CGIAR IRRI

Fundamentals of Genomic Prediction and Data-Driven Crop Breeding (November, 2025)

Introduction and Learning R Software

Module 1
November 24, 2025

Waseem Hussain, Mahender Anumalla and Margaret Catolos
Rice Breeding Innovations Platform
IRRI

IRRI International Rice Research Institute

Why should I learn R?

- R is a free open-source software and programming language.
    - Summarize, explore and model the data
- Reproducible research (code +text).
- Huge learning Resources and Community.
    - https://bookdown.org/.
    - https://www.r-bloggers.com/
    - https://gkhajduk.github.io/R-resources/
    - https://www.computerworld.com/article/2497464/top-r-language-resources-to-improve-your-data-skills.html
- Popular graphical capabilities.
- Dominant and useful variety of scientific disciplines.

In R Base *functionality is extended through packages*

What is R Studio

RStudio is an integrated development environment (IDE) for R

- Easy to control and manage the R scripts (point and click)
- View and interact with the objects in single environment.
- Easy to set your working directory and access files on your computer
- Graphics more accessible.

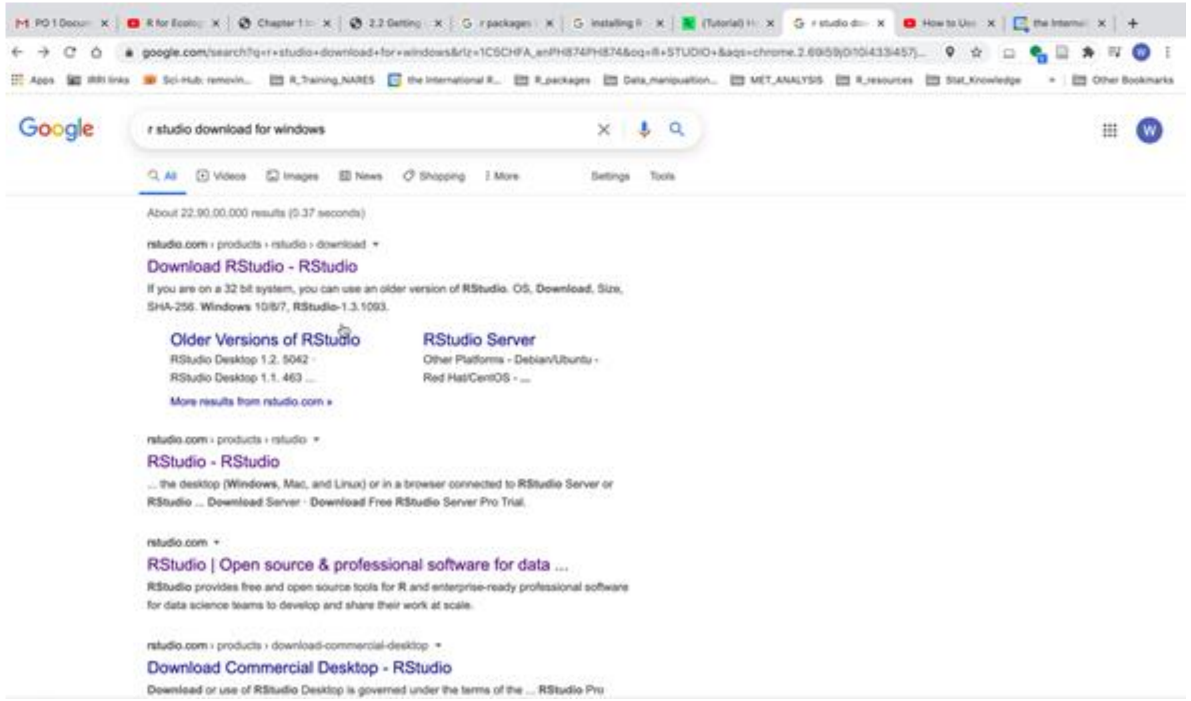
More features see the link:

https://rstudio.com/products/rstudio/features/

Installing R and R studio

https://rstudio.com/

https://www.r-project.org/



r studio download for windows

About 22,90,00,000 results (0.37 seconds)

rstudio.com › products › rstudio › download
Download RStudio - RStudio
If you are on a 32 bit system, you can use an older version of RStudio. OS, Download, Size, SHA-256. Windows 10/8/7, RStudio-1.3.1093.

Older Versions of RStudio
RStudio Desktop 1.2. 5042 · RStudio Desktop 1.1. 463 ...

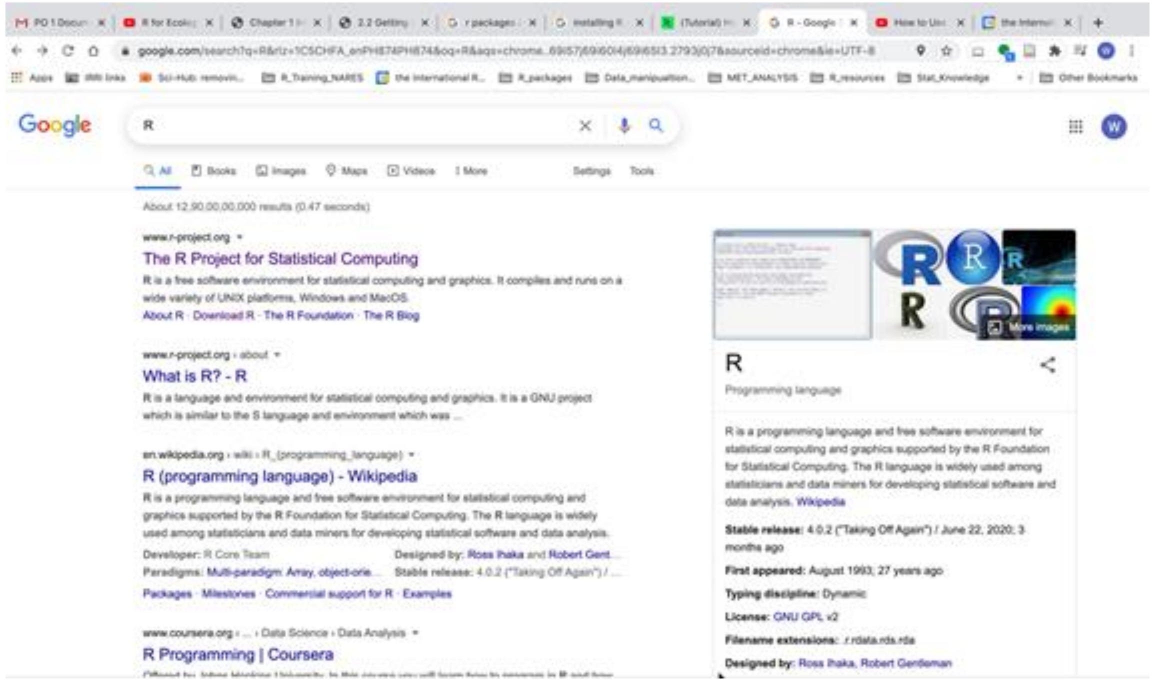
RStudio Server
Other Platforms - Debian/Ubuntu · Red Hat/CentOS · ...

More results from rstudio.com »

rstudio.com › products › rstudio
RStudio - RStudio
... the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio ... Download Server · Download Free RStudio Server Pro Trial.

rstudio.com
RStudio | Open source & professional software for data ...
RStudio provides free and open source tools for R and enterprise-ready professional software for data science teams to develop and share their work at scale.

rstudio.com › products › download-commercial-desktop
Download Commercial Desktop - RStudio
Download or use of RStudio Desktop is governed under the terms of the ... RStudio Pro



R

About 12,90,00,00,000 results (0.47 seconds)

www.r-project.org
The R Project for Statistical Computing
R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.
About R · Download R · The R Foundation · The R Blog

www.r-project.org › about
What is R? - R
R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was ...

en.wikipedia.org › wiki › R_(programming_language)
R (programming language) - Wikipedia
R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.
Developer: R Core Team · Designed by: Ross Ihaka and Robert Gent...
Paradigms: Multi-paradigm: Array, object-orie... · Stable release: 4.0.2 ("Taking Off Again") / ...
Packages · Milestones · Commercial support for R · Examples

www.coursera.org › ... › Data Science › Data Analysis
R Programming | Coursera

R
Programming language
R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Wikipedia
Stable release: 4.0.2 ("Taking Off Again") / June 22, 2020; 3 months ago
First appeared: August 1993; 27 years ago
Typing discipline: Dynamic
License: GNU GPL v2
Filename extensions: .r.rdata.rds.rda
Designed by: Ross Ihaka, Robert Gentleman

https://www.datacamp.com/community/tutorials/installing-R-windows-mac-ubuntu

Installing and Loading R packages

Installing and Loading R packages

What is R package?

•Bundles of codes build by the people to perform certain tasks

•Maintained at Comprehensive R Archive Network (CRAN)/Bioconductor/GitHub

➢**Install from CRAN**

install.package("ggplot2")

➢**Install from Bioconductor-** Packages for life sciences related data

BiocManager handles all of the packages hosted on Bioconductor

install.packages("BiocManager")

BiocManager::install("SNPRelate")

BiocManager::install("phyloseq")

➢Install from GitHub-

install.packages("devtools")

devtools::install_github("tidyr")

Resources
http://www.sthda.com/english/wiki/installing-and-using-r-packages
https://astrobiomike.github.io/R/installing_packages

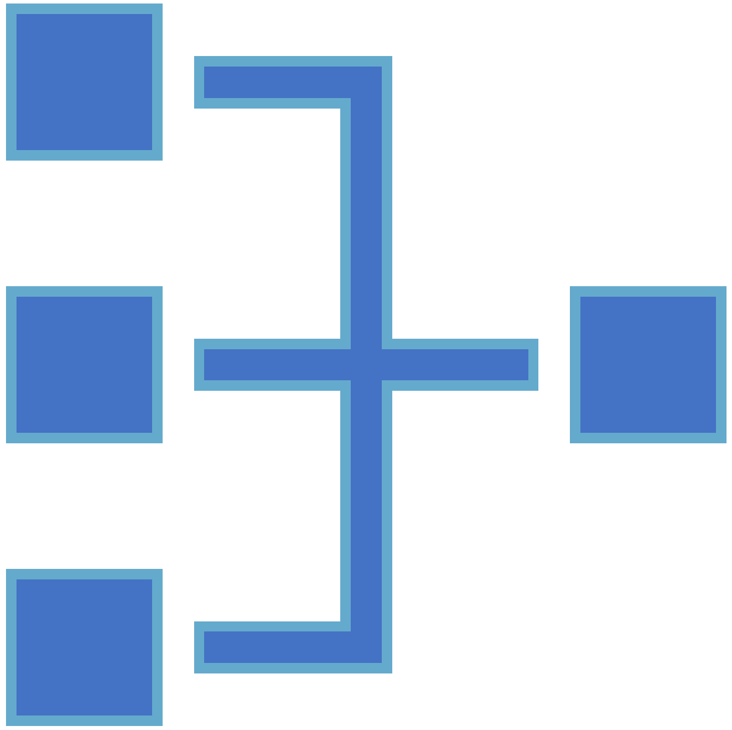
R Essentials, Data types and Structures

R essentials

R works on expression and objects

- **Users enters expression** (for example 2+2)
  - Expression involves operators or function calls.
  - Expression work on Objects
- **R evaluates it**
- **And Print the Results** , 4

What are Function Calls



- Calling a function which involves one or more variables.

  For example *sum(x)* or *plot (x)*. Here sum and plot are function calls.

- Function format is followed by a set of parentheses containing one or more arguments. *function(),*

  Example: *plot (height, weight),* Height and weight are the arguments

- More arguments on function *plot()*

  Plot(height, weight, pch=2, color="red"…..)

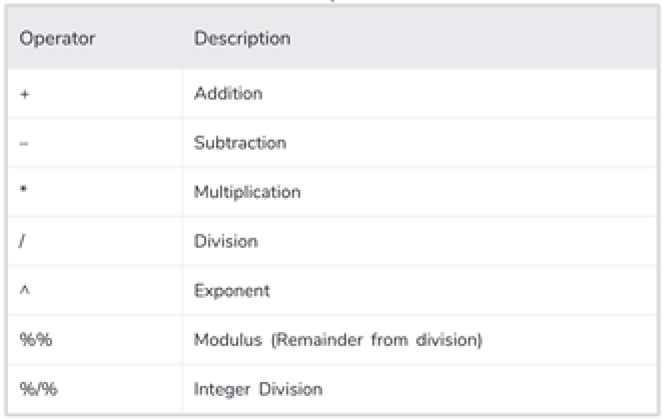
- Positional matching

  Plot(x=height, y=weight, pch=2…)

Operators in R

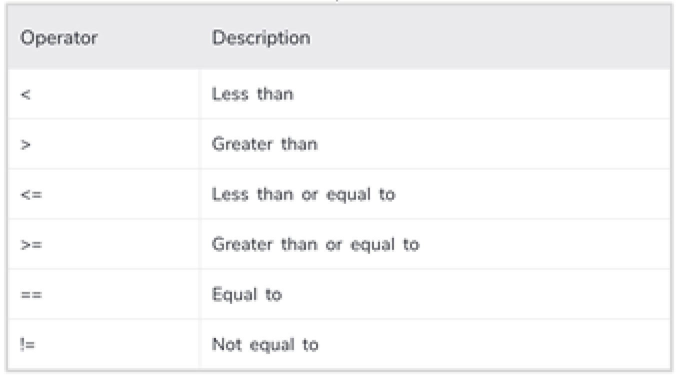
- Operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.
- R language is rich in built-in operators and provides following types of operators.

