Basic R

Bioinformatic analysis for cancer genomics

Huyha
Email: hagiahuy311@gmail.com
2025

Why we use R?

Contents

1. Overview and install R

What is R?

- A Programming/Statistical Language
- A powerful language for statistical computing and data analysis.
- Widely used in bioinformatics and life sciences

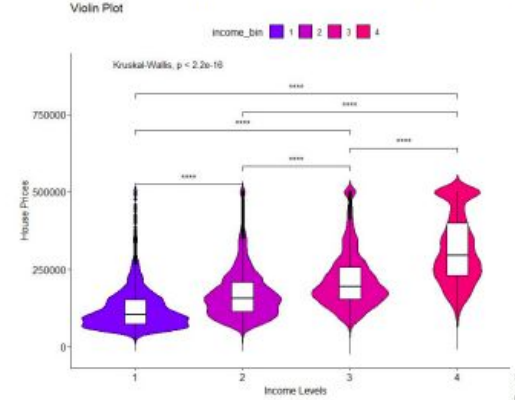
Why we use R?

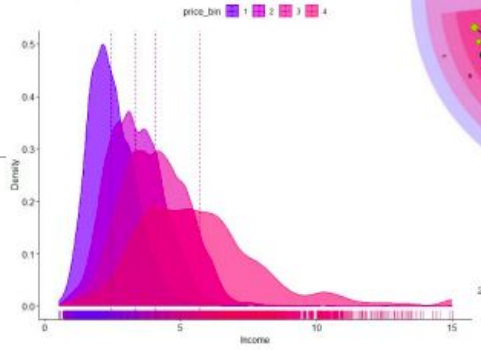
1. Comprehensive Statistical Tools
2. Rich Ecosystem of Bioinformatics Packages
3. Data Visualization
4. Handling High-Dimensional Data
5. Open Source and Active Community
6. Reproducibility
7. Free

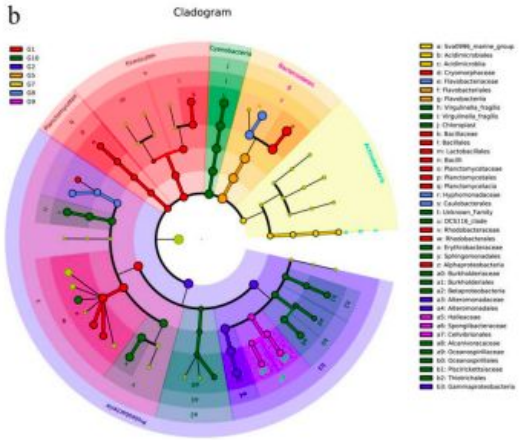
https://roelverbelen.netlify.app/resources/r/packages/

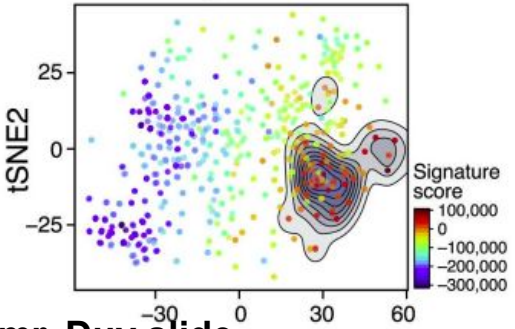
Why we use R?

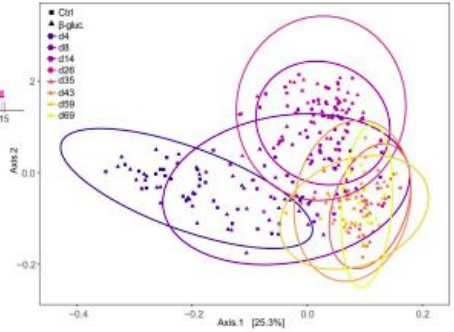
Generation of publication-quality graphs and figures.





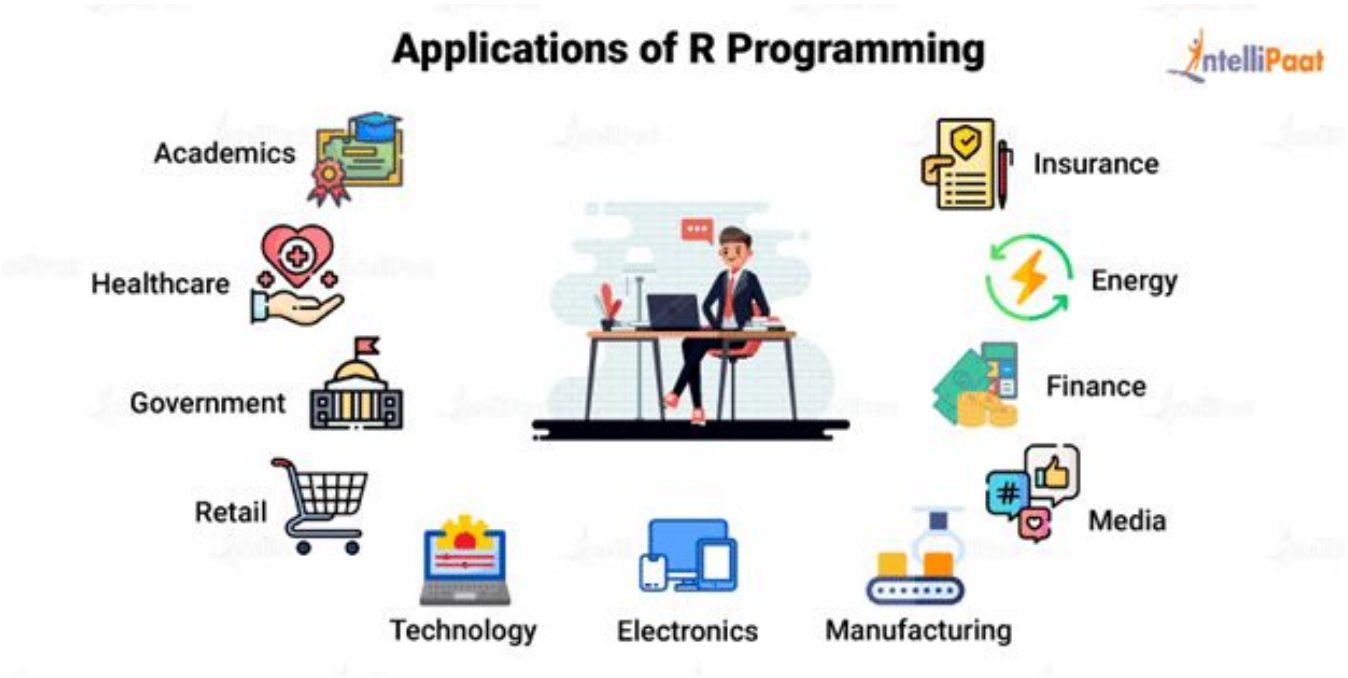






Adapted from mr. Duy slide

Why we use R?


Applications of R Programming
IntelliPaat
Academics
Healthcare
Government
Retail
Technology
Electronics
Manufacturing
Insurance
Energy
Finance
Media

Install R

CRAN
Mirrors
What's new?
Search
CRAN Team

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Task Views
Other

Documentation
Manuals
FAQs
Contributed

Donations
Donate

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2024-10-31, Pile of Leaves) R-4.4.2.tar.gz, read what's new in the latest version.
- The CRAN directory src/base-prerelease contains R alpha, beta, and rc releases as daily snapshots in time periods before a planned release.
- Between releases, the same directory src/base-prerelease contains snapshots of current patched and development versions.
  Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Alternatively, daily snapshots are available here.
- Source code of older versions of R is available here.
- Contributed extension packages.

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

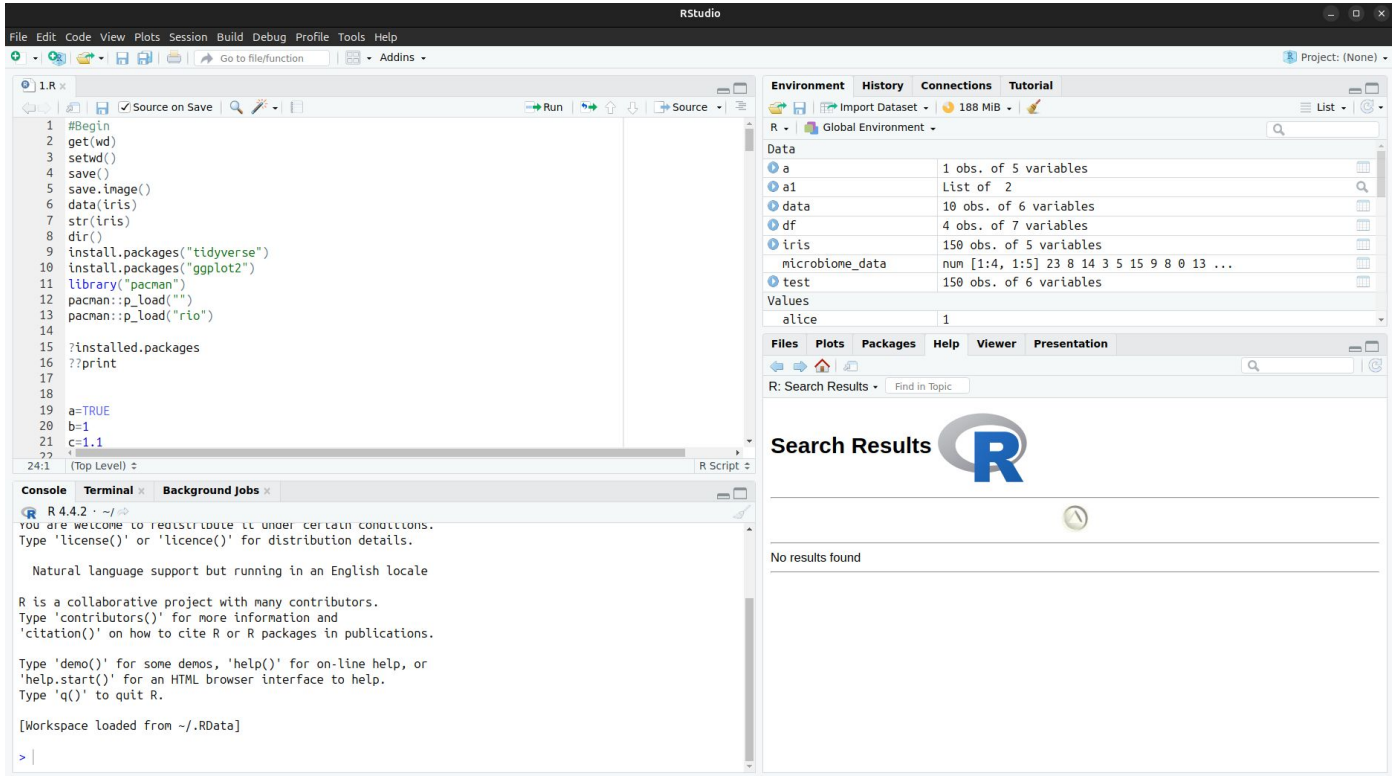
Supporting CRAN

- CRAN operations, most importantly hosting, checking, distributing, and archiving of R add-on packages for various platforms, crucially rely on technical, emotional, and financial support by the R community.

