

R Course: Beginner to Expert

Sumit.G

2025-10-25

Table of contents

1	Welcome	6
2	About this Course	7
2.1	How to Use	7
3	R Basics	8
4	R as a Calculator	9
5	Objects & Assignment	10
6	object naming rules	11
7	Basic Operations in R	12
8	Comments in R	14
9	Getting Help in R	15
10	Installing and Loading Packages in R	17
10.0.1	Saving and Loading Workspaces in R	17
11	Working Directory	18
12	Vectors (Atomic)	19
13	Exercises	20
14	Data Types & Data Structures	21
15	Vectors	22
16	Lists	36
17	Matrices	48
18	Data Frames	69

19 Factors	78
20 Arrays	80
21 Summary	81
22 Manipulating Vectors, Data Frames, and Lists	82
23 Vectors: Indexing & Vectorized Ops	83
24 Data Frames with dplyr	102
25 Read sas dataset	116
26 keeping selected columns	117
27 dropping columns	118
28 Rename	120
29 Arrange (sorting)	121
30 convert var names to lower case	122
31 Creating new variables	123
32 bind rows (set operator in SAS)	124
33 recode (similar to PROC FORMAT in SAS)	125
34 joins	126
34.1 innter join	127
34.2 left join	127
34.3 right join	128
34.4 Full Join	128
34.5 Semi Join	128
34.6 Anti join	129
35 Reshaping data with tidyr	131
36 counting with n()	132
37 dplyr distinct()	134
38 case_when()	135

39 create seq no.	136
40 Lists: lapply, purrr	137
41 Exercises	138
42 Reading SAS Datasets (+ Cleaning)	139
42.1 Handling Labels & Missing	141
42.2 Labelled to Factor (if needed)	141
42.3 Common Cleaning	141
43 Exercises	143
44 Base R Functions & Apply Family	144
45 Common Utilities	145
46 Apply Family	146
47 Subsetting Essentials	147
48 Exercises	148
49 Custom Functions & Validation	149
50 Writing Functions	150
51 Error Handling	151
52 Unit Testing with testthat	152
53 Document with roxygen2	153
54 Exercises	154
55 R Package Development	155
55.1 Setup	155
55.2 Create a Package	155
55.3 Add a Function	155
55.4 Build, Install, Check	155
55.5 Vignette & Website	156
56 Git in RStudio (Setup & Auth)	157
56.1 One-Time Setup	157
56.2 Initialize Git for the Current Project	157
56.3 Connect to GitHub	157

56.4	Typical Workflow	157
56.5	Remove Git from a Project (macOS/RStudio)	158
57	Creating ADaM: ADSL from SDTM-like Inputs	159
57.1	Build ADSL	160
58	TLFs: Table, Figure, Listing	162
58.1	Table 1: Baseline Characteristics by Treatment	163
58.2	Figure: (Toy) Survival Curve	163
58.3	Listing: Subject-Level Listing	164
59	Capstone: End-to-End Mini Workflow	166
59.1	Parameters	166
59.2	1) Read (or Synthesize) SDTM	166
59.3	2) Derive ADSL (Minimal Demo)	167
59.4	3) TLFs	168
59.5	4) Save Outputs	169
60	Appendix: Tips, Profiles, .libPaths	170
60.1	Useful Profiles	170
60.2	Custom Library Paths	170
60.3	Format vs formatC (quick recap)	170
60.4	POSIXct vs POSIXlt	171
60.5	Recommended Packages	171
60.6	Short Glossary	171

1 Welcome

2 About this Course

This Course from **R beginner** to **expert**, with a practical focus on **clinical programming** (CDISC/ADaM and TLFs).

Each chapter includes step-by-step explanations, runnable code, and short exercises.

2.1 How to Use

1. Install: R (4.2), RStudio (or VS Code).
2. Follow along chapter-by-chapter, running code and completing exercises.
3. Use sample data or your own clinical trial datasets (SAS/CSV). ## Structure (Highlights)
 - **Basics & Data:** R syntax, data types/structures, vectors/data frames/lists.
 - **I/O:** Read SAS datasets (with **haven**), handle labels, and clean raw data.
 - **Programming:** Base functions, write your own functions, validate with tests.
 - **DevOps:** Create an R package, connect Git in RStudio/GitHub.
 - **CDISC:** Build ADaM (ADSL) from SDTM-like inputs.
 - **TLFs:** Produce a baseline Table 1, a KM plot, and a listing.

Tip: If you don't have sample SDTM/ADaM data yet, the chapters generate **small synthetic data** as a fallback so everything runs end-to-end. ## contact For questions or feedback, reach out to **r2sas2025@gmail.com**

3 R Basics

4 R as a Calculator

```
1 + 1
```

```
[1] 2
```

```
3 * (4 + 5)
```

```
[1] 27
```

5 Objects & Assignment

```
x <- 10  
y <- 3.5  
x + y
```

```
[1] 13.5
```

6 object naming rules

- R variable names can contain letters, numbers, periods, and underscores. However, they cannot start with a number or underscore. R is case-sensitive, so `age`, `Age`, and `AGE` would be considered different variables.
- R variable names should be descriptive and meaningful. Avoid using reserved words or function names as variable names.
- A variable can have a short name (like `x` and `y`) or a more descriptive name (`age`, `carname`, `total_volume`). Rules for R variables are:
- A variable name must start with a letter and can be a combination of letters, digits, period(`.`) and underscore(`_`). If it starts with period(`.`), it cannot be followed by a digit.
- A variable name cannot start with a number or underscore (`_`) Variable names are case-sensitive (`age`, `Age` and `AGE` are three different variables) Reserved words cannot be used as variables (`TRUE`, `FALSE`, `NULL`, `if...`)
- Variable names should not contain spaces. Use underscore (`_`) or period (`.`) to separate words in a variable name.
- Variable names should be meaningful and descriptive. Avoid using single-letter variable names except for temporary variables in loops or functions.

7 Basic Operations in R

R supports various basic operations, including: * Arithmetic Operations: Addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (^). Example:

```
a <- 10
b <- 5
sum <- a + b
diff <- a - b
prod <- a * b
quot <- a / b
exp <- a ^ b
sum; diff; prod; quot; exp
```

```
[1] 15
```

```
[1] 5
```

```
[1] 50
```

```
[1] 2
```

```
[1] 1e+05
```

- Comparison Operations: Equal to (==), not equal to (!=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=). Example:

```
x <- 10
y <- 5
eq <- x == y
neq <- x != y
gt <- x > y
lt <- x < y
gte <- x >= y
lte <- x <= y
eq; neq; gt; lt; gte; lte
```

```
[1] FALSE
```

```
[1] TRUE
```

```
[1] TRUE
```

```
[1] FALSE
```

```
[1] TRUE
```

```
[1] FALSE
```

- Logical Operations: AND (&), OR (|), and NOT (!). Example:

```
p <- TRUE  
q <- FALSE  
and <- p & q  
or <- p | q  
not <- !p  
and; or; not
```

```
[1] FALSE
```

```
[1] TRUE
```

```
[1] FALSE
```

8 Comments in R

Comments in R are created using the `#` symbol. Anything following the `#` on the same line is considered a comment and is ignored by R during execution. Example:

```
# This is a comment
x <- 10 # Assigning value to x
y <- 5  # Assigning value to y
sum <- x + y # Calculating the sum of x and y
sum # Output the sum
```

```
[1] 15
```