Arithmetic Operators in R



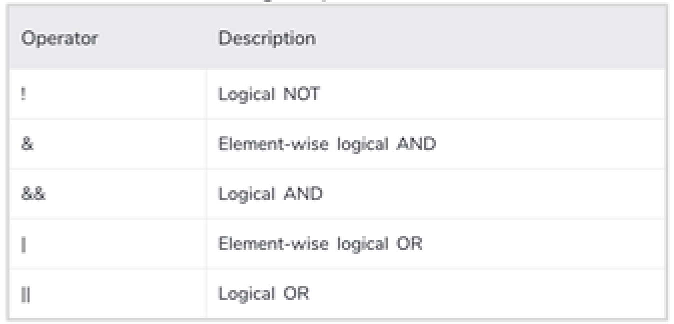
Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

Relational Operators in R



Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

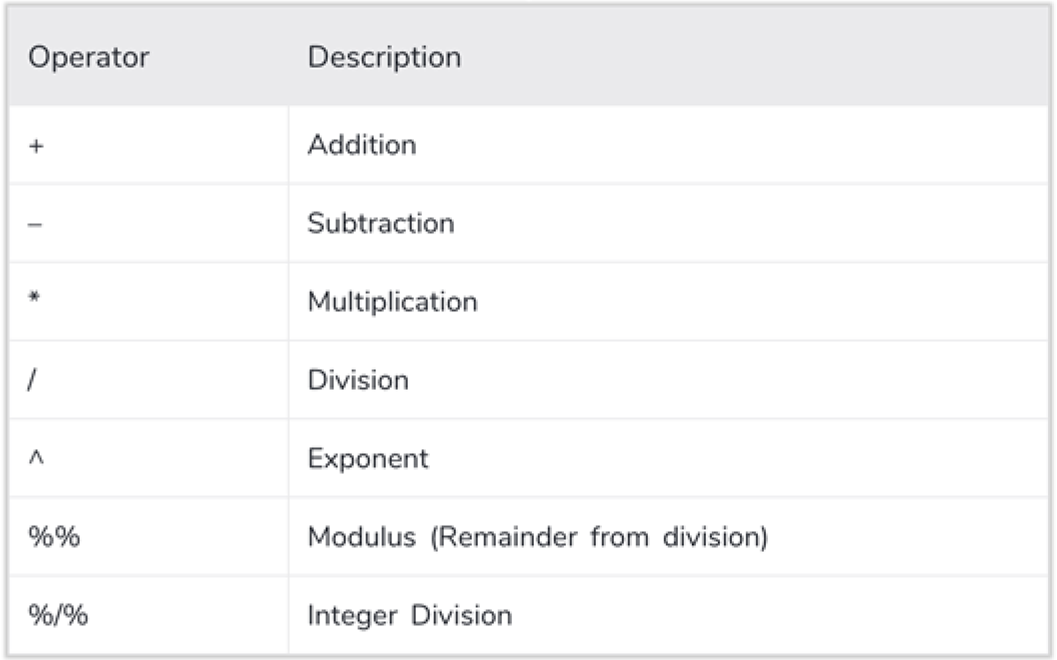
Logical Operators in R



Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR

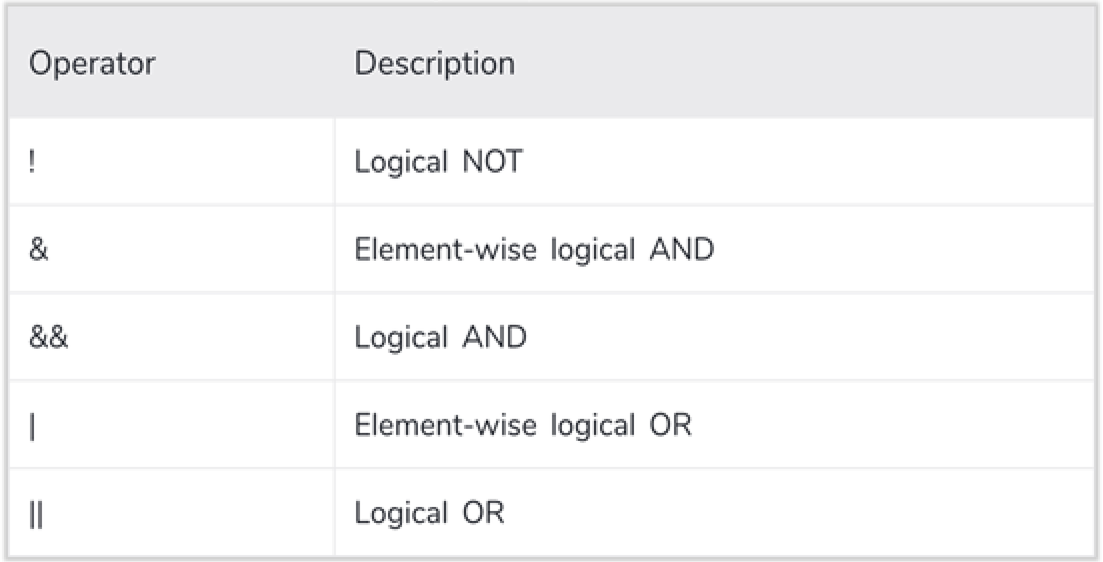
https://www.tutorialspoint.com/r/r_operators.htm

Arithmetic Operators in R



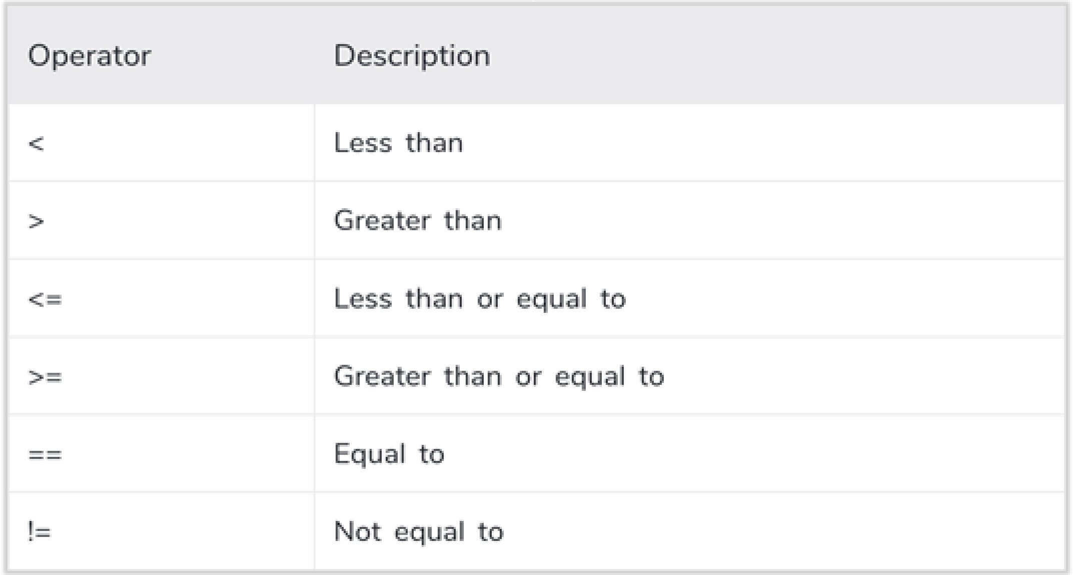
Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

Logical Operators in R



Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR

Relational Operators in R



Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Data types in R

Six data types are in R:

- **Character**: “Block”, “Replication”
- **Numeric** (real or decimal)- 2.4, 2, 10
- **Integer**: 2L, 3L
- **Logical**: TRUE, FALSE
- **Complex**: 1+6i

In R

class() - what kind of object is it

length()- how long it is

Data Structures in R

Vectors

- Vector is a basic data structure in R which contains a list of same elements
- Vectors are created using *function c()*
- *c()* concatenate the elements
- Examples
- *X<-c(1,2,3,4,5)* # five components
- Assignment operator in R is <-

Data Structures in R

Scalers

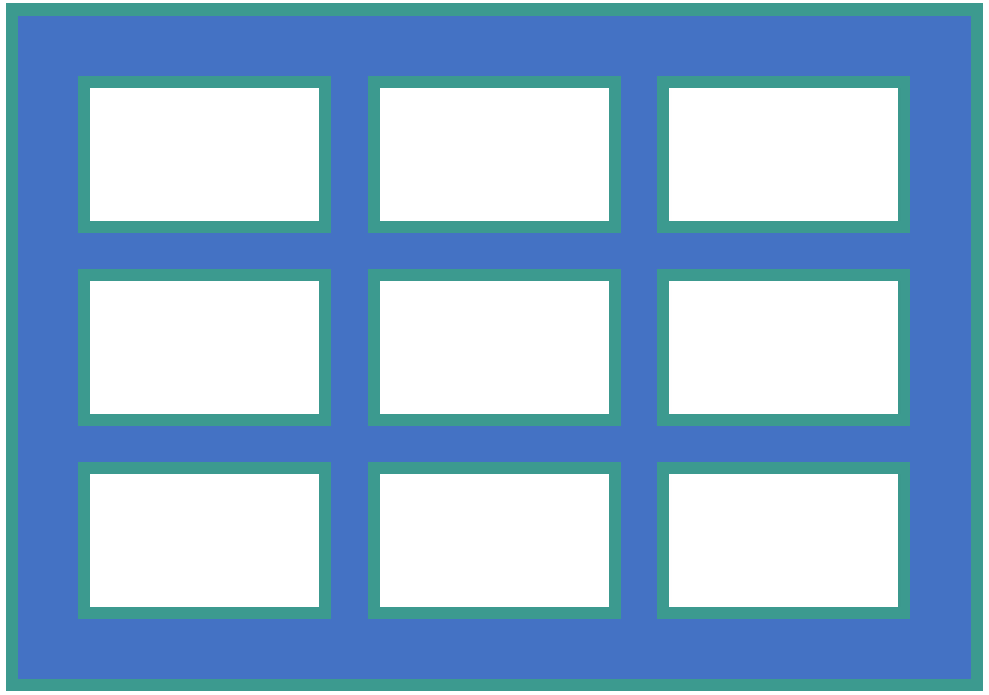
A scalar object is just a single value like a number or a name.

For example,

a <- 100

X<-”name”

Scalars don’t have to be numeric, they can also be **characters** (also known as strings)



Data Structures in R

Matrices

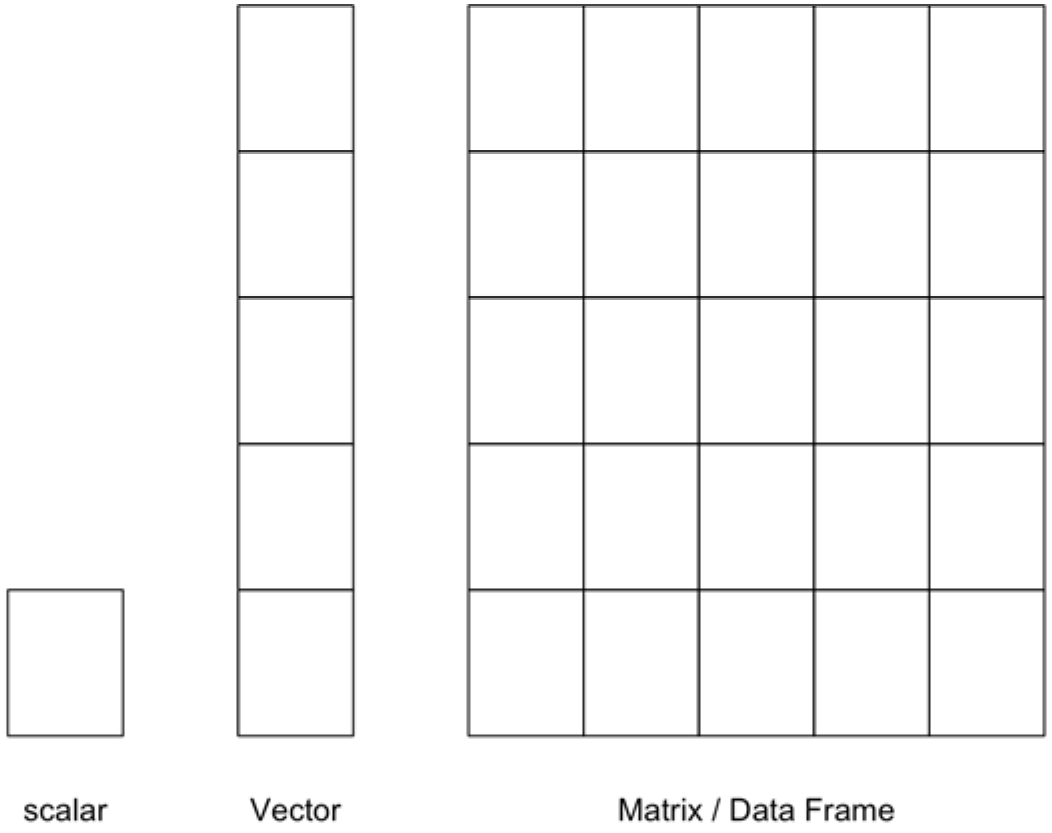
- Matrices are numeric array of rows and columns.

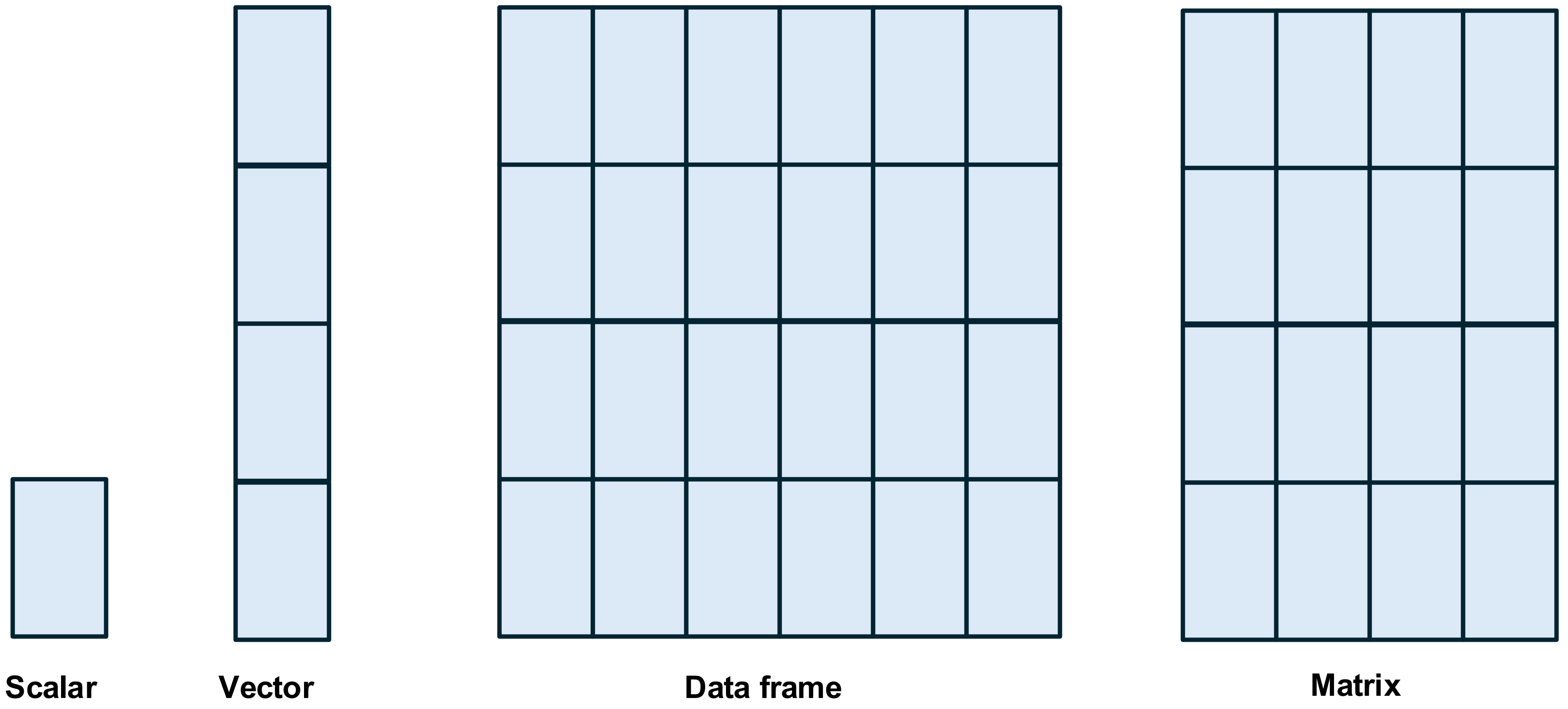
Think as Stacked version of vectors where each row and column is basically a vector.