  Please consider making financial contributions to the R Foundation for Statistical Computing.

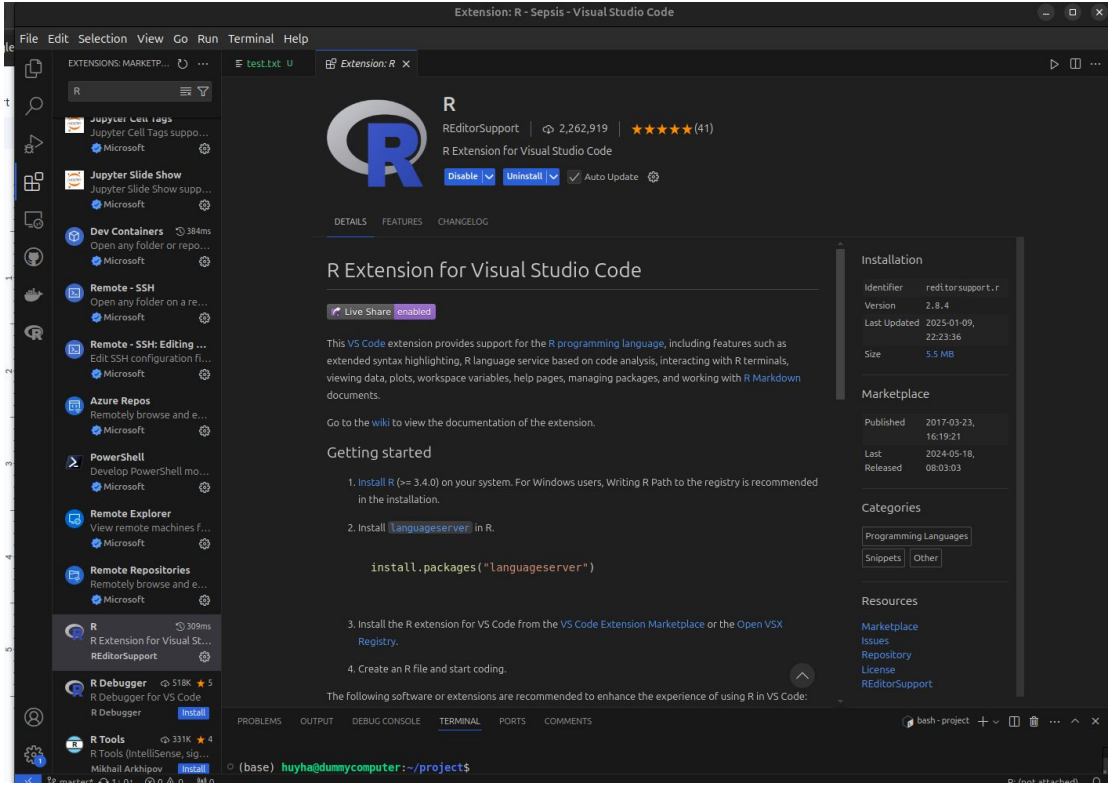
https://cran.rstudio.com/

R and Rstudio



https://posit.co/download/rstudio-desktop/

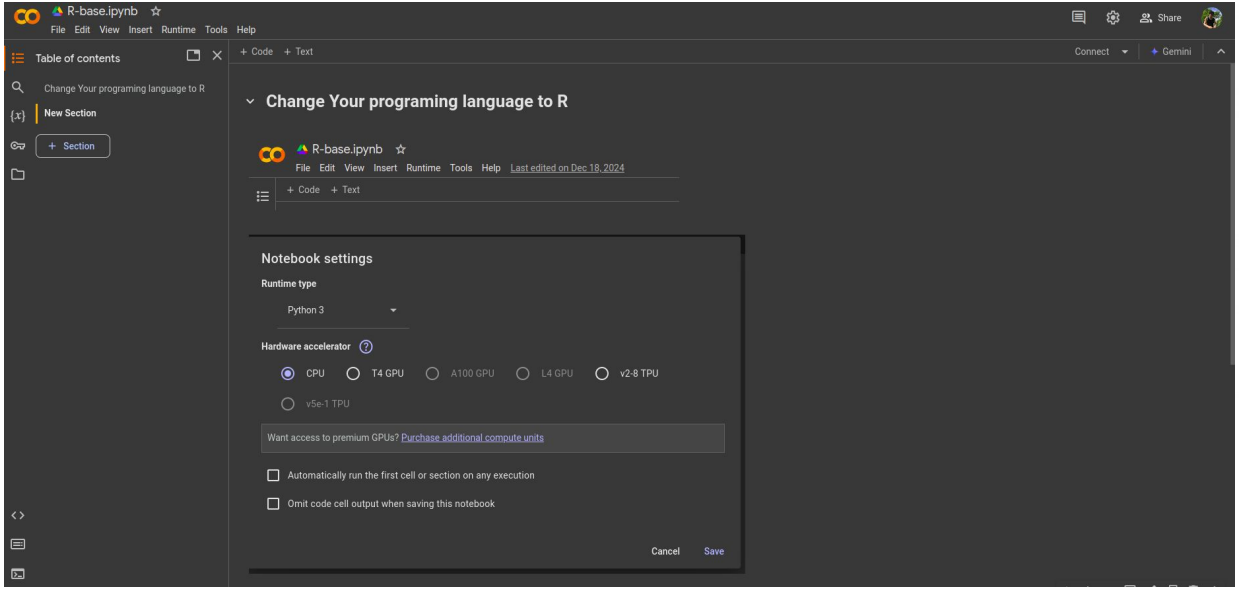
Orthers platforms for R



Visual code

https://code.visualstudio.com/download

Orthers platforms for R



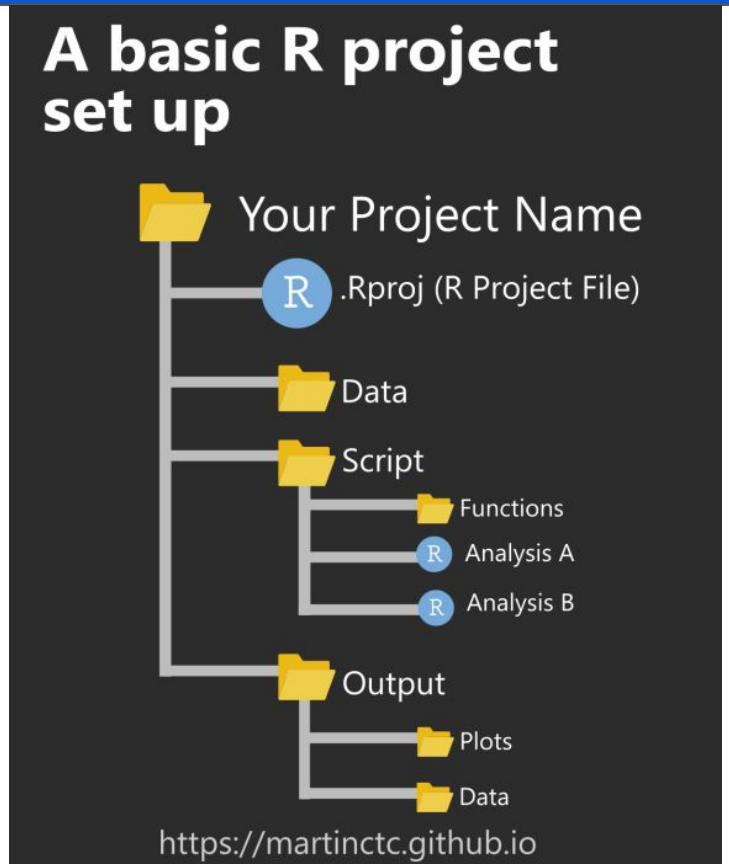
Google colab

https://colab.research.google.com

2. Basic R

Work directory

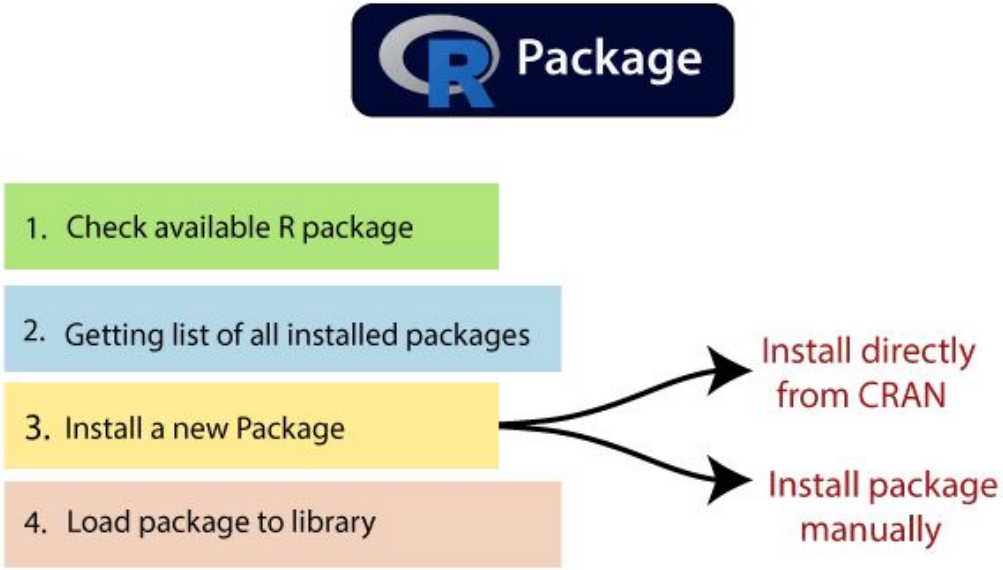
```
# First, have a look at the current working directory
getwd()

# Change to your desired directory
setwd()

# List the file in the directory
dir()
```



https://www.r-bloggers.com/2020/01/rstudio-projects-and-working-directories-a-beginners-guide/

Install and load package

```
# Get the list of installed packages
installed.packages()

# Install package
install.packages()

# Import package
library()

# get all packages currently loaded in the R environment.
search()

# Check installed packages location
libPaths()

# Update package
update.packages()
```



https://www.javatpoint.com/r-packages

Orther ways to install package

bioconductor.org/packages/release/bioc/html/DESeq2.html

Bioconductor

About Learn Packages Developers Search

Home > Bioconductor 3.20 > Software Packages > DESeq2

DESeq2

This is the **released** version of DESeq2; for the devel version, see DESeq2.

