with_header\$plant

Subsetting

```
df_with_header <-
```

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	O	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

```
df_with_header[,2]
```

Subsetting

```
df_with_header <-
```

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	0	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

```
df_with_header[3:4, c("maf", "n")]
```

if / else statements

```
if (condition) {  
    ...do something...  
} else {  
    ...do other thing...  
}
```

if / else statements

```
big_number <- 1.000
```

```
if (big_number > 100) {  
  print("this is a big number")  
} else {  
  print("not that big")  
}
```


if / else statements

```
big_number <- 1.000
```

```
if (big_number > 100) {  
  print("this is a big number")  
} else {  
  print("not that big")  
}
```



“not that big”

if / else statements

```
plant <- "corn"
```

```
if (plant == "corn") {  
  plant <- paste(plant, "is the best")  
} else {  
  plant <- "not corn"  
}
```

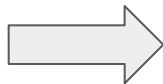
```
print(plant)
```

if / else statements

```
plant <- "corn"
```

```
if (plant == "corn") {  
  plant <- paste(plant, "is the best")  
} else {  
  plant <- "not corn"  
}
```

```
print(plant)
```



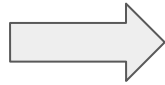
"corn is the best"

for loop

```
for (variable in vector) {  
    ...do something...  
}
```

I want to create a vector containing **only vowels** from the plant column

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	O	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90



df\$plant

C O R N



new_vector

O

Need to tell R some important things first...

```
# vowels I know  
vowels <- c("A", "E", "I", "O", "U")  
# create an empty vector to store values  
vowels_in_plant <- c()
```

Need to tell R some important things first...

```
# vowels I know  
vowels <- c("A", "E", "I", "O", "U")  
# create an empty vector to store values  
vowels_in_plant <- c()
```

... and then do what you gotta do

```
# get letter from first row of column "plant"  
letter <- df[1, "plant"]
```

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	O	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90

Need to tell R some important things first...

```
# vowels I know
vowels <- c("A", "E", "I", "O", "U")
# create an empty vector to store values
vowels_in_plant <- c()
```

... and then do what you gotta do

```
# get letter from first row of column "plant"
letter <- df[1, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}
```

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	O	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90



Need to tell R some important things first...

```
# vowels I know
vowels <- c("A", "E", "I", "O", "U")
# create an empty vector to store values
vowels_in_plant <- c()
```

... and then do what you gotta do

```
# get letter from first row of column "plant"
letter <- df[2, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}
```

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	O	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90



Need to tell R some important things first...

```
# vowels I know
vowels <- c("A", "E", "I", "O", "U")
# create an empty vector to store values
vowels_in_plant <- c()
```

... and then do what you gotta do

```
# get letter from first row of column "plant"
letter <- df[3, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}
```

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	O	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90



Need to tell R some important things first...

```
# vowels I know
vowels <- c("A", "E", "I", "O", "U")
# create an empty vector to store values
vowels_in_plant <- c()
```

... and then do what you gotta do

```
# get letter from first row of column "plant"
letter <- df[4, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}
```

	maf	plant	treat	n
1	1	C	TRUE	333
2	0.2	O	TRUE	2
3	0	R	TRUE	758
4	1	N	FALSE	90



```
# vowels I know
vowels <- c("A", "E", "I", "O", "U")
# create an empty vector to store values
vowels_in_plant <- c()

# get letter from first row of column "plant"
letter <- df[1, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}

# get letter from first row of column "plant"
letter <- df[2, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}

# get letter from first row of column "plant"
letter <- df[3, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}

# get letter from first row of column "plant"
letter <- df[4, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}
```

```
# vowels I know
vowels <- c("A", "E", "I", "O", "U")
# create an empty vector to store values
vowels_in_plant <- c()
```

```
# get letter from first row of column "plant"
letter <- df[1, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}
```

```
# get letter from first row of column "plant"
letter <- df[2, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}
```

```
# get letter from first row of column "plant"
letter <- df[3, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}
```

```
# get letter from first row of column "plant"
letter <- df[4, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}
```



```
# vowels I know
vowels <- c("A", "E", "I", "O", "U")
# create an empty vector to store values
vowels_in_plant <- c()
```

```
for (row in 1:4) {
  # get letter from column "plant"
  letter <- df[row, "plant"]
  # if letter is a vowel
  if (letter %in% vowels) {
    # append letter to vector
    vowels_in_plant <- append(vowels_in_plant, letter)
  }
}
```

```

# vowels I know
vowels <- c("A", "E", "I", "O", "U")
# create an empty vector to store values
vowels_in_plant <- c()

# get letter from first row of column "plant"
letter <- df[1, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}

# get letter from first row of column "plant"
letter <- df[2, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}

# get letter from first row of column "plant"
letter <- df[3, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}

# get letter from first row of column "plant"
letter <- df[4, "plant"]
# if letter is a vowel
if (letter %in% vowels) {
  # append letter to vector
  vowels_in_plant <- append(vowels_in_plant, letter)
}