Combination of n vectors

matrix(data, nrow, ncol, byrow = FALSE)

See demo how to create and deal with matrices




Scalar
Vector
Data frame
Matrix

Data Structures in R

LISTS

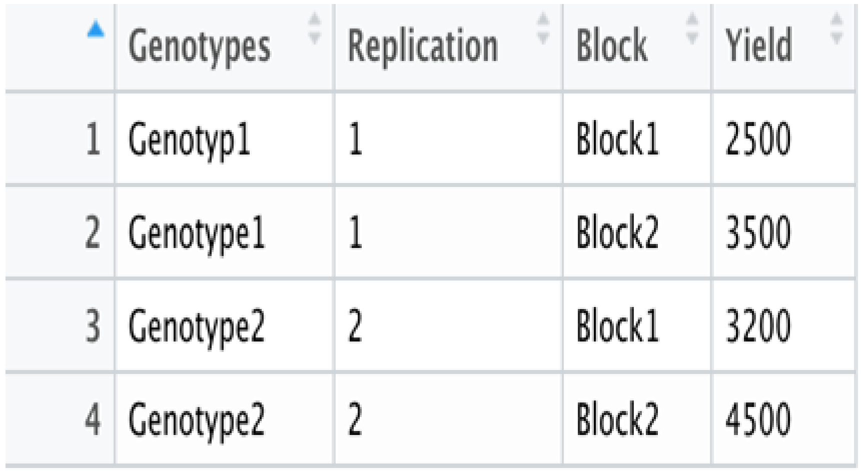
List is a data structure having components of mixed data types.

- To create a list we use function list()
- Demo in r

Data Structures in R

Data Frames

- ➢ More general than matrix, which has columns with different modes (numeric, character, factor).
- ➢ A data frame can be constructed by the *data.frame()*



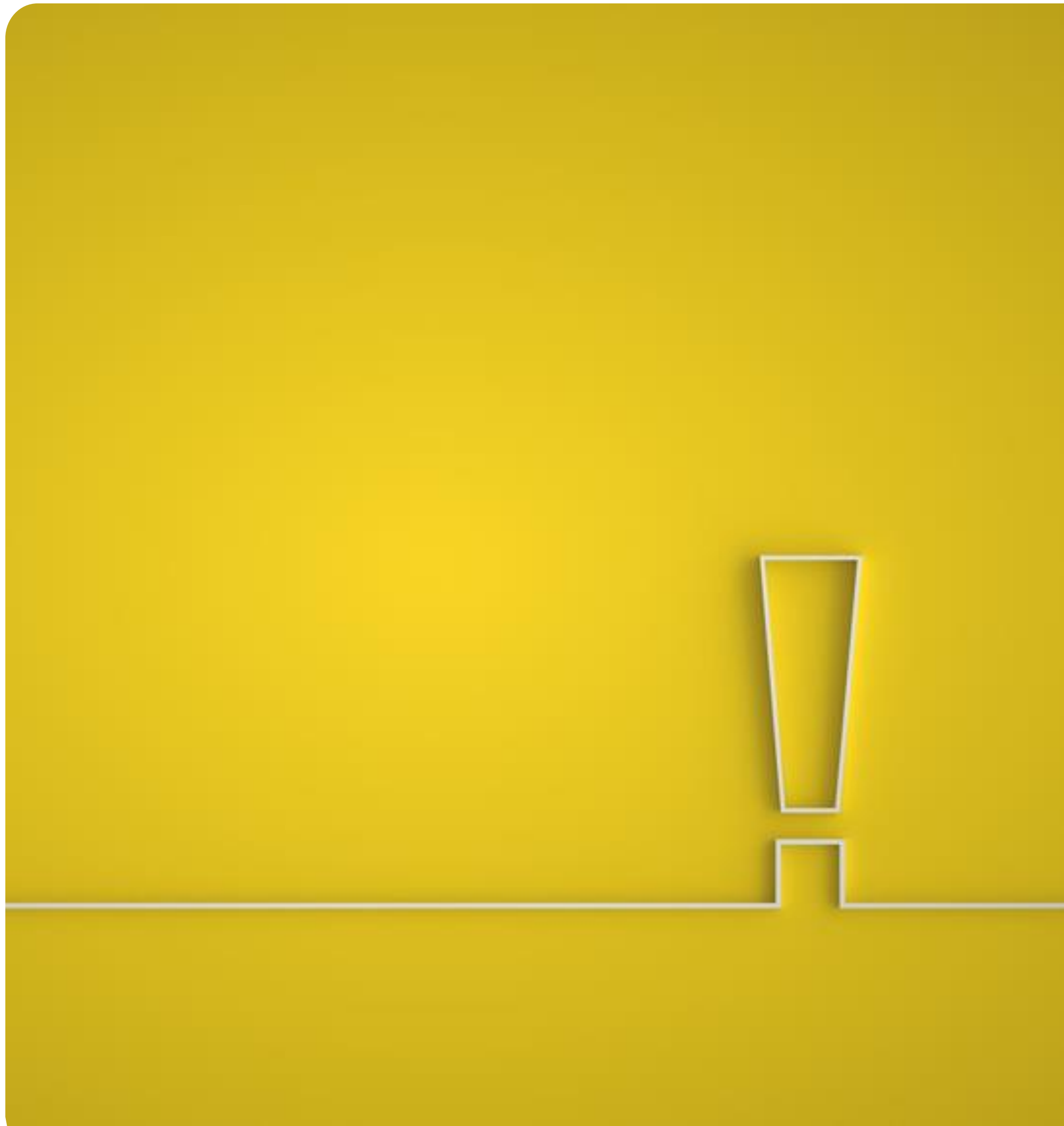
	Genotypes	Replication	Block	Yield
1	Genotyp1	1	Block1	2500
2	Genotype1	1	Block2	3500
3	Genotype2	2	Block1	3200
4	Genotype2	2	Block2	4500

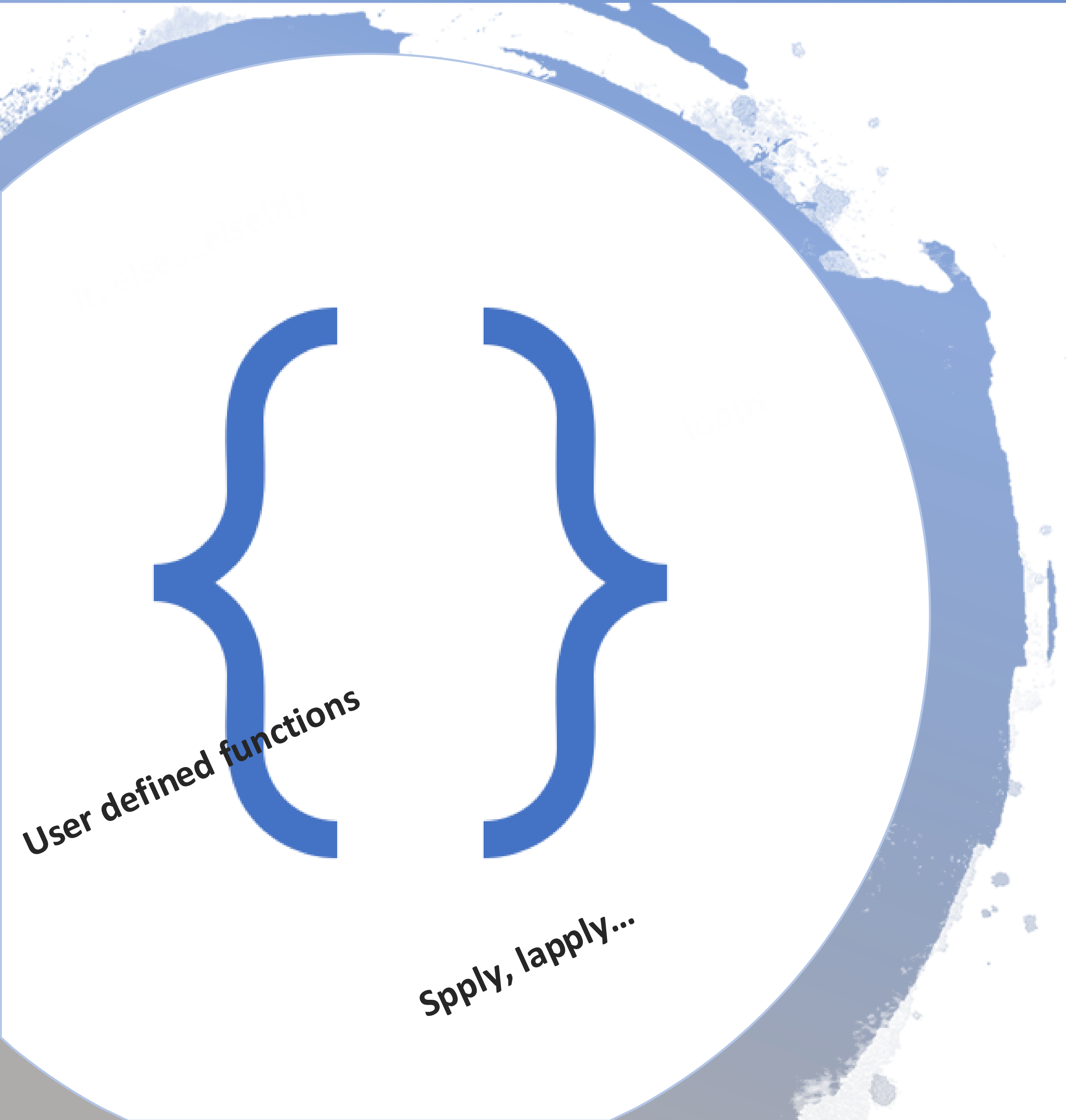
- Demo in r

Data Structures in R

FACTORS

- Factors are categorical variables with different levels or subdivisions.
- For example in plant Breeding Replications can be treated as factors with number of replications as factor levels
- RCBD design with 3 replications
- Replication has 3 levels
- *levels()* in R is used to determine number of levels





Data manipulations, Control structures and Functions

Control Structure s

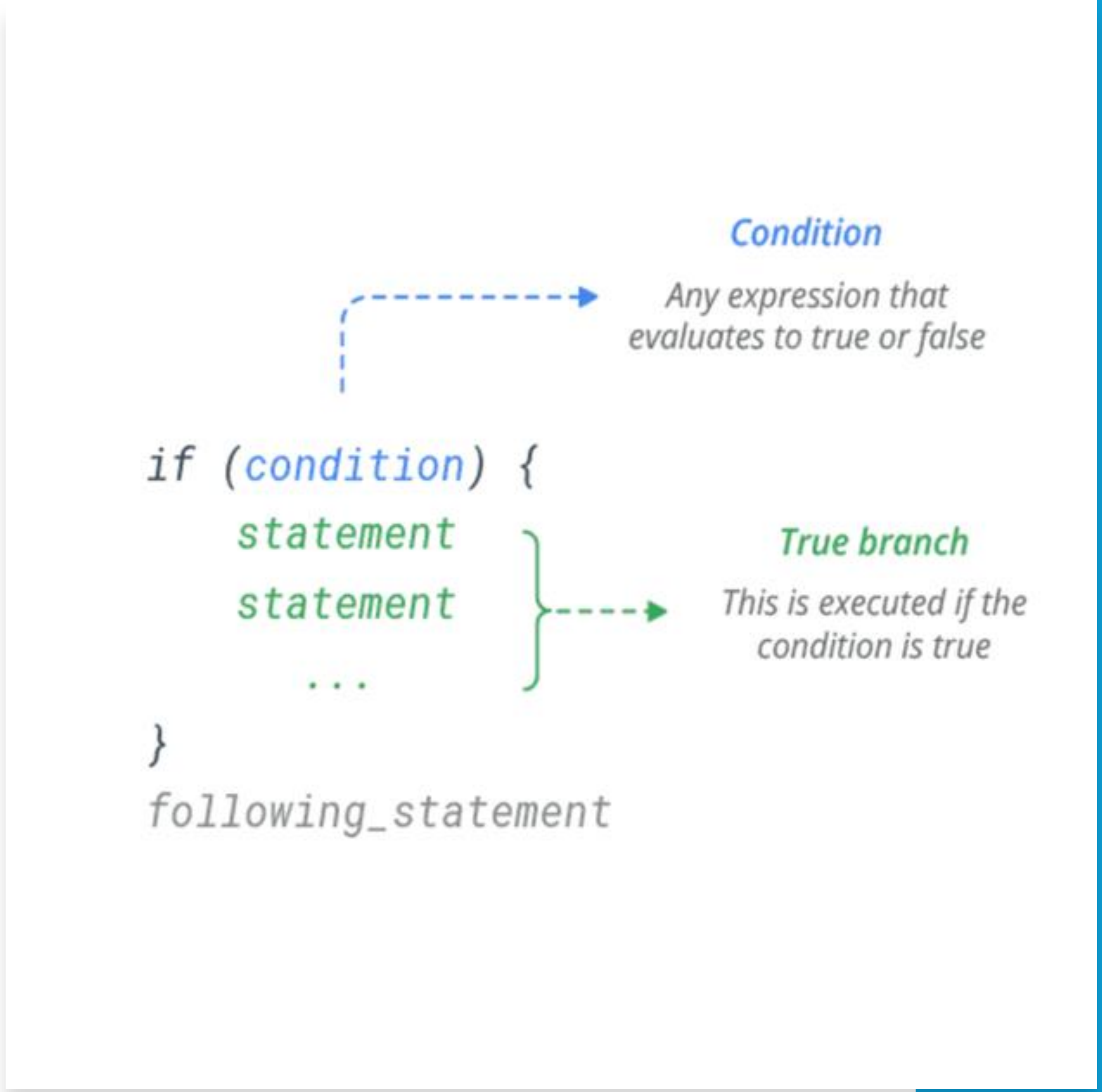
Allow users to control the flow of execution of a series of R expressions.