Differential gene expression analysis based on the negative binomial d

platforms all | rank 28 / 2289 | support 101 / 108 | in Bioc 12 years | build ok | updated before release | dependencies 75

DOI: 10.18129/B9.bioc.DESeq2

Bioconductor version: Release (3.20)

Estimate variance-mean dependence in count data from high-throughput sequencing assays and test for differential expression based on a model using t

Author: Michael Love [aut, cre], Constantin Ahlmann-Eltze [ctb], Kwame Forbes [ctb], Simon Anders [aut, ctb], Wolfgang Huber [aut, ctb], RADIANT EU FP7

Maintainer: Michael Love <michaelisaiahlove at gmail.com>

Citation (from within R, enter citation("DESeq2")):

Love MI, Huber W, Anders S (2014). "Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2." *Genome Biology*, **15**, 550. do

Installation

To install this package, start R (version "4.4") and enter:

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("DESeq2")
```

github.com/vegandevs/vegan

vegandevs / vegan

Code | Issues 38 | Pull requests 1 | Discussions | Actions | Projects | Wiki | Security | Insights

vegan Public

Watch 35 | Fork 97

master | 157 Branches | 37 Tags | Go to file | Add file | Code

jarioksa NEWS about not depending on lattice and cascading changes in per... b78d3a4 · 3 weeks ago 5,783 Commits

.github	remove rhub.yaml	3 months ago
R	add permulattice to use lattice graphics without library(l...	3 weeks ago
data	add data set on geographical coordinates and sizes of Si...	9 years ago
inst	Delete inst/NEWS.md	9 months ago
man	Merge branch 'startup-message'	3 weeks ago
src	getF.c: use vector dummy also in qrXw	2 months ago
tests	add permulattice to use lattice graphics without library(l...	3 weeks ago
vignettes	vignettes became aware of dbrda	10 months ago
.Rbuildignore	Update gh action	9 months ago
.gitattributes	add appveyor CI support	11 years ago
.gitignore	ignore apple junk	6 months ago
DESCRIPTION	lattice: do not depend on, but import from lattice	last month
LICENSE	Add a LICENSE file with the GPL-2 license text	3 years ago
NAMESPACE	add permulattice to use lattice graphics without library(l...	3 weeks ago
NEWS.md	NEWS about not depending on lattice and cascading cha...	3 weeks ago

About

R package for popular ordin null models &

vegandevs.

r ecology ecological-mode

Readme

GPL-2.0 lice

Activity

Custom pro

470 stars

35 watching

97 forks

Report repositor

Releases 23

CRAN Relea 2 weeks ago

+ 22 releases

Packages

No packages publi

Bioconductor **Github**

Install and load package

Search and download these packages:

- tidyverse
- readr
- ggplot2

Help and manual

```
# Access the help file
?mean

# If unsure of the precise name
# search doc across all installed
packages
??mean
```

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

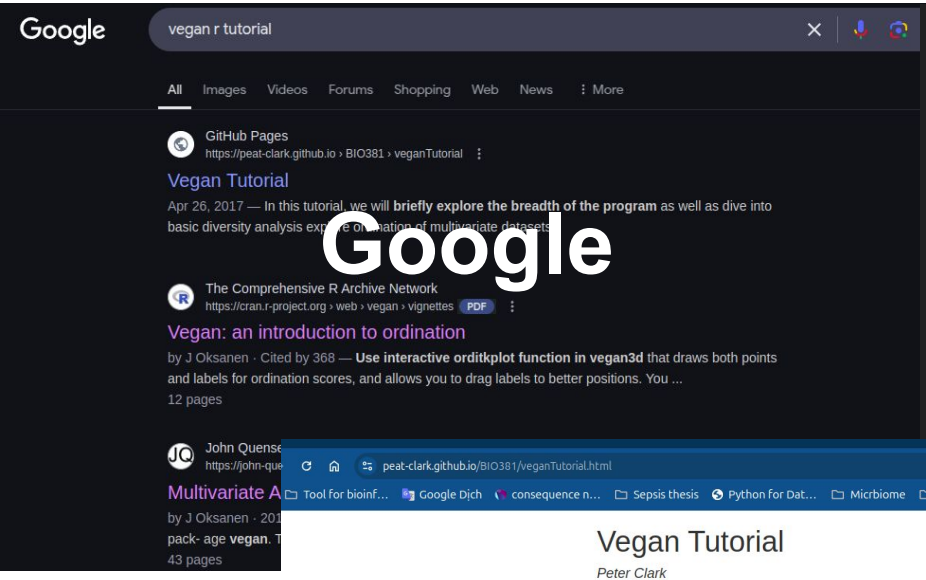
Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

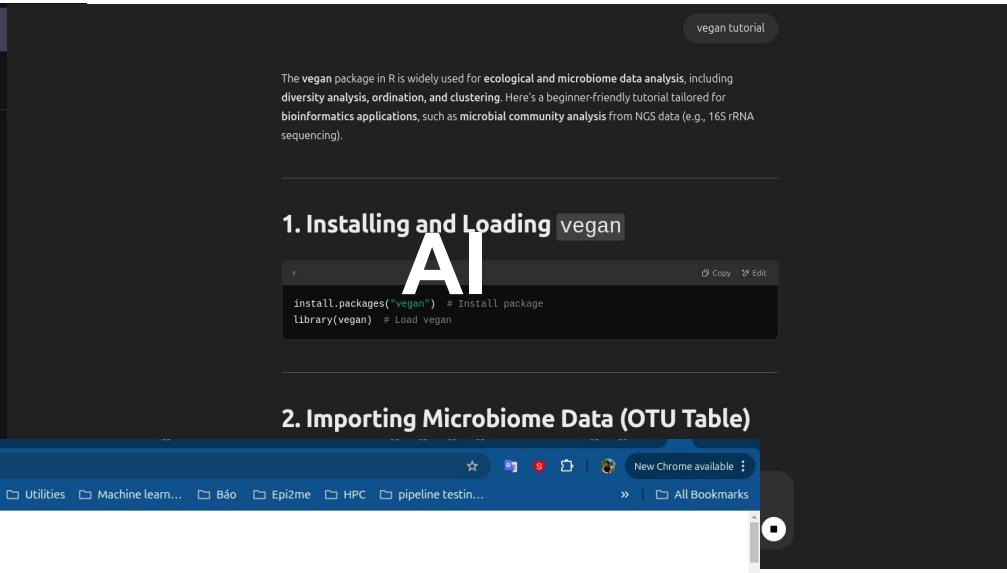
Arguments

x	an **R** object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.

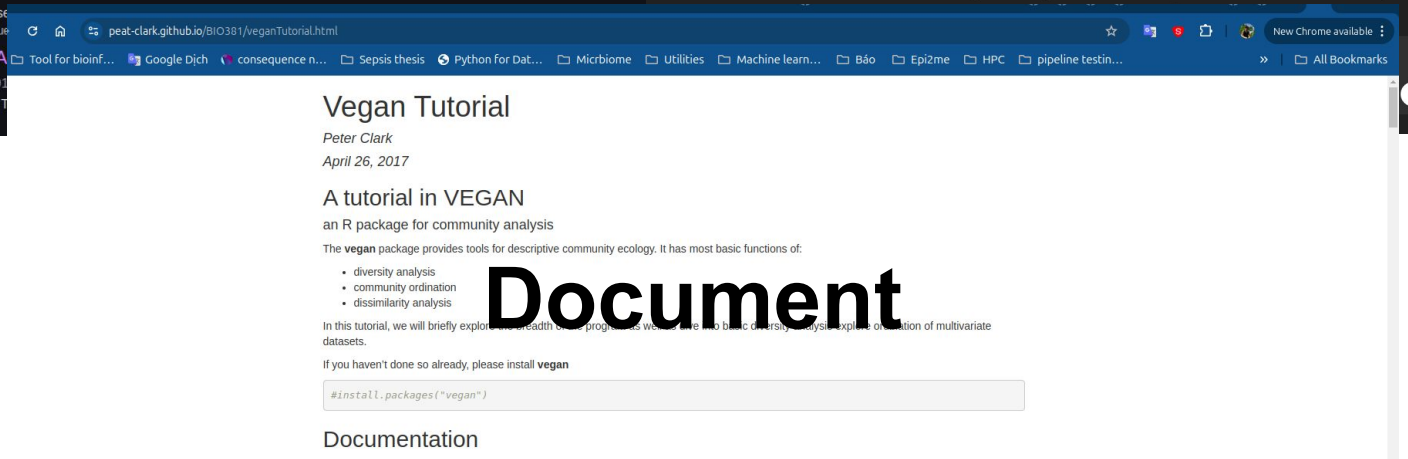
Package tutorial



Google

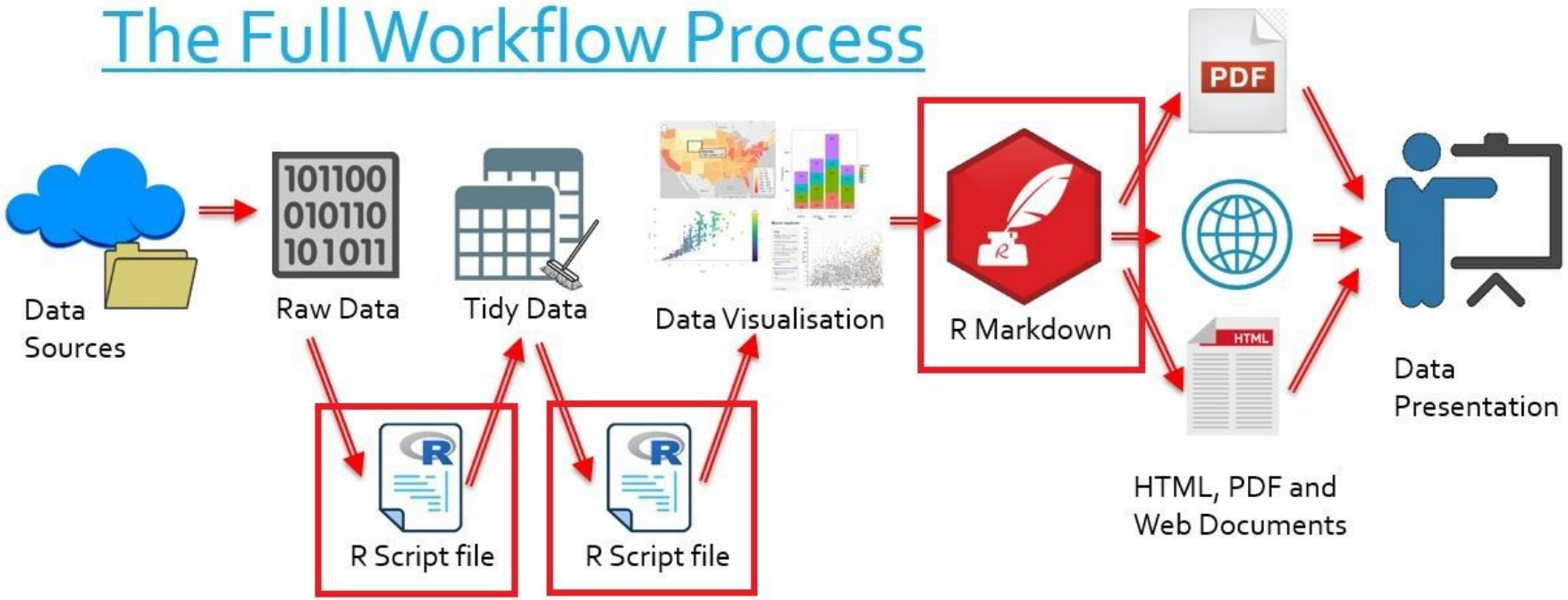


AI

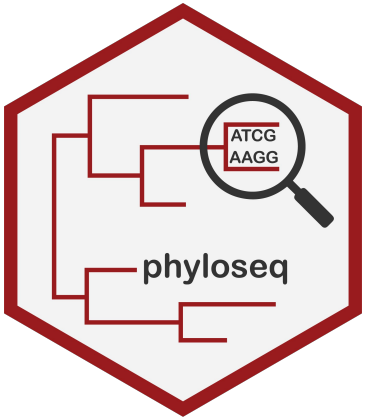


Document

Workflow in R tutorial



Workflow in R tutorial



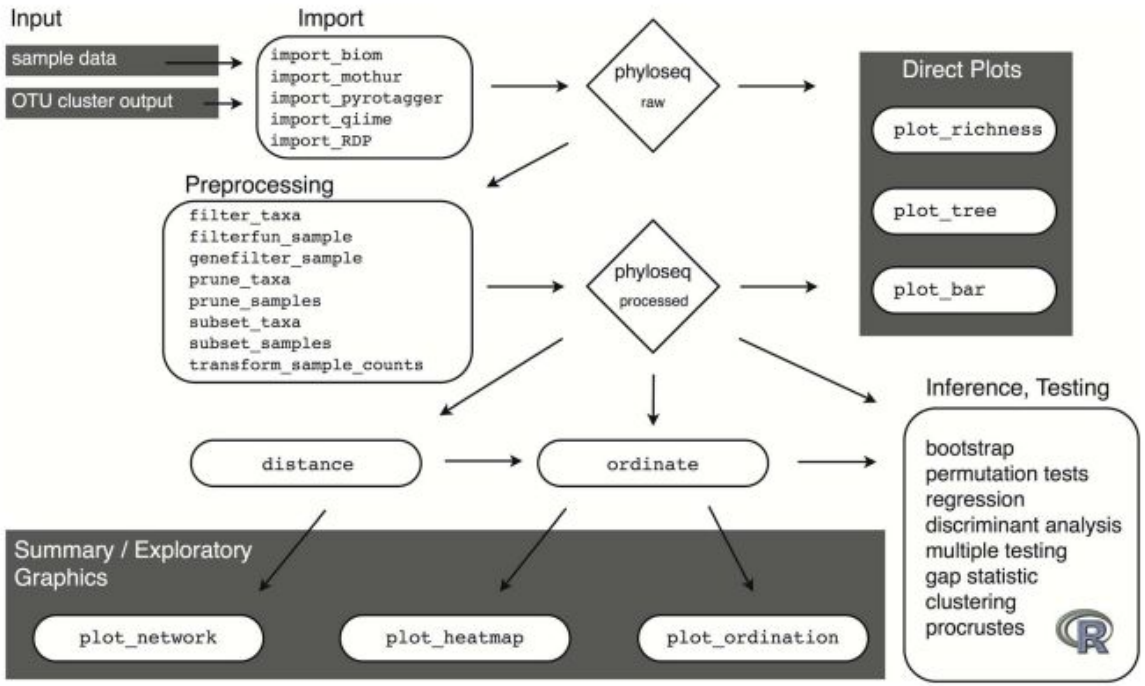


Figure 2. Analysis workflow using phyloseq. The workflow starts with the results of OTU clustering and independently-measured sample data (Input, top left), and ends at various analytic procedures available in R for inference and validation. In between are key functions for preprocessing and graphics. Rounded rectangles and diamond shapes represent functions and data objects, respectively, further described in Figure 3.
doi:10.1371/journal.pone.0061217.g002

Loading and Saving CSV Files in R