9 Getting Help in R

R provides several ways to get help and documentation for functions and packages: *

- `?function_name`: Displays the documentation for a specific function. Example:

```
?mean
```

- `help(function_name)`: Another way to access the documentation for a function. Example:

```
help(mean)
```

- `help.search("keyword")`: Searches for help topics related to a specific keyword. Example:

```
help.search("regression")
```

- `example(function_name)`: Shows examples of how to use a specific function. Example:

```
example(mean)
```

```
mean> x <- c(0:10, 50)
```

```
mean> xm <- mean(x)
```

```
mean> c(xm, mean(x, trim = 0.10))  
[1] 8.75 5.50
```

- `vignette("package_name")`: Opens the vignette (detailed documentation) for a specific package. Example:

```
vignette("dplyr")
```

```
starting httpd help server ... done
```

- `??keyword`: Searches for help topics related to a specific keyword (similar to `help.search`). Example:

```
??regression
```


10 Installing and Loading Packages in R

R has a vast ecosystem of packages that extend its functionality. To use a package, you need to install it first and then load it into your R session. * Installing a Package: Use the `install.packages("package_name")` function to install a package from CRAN. Example:

```
install.packages("ggplot2")
```

- Loading a Package: Use the `library(package_name)` function to load an installed package into your R session. Example:

```
library(ggplot2)
```

```
# Now you can use functions from the ggplot2 package
```

10.0.1 Saving and Loading Workspaces in R

You can save your R workspace (all objects in memory) to a file and load it later * Saving Workspace: Use the `save.image("file_name.RData")` function to save the entire workspace to a file. Example:

```
save.image("my_workspace.RData")
```

- Loading Workspace: Use the `load("file_name.RData")` function to load a saved workspace from a file. Example:

```
load("my_workspace.RData")
```

11 Working Directory

```
getwd()
```

```
[1] "/home/runner/work/r4sas/r4sas"
```

```
# setwd("/path/you/want") # avoid in reproducible code; prefer here::here() for projects
```

12 Vectors (Atomic)

```
nums <- c(1, 2, 3, 4)
chars <- c("a", "b", "c")
logical <- c(TRUE, FALSE, TRUE)
typeof(nums); typeof(chars); typeof(logical)
```

```
[1] "double"
```

```
[1] "character"
```

```
[1] "logical"
```

13 Exercises

1. Create an object `z` that stores $(5^2 + 7)/3$.
2. Use `?seq` and create a sequence from 0 to 1 by 0.1.
3. Inspect `typeof()` for a few objects you create.

14 Data Types & Data Structures

R has several built-in data structures to store and manipulate different types of data. These include vectors, lists, matrices, data frames, and factors. Below is an overview of each structure along with code examples.

15 Vectors

Vectors are the simplest data structure in R. They store elements of the same type (numeric, character, logical, etc.).

```
# Creating numeric and character vectors
numeric_vector <- c(1, 2, 3, 4)
char_vector1 <- c("apple", "banana", "cherry")
char_vector2 <- c(2, 3, 4, 5, "a")
logical_vector <- c(TRUE, FALSE, TRUE)

# Accessing elements
numeric_vector[1] # Access the first element
```

```
[1] 1
```

```
v_logical <- c(T,F,T) # logical vector
v_logical
```

```
[1] TRUE FALSE TRUE
```

```
is.vector(v_logical)
```

```
[1] TRUE
```

```
is.atomic(v_logical)
```

```
[1] TRUE
```

```
typeof(v_logical)
```

```
[1] "logical"
```

```
v_integer <- c(1L,2L,5L) # integer vector  
v_integer
```

```
[1] 1 2 5
```

```
is.vector(v_integer)
```

```
[1] TRUE
```

```
is.atomic(v_integer)
```

```
[1] TRUE
```

```
typeof(v_integer)
```

```
[1] "integer"
```

```
v_double <- c(1.3,2.1,5.0) # double vector  
v_double
```

```
[1] 1.3 2.1 5.0
```

```
is.vector(v_double)
```

```
[1] TRUE
```

```
is.atomic(v_double)
```

```
[1] TRUE
```

```
typeof(v_double)
```

```
[1] "double"
```

```
v_character <- c("a", "b", "c") # character vector
v_character
```

```
[1] "a" "b" "c"
```

```
is.vector(v_character)
```

```
[1] TRUE
```

```
is.atomic(v_character)
```

```
[1] TRUE
```

```
typeof(v_character)
```

```
[1] "character"
```

```
v_NULL <- NULL # NULL
v_NULL
```

```
NULL
```

```
typeof(v_NULL)
```

```
[1] "NULL"
```

```
# Mix type vector (type coercion or conversion)
v_mix <- c(T, 1L, 1.25, "a")
v_mix # all elements converted to characters (based on hierarchy)
```

```
[1] "TRUE" "1"    "1.25" "a"
```

```
is.vector(v_mix)
```

```
[1] TRUE
```



```
typeof(v_mix)
```

```
[1] "character"
```

```
# Vector properties
```

```
v <- c(1,2,3,4,5)
```

```
# vector length
```

```
length(v)
```

```
[1] 5
```

```
# type
```

```
typeof(v)
```

```
[1] "double"
```

```
class(v)
```

```
[1] "numeric"
```

```
# naming elements
```

```
names(v) # without names
```

```
NULL
```

```
vnames <- c("first", "second", "third", "fourth", "fifth") # element names
```

```
names(v) <- vnames # naming elements
```

```
v
```

```
first second  third fourth  fifth
      1      2      3      4      5
```

```
names(v) # new names
```

```
[1] "first" "second" "third" "fourth" "fifth"
```

```
# Create vector, access elements, modify vector
```

```
# create using c()
```

```
v <- c(1,3,5,8,0)
```

```
# create using operator :
```

```
1:100
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
[91] 91 92 93 94 95 96 97 98 99 100
```

```
10:-10
```

```
[1] 10 9 8 7 6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -
7 -8
[20] -9 -10
```

```
# using sequence seq()
```

```
v <- seq(from = 1, to = 100, by = 1)
```

```
v
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
[91] 91 92 93 94 95 96 97 98 99 100
```

```
v <- seq(from = 0, to = 1, by = 0.01)
```

```
v
```

```
[1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14
[16] 0.15 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29
[31] 0.30 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41 0.42 0.43 0.44
[46] 0.45 0.46 0.47 0.48 0.49 0.50 0.51 0.52 0.53 0.54 0.55 0.56 0.57 0.58 0.59
[61] 0.60 0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69 0.70 0.71 0.72 0.73 0.74
[76] 0.75 0.76 0.77 0.78 0.79 0.80 0.81 0.82 0.83 0.84 0.85 0.86 0.87 0.88 0.89
[91] 0.90 0.91 0.92 0.93 0.94 0.95 0.96 0.97 0.98 0.99 1.00
```

```
v <- seq(from = 0, to = 10, length.out = 5)
v
```

```
[1] 0.0 2.5 5.0 7.5 10.0
```

```
# let's create a vector for accessing vector elements
v <- 1:10
names(v) <- c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j")
v
```

```
 a  b  c  d  e  f  g  h  i  j
1  2  3  4  5  6  7  8  9 10
```

```
# access vector elements using integer vector index
v[c(1,5,10)]
```

```
 a  e  j
1  5 10
```

```
v[1:5] # range index selection (slicing)
```

```
 a b c d e
1 2 3 4 5
```

```
v[seq(from = 1, to = 9, by = 2)]
```

```
 a c e g i
1 3 5 7 9
```

```
v[10:1] # reverse order selection
```

```
 j  i  h  g  f  e  d  c  b  a
10 9  8  7  6  5  4  3  2  1
```

```
v[c(10,1,5,3)] # mix order selection
```

```
 j  a  e  c
10 1  5  3
```

```
# access vector elements using logical vector index
v[c(T,F,F,F,F,F,F,F,F)] # access first element
```

```
a
1
```

```
v[c(F,F,F,F,F,F,F,T,T,T)] # access last three elements
```

```
h i j
8 9 10
```

```
# access elements using names
v[c("a","c","e")]
```

```
a c e
1 3 5
```

```
v[c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j")]
```

```
a b c d e f g h i j
1 2 3 4 5 6 7 8 9 10
```

```
# modify vector elements
v
```

```
a b c d e f g h i j
1 2 3 4 5 6 7 8 9 10
```

```
v[2] <- 20 # alter second element
v
```

```
a b c d e f g h i j
1 20 3 4 5 6 7 8 9 10
```

```
v[c(1,5,10)] <- c(0,0,0) # alter multiple elements
v
```

```
a b c d e f g h i j
0 20 3 4 0 6 7 8 9 0
```

```
# modify elements with value 0
v
```