Commonly used control structures are:

- ***if and else***: testing a condition and acting on it
- ***for***: execute a loop a fixed number of times
- ***while***: execute a loop *while* a condition is true
- ***repeat***: execute an infinite loop (must break out of it to stop)
- ***break***: break the execution of a loop
- ***next***: skip an interation of a loop

If statement

Execute a block of code, if a specified condition is true

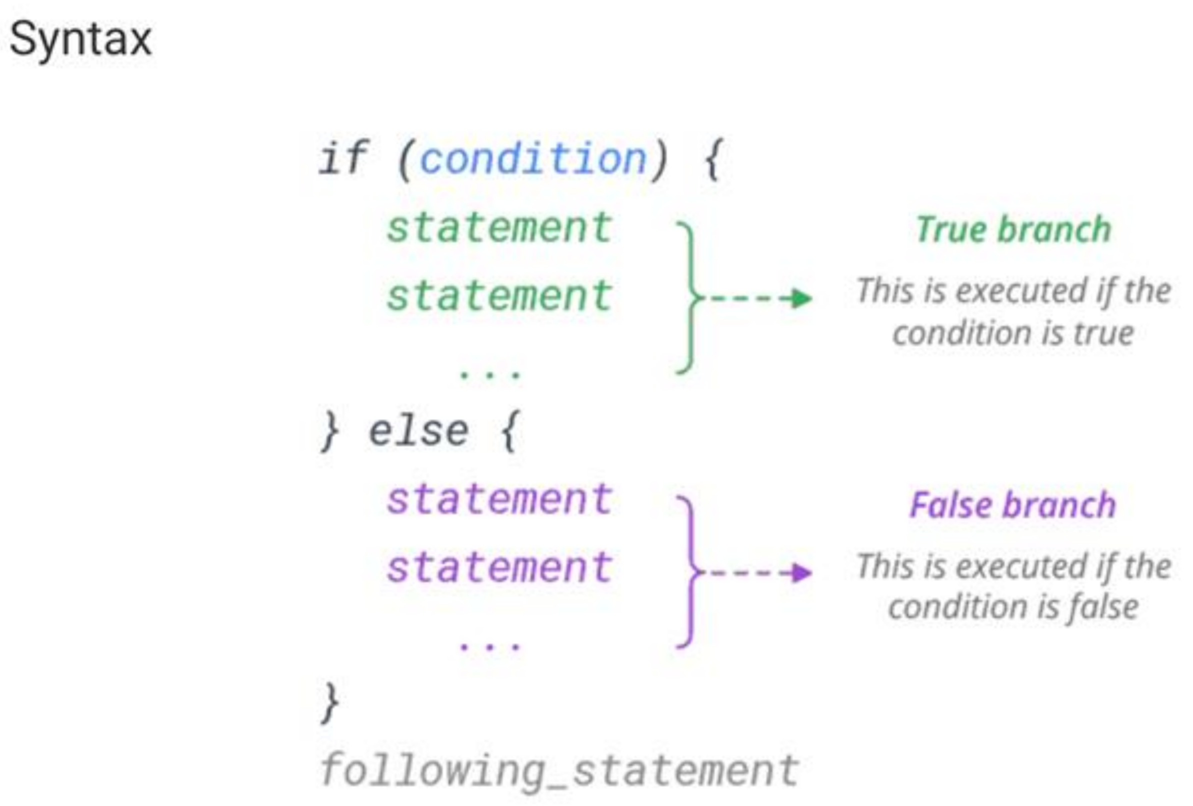

```
if (condition) {
    statement
    statement
    ...
}
following_statement
```
Condition
Any expression that evaluates to true or false
True branch
This is executed if the condition is true

Demo in R

Adopted from: https://www.learnbyexample.org/r-if-else-elseif-statement

If and else statement

*E*xecute a block of code, if the condition is false


Syntax

```
if (condition) {
    statement
    statement
       ...
} else {
    statement
    statement
       ...
}
following_statement
```

True branch
This is executed if the condition is true
False branch
This is executed if the condition is false

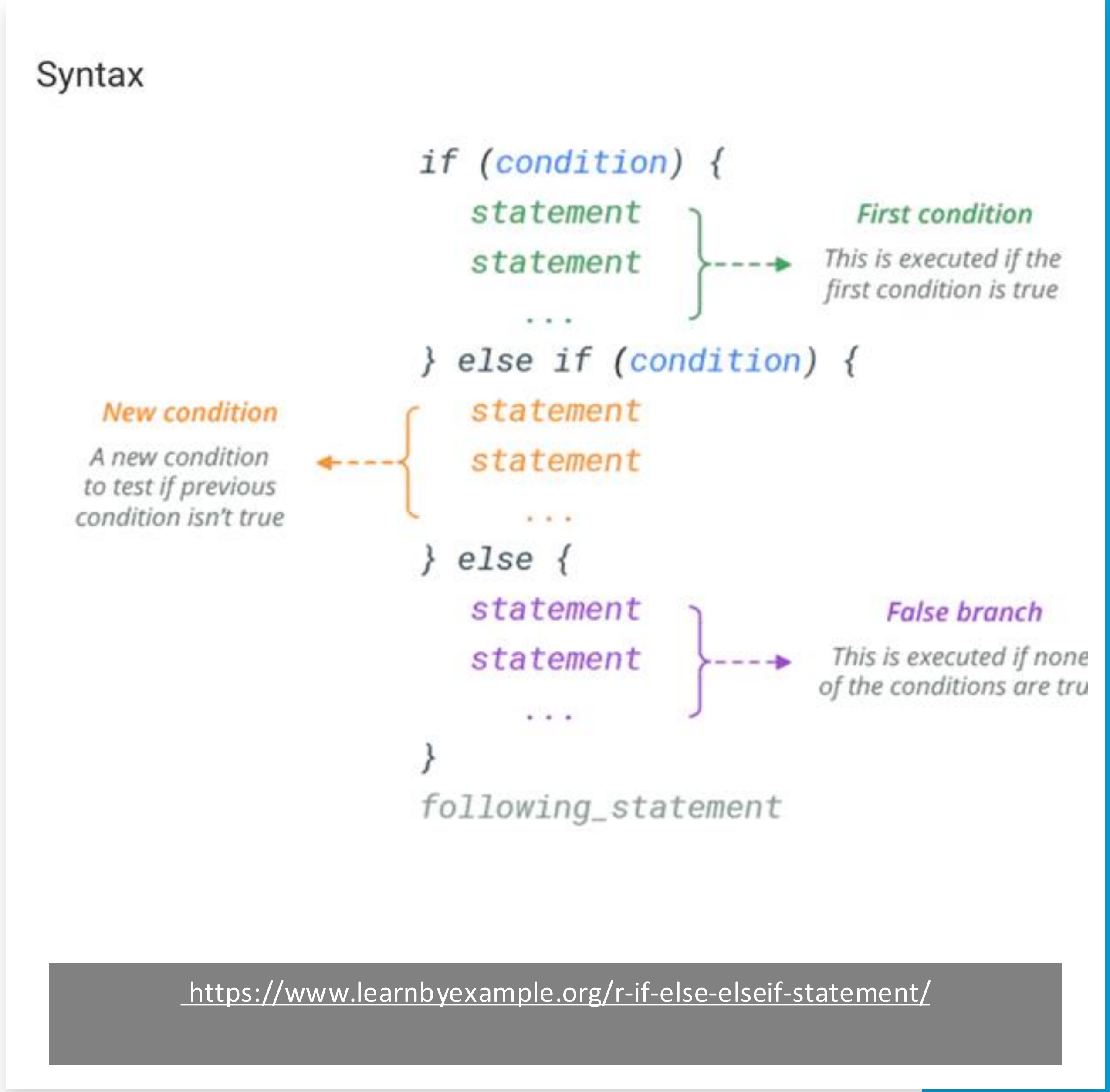
Adopted from: https://www.learnbyexample.org/r-if-else-elseif-statement/

Demo in R

Else If statement

Specify a new condition to test, if the first condition is false.

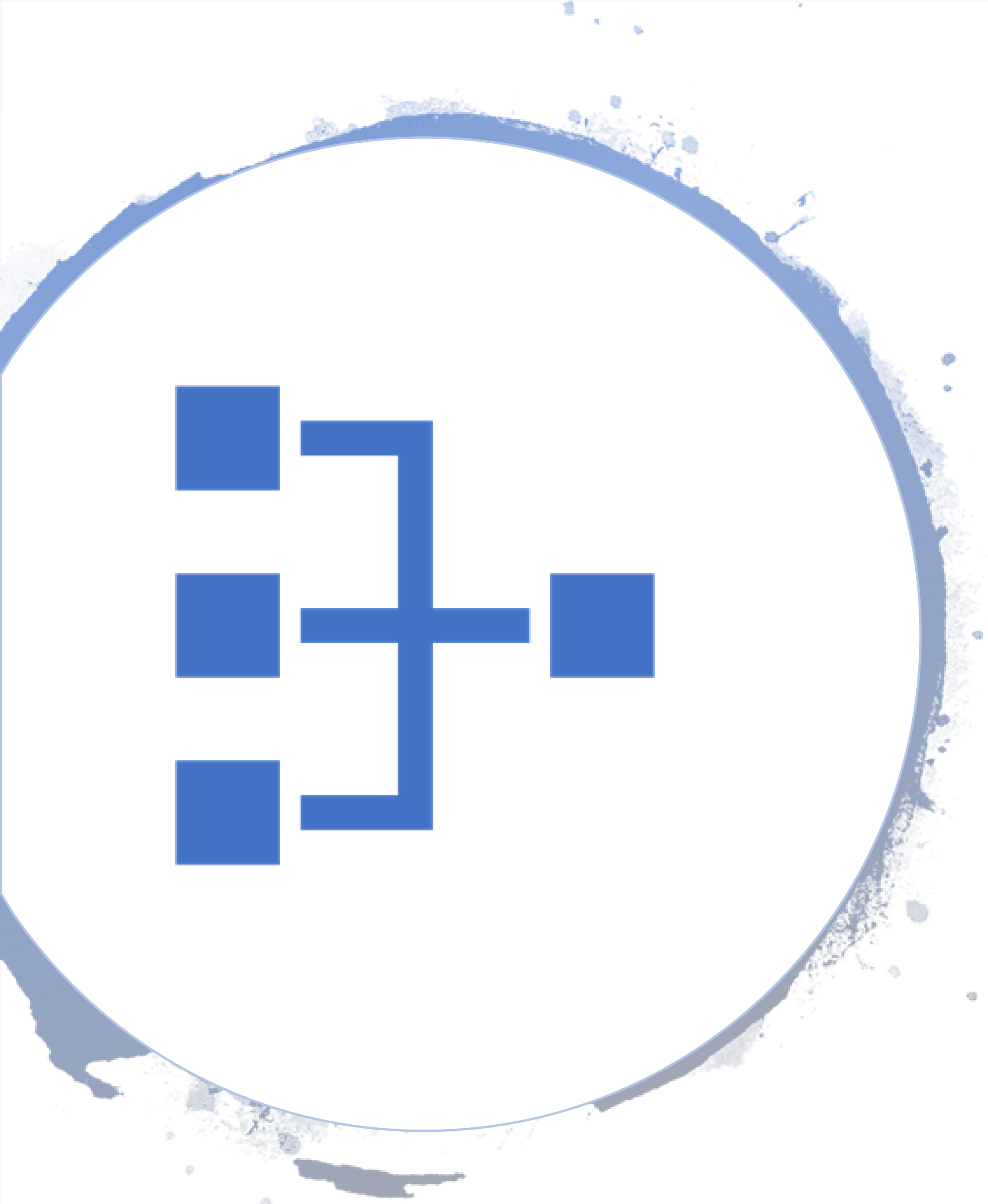
Demo in R


Syntax

```
if (condition) {
    statement
    statement
    ...
} else if (condition) {
    statement
    statement
    ...
} else {
    statement
    statement
    ...
}
following_statement
```

First condition
This is executed if the first condition is true
New condition
A new condition to test if previous condition isn't true
False branch
This is executed if none of the conditions are tru

https://www.learnbyexample.org/r-if-else-elseif-statement/



Multiple Condition Statements

Join two or more conditions into a single if statement

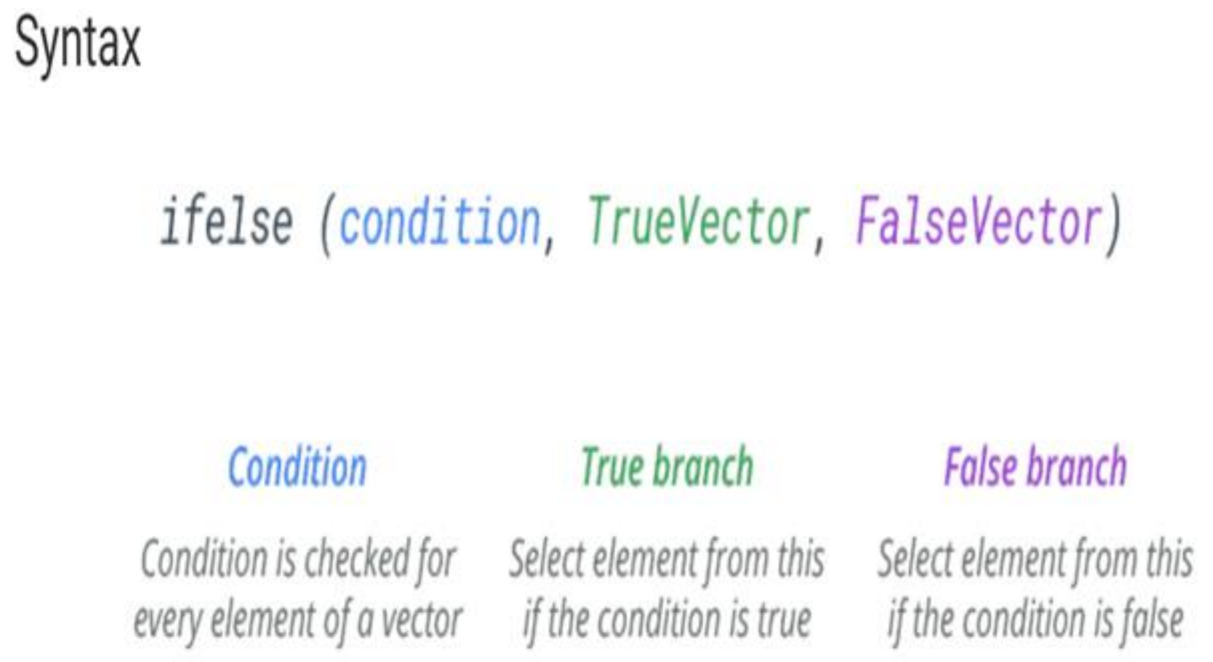
Logical operators: && (and), || (or) and ! (not). && (and) expression is True, if all the conditions are true.

Demo in R

https://www.learnbyexample.org/r-if-else-elseif-statement/

Conditional statements act on single element!

Ifelse() functior


Syntax

```
ifelse (condition, TrueVector, FalseVector)
```

Condition	True branch	False branch
Condition is checked for every element of a vector	Select element from this if the condition is true	Select element from this if the condition is false

function checks the condition for every element of a vector and selects elements from the specified vector depending upon the result.

Adopted from: https://www.learnbyexample.org/r-if-else-elseif-statement/

Demo in R

Apply functions

Repetitively perform an action on multiple chunks of data.

Runs faster than loop and requires less coding

Basic function

apply(X, MARGIN, FUN).