```
#Standard use, small files (base R)

#Load a CSV file

data <- read.csv("gene_expression.csv", header =
TRUE) head(data) # View first few rows

#Save a CSV file

write.csv(data, "output.csv", row.names = FALSE)
```

`header = TRUE`: Treats the first row as column names.
`sep = ","`: (Default) Assumes comma-separated values.

Loading and Saving CSV Files in R

```
#Using read.table() (More Control)

#Load CSV with custom delimiter

data <- read.table("gene_expression.csv", sep = ",",
header = TRUE)

#Save CSV with write.table()

write.table(data, "output.csv", sep = ",", row.names
= FALSE, quote = FALSE)
```

`header = TRUE`: Treats the first row as column names.
`sep = ","`: (Default) Assumes comma-separated values.

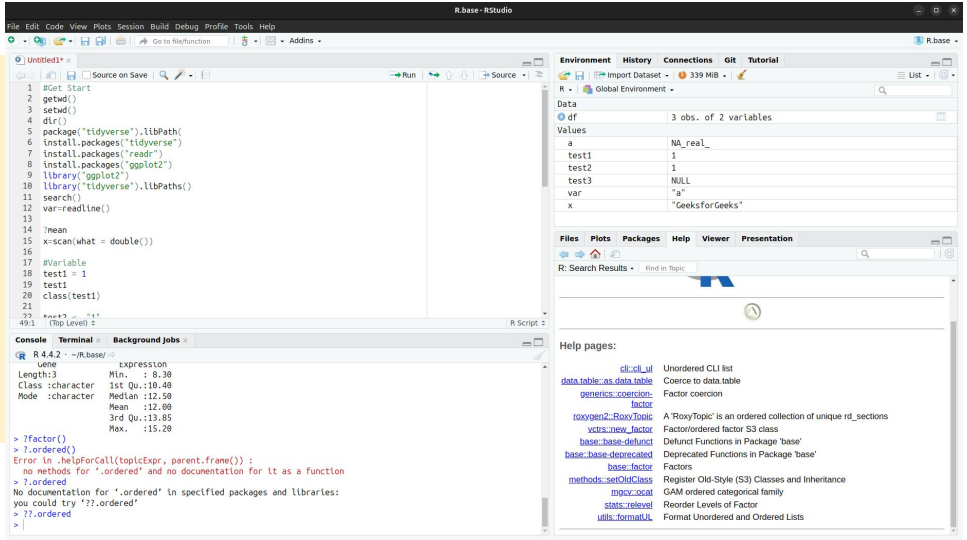
Loading and Saving CSV Files in R

```
#Standard use, small files

read.csv() / write.csv()

#Custom delimiters (e.g.,
tab-separated)

read.table() / write.table()

#Tidyverse compatibility, easy use

read_csv() / write_csv()
```

Save and quit

An R workspace image contains all the information held in the R session at the time of exit and is saved as a .RData file

```
# Save current workspace
save.image(file="mysession.RData")
# exit R
q()
# Load workspace
load('myession.RData')
```



R-base overview

Next Article:
R Programming Language - Introduction

R Tutorial | Learn R Programming Language

Last Updated : 03 Dec, 2024

R is an **interpreted** programming language widely used for statistical computing, data analysis and visualization. R language is open-source with large community support. R provides structured approach to data manipulation, along with decent libraries and packages like Dplyr, Ggplot2, shiny, Janitor and more.

Hello World Program in R Language

Here is an example of the first Hello World program in R Programming Language. To print in R language you just need to use a Print function.

```
# Code
print("Hello World!")
```

Output

```
Hello World!
```

R-Basics

- Introduction to R Programming Language
- Interesting Facts about R
- R vs Python
- How to Install R Studio on Windows and Linux?
- Creation and Execution of R File in R Studio
- Hello World in R Programming

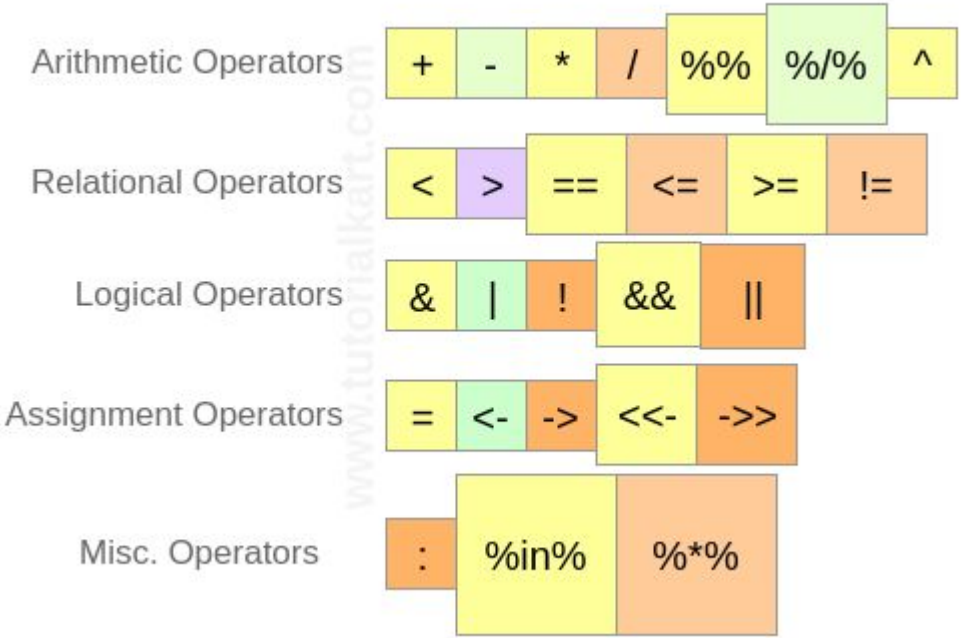
Fundamentals of R

- Basic Syntax
- Comments
- Operators

https://www.geeksforgeeks.org/r-tutorial/

3. Operators

Operator



https://www.tutorialkart.com/r-tutori al/r-operators/#gsc.tab=0

Arithmetic Operator

+	-	*	/	%%	%/%	^