a	b	c	d	e	f	g	h	i	j
0	20	3	4	0	6	7	8	9	0

```
v[v==0] # filter with condition
```

a	e	j
0	0	0

```
v[v==0] <- 1000
v
```

a	b	c	d	e	f	g	h	i	j
1000	20	3	4	1000	6	7	8	9	1000

```
# truncate vector to first 3 elements
v <- v[1:3]
v
```

a	b	c
1000	20	3

```
# transpose vector change row to column vector or vice versa
v
```

a	b	c
1000	20	3

```
t(v)
```

	a	b	c
[1,]	1000	20	3

```
# delete or remove a vector
v <- NULL
v
```

NULL

```
rm(v)

# combine 2 different vectors
v1 <- 1:3
v2 <- 100:105
v1
```

```
# combine 2 different vectors
v1 <- 1:3
v2 <- 100:105
v1
```

```
v1 <- 1:3
v2 <- 100:105
v1
```

```
v2 <- 100:105
v1
```

[1] 1 2 3

v2

```
[1] 100 101 102 103 104 105
```

```
v3 <- c(v1,v2) # combine vectors
v3
```

```
[1] 1 2 3 100 101 102 103 104 105
```

```
# repet elements of a vector
rep(x = v1, times = 2)
```

```
rep(x = v1, times = 2)
```

[1] 1 2 3 1 2 3

```
rep(x = v1, times = 100)
```

[illegible]

```
rep(10,10)
```

```
[1] 10 10 10 10 10 10 10 10 10 10
```

```
# Vector arithmetics
```

```
# vector - scalar (scalar with each vector element)
```

```
v <- 1:5
```

```
a <- 10
```

```
v
```

```
[1] 1 2 3 4 5
```

```
a
```

```
[1] 10
```

```
# Addition +
```

```
v + a
```

```
[1] 11 12 13 14 15
```

```
# Subtraction -
```

```
v - a
```

```
[1] -9 -8 -7 -6 -5
```

```
# Multiplication *
```

```
v * a
```

```
[1] 10 20 30 40 50
```

```
# Division /
```

```
v / a
```

```
[1] 0.1 0.2 0.3 0.4 0.5
```

```
# Exponent ^ **
```

```
v^a
```

```
[1]      1    1024  59049 1048576 9765625
```

```
# Modulus (Remainder from division) %%  
v %% 2
```

```
[1] 1 0 1 0 1
```

```
# Integer Division %/%  
v %/% 2
```

```
[1] 0 1 1 2 2
```

```
# Other functions on vector elements  
sqrt(v)
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```
log(v)
```

```
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

```
sum(v)
```

```
[1] 15
```

```
# vector - vector (vector element to element | member-by-member)  
v1 <- seq(10,30,10)  
v2 <- rep(3,3)  
  
# Addition +  
v1 + v2
```

```
[1] 13 23 33
```

```
# Subtraction -  
v1 - v2
```

```
[1] 7 17 27
```



```
# Multiplication *  
v1 * v2
```

```
[1] 30 60 90
```

```
# Division /  
v1 / v2
```

```
[1] 3.333333 6.666667 10.000000
```

```
# Exponent ^ **  
v1^v2
```

```
[1] 1000 8000 27000
```

```
# Modulus (Remainder from division) %%  
v1 %% v2
```

```
[1] 1 2 0
```

```
# Integer Division %/%  
v1 %/% v2
```

```
[1] 3 6 10
```

```
# Vector-matrix style multiplication  
v1
```

```
[1] 10 20 30
```

```
v2
```

```
[1] 3 3 3
```

```
10*3 + 20*3 + 30*3
```

```
[1] 180
```

```
t(v1) %*% v2
```

```
      [,1]  
[1,] 180
```

```
v1 %*% v2
```

```
      [,1]  
[1,] 180
```

```
v1 %*% t(v2)
```

```
      [,1] [,2] [,3]  
[1,]    30    30    30  
[2,]    60    60    60  
[3,]    90    90    90
```

```
# Recycling rule  
v1 <- c(1,1,1)  
v2 <- 1:6  
v1
```

```
[1] 1 1 1
```

```
v2
```

```
[1] 1 2 3 4 5 6
```

```
v1 + v2
```

```
[1] 2 3 4 5 6 7
```

```
# Set operations  
v1 <- c("a", "b", "c")  
v2 <- c("c", "d", "e")  
v1
```

```
[1] "a" "b" "c"
```

```
v2
```

```
[1] "c" "d" "e"
```

```
union(v1,v2) # union of both sets (all unique elements)
```

```
[1] "a" "b" "c" "d" "e"
```

```
intersect(v1,v2) # intersection of both sets (elements in both sets)
```

```
[1] "c"
```

```
setdiff(v1,v2) # difference of elements (elements in v1 and not in v2)
```

```
[1] "a" "b"
```

```
identical(v1, v2) # check if vectors are identical
```

```
[1] FALSE
```

```
identical(c(1,2,3), c(1,2,3))
```

```
[1] TRUE
```

16 Lists

A list can contain elements of different types, including other lists or vectors or data structures.