- ❖X is an array or matrix.
- ❖Margin specifies whether you want to apply the function across rows (1) or columns (2)
- ❖FUN is the function you want to use

Demo in R

1. *lapply* functions

lapply() operates on list and always returns a list, 'l' in lapply() refers to 'list'

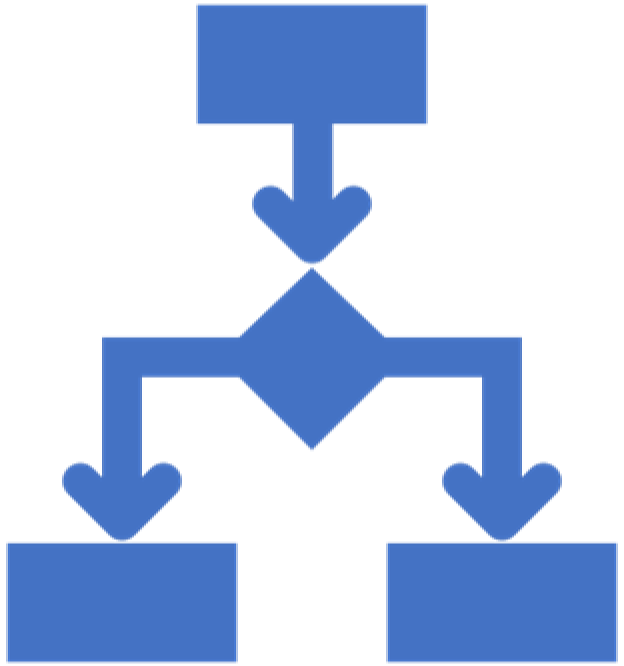
lapply(X, FUN, …)

X is a list

Fun, function to be applied

…. Additional arguments passed to function

- ❖ lapply() always returns a list whereas apply() can return a vector, list, matrix or array.
- ❖ No scope of MARGIN in lapply(), always to columns

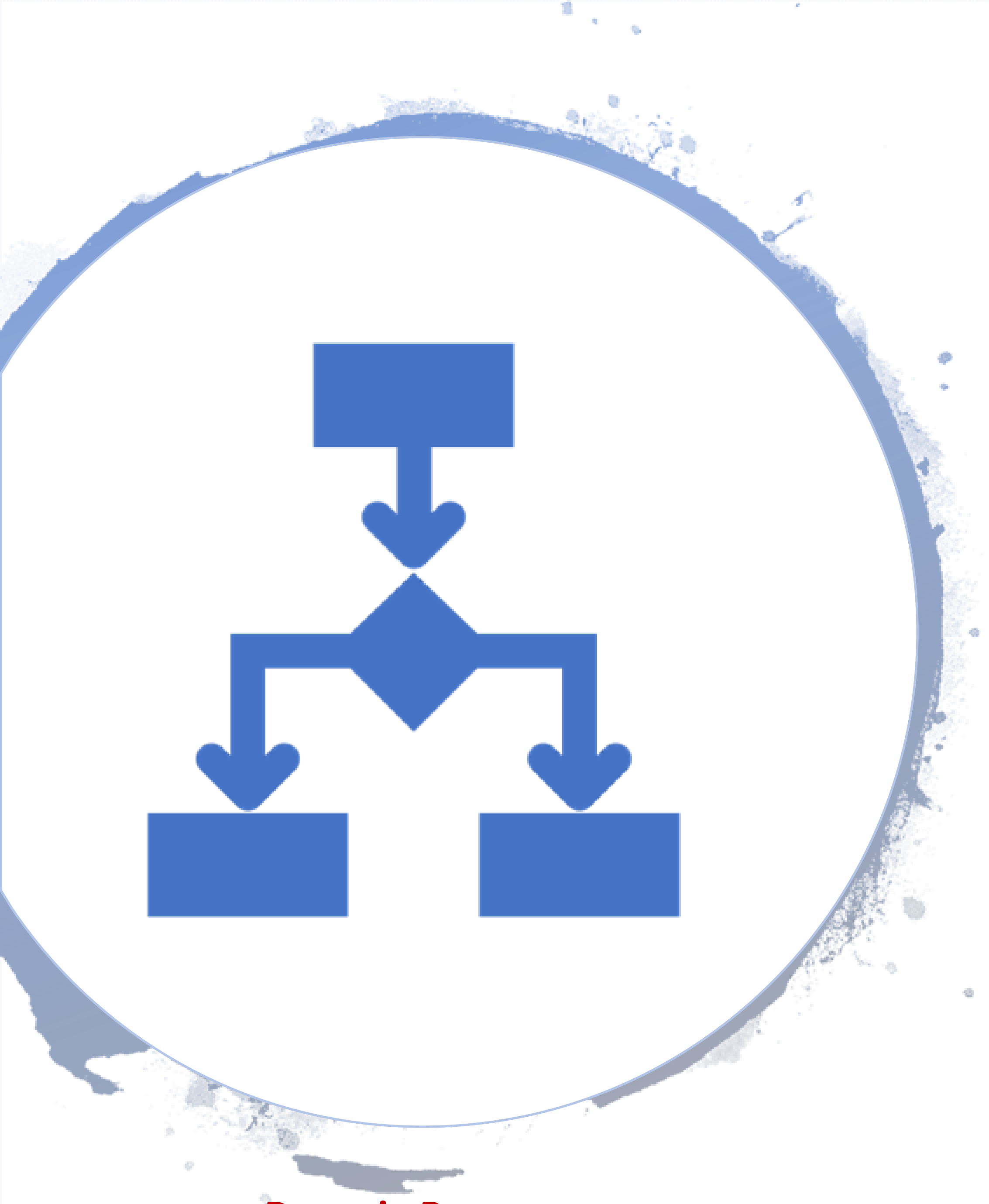


Demo in R

2. *sapply* functions

syntax for sapply() is as follows:
sapply(x, fun,…..)

- ❖sapply() and lapply() work basically the same.
- ❖The only difference is that lapply() always returns a list, whereas sapply() tries to simplify the result into a vector or matrix.
- ❖Additional argument if simplify = F then sapply() returns a list similar to lapply()



Demo in R

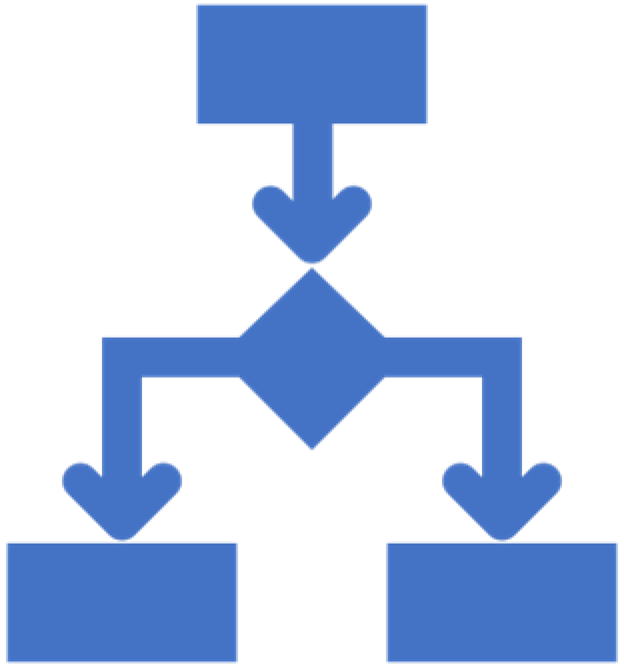
3. *tapply* functions

tapply() function breaks the data set up into groups and applies a function to each group.

The syntax:

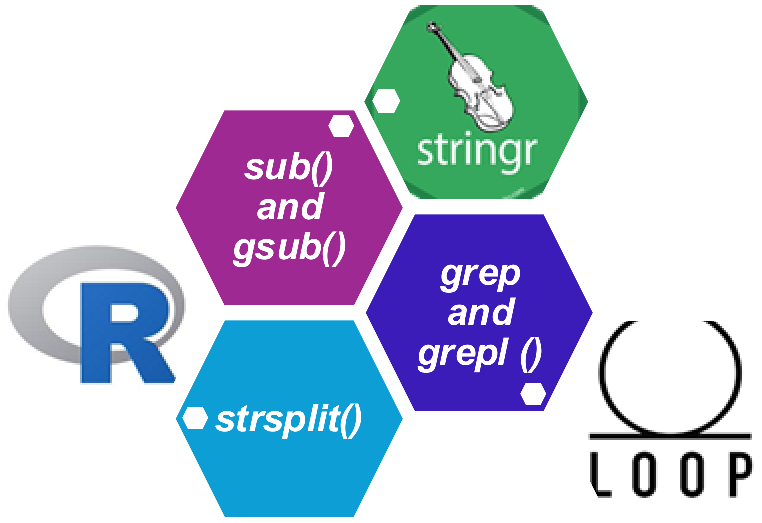
tapply(x,INDEX,FUN,…,simplify)

- x is required vector
- A grouping factor or a list of factors
- The function to be applied
- ….. Additional arguments
- Simplify return simplified results



Demo in R

Loops and String Manipulations in R



Loops in R
(Cycling or iterating)

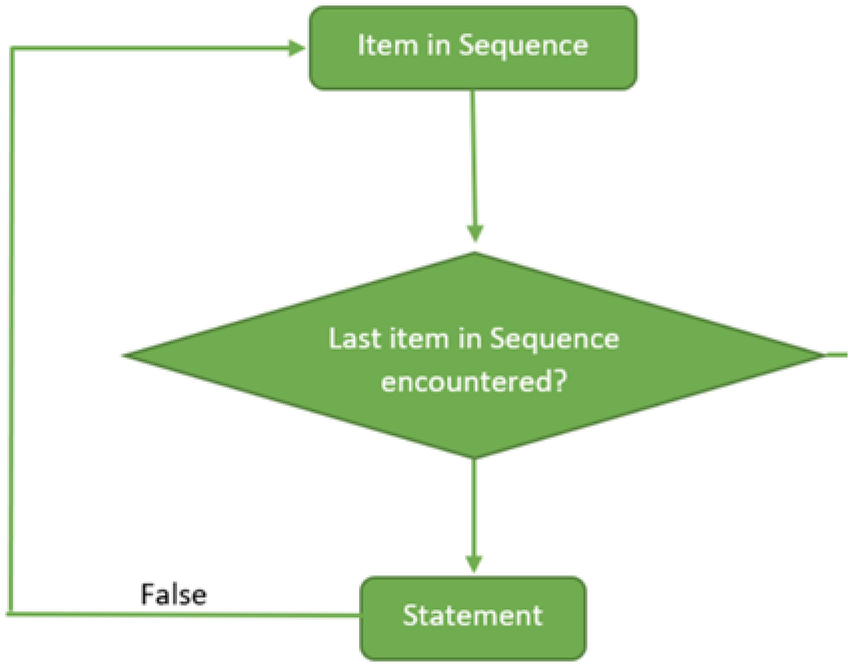
Control statement that allows multiple executions of a statement or a set of statements

For loop

- ➢ Loops our texts, data frames etc.
- ➢ Loops repeatedly depending upon the number of elements

Syntex

```
        for (var in vector) {
            statement(s)
                 }
```



https://www.geeksforgeeks.org/loops-in-r-for-while-repeat/

Loops in R
(Cycling or iterating)

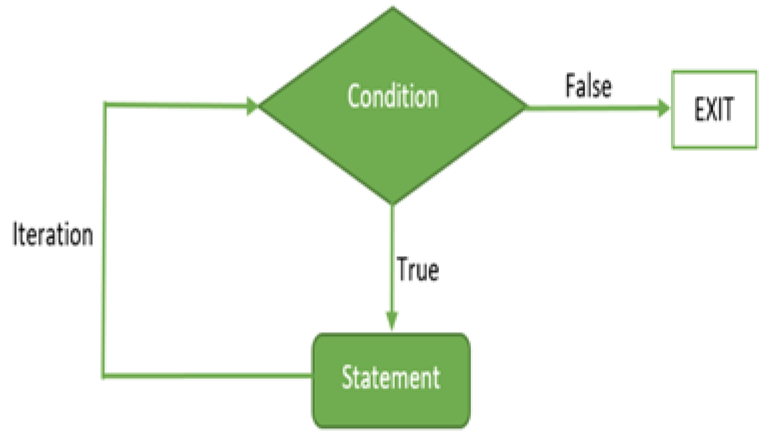
Control statement that allows multiple executions of a statement or a set of statements.

while loop

- Runs a statement or a set of statements repeatedly unless the given condition becomes false.
- Entry controlled loop.

Syntex

while (condition)
{ statement }



https://www.geeksforgeeks.org/loops-in-r-for-while-repeat/

Loops in R (Cycling or iterating)

Control statement that allows multiple executions of a statement or a set of statements.

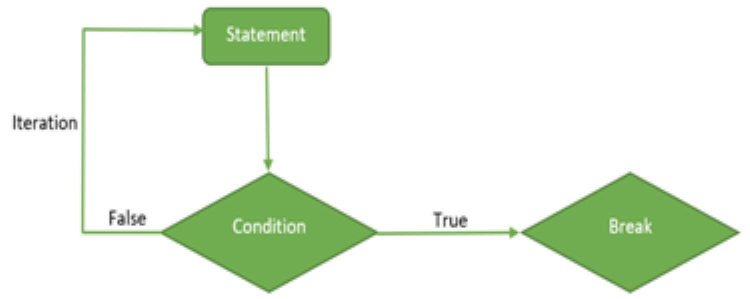
repeat loop

- run the same statement or a group of statements repeatedly until the stop condition has been encountered.
- Iterate infinitely if no condition given

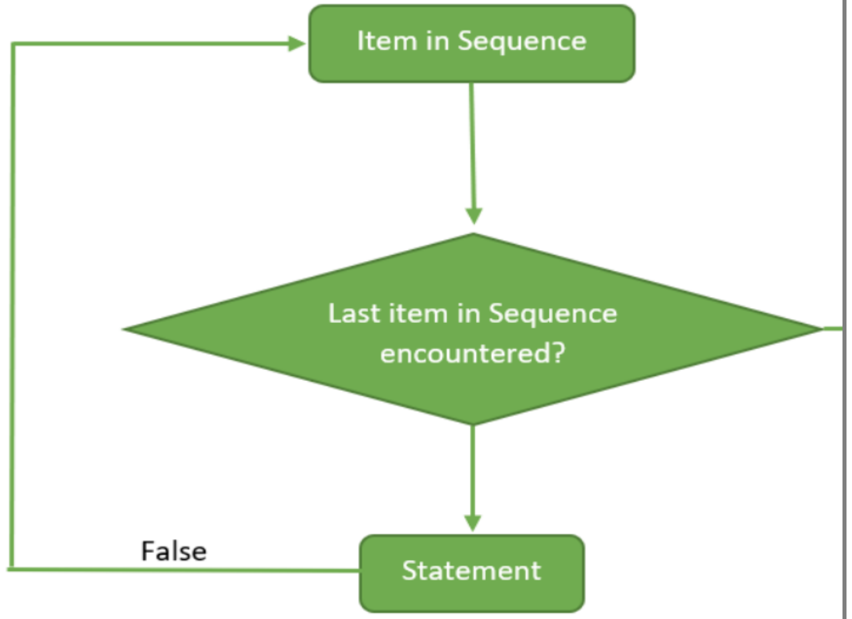
Syntex

```
repeat
{ statement
if( condition )
{ break }
}
```