```
# R Arithmetic Operators Example for integers

a <- 7.5
b <- 2
print ( a+b )      #1 Addition

print ( a-b )      #2 Subtraction

print ( a*b )      #3 Multiplication

print ( a/b )      #4 Division

print ( a%%b )  #5 Reminder

print ( a%/%b ) #6 Quotient

print ( a^b )      #7 Power of
```

```
$ Rscript r_op_arithmetic.R
[1] 9.5

[2] 5.5

[3] 15

[4] 3.75

[5] 1.5

[6] 3

[7] 56.25
```

Arithmetic Operator

+	-	*	/	%%	%/%	^

```
# R Operators - R Arithmetic Operators Example for vectors

a <- c(8, 9, 6)
b <- c(2, 4, 5)

print ( a+b )#1 addition

print ( a-b )#2 subtraction

print ( a*b )#3 multiplication

print ( a/b ) #4 Division

print ( a%%b )#5 Reminder

print ( a%/%b )#6 Quotient

print ( a^b )#7 Power of
```

```
$ Rscript r_op_arithmetic.R

[1] 10 13 11

[2] 6 5 1

[3] 16 36 30

[4] 4.00 2.25 1.20

[5] 0 1 1

[6] 4 2 1

[7]   64 6561 7776
```

Arithmetic Operator

+	-	*	/	%%	%/%	^

Arithmetic Operators

$$10^2 + \frac{3 \times 60}{8} - 3$$

```
R> 10^2+3*60/8-3
[1] 119.5
```

$$\frac{5^3 \times (6-2)}{61-3+4}$$

```
R> 5^3*(6-2)/(61-3+4)
[1] 8.064516
```

$$2^{2+1} - 4 + 64^{-2^{2.25-\frac{1}{4}}}$$

```
R> 2^(2+1)-4+64^((-2)^(2.25-1/4))
[1] 16777220
```

$$\left(\frac{0.44 \times (1-0.44)}{34}\right)^{\frac{1}{2}}$$

```
R> (0.44*(1-0.44)/34)^(1/2)
[1] 0.08512966
```

Performing calculation in R

Arithmetic Operator

+	-	*	/	%%	%/%	^

Classwork

a. Using R, verify that

$$\frac{6a + 42}{3^{4.2-3.62}} = 29.50556$$

when $a = 2.3$.

Relational Operator

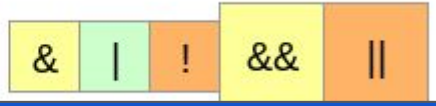

```
# R Operators - R Relational Operators Example for Numbers

a <- 7.5
b <- 2

print ( a>b )      #1 greater than

print ( a<b )      #2 less than

print ( a==b )     #3 equal to

print ( a<=b )     #4 less than or equal to

print ( a>=b )     #5 greater than or equal to

print ( a!=b )     #6 not equal to
```

Relational Operator


```
# R Operators - R Relational Operators Example for Numbers

a <- 7.5
b <- 2

print ( a>b )      #1 greater than

print ( a<b )      #2 less than

print ( a==b )     #3 equal to

print ( a<=b )     #4 less than or equal to

print ( a>=b )     #5 greater than or equal to

print ( a!=b )     #6 not equal to
```

```
$ Rscript r_op_relational.R
[1] TRUE

[2] FALSE

[3] FALSE

[4] FALSE

[5] TRUE

[6] TRUE
```

Logical Operator




```
# R Operators - R Logical Operators Example for basic logical elements

a <- 0 (TRUE)
b <- 2 (FALES)

print ( a & b ) #1 logical AND element wise

print ( a | b ) #2 logical OR element wise

print ( !a ) #3 logical NOT element wise

print ( a && b ) #4 logical AND consolidated for all elements

print ( a || b ) #5 logical OR consolidated for all elements
```

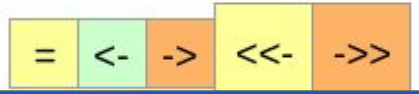
Logical Operator

&	|	!	&&	||

```
# R Operators - R Logical Operators Example for basic logical elements

a <- 0 (TRUE)
b <- 2 (FALES)

print ( a & b ) #1 logical AND element wise

print ( a | b ) #2 logical OR element wise

print ( !a ) #3 logical NOT element wise

print ( a && b ) #4 logical AND consolidated for all elements

print ( a || b ) #5 logical OR consolidated for all elements
```

```
$ Rscript r_op_logical.R
[1] FALSE

[2] TRUE

[3] TRUE

[4] FALSE

[5] TRUE
```

Assignment Operator



R Variable can be assigned a value using one of the following three operators :

1. Equal Operator =
2. Leftward Operator <-
3. Rightward Operator ->

```
#Assign variable
 x = 'hello'
print(x)
[1] "hello"
x <- 'learn r'
print(x)
[1] "learn r"
'r programming language' -> x;  print(x)
[1] "r programming language"
```

Miscellaneous Operator

Operator	Description	Usage
:	Creates series of numbers from left operand to right operand	a:b
%in%	Identifies if an element(a) belongs to a vector(b)	a %in% b
%*%	Performs multiplication of a vector with	

```
[1] 23 24 25 26 27 28 29 30 31
```

```
a = c(25, 27, 76)
b = 27
print ( b %in% a )
```

```
[1] TRUE
```

Miscellaneous Operator

;	:	%in%	%*%

```
mat = matrix(c(1,2,3,4,5,6),nrow=2,ncol=3)
    print (mat)
    print( t(mat))
    pro = mat %*% t(mat)
    print(pro)
```

```
Output :[,1] [,2] [,3]
#original matrix of order 2x3
        [1,]    1    3    5
        [2,]    2    4    6
             [,1] [,2]
#transposed matrix of order 3x2
        [1,]    1    2
        [2,]    3    4
        [3,]    5    6
             [,1] [,2]
#product matrix of order 2x2
        [1,]   35   44
        [2,]   44   56
```

Special Value(Inf, NaN, NA, NULL)

Special Value	Meaning	Example Usage
Inf	Positive or negative infinity	1/0, -1/0
NaN	Undefined or unrepresentable value	0/0, sqrt(-1)
NA	Missing value	mean(c(1, NA), na.rm=TRUE)
NULL	No value or undefined value	list(a = 1, b = NULL)

Classwork

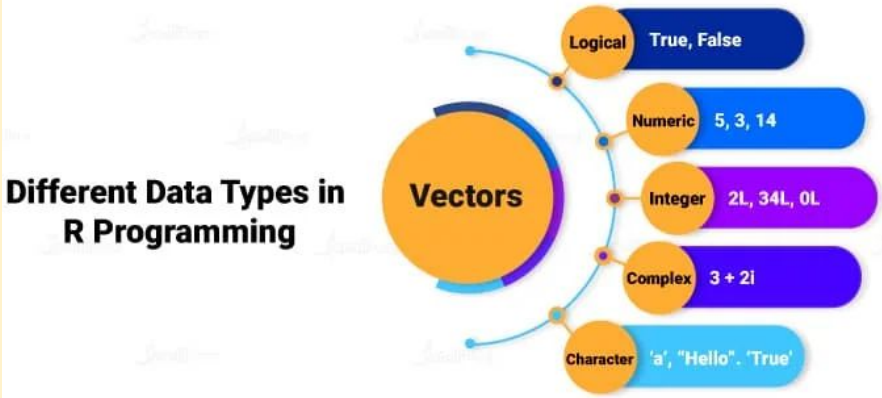
Classwork : Replicate all the operation codes above

3. Data Type and Data Structure

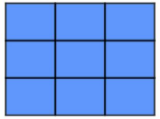
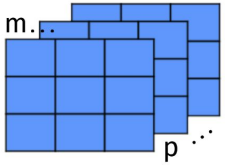
DataType

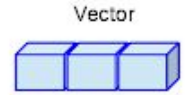
```
> x <- TRUE
> print(class(x))
[1] "logical"

> x <- 67.54
> print(class(x))
[1] "numeric"

x <- 63L
> print(class(x))
[1] "integer"
```

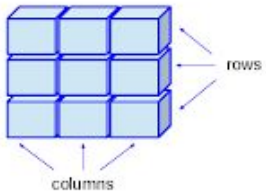
```
> x <- 6 + 4i
> print(class(x))
[1] "complex"

> x <- "hello"
> print(class(x))
[1] "character"

> x <- charToRaw("hello")
> print(class(x))
[1] "raw"
```

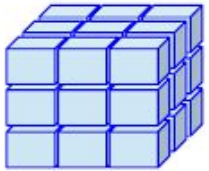


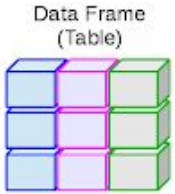
Data structure

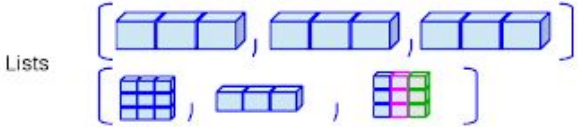
	Dimensions	Mode (data "type")	Example
Vector	1 m… 	Identical	`c(10,0.2,34,48,53)`
Matrix	n ⋮ m… 	Identical	`matrix(c(1,2,3, 11,12,13), nrow = 2, ncol = 3)`
Data frame	n ⋮ m… 	Can be different	`data.frame(x = 1:3, y = 5:7)`
Array	n ⋮ m…  p …	Identical	`array(data = 1:3, dim = c(2,4,2))`
List	{ Vector, Matrix, Data frame, Array }	Can be different	`list(x = cars[,1], y = cars[,2])`





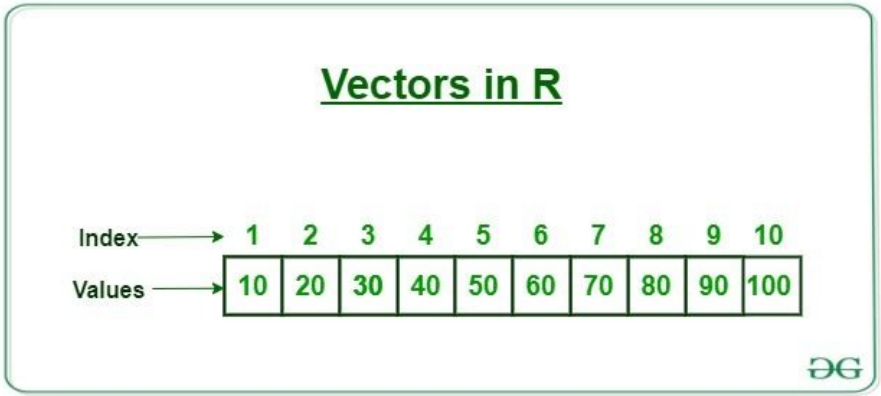






Vector and index