```
# Creating a list
my_list <- list(name = "John", age = 25, scores = c(90, 85, 88))

# Accessing elements by name
my_list$name # Output: "John"
```

```
[1] "John"
```

```
# Create a list (and name elements)

# lets create some variables (different types)
a <- 10
b <- 2L
c <- TRUE
d <- "word"
v <- 1:10
names(v) <- paste("i", v, sep = "")
M <- matrix(data = seq(10,40,by = 10), nrow = 2, dimnames = list(c("r1", "r2"), c("c1", "c2")))
A <- array(data = 1:8, dim = c(2,2,2), dimnames = list(c("r1", "r2"), c("c1", "c2"), c("M1", "M2", "M3")))

# create list and include all variables (elements)
lst <- list(a, b, c, d, v, M, A)
lst
```

```
[[1]]
[1] 10
```

```
[[2]]
[1] 2
```

```
[[3]]
[1] TRUE
```

```
[[4]]  
[1] "word"
```

```
[[5]]  
  i1 i2 i3 i4 i5 i6 i7 i8 i9 i10  
  1  2  3  4  5  6  7  8  9  10
```

```
[[6]]  
      c1 c2  
r1 10 30  
r2 20 40
```

```
[[7]]  
, , M1
```

```
      c1 c2  
r1  1  3  
r2  2  4
```

```
, , M2
```

```
      c1 c2  
r1  5  7  
r2  6  8
```

```
str(lst) # check list structure
```

```
List of 7  
 $ : num 10  
 $ : int 2  
 $ : logi TRUE  
 $ : chr "word"  
 $ : Named int [1:10] 1 2 3 4 5 6 7 8 9 10  
 ..- attr(*, "names")= chr [1:10] "i1" "i2" "i3" "i4" ...  
 $ : num [1:2, 1:2] 10 20 30 40  
 ..- attr(*, "dimnames")=List of 2  
 .. ..$ : chr [1:2] "r1" "r2"  
 .. ..$ : chr [1:2] "c1" "c2"  
 $ : int [1:2, 1:2, 1:2] 1 2 3 4 5 6 7 8  
 ..- attr(*, "dimnames")=List of 3  
 .. ..$ : chr [1:2] "r1" "r2"
```

```
.. ..$ : chr [1:2] "c1" "c2"  
.. ..$ : chr [1:2] "M1" "M2"
```

```
typeof(lst) # check type
```

```
[1] "list"
```

```
class(lst) # check class
```

```
[1] "list"
```

```
is.list(lst) # check if object is list
```

```
[1] TRUE
```

```
# name each list member  
names(lst) <- c("a", "b", "c", "d", "v", "M", "A")  
lst
```

```
$a  
[1] 10
```

```
$b  
[1] 2
```

```
$c  
[1] TRUE
```

```
$d  
[1] "word"
```

```
$v  
 i1 i2 i3 i4 i5 i6 i7 i8 i9 i10  
  1  2  3  4  5  6  7  8  9  10
```

```
$M  
   c1 c2  
r1 10 30  
r2 20 40
```

```
$A  
, , M1
```

```
      c1 c2  
r1  1  3  
r2  2  4
```

```
, , M2
```

```
      c1 c2  
r1  5  7  
r2  6  8
```

```
# alternative: define names as tags when list is created  
list(a=a, b=b, c=c, d=d, v=v, M=M, A=A)
```

```
$a  
[1] 10
```

```
$b  
[1] 2
```

```
$c  
[1] TRUE
```

```
$d  
[1] "word"
```

```
$v  
  i1  i2  i3  i4  i5  i6  i7  i8  i9 i10  
  1   2   3   4   5   6   7   8   9  10
```

```
$M  
      c1 c2  
r1 10 30  
r2 20 40
```

```
$A  
, , M1
```

```
      c1 c2
```

```
r1  1  3
r2  2  4
```

```
, , M2
```

```
      c1 c2
r1    5  7
r2    6  8
```

```
# Access list elements
```

```
# single square bracket [] (return a list)
```

```
lst1 <-lst[1] # access first list elements (return a list)
```

```
str(lst1)
```

```
List of 1
```

```
$ a: num 10
```

```
class(lst1)
```

```
[1] "list"
```

```
lst123 <-lst[c(1,2,3)] # access first three elements with index vector (return a list)
lst123
```

```
$a
```

```
[1] 10
```

```
$b
```

```
[1] 2
```

```
$c
```

```
[1] TRUE
```

```
class(lst123)
```

```
[1] "list"
```



```
# double square brackets [[]] (return original member)
ele <-lst[[5]] # extract 5th member-element (returns original element)
ele
```

```
  i1  i2  i3  i4  i5  i6  i7  i8  i9 i10
    1   2   3   4   5   6   7   8   9  10
```

```
is.vector(ele)
```

```
[1] TRUE
```

```
# use $ operator - extract by member name (return original member)
ele <- lst$M
ele
```

```
      c1 c2
r1 10 30
r2 20 40
```

```
class(ele)
```

```
[1] "matrix" "array"
```

```
# Modify list

# remove element from a list
lst
```

```
$a
[1] 10
```

```
$b
[1] 2
```

```
$c
[1] TRUE
```

```
$d
[1] "word"
```

```
$v
  i1 i2 i3 i4 i5 i6 i7 i8 i9 i10
  1  2  3  4  5  6  7  8  9  10
```

```
$M
    c1 c2
r1 10 30
r2 20 40
```

```
$A
, , M1
```

```
    c1 c2
r1  1  3
r2  2  4
```

```
, , M2
```

```
    c1 c2
r1  5  7
r2  6  8
```

```
lst[1] <- NULL # remove first member
lst
```

```
$b
[1] 2
```

```
$c
[1] TRUE
```

```
$d
[1] "word"
```

```
$v
  i1 i2 i3 i4 i5 i6 i7 i8 i9 i10
  1  2  3  4  5  6  7  8  9  10
```

```
$M
    c1 c2
r1 10 30
```

```
r2 20 40
```

```
$A
```

```
, , M1
```

```
      c1 c2  
r1    1  3  
r2    2  4
```

```
, , M2
```

```
      c1 c2  
r1    5  7  
r2    6  8
```

```
# add element to a list (at the end)  
length(lst)
```

```
[1] 6
```

```
lst[7] <- 1000  
lst
```

```
$b
```

```
[1] 2
```

```
$c
```

```
[1] TRUE
```

```
$d
```

```
[1] "word"
```

```
$v
```

```
  i1  i2  i3  i4  i5  i6  i7  i8  i9 i10  
   1   2   3   4   5   6   7   8   9  10
```

```
$M
```

```
      c1 c2  
r1    10 30  
r2    20 40
```

```
$A
```

```
, , M1
```

```
      c1 c2  
r1  1  3  
r2  2  4
```

```
, , M2
```

```
      c1 c2  
r1  5  7  
r2  6  8
```

```
[[7]]  
[1] 1000
```

```
# update value of a member in alist  
lst[[7]] <- 500  
lst[7]
```

```
[[1]]  
[1] 500
```

```
# update value within a vector (on a list)  
lst[[4]][5] <- 5000  
lst[[4]]
```

```
  i1  i2  i3  i4  i5  i6  i7  i8  i9  i10  
   1   2   3   4 5000   6   7   8   9  10
```

```
# convert list to a vector  
vec <- unlist(lst)  
vec
```

```
      b      c      d  v.i1  v.i2  v.i3  v.i4  v.i5  v.i6  v.i7  v.i8  
    "2" "TRUE" "word"   "1"   "2"   "3"   "4" "5000"   "6"   "7"   "8"  
v.i9 v.i10      M1      M2      M3      M4      A1      A2      A3      A4      A5  
    "9"   "10"   "10"   "20"   "30"   "40"   "1"    "2"    "3"    "4"    "5"  
     A6     A7     A8  
    "6"    "7"    "8" "500"
```