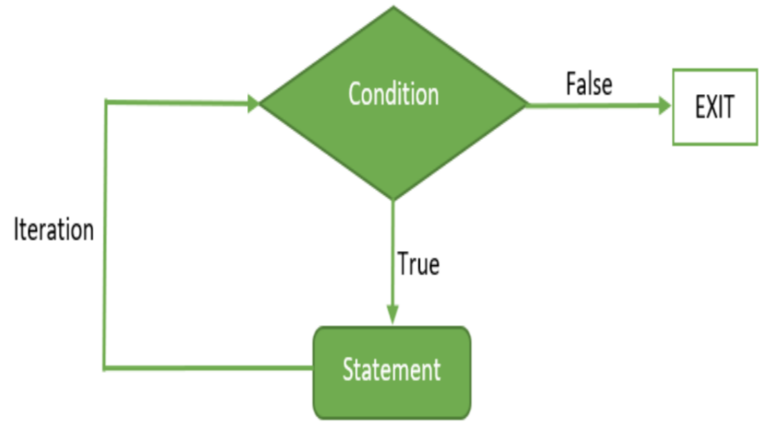
Flow Diagram:



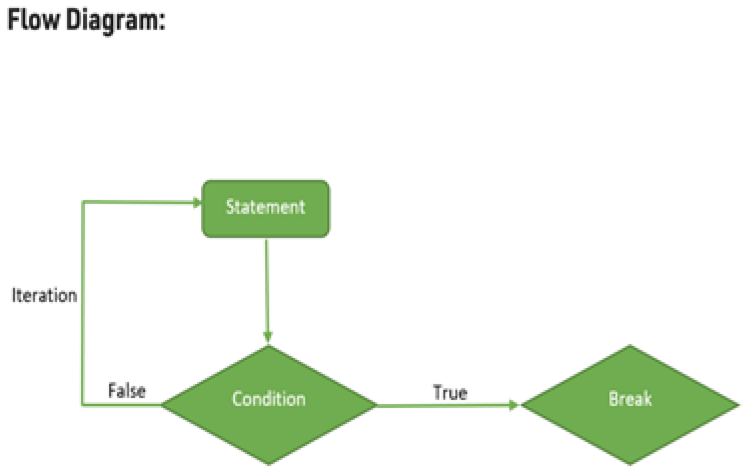
https://www.geeksforgeeks.org/loops-in-r-for-while-repeat/



For loop



while loop



repeat loop

String Manipulations in R

Basic String Manipulations

- *nchar()* number of characters
- *tolower()* convert to lower case
- *toupper()* convert to upper case
- *casefold()* case folding
- *chartr()* character translation
- abbreviate() abbreviation
- *substr()* substrings of a character vector

https://www.gastonsanchez.com/Handling_and_Processing_Strings_in_R.pdf

Set Operations

- ***union()*** set union
- ***intersect()*** intersection
- ***setdiff()*** set difference
- ***setequal()*** equal sets identical() exact equality
- ***is.element()*** is element
- ***%in%()*** contains
- ***sort()*** sorting
- ***rep()*** repetition

Other string functions

- ***paste()*** concatenates several characters
- *paste(..., sep = " ", collapse = NULL)*
- ***print()*** generic printing
- ***cat()*** concatenation

Your assignment what they does?

Working Directories

R is always pointed at a directory on your computer.

- **Check current directory**

  getwd()

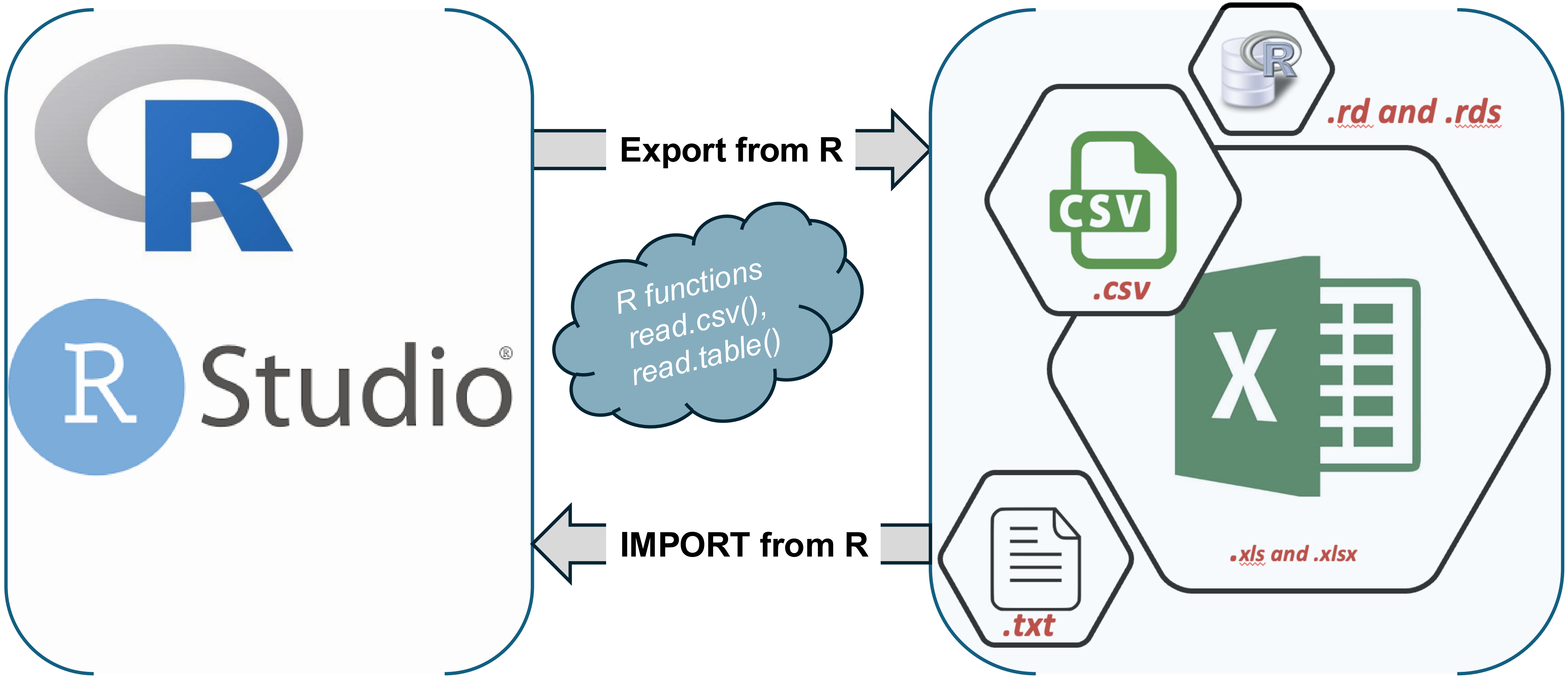
- **Set working directory**

  setwd()

- **Create working directory**

  dir.create()

Import and Export of Data in R



Importing Data from .csv and .txt files

Depending upon format, several variants are available

- **read.csv**(): for reading **“comma separated value”** files (“.csv”).
- ***read.table():*** for reading “text files” (“.txt”)

Syntax:

read.table(filename or path, header = FALSE, sep = “”)

Read.csv(filename or path, header=TRUE, sep=“”)

http://www.sthda.com/english/wiki/importing-data-into-r

Importing Data from excel file

Reading from Excel files

readxl package comes with the function **read_excel**() to read xls and xlsx files

```
my_data <- read_excel("my_file.xlsx", sheet = "data")
my_data <- read_excel("my_file.xlsx", sheet = 2)
my_data <- read_excel("my_file.xlsx", na = "---")
```

http://www.sthda.com/english/wiki/reading-data-from-excel-files-xls-xlsx-into-r

Exporting Data files

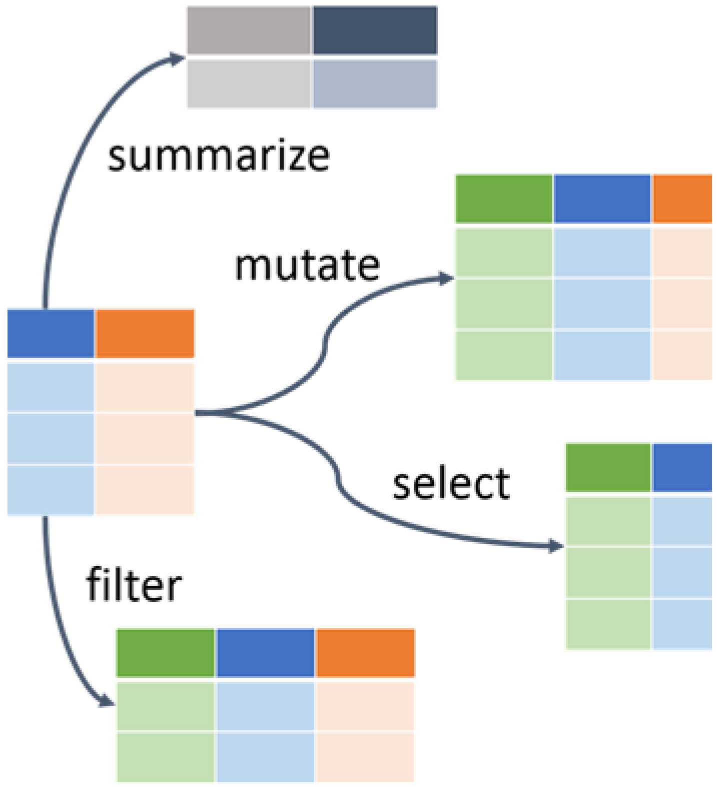
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE….)

write.csv(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE….)

http://www.sthda.com/english/wiki/exporting-data-from-r

Data Wrangling and Manipulations using R packages

dplyr R Package

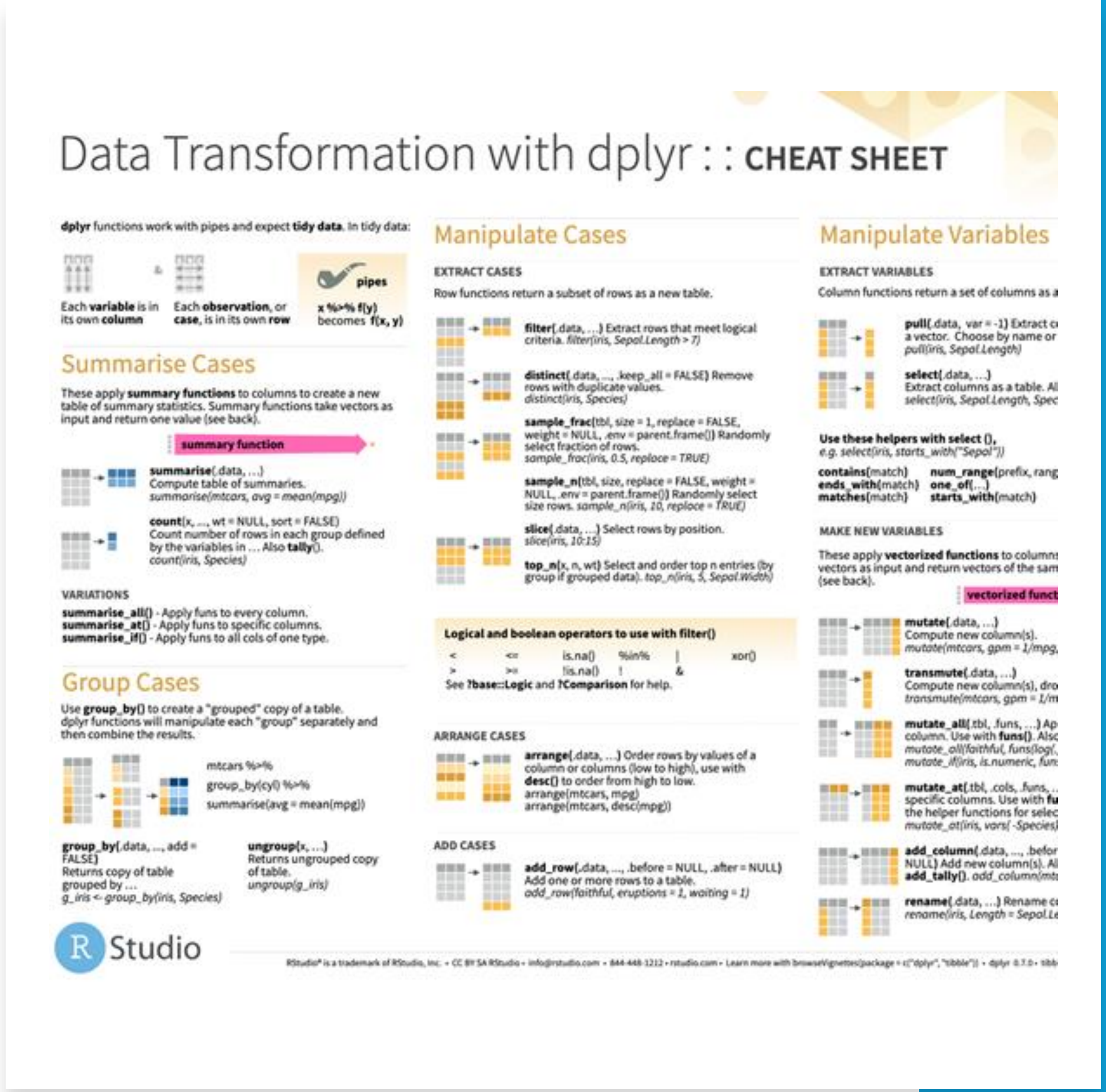


- ***filter()*** chooses rows based on column values.
- ***slice()*** chooses rows based on location.
- ***arrange()*** changes the order of the rows.
- ***select()*** changes whether or not a column is included.
- ***rename()*** changes the name of columns.
- ***mutate()*** changes the values of columns and creates new columns.
- ***relocate()*** changes the order of the columns.
- ***summarise()*** collapses a group into a single row.