```
# we can use the c function to combine the values
as a vector.
# By default the type will be double
X<- c(61, 4, 21, 67, 89, 2)
X
[1] 61  4 21 67 89  2
# seq() function for creating
# a sequence of continuous values.
# length.out defines the length of vector.
Y<- seq(1, 10, length.out = 5)
Y
[1]  1.00  3.25  5.50  7.75 10.00
# use':' to create a vector
# of continuous values.
Z<- 2:7
Z
[1] 2 3 4 5 6 7
```




```
vector[1]
[1] 10

vector[c(1,3)]
[1] 10 30

Vector[7:10]
[1] 70 80 90 100
```

Operator in Vector

```
# Numeric vector
numbers <-c(1,2,3,4,5)

# Character vector
names<-c("Alice","Bob","Charlie")

# Addition
result <- numbers + 2
print(result)
[1] 3 4 5 6 7

# Multiplication
result <- numbers *2
print(result)
[1] 2 4 6 8 10
```

```
# Adding two vectors
vector1 <-c(1,2,3)
vector2 <-c(4,5,6)
result <- vector1 + vector2
print(result)
[1] 5 7 9
```

Vector

Accessing Elements:

```
number<-c(1,2,3,4,5,6,7,8)
# Access the second element
second_element <- numbers[2]
print(second_element)
[1] 2

number[3]<-5
number
[1] 1 2 -5 4 5 6 7 8

number<-number[-3]
number [1] 1 2 4 5 6 7 8
```

Logical Subset Vector

```
# Get elements greater than 3 (logical subset)
gt_than_3 <- numbers[numbers >3]
print(greater_than_three) [1] 4 5
```

Subset Vector

```
# Get a subset of the first three elements
subset_vector <- numbers[1:3]
print(subset_vector) [1] 1 2 3
```

Vector

Vector Naming

```
# Name the elements of the vector
names(numbers)<-c("First","Second","Third","Fourth","Fifth")
print(numbers)
[1] First Second Third Fourth Fifth
# 1 2 3 4 5
```

```
# Combine vectors
vector1<-c(1,2,3)
vector2<-c(4,5,6)
combined_vector <-c(vector1, vector2)
print(combined_vector)
[1] 1 2 3 4 5 6
```

```
# Get the type of the vector vector_type <-
typeof(numbers) print(vector_type)
[1] "double"
```

Vector factor

```
# Creating a factor from a character vector
colors <-c("red","green","blue","red","green")
color_factor <- factor(colors)
print(color_factor)

[1] red green blue red green
Levels: blue green red

# Specifying the order of levels
ordered_factor <- factor(colors, levels =c("red","green","blue"))

print(ordered_factor)
[1] red green blue red green
Levels: red green blue
```

Recycle rule

The Recycling Rule

→ How R handles operations between vectors of unequal lengths.
→ R will "recycle" the shorter vector by repeating its elements until it matches the length of the longer vector.

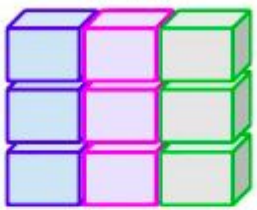
```
# Shorter vector is recycled to match the length of the longer vector short_vector

short_vector <-c(1,2)
long_vector <-c(10,20,30,40)
result <- long_vector + short_vector
print(result)

[1] 11 22 31 42
```

https://www.gastonsanchez.com/R-coding-basics/vectors4.html#recycling

Vector functions

```
# Sequences with seq()
> seq(from=3, to=27, by=3)
[1] 3 6 9 12 15 18 21 24 27

# Repetition with rep()
> rep(x=1,times=4) [1] 1 1 1 1
> rep(x=c(3,62,8.3),times=3)
[1] 3.0 62.0 8.3 3.0 62.0 8.3 3.0 62.0 8.3

# Sorting with sort()
> sort(x=c(2.5,-1,-10,3.44),decreasing=FALSE)
[1] -10.00 -1.00 2.50 3.44
> sort(x=c(2.5,-1,-10,3.44),decreasing=TRUE)
[1] 3.44 2.50 -1.00 -10.00

# Finding a Vector length with length()
> length(x=c(3,2,8,1))
[1] 4
```

Data frame



1	"S"	TRUE
7	"A"	FALSE
3	"U"	TRUE
numeric	character	logical

Definition:
A data frame is a table or a 2-dimensional array-like structure in R, where each column can contain different types of data (numeric, character, factor, etc.).

Structure:
Similar to a spreadsheet or SQL table, with rows representing observations and columns representing variables.

Creating Data frame

```
# Create a data frame with three columns
df <- data.frame(ID =1:4,
                 Name=c("Alice","Bob","Charlie","Diana"),
                 Score =c(85,92,88,76))
print(df)

  ID    Name    Score
1  1   Alice       85
2  2     Bob       92
3  3 Charlie       88
4  4   Diana       76
```

Accessing data in Data frame

Using $ to Access Columns:

```
# Access the 'Name'
columnnames<- df$Name
print(names)

[1] "Alice" "Bob" "Charlie" "Diana"
```

Using Indexing

```
# Access the element in the 2nd row, 3rd
column element <- df[2,3] print(element)

[1] 92
```

Data frame

Adding a New Column

```
# Add a new column 'Passed'

df$Passed <- df$Score >80 print(df)

  ID  Name    Score  Passed
1 1   Alice   85     TRUE
2 2   Bob     92     TRUE
3 3   Charlie 88     TRUE
4 4   Diana   76     FALSE
```

Subsetting Data Frames:

```
# Subsetting a dataframe with condition

high_scores <- df[df$Score >80,]
print(high_scores)

  ID  Name    Score  Passed
1 1   Alice   85     TRUE
2 2   Bob     92     TRUE
3 3   Charlie 88     TRUE
```

Data frame

Row Binding

```
# Combine data frames by adding rows
df_new <- data.frame(ID=5,
                     Name="Eve",
                     Score=90)
combined_df <- rbind(df, df_new)

print(combined_df)
  ID Name Score Passed
1  1 Alice    85   TRUE
2  2  Bob    92   TRUE
3  3 Charlie 88   TRUE
4  4 Diana    76  FALSE
5  5  Eve    90   TRUE
```

Column Binding

```
# Combine data frames by adding columns
Extra_info<-data.frame(Age=c(23,25,22,21,24))
full_df <- cbind(combined_df,extra_info)

print(full_df)
  ID Name Score Passed Age
1  1 Alice    85   TRUE  23
2  2  Bob    92   TRUE  25
3  3 Charlie 88   TRUE  22
4  4 Diana    76  FALSE  21
5  5  Eve    90   TRUE  24
```

Viewing and Inspecting Data Frames

```
# Viewing data
View(df)

# Explore the structure of the data
str(df)

'data.frame': 4 obs. of 4 variables:
$ ID : int 1 2 3 4
$ Name : chr "Alice" "Bob" "Charlie" "Diana"
$ Score : num 85 92 88 76 $ Passed: logi TRUE TRUE TRUE FALSE
```

Accessing data in Data frame

Summary Statistics

To get a summary of each column.

```
summary(df)

ID              Name              Score           Passed
Min. :1.00      Length:4          Min. :76.00     Mode :logical
1st Qu.:1.75    Class :character  1st Qu.:82.75   FALSE:1
Median :2.50    Mode :character   Median :86.50   TRUE :3
Mean :2.50                        Mean :85.25
3rd Qu.:3.25                      3rd Qu.:89.00
Max. :4.00                        Max. :92.00
```

Data frame

Subset Rows Based on Conditions

```
# Get rows where Score is greater than 80
high_scores <- df[df$Score >80,]
print(high_scores)
        ID     Name Score Passed
        1 1 Alice 85 TRUE
        2 2 Bob 92 TRUE
        3 3 Charlie 88 TRUE
```

Select Specific Columns

```
# Select only the 'Name' and 'Score'
columns name_score <- df[,c("Name","Score")]
 print(name_score)
Name Score
1 Alice 85
2 Bob 92
3 Charlie 88
4 Diana 76
```

Adding and Modifying Columns

Add a New Column

```
# Add a column indicating if the score is above average
df$Above_Average <- df$Score > mean(df$Score)
print(df)

ID Name Score Passed Above_Average
1 1 Alice 85 TRUE FALSE
2 2 Bob 92 TRUE TRUE
3 3 Charlie 88 TRUE TRUE
4 4 Diana 76 FALSE FALSE
```

Modify an Existing Column

```
# Adjust the score by adding 5 points to each student
df$Score <- df$Score + 5
print(df)

ID Name Score Passed Above_Average
1 1 Alice 90 TRUE FALSE
2 2 Bob 97 TRUE TRUE
3 3 Charlie 93 TRUE TRUE
4 4 Diana 81 FALSE FALSE
```

Accessing data in Data frame

Sort by a Single Column

```
# Sort the data frame by 'Score' in descending order
df_sorted <- df[order(-df$Score),]
print(df_sorted)

ID Name Score Passed Above_Average
2 2 Bob 97 TRUE TRUE
3 3 Charlie 93 TRUE TRUE
1 1 Alice 90 TRUE FALSE
4 4 Diana 81 FALSE FALSE
```

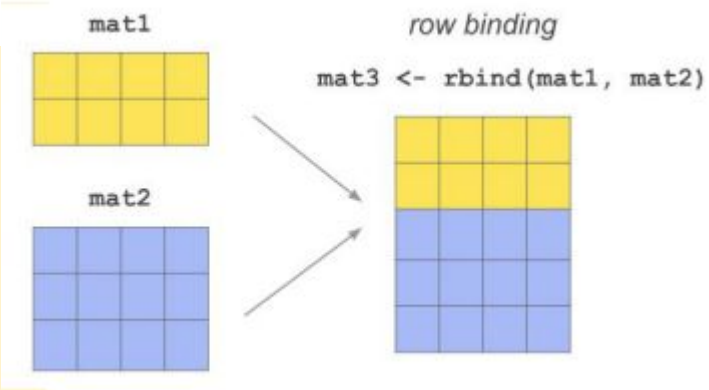
Sort by Multiple Columns Binding