```
is.vector(vec)
```

```
[1] TRUE
```

```
# Merging lists & nested lists

# create another list
lst1 <- list(e11 = c(1,5,10), e12 = TRUE)

# merge both lists
lst_merged <- c(lst, lst1)
lst_merged
```

```
$b
```

```
[1] 2
```

```
$c
```

```
[1] TRUE
```

```
$d
```

```
[1] "word"
```

```
$v
```

i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
1	2	3	4	5000	6	7	8	9	10

```
$M
```

	c1	c2
r1	10	30
r2	20	40

```
$A
```

```
, , M1
```

	c1	c2
r1	1	3
r2	2	4

```
, , M2
```

	c1	c2
--	----	----

```
r1 5 7
r2 6 8
```

```
[[7]]
[1] 500
```

```
$el1
[1] 1 5 10
```

```
$el2
[1] TRUE
```

```
str(lst_merged)
```

```
List of 9
 $ b : int 2
 $ c : logi TRUE
 $ d : chr "word"
 $ v : Named num [1:10] 1 2 3 4 5000 6 7 8 9 10
 ..- attr(*, "names")= chr [1:10] "i1" "i2" "i3" "i4" ...
 $ M : num [1:2, 1:2] 10 20 30 40
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : chr [1:2] "r1" "r2"
 .. ..$ : chr [1:2] "c1" "c2"
 $ A : int [1:2, 1:2, 1:2] 1 2 3 4 5 6 7 8
 ..- attr(*, "dimnames")=List of 3
 .. ..$ : chr [1:2] "r1" "r2"
 .. ..$ : chr [1:2] "c1" "c2"
 .. ..$ : chr [1:2] "M1" "M2"
 $ : num 500
 $ el1: num [1:3] 1 5 10
 $ el2: logi TRUE
```

```
names(lst_merged)
```

```
[1] "b" "c" "d" "v" "M" "A" "" "el1" "el2"
```

```
# nested list (recursive procedure)
list3 <- list(1, c(T,F,F)) # list sub-level 3
```

```
list2 <- list(list3) # list sub-level 2
list1 <- list(list2) # list sub-level 1

str(list1)
```

```
List of 1
 $ :List of 1
  ..$ :List of 2
   .. ..$ : num 1
   .. ..$ : logi [1:3] TRUE FALSE FALSE
```

```
# extract list level 2
list1[[1]]
```

```
[[1]]
[[1]][[1]]
[1] 1
```

```
[[1]][[2]]
[1] TRUE FALSE FALSE
```

```
# extract list level 3
list1[[1]][[1]]
```

```
[[1]]
[1] 1
```

```
[[2]]
[1] TRUE FALSE FALSE
```

```
# extract 1st member from list level 3
list1[[1]][[1]][[1]]
```

```
[1] 1
```

```
# extract 2nd member from list level 3
list1[[1]][[1]][[2]]
```

```
[1] TRUE FALSE FALSE
```

17 Matrices

A matrix is a two-dimensional structure that contains elements of the same type (numeric, character, or logical).

```
# Creating a 3x3 numeric matrix
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)

# Accessing elements
my_matrix[1, 2] # Access the element in row 1, column 2
```

[1] 4

```
# using matrix()
M <- matrix(data = 1:9, nrow = 3, ncol = 3)
M
```

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

```
M <- matrix(data = 1:9, nrow = 3, ncol = 3, byrow = T)
M
```

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

```
matrix(data = 1:6, nrow = 2, ncol = 3)
```

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	2	4	6


```
# by merging multiple vectors
v1 <- c(1,2,3)
v2 <- c(4,5,6)
v3 <- c(7,8,9)

rbind(v1,v2,v3)
```

```
      [,1] [,2] [,3]
v1      1    2    3
v2      4    5    6
v3      7    8    9
```

```
cbind(v1,v2,v3)
```

```
      v1 v2 v3
[1,]  1  4  7
[2,]  2  5  8
[3,]  3  6  9
```

```
# by altering vector dimension
v <- 1:9
v
```

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

```
dim(v) <- c(3,3)
v
```

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

```
# Matrix properties
```

```
# rownames & colnames
```

```
M <- matrix(1:12, nrow = 4, dimnames = list(c("r1","r2","r3", "r4"), c("c1","c2","c3")))
M
```

```

      c1 c2 c3
r1  1  5  9
r2  2  6 10
r3  3  7 11
r4  4  8 12

```

```
rownames(M)
```

```
[1] "r1" "r2" "r3" "r4"
```

```
colnames(M)
```

```
[1] "c1" "c2" "c3"
```

```
# matrix dimension
dim(M)
```

```
[1] 4 3
```

```
# get all attributes
attributes(M)
```

```
$dim
[1] 4 3
```

```
$dimnames
$dimnames[[1]]
[1] "r1" "r2" "r3" "r4"
```

```
$dimnames[[2]]
[1] "c1" "c2" "c3"
```

```
# change rownames & colnames
rownames(M) <- paste("row ", 1:4, sep = "")
colnames(M) <- paste("col ", 1:3, sep = "")
attributes(M)
```

```

$dim
[1] 4 3

$dimnames
$dimnames[[1]]
[1] "row 1" "row 2" "row 3" "row 4"

$dimnames[[2]]
[1] "col 1" "col 2" "col 3"

```

```
M
```

	col 1	col 2	col 3
row 1	1	5	9
row 2	2	6	10
row 3	3	7	11
row 4	4	8	12

```

# class and type
class(M)

```

```
[1] "matrix" "array"
```

```
typeof(M)
```

```
[1] "integer"
```

```

# check for matrix
is.matrix(M)

```

```
[1] TRUE
```

```

# Access matrix elements

# integer vector as index
M

```

	col 1	col 2	col 3
row 1	1	5	9
row 2	2	6	10
row 3	3	7	11
row 4	4	8	12

```
M[2,3]
```

```
[1] 10
```

```
M[c(1,2),3]
```

row 1	row 2
9	10

```
M[c(2,3),] # selected rows and all columns
```

	col 1	col 2	col 3
row 2	2	6	10
row 3	3	7	11

```
M[,c(2,3)] # selected columns and all rows
```

	col 2	col 3
row 1	5	9
row 2	6	10
row 3	7	11
row 4	8	12

```
# logical vector as index
M[c(T,T,F,F), c(T,T,T)]
```

	col 1	col 2	col 3
row 1	1	5	9
row 2	2	6	10

```
# character vector as index
M[c("row 2", "row 3"), c("col 1", "col 2")]
```

	col 1	col 2
row 2	2	6
row 3	3	7

```
# range of indexes (slicing rows and columns)
M[1:3,2:3]
```

	col 2	col 3
row 1	5	9
row 2	6	10
row 3	7	11

```
# Access matrix elements

# modify 1 element
M
```

	col 1	col 2	col 3
row 1	1	5	9
row 2	2	6	10
row 3	3	7	11
row 4	4	8	12

```
M[1,1] <- 10
M
```

	col 1	col 2	col 3
row 1	10	5	9
row 2	2	6	10
row 3	3	7	11
row 4	4	8	12

```
# modify more than one element
M[2:3,3] <- 20
M
```

	col 1	col 2	col 3
row 1	10	5	9
row 2	2	6	20
row 3	3	7	20
row 4	4	8	12

```
# modify elements based on condition
M[M>10] <- 0
M
```

	col 1	col 2	col 3
row 1	10	5	9
row 2	2	6	0
row 3	3	7	0
row 4	4	8	0