Pipe the functions using %>%

https://rdrr.io/cran/dplyr/f/vignettes/dplyr.Rmd
https://www.r-bloggers.com/2019/04/how-to-filter-in-r-a-detailed-introduction-to-the-dplyr-filter-function/

dplyr cheet sheet



Data Transformation with dplyr : : **CHEAT SHEET**

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, …, wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, …, add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

RStudio

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, …, .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, …, .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a

pull(.data, var = -1) Extract c
a vector. Choose by name or
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Al
select(iris, Sepal.Length, Spec

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, rang
ends_with(match)	**one_of**(…)
matches(match)	**starts_with**(match)

MAKE NEW VARIABLES

These apply **vectorized functions** to column
vectors as input and return vectors of the sam
(see back).

vectorized funct

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg

transmute(.data, …)
Compute new column(s), dro
transmute(mtcars, gpm = 1/m

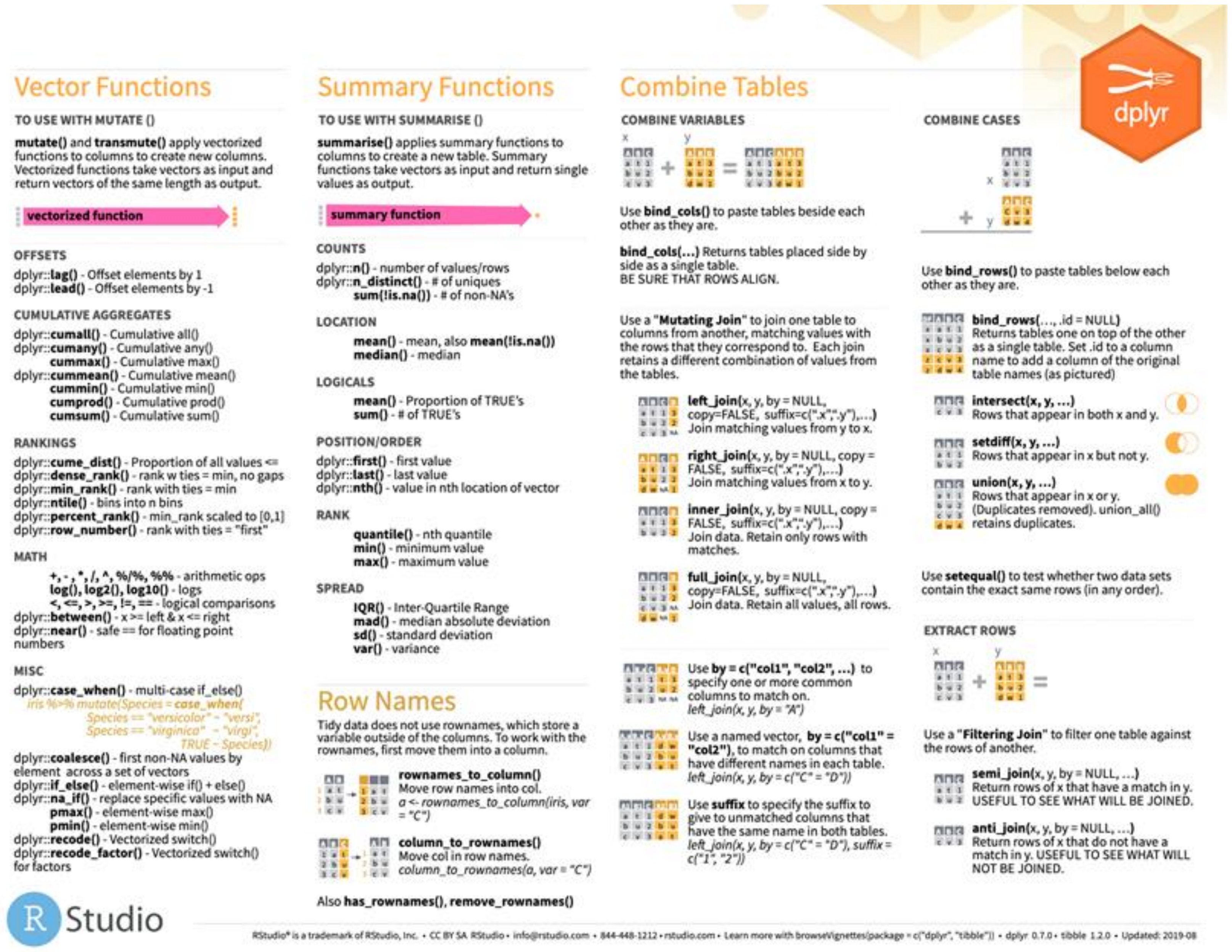
mutate_all(.tbl, .funs, …) Ap
column. Use with **funs()**. Als
mutate_all(faithful, funs(log(
mutate_if(iris, is.numeric, fun

mutate_at(.tbl, .cols, .funs, …
specific columns. Use with **fu**
the helper functions for selec
mutate_at(iris, vars(-Species)

add_column(.data, …, .befor
NULL) Add new column(s). Al
add_tally(). *add_column(mt*

rename(.data, …) Rename c
rename(iris, Length = Sepal.Le

dplyr chee sheet



Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()

```
iris %>% mutate(Species = case_when(
              Species == "versicolor" ~ "versi",
              Species == "virginica"  ~ "virgi",
                                TRUE ~ Species))
```

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL**)**
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

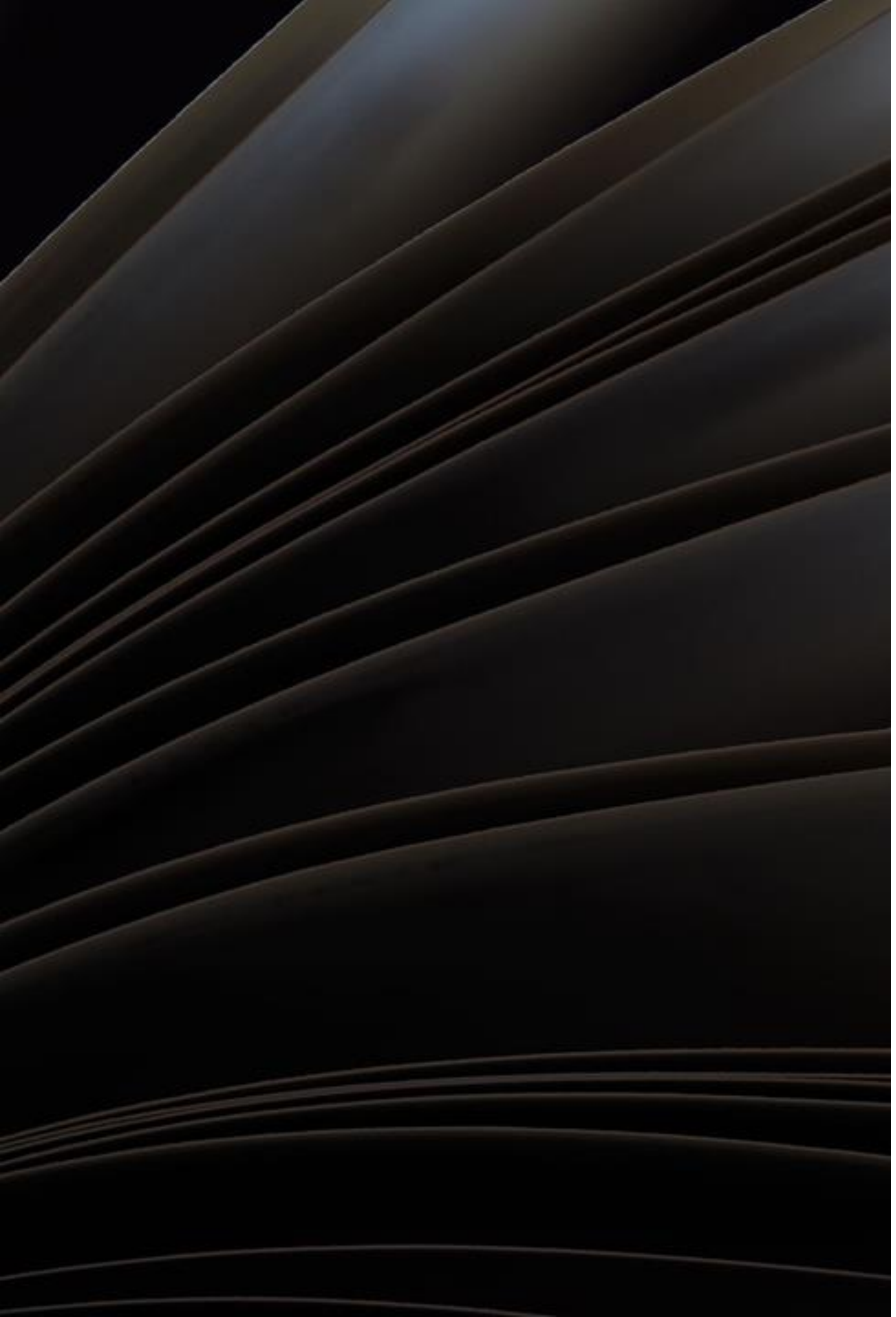
Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.



Reshape Package

- reshape2 is based around two key functions: melt and cast:
- melt takes wide-format data and melts it into long-format data.
- cast takes long-format data and casts it into wide-format data.
- Think of working with metal: if you melt metal, it drips and becomes long. If you cast it into a mould, it becomes wide.

https://uc-r.github.io/tidyr
https://ademos.people.uic.edu/Chapter9.html

Reshaping Data with Tidyr

tidyr is a one such package which was built for the sole purpose of simplifying the process of creating tidy data.

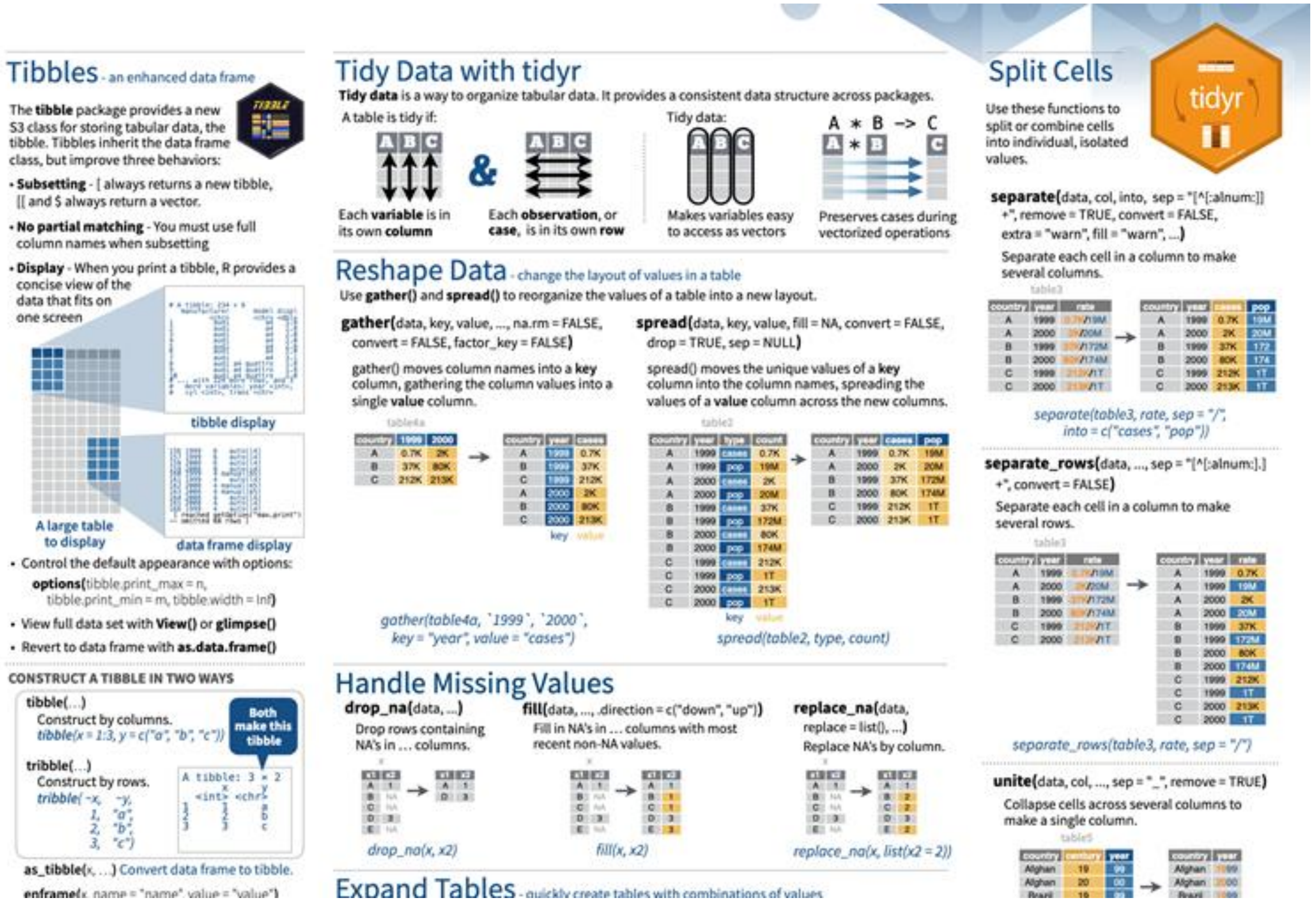
- **gather()** makes “wide” data longer
- **spread()** makes “long” data wider
- **separate()** splits a single column into multiple columns
- **unite()** combines multiple columns into a single column

Go through this PPT

https://rpubs.com/bradleyboehmke/data_processing

https://hbctraining.github.io/Intro-to-R/lessons/08_intro_tidyverse.html

tidyr cheet sheet



Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen

- Control the default appearance with options:
  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf)
- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(...)
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(...)
Construct by rows.
tribble(~x, ~y,
1, "a",
2, "b",
3, "c")

```
A tibble: 3 × 2
  x     y
<int> <chr>
1     a
2     b
3     c
```

as_tibble(x, ...) Convert data frame to tibble.

enframe(x, name = "name", value = "value")

Tidy Data with tidyr

Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

Tidy data:

Makes variables easy to access as vectors

A * B -> C

Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...)
Drop rows containing NA's in ... columns.
drop_na(x, x2)

fill(data, ..., .direction = c("down", "up"))
Fill in NA's in ... columns with most recent non-NA values.
fill(x, x2)

replace_na(data, replace = list(), ...)
Replace NA's by column.
replace_na(x, list(x2 = 2))

Expand Tables - quickly create tables with combinations of values

Split Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...)

Separate each cell in a column to make several columns.

separate(table3, rate, sep = "/", into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE)

Separate each cell in a column to make several rows.