```
# Sort by 'Passed' (descending) and then by 'Score' (ascending)
df_sorted_multi <- df[order(-df$Passed,df$Score),]
print(df_sorted_multi)

ID Name Score Passed Above_Average
1 1 Alice 90 TRUE FALSE
3 3 Charlie 93 TRUE TRUE
2 2 Bob 97 TRUE TRUE
4 4 Diana 81 FALSE FALSE
```

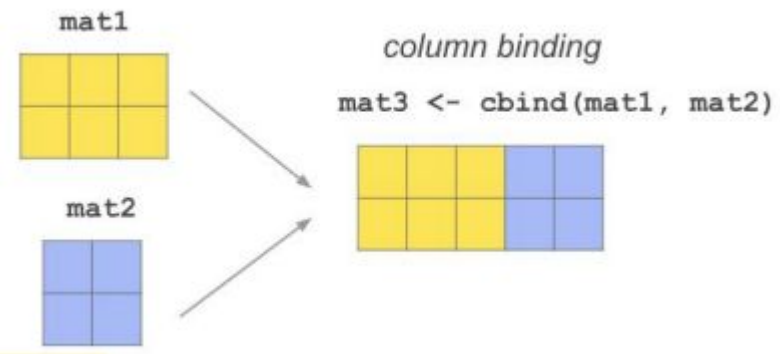
Accessing data in Data frame

Row Binding

```
# Bind new dataframe rows to an existed one
new_students <- data.frame
        (ID = 5,
        Name = "Eve",
        Score = 89,
        Passed = TRUE,
        Above_Average = FALSE)
        df_combined <- rbind(df, new_students)
        print(df_combined)

        ID Name Score Passed Above_Average
        1 1 Alice 90 TRUE FALSE
        2 2 Bob 97 TRUE TRUE
        3 3 Charlie 93 TRUE TRUE
        4 4 Diana 81 FALSE FALSE
        5 5 Eve 89 TRUE FALSE
```



Data frame



Column Binding

```
# Add a new column for student
Age ages <- data.frame(Age = c(23,25,22,21,24))
df_with_age <- cbind(df_combined, ages)
print(df_with_age)

ID Name Score Passed Above_Average Age
1 1 Alice 90 TRUE FALSE 23
2 2 Bob 97 TRUE TRUE 25
3 3 Charlie 93 TRUE TRUE 22
4 4 Diana 81 FALSE FALSE 21
5 5 Eve 89 TRUE FALSE 24
```

Data frame

Remove a Column

```
# Remove the 'Passed' column
df_no_passed <-
df[,!(names(df)%in%"Passed")]
print(df_no_passed)

ID Name Score Above_Average
1 1 Alice 90 FALSE
2 2 Bob 97 TRUE
3 3 Charlie 93 TRUE
4 4 Diana 81 FALSE
```

Rename a Column

```
# Rename 'Score' to 'Final_Score'
names(df)[names(df)=="Score"]<-"Final_Score"
print(df)

ID Name Final_Score Passed Above_Average
1 1 Alice 90 TRUE FALSE
2 2 Bob 97 TRUE TRUE
3 3 Charlie 93 TRUE TRUE
4 4 Diana 81 FALSE FALSE
```

Data frame

Key Variable	Variable A	Variable B	Variable C	Variable D
1	3.1	7.3	1	23
2	4.5	9.9	0	21
3	5.0	8.5	0	44
4	1.0	8.4	1	50

Key Variable	Variable E	Variable F	Variable G	Variable H
1	86	Red	4.9	19
2	95	Green	5.0	20
3	78	Red	5.0	14
4	91	Blue	4.1	13

...

Key Variable	Variable A	Variable B	Variable C	Variable D	Variable E	Variable F	Variable G	Variable H
1	3.1	7.3	1	23	86	Red	4.9	19
2	5.0	8.5	0	44	95	Green	5.0	20
3	5.0	8.5	0	44	78	Red	5.0	14
4	1.0	8.4	1	50	91	Blue	4.1	13

Accessing data in Data frame

Merging Data Frames

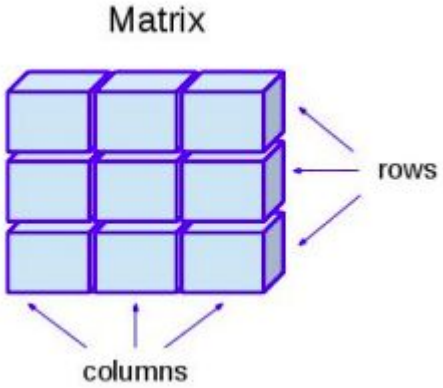
```
# Merge two data frames by the 'ID' column
df_info <- data.frame(ID =1:4, Gender = c("F","M","M","F"))
df_merged <- merge(df, df_info, by ="ID")
print(df_merged)

ID Name Score Passed Above_Average Gender
1 1 Alice 90 TRUE FALSE F
2 2 Bob 97 TRUE TRUE M
3 3 Charlie 93 TRUE TRUE M
4 4 Diana 81 FALSE FALSE F
```

Key Functions in Data frame

Other Key Functions

- **nrow(df): Number of rows.**
- **ncol(df): Number of columns.**
- **dim(df): Dimensions (rows, columns).**
- **names(df): Column names.**

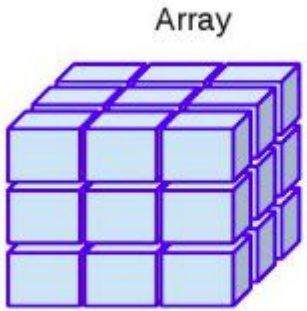
Matrix and array



Matrix

Definition: A matrix is a two-dimensional (2D) data structure in R where all elements are of the same data type (numeric, character, or logical).

Structure: Consists of rows and columns.



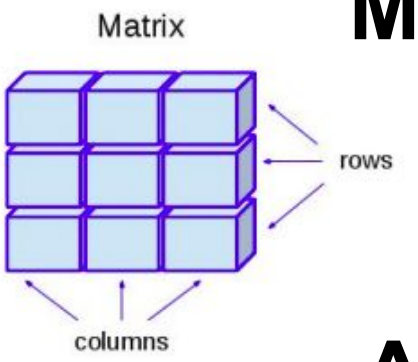
Array

Definition: An array is a multi-dimensional data structure in R that can have more than two dimensions. All elements must be of the same type.

Structure: Arrays can be thought of as matrices extended to more dimensions.

Matrix and array

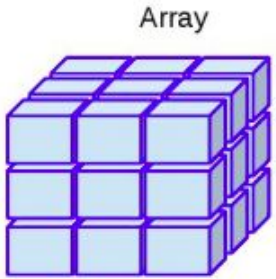
Matrix




```
# Create a 3x3 numeric
matrix mat <- matrix(1:9, nrow =3, ncol =3)
print(mat)
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Array




```
# Create a 3x3x2 array
arr <- array(1:18,dim=c(3,3,2)) print(arr)
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18
```

Matrix

Classwork 6: Create a microbiome feature matrix

1.1 Create a matrix named `microbiome_data` representing the abundance of 5 bacterial species across 4 different samples. Use the following data

```
microbiome_data <- matrix(c(23, 5, 0, 12, 9, 8, 15, 13, 7, 2,
14, 9, 6, 11, 1, 3, 8, 2, 10, 5), nrow = 4, byrow = TRUE)
```

1.2 Assign row names as `"Sample_1"`, `"Sample_2"`, `"Sample_3"`, and `"Sample_4"`, and column names as `"Species_1"`, `"Species_2"`, `"Species_3"`, `"Species_4"`, and `"Species_5"`.

	Species_1	Species_2	Species_3	Species_4	Species_5
Sample_1	23	5	0	12	9
Sample_2	8	15	13	7	2
Sample_3	14	9	6	11	1
Sample_4	3	8	2	10	5

Matrix

Classwork 6: Create a microbiome feature matrix

Part 2: Basic Matrix Operations

2.1 Extract the abundance data for `"Species_3"` across all samples.

2.2 Extract the data for `"Sample_2"` across all species.

2.3 Calculate the total abundance for each sample. (Use rowSums function)

2.4 Calculate the average abundance for each species across all samples. (Use colMeans function)

Matrix

Classwork 6: Create a microbiome feature matrix

Part 3: Advanced Matrix Operations

3.1 Transpose the `microbiome_data` matrix to switch rows and columns.

3.2 Identify the sample with the highest abundance of `"Species_1"`.

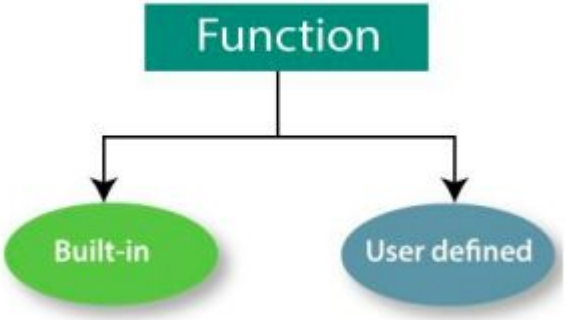
- **Hint**: Use the `which.max()` function to find the index.

3.3 Add a new species (`"Species_6"`) with the following abundance data: `[7, 10, 3, 5]`.

- **Hint**: Use the `cbind()` function to add a new column.

4. Functions

R function



- Useful Built-in function
- Create an R function

https://www.statmethods.net/management/functions.html

https://iqss.github.io/dss-workshops/R/Rintro/base-r-cheat-sheet.pdf

Useful Built-in function

Data Manipulation

- subset(): Extract subsets of data.
- merge(): Combine data frames by common columns or row names.
- apply(): Apply a function over the margins of an array or matrix.
- tapply(): Apply a function over subsets of a vector.
- reshape(): Reshape data between wide and long formats.
- cut(): Divide continuous variables into intervals.
- aggregate(): Compute summary statistics over subsets of data.

Statistical Analysis

- summary(): Provide a summary of an object.
- cor(): Calculate correlation between variables.
- lm(): Fit linear models.
- table(): Create a contingency table of counts.

Useful Built-in function

Data Cleaning

- na.omit(): Remove missing values from an object.
- is.na(): Identify missing values.
- duplicated(): Identify duplicate elements.

Data Visualization

- plot(): Generic X-Y plotting.
- hist(): Create a histogram.
- boxplot(): Create a boxplot.
- pairs(): Create a matrix of scatterplots.