```
# transpose a matrix
t(M)
```

	row 1	row 2	row 3	row 4
col 1	10	2	3	4
col 2	5	6	7	8
col 3	9	0	0	0

```
# add row to matrix
M
```

	col 1	col 2	col 3
row 1	10	5	9
row 2	2	6	0
row 3	3	7	0
row 4	4	8	0

```
rbind(M, c(0,0,0))
```

	col 1	col 2	col 3
row 1	10	5	9
row 2	2	6	0
row 3	3	7	0
row 4	4	8	0
	0	0	0

```
# add column to matrix
cbind(M, c(0,0,0, 0))
```

```
      col 1 col 2 col 3
row 1    10     5     9 0
row 2     2     6     0 0
row 3     3     7     0 0
row 4     4     8     0 0
```

```
# alter matrix dimensions
dim(M)
```

```
[1] 4 3
```

```
dim(M) <- c(3,4) # names are dropped
M
```

```
      [,1] [,2] [,3] [,4]
[1,]    10     4     7     0
[2,]     2     5     8     0
[3,]     3     6     9     0
```

```
# merge 2 matrices
M1 <- matrix(data = rep(0,4), nrow = 2, ncol = 2)
M2 <- matrix(data = rep(1,4), nrow = 2, ncol = 2)
M1
```

```
      [,1] [,2]
[1,]     0     0
[2,]     0     0
```

```
M2
```

```
      [,1] [,2]
[1,]     1     1
[2,]     1     1
```

```
rbind(M1,M2)
```

```
      [,1] [,2]  
[1,]    0    0  
[2,]    0    0  
[3,]    1    1  
[4,]    1    1
```

```
cbind(M1,M2)
```

```
      [,1] [,2] [,3] [,4]  
[1,]    0    0    1    1  
[2,]    0    0    1    1
```

```
# Matrix arithmetics
```

```
# matrix - scalar (scalar with each vector element)  
M
```

```
      [,1] [,2] [,3] [,4]  
[1,]   10    4    7    0  
[2,]    2    5    8    0  
[3,]    3    6    9    0
```

```
a <- 10
```

```
# Addition +  
M + a
```

```
      [,1] [,2] [,3] [,4]  
[1,]   20   14   17   10  
[2,]   12   15   18   10  
[3,]   13   16   19   10
```

```
# Subtraction -  
M - a
```

```
      [,1] [,2] [,3] [,4]  
[1,]    0   -6   -3  -10  
[2,]   -8   -5   -2  -10  
[3,]   -7   -4   -1  -10
```



```
# Multiplication *
```

```
M * a
```

```
      [,1] [,2] [,3] [,4]
[1,]  100   40   70    0
[2,]   20   50   80    0
[3,]   30   60   90    0
```

```
# Division /
```

```
M / a
```

```
      [,1] [,2] [,3] [,4]
[1,]   1.0  0.4  0.7    0
[2,]   0.2  0.5  0.8    0
[3,]   0.3  0.6  0.9    0
```

```
# Exponent ^ **
```

```
M^a
```

```
      [,1]      [,2]      [,3] [,4]
[1,] 1.0000e+10  1048576  282475249    0
[2,] 1.0240e+03   9765625 1073741824    0
[3,] 5.9049e+04  60466176 3486784401    0
```

```
# Modulus (Remainder from division) %%
```

```
M %% 2
```

```
      [,1] [,2] [,3] [,4]
[1,]    0    0    1    0
[2,]    0    1    0    0
[3,]    1    0    1    0
```

```
# Integer Division %/%
```

```
M %/% 2
```

```
      [,1] [,2] [,3] [,4]
[1,]     5     2     3     0
[2,]     1     2     4     0
[3,]     1     3     4     0
```

```
# Other functions on matrix elements
sqrt(M)
```

```
      [,1]      [,2]      [,3] [,4]
[1,] 3.162278 2.000000 2.645751    0
[2,] 1.414214 2.236068 2.828427    0
[3,] 1.732051 2.449490 3.000000    0
```

```
log(M)
```

```
      [,1]      [,2]      [,3] [,4]
[1,] 2.3025851 1.386294 1.945910 -Inf
[2,] 0.6931472 1.609438 2.079442 -Inf
[3,] 1.0986123 1.791759 2.197225 -Inf
```

```
sum(M)
```

```
[1] 54
```

```
# matrix - vector (matrix element to element | member-by-member)
M1 <- matrix(data = 1:9, nrow = 3, byrow = T)
M2 <- matrix(data = rep(3,9), nrow = 3)

# Addition +
M1 + M2
```

```
      [,1] [,2] [,3]
[1,]    4    5    6
[2,]    7    8    9
[3,]   10   11   12
```

```
# Subtraction -
M1 - M2
```

```
      [,1] [,2] [,3]
[1,]   -2   -1    0
[2,]    1    2    3
[3,]    4    5    6
```

```
# Multiplication *
M1 * M2
```

```
      [,1] [,2] [,3]
[1,]    3    6    9
[2,]   12   15   18
[3,]   21   24   27
```

```
# Division /
M1 / M2
```

```
      [,1]      [,2] [,3]
[1,] 0.3333333 0.6666667 1
[2,] 1.3333333 1.6666667 2
[3,] 2.3333333 2.6666667 3
```

```
# Exponent ^ **
M1^M2
```

```
      [,1] [,2] [,3]
[1,]    1    8   27
[2,]   64  125  216
[3,]  343  512  729
```

```
# Modulus (Remainder from division) %%
M1 %% M2
```

```
      [,1] [,2] [,3]
[1,]    1    2    0
[2,]    1    2    0
[3,]    1    2    0
```

```
# Integer Division %/%
M1 %/% M2
```

```
      [,1] [,2] [,3]
[1,]    0    0    1
[2,]    1    1    2
[3,]    2    2    3
```

```
# matrix-matrix style multiplication
M1
```

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

```
M2
```

	[,1]	[,2]	[,3]
[1,]	3	3	3
[2,]	3	3	3
[3,]	3	3	3

```
t(M1) %*% M2
```

	[,1]	[,2]	[,3]
[1,]	36	36	36
[2,]	45	45	45
[3,]	54	54	54

```
M1 %*% M2
```

	[,1]	[,2]	[,3]
[1,]	18	18	18
[2,]	45	45	45
[3,]	72	72	72

```
# matrix algebra (matrix based functions)
M <- matrix(data = c(1,5,3,2,4,7,4,6,2), nrow = 3, byrow = T)

# get diagonal elements
diag(M)
```

```
[1] 1 4 2
```

```
# get matrix determinant
det(M)
```

```
[1] 74
```

```
# get inverse of a matrix M(-1)
solve(M)
```

```
      [,1]      [,2]      [,3]
[1,] -0.45945946  0.1081081  0.31081081
[2,]  0.32432432 -0.1351351 -0.01351351
[3,] -0.05405405  0.1891892 -0.08108108
```

```
# get eigen values
eigen(M)
```

```
eigen() decomposition
```

```
$values
```

```
[1] 11.778446+0.0000000i -2.389223+0.7578106i -2.389223-0.7578106i
```

```
$vectors
```

```
      [,1]      [,2]      [,3]
[1,] 0.4687233+0i  0.5211486+0.2411697i  0.5211486-0.2411697i
[2,] 0.6544420+0i -0.6642393+0.0000000i -0.6642393+0.0000000i
[3,] 0.5932993+0i  0.4573822-0.1408153i  0.4573822+0.1408153i
```