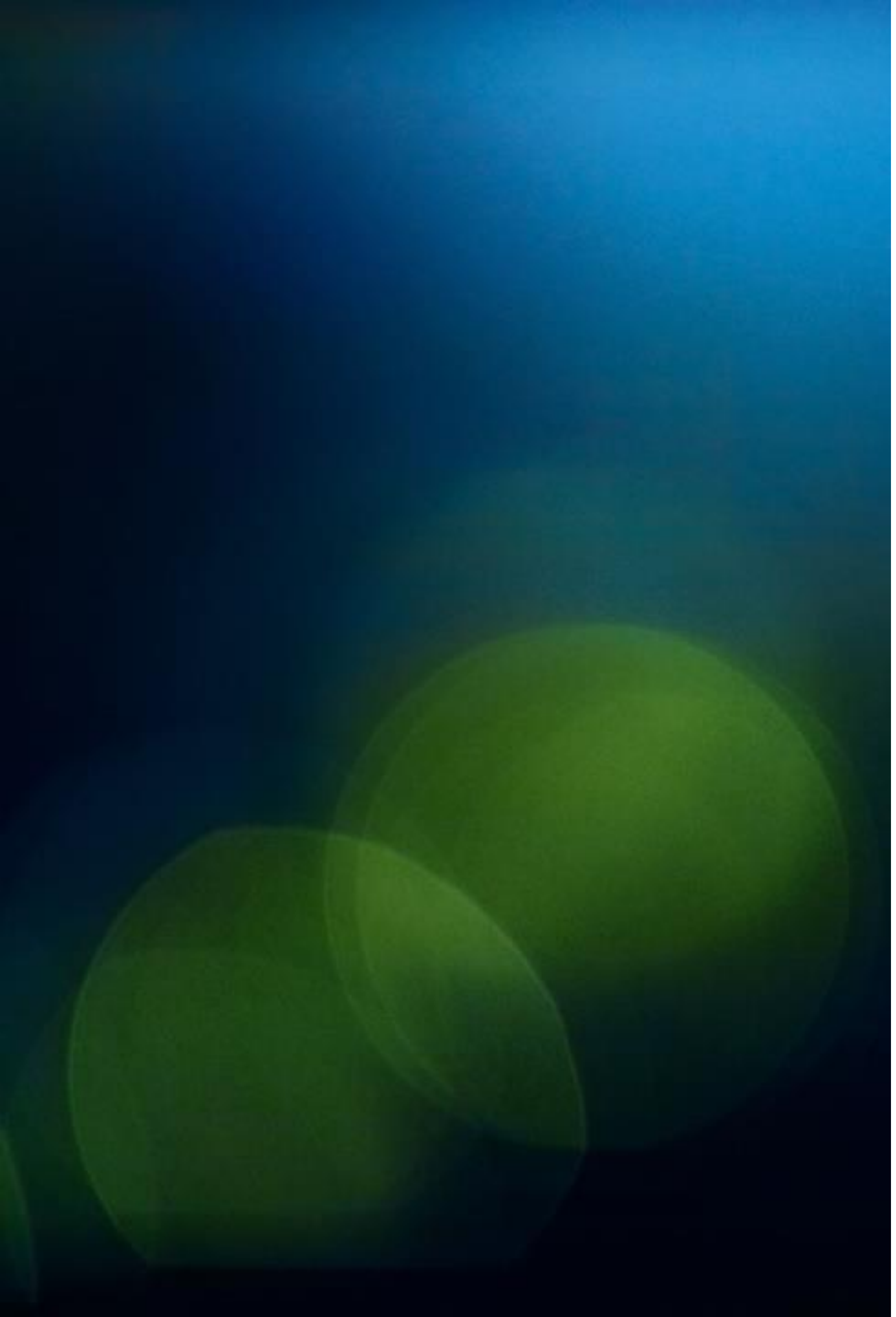
separate_rows(table3, rate, sep = "/")

unite(data, col, ..., sep = "_", remove = TRUE)

Collapse cells across several columns to make a single column.

Additional Useful Resources

- ❖ https://dplyr.tidyverse.org/
- ❖ https://uc-r.github.io/tidyr
- ❖ https://bookdown.org/mikemahoney218/IDEAR/data-wrangling.html
- ❖ https://exeter-data-analytics.github.io/AdVis/data-wrangling.html
- ❖ https://www.tidyverse.org/packages/
- ❖ https://atrebas.github.io/post/2019-03-03-datatable-dplyr/

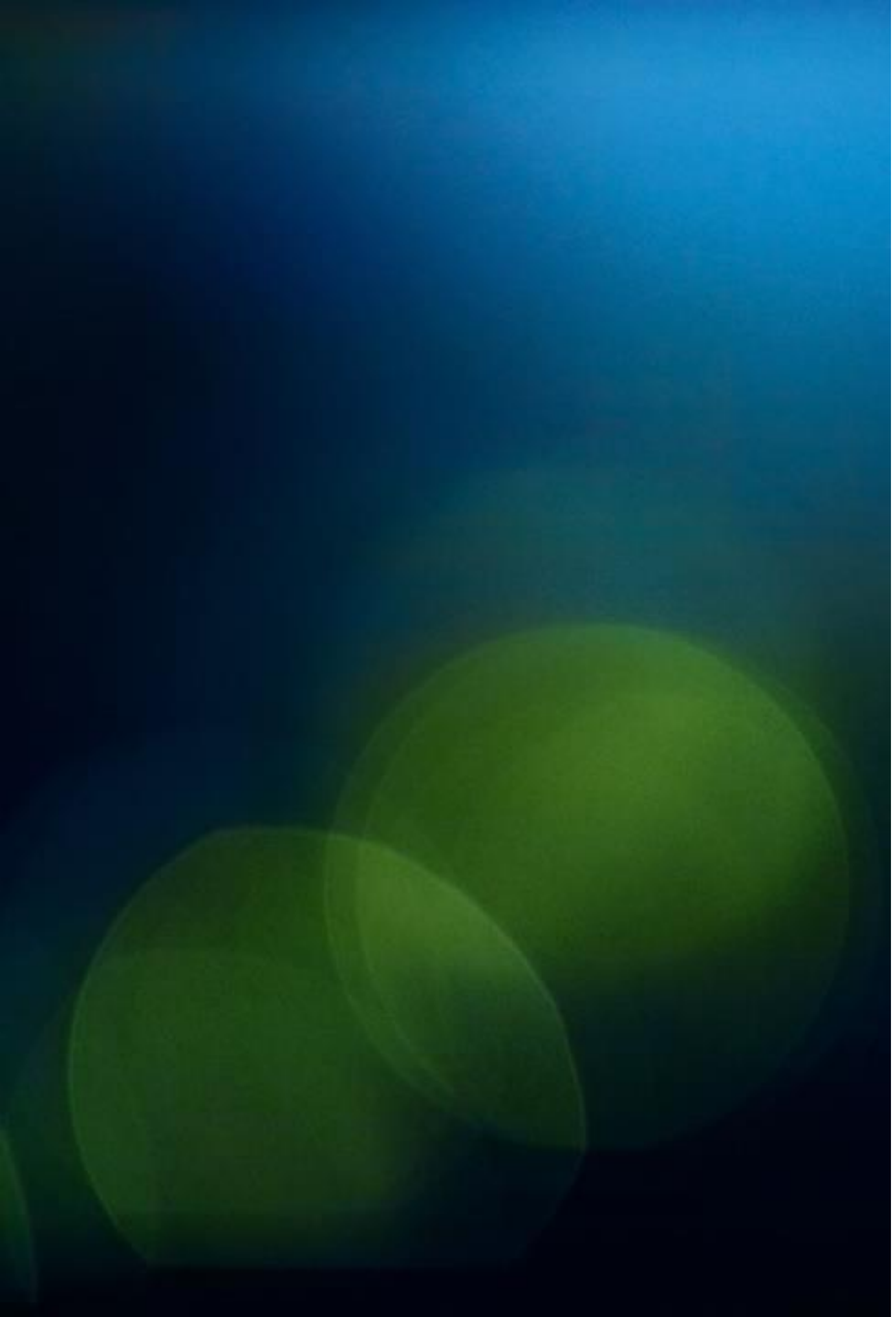


Regular Expressions
(Patter matching and substitution)

A regular expression (a.k.a. regex) is a special text string for describing a certain amount of text.

Regular expression is a pattern that describes a set of strings.

For example, searching word "programming" in a large text document.



Regular Expressions
(Patter matching and substitution)

grep: match a pattern

grepl: similar to grep, output as logical

regexpr: similar to grepl, output different and detailed

gregexpr: similar to regexpr, output as list

sub(): replacing one pattern with another one

gsub(): replacing one pattern with another one (all occurrences)

Writing R functions

Pieces of code that perform a desired operation on given input(s) and return the output back to the user.

Syntax

```
functionName <- function(argument1, argumen2…) {
    #function Body
    return(varc)
}
```

- **Function Name** − Name of the function.
- **Arguments** − An argument is a placeholder. When a function is invoked, you pass a value to the argument.
- **Function Body** − Collection of statements that defines what the function does.
- **Return Value** − The return value of a function is the last expression in the function body to be evaluated.

Demo in R

Generating Reproducible Reports and Interactive Visualizations in R

knitr

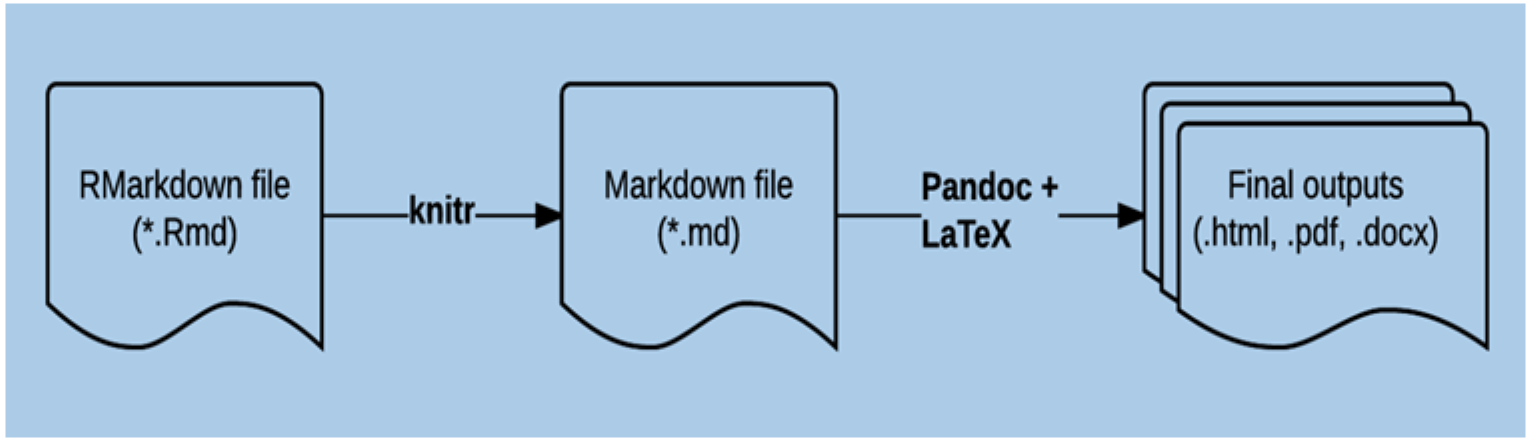
rmarkdown

plotly

IRRI 60

R Markdown

- Save the codes, execute them, and
- Generate high quality reproducible reports:
  - Edit any time
  - Seamless Visualization
  - Easy sharing.
- Knitr combines elements of R code and Markdown
- Convert the Analysis into Word, PDF, HTML etc.



https://rmarkdown.rstudio.com/index.html
https://rmarkdown.rstudio.com/articles_intro.html
https://rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

Workflow of Generating Reports

Open the File (.rmd)

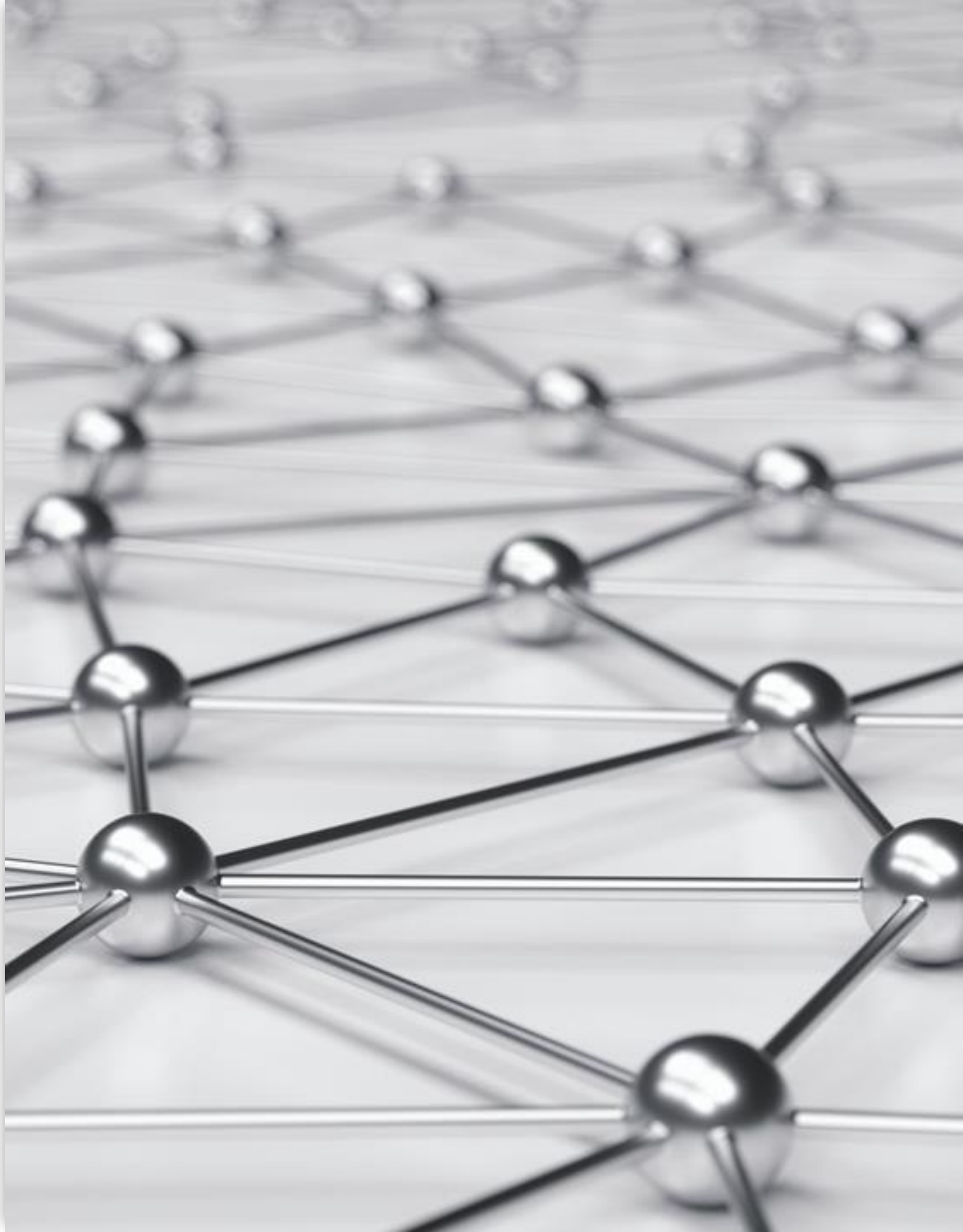
Write the Code

Embed the Code and Text

Render to generate the report

Useful resources to Learn R markdown

- https://bookdown.org/yihui/bookdown/figures.html
- https://bookdown.org/yihui/bookdown/r-code.html
- https://bioconnector.github.io/workshops/r-rmarkdown.html
- http://bioconnector.github.io/markdown/#!rmarkdown.md
- https://holtzy.github.io/Pimp-my-rmd/
- http://jianghao.wang/post/2017-12-08-rmarkdown-templates/



Demo in R