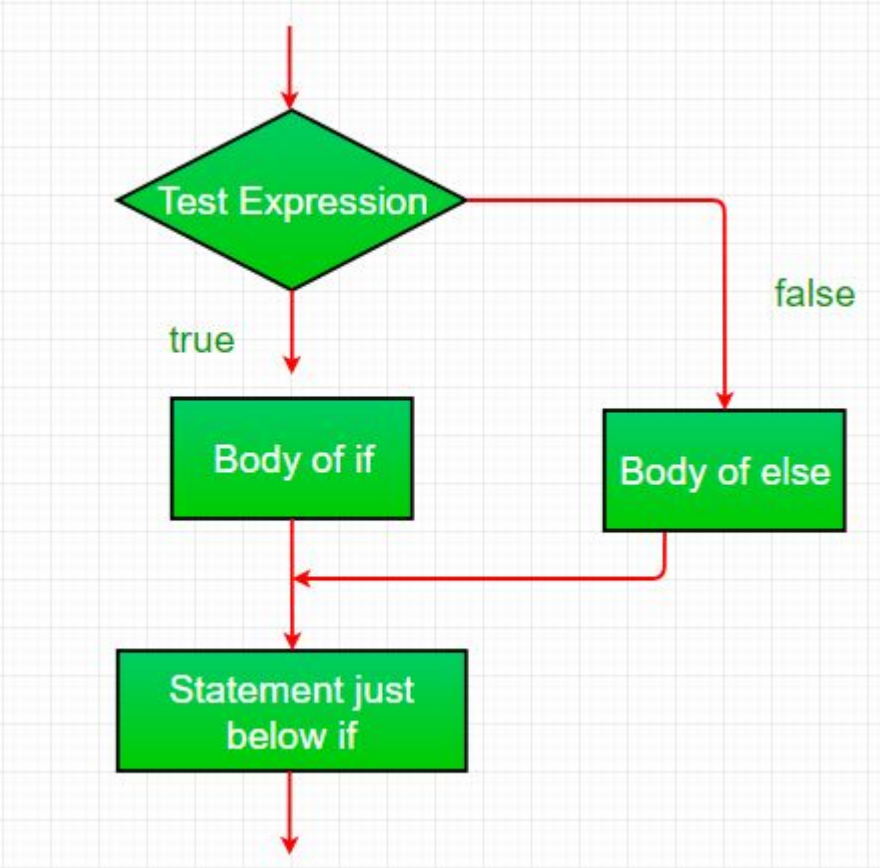
Useful Built-in function

Utility Functions

- str(): Display the structure of an R object.
- paste(): Concatenate strings.
- seq(): Generate a sequence of numbers.
- rep(): Repeat elements of a vector.

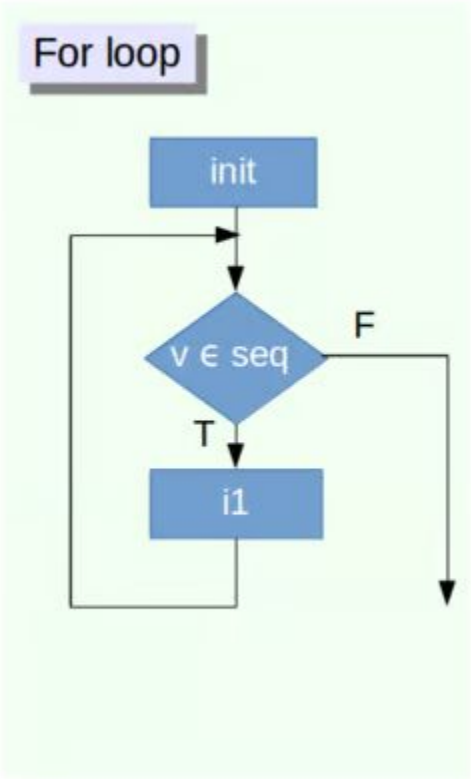
5. Decision Making

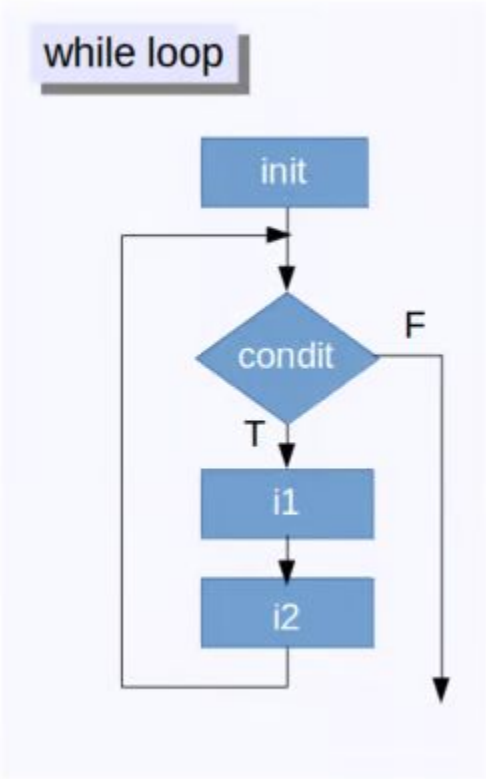
Decision making

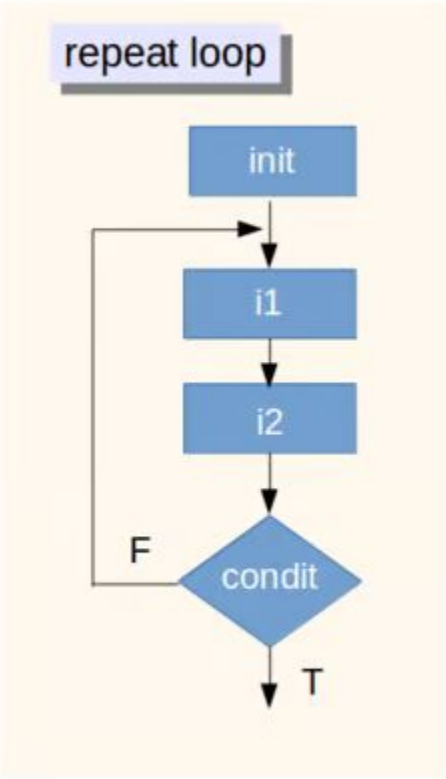


6. Control Flow

Control flow







R cheat sheet

r-cheat-sheet.key 1 / 2 100%

1

2

Base R
Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.
help.search('weighted mean')
Search the help files for a word or phrase.
help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.
class(iris)
Find the class an object belongs to.

Using Libraries

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

Vectors

Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x) Return x sorted.
rev(x) Return x reversed.
table(x) See counts of values.
unique(x) See unique values.

Selecting Vector Elements

By Position

x[4]	The fourth element.
x[-4]	All but the fourth.
x[2:4]	Elements two to four.
x[-(2:4)]	All elements except two to four.
x[c(1, 5)]	Elements one and five.

By Value

x[x == 10]	Elements which

Programming

For Loop

```
for (variable in sequence){
   Do something
}
```

Example

```
for (i in 1:4){
   j <- i + 10
   print(j)
}
```

While Loop

```
while (condition){
   Do something
}
```

Example

```
while (i < 5){
   print(i)
   i <- i + 1
}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){
   Do something
   return(new_variable)
}
```

Example

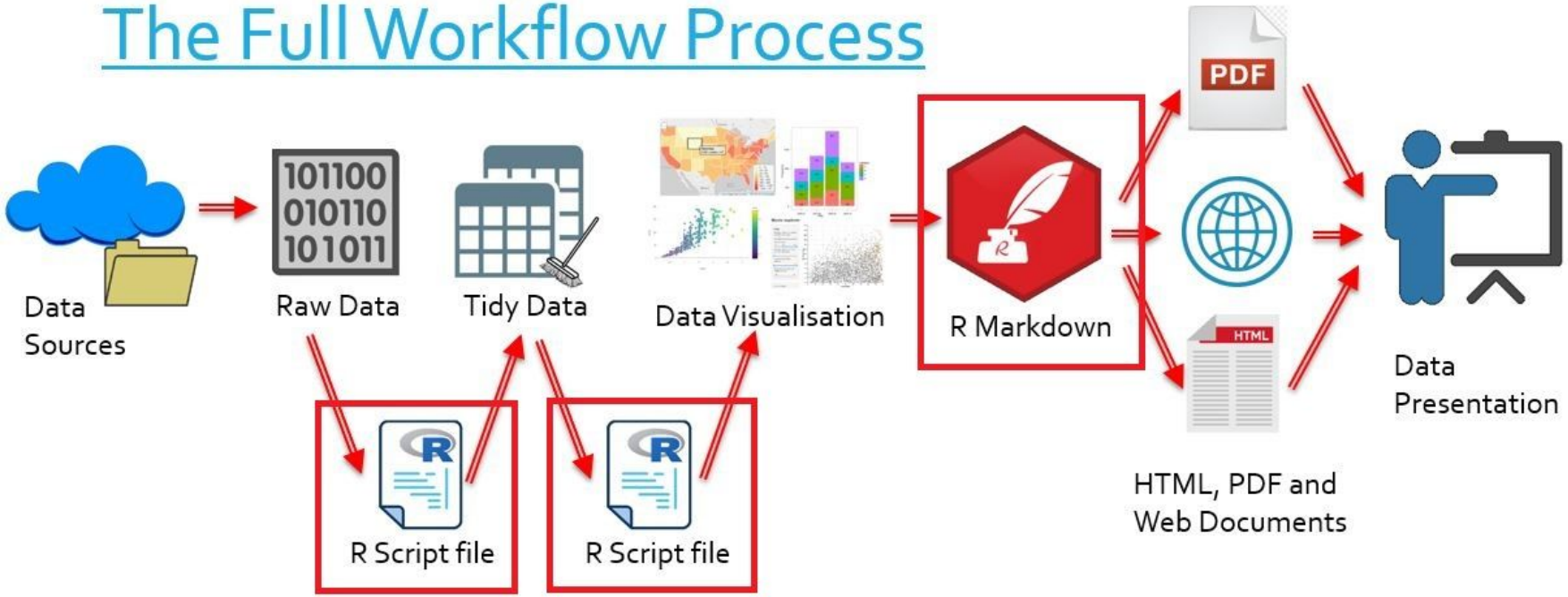
```
square <- function(x){
   squared <- x*x
   return(squared)
}
```

Reading and Writing Data

Input	Ouput	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read.table/

https://iqss.github.io/dss-workshops/R/Rintro/base-r-cheat-sheet.pdf

Summary in R tutorial



Thanks you