```
# calculate sum over rows or columns
M
```

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

```
rowSums(M)
```

```
[1]  9 13 12
```

```
colSums(M)
```

```
[1] 7 15 12
```

```
# Lets solve simple matrix equation
# A * X = B
A <- matrix(data = c(1,2,4,5), nrow = 2, byrow = T)
B <- matrix(data = c(5,24,17,66), nrow = 2, byrow = T)
# X = A(-1) * B
X <- solve(A) %*% B
X
```

```
      [,1] [,2]
[1,]     3     4
[2,]     1    10
```

```
# test
A %*% X # should get B
```

```
      [,1] [,2]
[1,]     5    24
[2,]    17    66
```

```
# summarizing a matrix (apply)
M
```

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

```
# sum of rows
apply(X = M, MARGIN = 1, FUN = sum)
```

```
[1] 9 13 12
```

```
# sum of columns
apply(X = M, MARGIN = 2, FUN = sum)
```

[1] 7 15 12

```
# create matrix of random numbers  
rnorm(n = 1000, mean = 0, sd = 2)
```

```
[1] -1.562230903 -0.800173792 -1.035944733 0.925116543 0.232035556  
[6] 0.478319852 -3.856108058 0.966679677 2.685036443 -1.906912653  
[11] 0.311398495 -0.474847418 -5.012020659 0.197929667 -1.245245633  
[16] -1.345258822 1.624429682 1.308278897 -1.130145460 1.655356171  
[21] -1.925639320 1.359722825 1.924726702 3.109197880 0.508911559  
[26] -1.383393961 -0.599841021 -0.284410661 -0.119095776 -2.457163208  
[31] -3.649669825 0.943306559 0.954446643 -1.695864828 1.145256456  
[36] 2.271903901 -0.123601327 -2.621033530 0.247970495 -2.048007317  
[41] -0.241901130 -0.402653635 -1.901625190 -3.925267127 -2.603487463  
[46] 2.011330806 -0.335312741 -2.796132993 1.041815983 2.657451847  
[51] -0.079022010 0.663988184 -2.427701357 -1.543168569 1.900419913  
[56] 2.089460333 1.549019939 1.892515204 4.084003882 -1.109956564  
[61] 3.312897689 -0.023215130 -0.690392565 0.986734319 0.300177739  
[66] -0.026264161 1.113762201 -0.520335484 0.865070711 1.064200327  
[71] 1.103738551 1.568937249 -0.353302568 2.372226403 1.026323499  
[76] 0.061820549 0.237572961 1.122208698 3.157843270 1.452800910  
[81] 1.101082117 -1.310319634 0.193244706 -1.302337797 1.560170926  
[86] -1.628354165 -0.553471264 -0.009660685 1.719728843 3.119451945  
[91] -3.224014593 1.674676701 -0.192901224 -2.003810109 1.224244268  
[96] 1.591743969 1.145074045 1.857528051 3.085838352 0.030941505  
[101] 0.798552558 -0.160499176 -0.343736671 -1.102148988 0.475123867  
[106] -0.900970563 -1.597699395 3.664117928 2.711501889 2.612174816  
[111] 0.041178692 0.422065024 -1.686034743 0.652551256 0.300934116  
[116] 1.173901284 -0.121832443 2.467411224 2.305429934 2.332834748  
[121] 4.260347339 -0.923521198 -0.055515371 1.804705848 -2.494627150  
[126] -1.901584162 -1.507390810 -1.885479599 -0.242485457 2.105537958  
[131] -0.294279222 -1.411442742 0.147405556 -0.788697140 -0.911636560  
[136] -0.794362652 2.299570403 -0.660182754 2.511660687 -0.342442114  
[141] -0.568459716 0.594787199 -0.659687652 -1.655594683 0.558018911  
[146] -0.272704676 0.158507232 0.008536762 -0.848263475 -0.454991126  
[151] -1.859979340 0.315352401 -0.266149315 -2.286229028 -0.070390841  
[156] 3.531929247 0.341163964 -0.207433363 -0.903945053 2.277658188  
[161] -0.078479745 -4.320188183 -1.571525493 0.331957121 -1.885592229  
[166] 2.802186555 0.209844113 0.537475250 -2.192400422 -2.717467087  
[171] -1.687610957 -0.351191121 0.635854078 1.694125926 -0.080563761  
[176] -1.732960209 -0.272096061 -0.957615923 -0.030429444 -0.474435952  
[181] -3.902527607 1.722363572 -1.304861330 -0.902688819 0.013496836
```

[186]	1.474082726	3.643215680	5.007628289	-1.173283087	-1.317355348
[191]	-0.792118505	-3.393873747	4.230225404	0.319066046	1.009542696
[196]	1.563372153	1.282933564	2.226567250	1.576169586	3.403248441
[201]	-0.366022942	-0.564159566	-1.306166670	-1.760695959	0.018787351
[206]	-1.967423438	-2.708688407	-0.193960099	0.704835775	-0.328997128
[211]	1.500802142	-1.707217972	-0.611247457	0.917293756	1.191567313
[216]	-0.943041108	-4.037704413	-2.596454869	2.892691831	-1.426305885
[221]	-2.808497691	1.933964800	-1.862882144	0.451355987	-0.767764230
[226]	-1.812864016	0.524635966	1.274923692	-2.151825192	3.753409785
[231]	-0.532908297	0.885323720	0.087242171	-0.734506805	1.027557596
[236]	-1.550550079	1.074780132	0.673287825	0.882900511	0.841124537
[241]	-1.993122977	1.030290445	3.529150044	0.212042918	-4.886248364
[246]	2.752831541	-1.075661643	0.417589179	-0.793177193	-2.289241193
[251]	-1.985520587	-0.091945497	-4.322324815	-0.634156275	4.687803464
[256]	-0.630208008	0.231241104	0.132259578	-0.220649394	0.725373624
[261]	-0.556229987	-0.817071629	0.195295486	-0.524250255	3.605939508
[266]	-0.505080659	-2.112618597	-0.883487791	0.888766995	0.293077334
[271]	-1.414444778	1.328407335	2.726422823	-1.890162948	1.927936619
[276]	-1.628087943	1.616137047	0.518016521	-2.286690916	-0.216829609
[281]	-2.735487008	1.053041022	-1.822207756	1.305153457	-0.106942910
[286]	0.037680212	-0.402386607	-0.220512378	2.550671697	0.628482730
[291]	4.699476521	1.276883160	-2.628319867	1.952272607	3.650119528
[296]	0.571362816	-1.948310060	1.334406228	-1.516592635	-1.894661444
[301]	-1.893898882	-2.408029207	1.006156035	0.358056030	0.476739136
[306]	2.912692444	-0.305129729	1.583197291	-1.888777313	-1.865050495
[311]	0.605821776	0.297544690	1.457900582	-1.021480689	-0.346119389
[316]	-1.651840140	0.273312144	5.543414108	-1.954949145	-2.063534707
[321]	0.085262397	2.191449184	-1.464614337	-3.247230102	-2.797381484
[326]	2.715288675	-0.660278704	-0.327349789	-3.662415143	3.962736706
[331]	1.722290552	2.904950320	-1.669084370	0.043620155	2.142423273