```

```

# vowels I know
vowels <- c("A", "E", "I", "O", "U")
# create an empty vector to store values
vowels_in_plant <- c()

for (row in 1:4) {
  # get letter from column "plant"
  letter <- df[row, "plant"]
  # if letter is a vowel
  if (letter %in% vowels) {
    # append letter to vector
    vowels_in_plant <- append(vowels_in_plant, letter)
  }
}

```

while loop

```
while (condition) {  
    ...do something...  
}
```

```
# initial number of seeds
total_seeds <- 0
# initial number of surprise packages opened
surprise_packages <- 0

while(total_seeds < 1000) {

  # open surprise packages and add to total number of seeds
  total_seeds <- total_seeds + sample(x = 1:100, size = 1)
  # increase number of surprise packages opened
  surprise_packages <- surprise_packages + 1

}

# let me know how many packages I opened
print(surprise_packages)
```



```
# initial number of seeds
total_seeds <- 0
# initial number of surprise packages opened
surprise_packages <- 0
```

```
while(total_seeds < 1000) {

  # open surprise packages and add to total number of seeds
  total_seeds <- total_seeds + sample(x = 1:100, size = 1)
  # increase number of surprise packages opened
  surprise_packages <- surprise_packages + 1

}
```

```
# let me know how many packages I opened
print(surprise_packages)
```

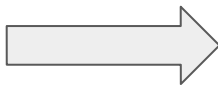
```
# initial number of seeds
total_seeds <- 0
# initial number of surprise packages opened
surprise_packages <- 0

while(total_seeds < 1000) {

  # open surprise packages and add to total number of seeds
  total_seeds <- total_seeds + sample(x = 1:100, size = 1)
  # increase number of surprise packages opened
  surprise_packages <- surprise_packages + 1

}

# let me know how many packages I opened
print(surprise_packages)
```



First time → **19**

Second time → **17**

Third time → **21**

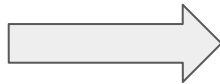
```
# initial number of seeds
total_seeds <- 0
# initial number of surprise packages opened
surprise_packages <- 0

while(total_seeds < 1000) {

  # open surprise packages and add to total number of seeds
  total_seeds <- total_seeds + sample(x = 1:100, size = 1)
  # increase number of surprise packages opened
  surprise_packages <- surprise_packages + 1

}

# let me know how many packages I opened
print(surprise_packages)
```



Infinite loop

(will never finished running)

Getting help: a matter of asking the right questions

(google == “best friend”)

Read the docs

1

Use `?function` in R, go to package website, look for tutorials/examples online

Google error/question

2

Spend some time reading through the comments in StackOverflow or other forum post, and know that you may need to try slightly different things

Learn how to ask the right question

3

This is a skill that you get better as you go, but usually involves being precise on what you ask and use certain terminology of your programming language

Learn how to ask the right question

3

“how to get lines of my table according to a number in R”

vs

“how to **subset** my **data frame** in R”

4

Ask your friend :)

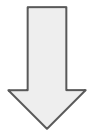
They don't need to be experts, but they might have had similar problems before or they can help you debug

Pseudocoding: write your code in a human language

(then translate to computer language)

Break down a complex task into a
series of **smaller and simpler** tasks

Calculate allele frequencies for each marker in this file



```
load the file
```

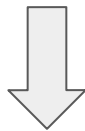
```
for each marker in that data frame
```

```
    count total number of alleles
```

```
    count how many times each allele shows up
```

```
    divide the allele count by total
```

Calculate allele frequencies for each marker in this file



```
load the file
```

```
create empty vector to store values
```

```
for each marker in that data frame
```

```
  count total number of alleles
```

```
  count how many times each allele shows up
```

```
  divide the allele count by total
```

```
append frequencies to vector
```

```
write file with allele frequencies
```

This is a skill and takes time to develop!

Writing you pseudocode as comments in your script, helps you visualize what you need to do and make your translation to R language easier

Your turn!

Hands-on exercises



1

If you want to practice some R basics:

simple_marker_analysis.R

2

If you know the basics and want some challenge:

need4speed.R

Extra

More advanced stuff...



apply() function

Do something at each iteration

```
apply(x, MARGIN, FUN)
```

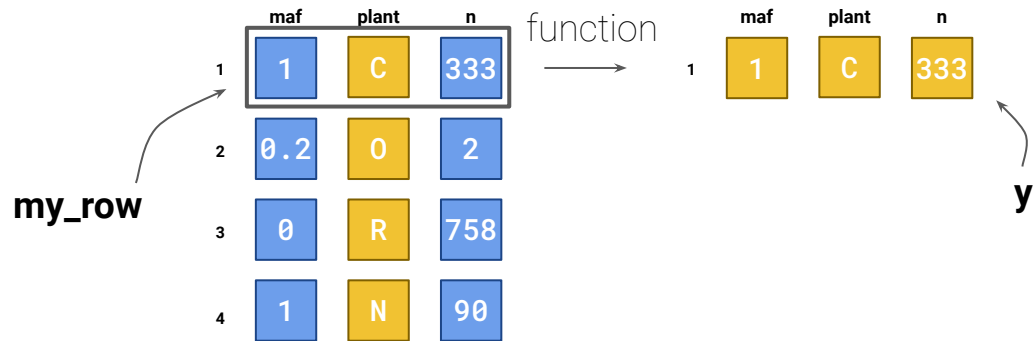
Iterate through data frame by:

Row → MARGIN = 1

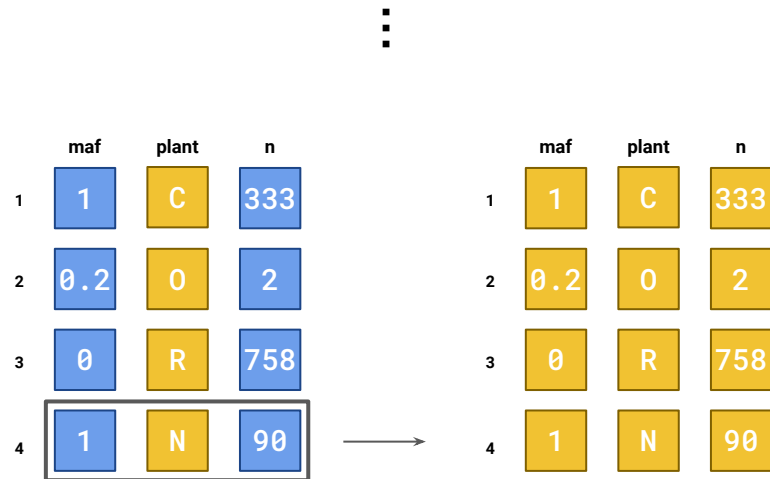
Column → MARGIN = 2

Element → MARGIN = c(1, 2)

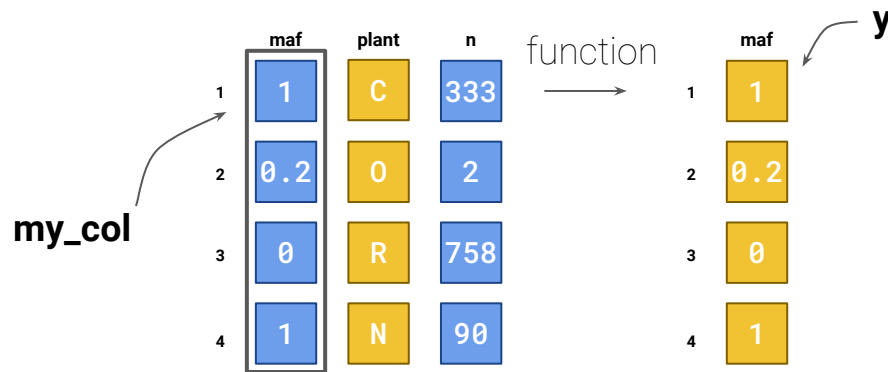
apply() function



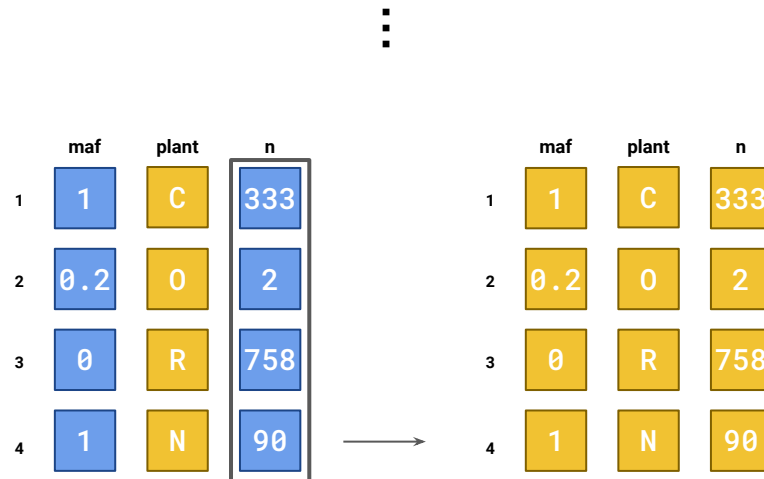
```
apply(my_df,  
      MARGIN = 1,  
      function(my_row) {  
        y <- as.character(my_row)  
        return(y)  
      })
```



apply() function



```
apply(my_df,  
      MARGIN = 2,  
      function(my_col) {  
        y <- as.character(my_col)  
        return(y)  
      })
```



Plotting with ggplot2

<https://datacarpentry.org/R-ecology-lesson/04-visualization-ggplot2.html>

Working directory

Know where you are:

getwd()

Change where you are:

setwd()

```
getwd()
```

```
[1] "/rcloud/project/"
```

```
read.file("file.txt")
```

```
[1] error
```

```
read.file("data/file.txt")
```

```
getwd()
```

```
[1] "/rcloud/project/"
```

```
read.file("file.txt")
```

```
[1] error
```

```
setwd("data")
```

```
getwd()
```

```
[1] "/rcloud/project/data"
```

```
read.file("data/file.txt")
```

```
[1] error
```

```
read.file("file.txt")
```

Keep learning!

Online courses:

www.datacamp.com

www.edx.org

www.coursera.org

Data carpentry:

datacarpentry.org/

Style guide:

style.tidyverse.org/