[336]	-2.963040723	-1.551103484	0.408506750	0.139764604	4.290297167
[341]	2.200013445	0.330455451	-1.470347243	2.333353498	1.112683156
[346]	1.474301833	3.134898412	-0.783741836	-0.318667058	-0.825683968
[351]	1.689087735	1.255683219	3.689536712	-0.934864954	-2.969384119
[356]	-0.125708377	-0.989795205	2.402011857	-1.129648163	-0.548558540
[361]	-1.975841260	-0.193710555	0.876106760	-0.583578244	2.354397844
[366]	-0.802023187	0.982352058	-2.421516262	2.331595420	-0.132704650
[371]	2.463598713	-0.025052486	-3.127160600	-1.941534603	1.326263115
[376]	1.615527604	-1.492684041	-0.875567297	-0.722316676	0.517828946
[381]	-0.947678174	-1.729208859	-0.954798404	3.613485643	3.937190777
[386]	-2.919179392	0.837885009	-5.944639003	-0.781035939	-3.063630263
[391]	-2.189825024	0.355641397	-0.936268005	2.805371126	-2.389095884
[396]	3.126340896	0.988988597	0.819385261	-1.276712075	-1.247475864

[401]	-1.481487763	1.797096832	-1.153610059	0.335964511	-0.669556362
[406]	-1.102704720	-0.569488075	0.087245121	0.732221983	-0.367168614
[411]	-2.407458035	2.081528702	3.000285792	1.610977670	1.405540923
[416]	-0.543451438	-2.468104949	1.394801992	-2.138453409	-0.037326384
[421]	-2.359930736	0.143014953	-1.106136164	-0.559090908	0.568418832
[426]	4.650263681	1.766266334	-0.372955603	-1.574182951	3.106523543
[431]	-0.866189481	0.184142650	2.129884519	0.982446648	0.043416246
[436]	1.829858091	-0.707201531	3.274823508	0.629190389	0.485494166
[441]	2.458290490	-1.521580587	2.935062961	-2.033601369	1.412512209
[446]	1.270228108	0.166889175	0.382945363	-0.712904121	-1.367245900
[451]	2.012263946	2.511349103	-0.699138068	2.010065067	0.882768330
[456]	-0.158992139	-0.561938306	0.692382117	2.359632878	-1.598101365
[461]	0.709753611	0.522718660	2.012172937	-1.379163291	-1.419606244
[466]	3.306121395	-2.511956141	-0.963702053	-3.255281787	-1.259259841
[471]	1.542146774	-6.051684614	-0.273835763	-3.301474789	0.018876337
[476]	-0.166444174	2.113700450	0.489028684	2.042682395	2.358090848
[481]	1.308374350	5.086198057	-0.227243017	-0.753261151	4.868605533
[486]	2.461398730	-0.224812019	1.573224271	1.585157479	1.008597606
[491]	0.795203922	0.417009392	-0.813145098	-4.342184007	-2.947928765
[496]	0.274879598	0.859352758	3.235660742	-1.358195318	1.769795175
[501]	3.117035260	2.067507990	0.200867347	0.618914385	-1.107331744
[506]	-1.086318514	-2.261294774	-3.026930959	2.007741292	-1.019539956
[511]	0.581689686	-0.831116640	-0.453655929	0.259274513	-0.967698318
[516]	0.616083649	0.885772081	1.761680628	-1.439959851	-0.329120198
[521]	2.926135495	-2.330973058	3.130361850	0.987095412	3.144738513
[526]	-0.232164343	-3.753337080	1.532622493	-1.649475565	-2.731726817
[531]	1.596456234	-0.727906858	-0.524299177	-0.443229975	-1.280931617
[536]	-3.657376556	1.147303829	-3.123434996	0.982329366	0.405664755
[541]	0.158331654	0.922872389	-1.471268864	1.443668823	3.352875016
[546]	1.653094872	0.851997204	-1.169889170	-1.833577891	0.036765324
[551]	1.907646290	0.207469783	0.589303156	-1.596806828	-2.519913921
[556]	2.670901920	-3.436029978	1.908015600	0.534528958	0.606776440
[561]	-0.772194332	0.463682455	-1.571076293	1.241873715	1.831596305
[566]	1.970834433	-0.069465894	-2.046063835	2.822814104	0.629799848
[571]	0.301427429	-0.467371011	2.078642762	-2.885728326	1.622011499
[576]	-0.518383866	0.100918171	-1.005384108	1.421794889	2.345754682
[581]	2.593550871	0.992568486	-0.360741878	-0.067454665	-0.388298755
[586]	-1.525576766	-2.016248360	0.749505846	2.434949355	0.829179561
[591]	3.208960940	-0.874428604	1.171047123	2.461397125	0.461840323
[596]	0.910948792	-0.020566017	-0.802454078	-2.618596732	-0.284443440
[601]	-1.569480318	0.356514063	-0.787289302	0.451817197	-3.847702671
[606]	-0.944621120	-1.796139489	-1.851918693	-3.392811556	0.227479448
[611]	-1.883850846	-2.957246462	-4.466570835	1.704614649	-3.228601775

[616]	1.132596686	0.585972476	-1.903100980	1.407303818	-4.146764175
[621]	0.834548869	-2.410010882	-0.119041332	-1.731805625	-3.143502687
[626]	-0.975903919	1.186068573	1.022042225	-0.984256942	-2.848960200
[631]	-0.575157651	-2.410828356	-2.015923748	1.376994213	-1.195203811
[636]	-2.472839684	-1.271801547	2.269082457	-0.138058653	0.430991652
[641]	-1.656357584	-0.974128177	0.141453426	-1.484446782	-0.647315590
[646]	2.235637743	0.213823718	2.008601500	-0.448311504	-0.244210412
[651]	-0.502035775	-1.797142860	-0.800951163	-1.848830679	-1.515639640
[656]	-3.396088659	0.927337644	-2.512806491	-1.240471909	-1.396738553
[661]	-1.707041625	-0.979730446	2.667377006	1.180229554	-0.444101690
[666]	1.912734425	-2.648301488	0.530555407	0.640615525	-0.671262081
[671]	1.094731863	1.128817791	-1.932821705	-0.677219115	1.654999673
[676]	-0.096136453	0.031990610	-0.079044161	1.258639571	-1.692720805
[681]	0.951102147	3.827033742	1.036275805	-0.824073827	0.500329912
[686]	0.475891419	-0.241166537	0.737813362	2.243807424	-0.111735022
[691]	2.477878805	0.822358478	2.186751331	3.186292909	0.490971559
[696]	-4.522424854	-1.681761823	2.575881707	1.092150147	0.289108061
[701]	0.524781105	1.801668995	2.005249258	-0.089029927	2.052496269
[706]	-0.660701619	-1.214735502	1.278733032	-0.171870437	0.558131480
[711]	-1.347580456	-0.754766930	-0.914821147	-2.540909070	4.515958818
[716]	-1.641232952	-2.542868432	0.082282401	0.846601861	0.125275676
[721]	3.719874352	-0.674509013	-0.478505895	-2.824931446	-0.894389649
[726]	-0.841045289	-1.606137129	-2.179829359	2.774637441	-0.002286994
[731]	-2.845169262	1.250094668	-0.025054065	-3.222484315	2.320527729
[736]	-0.868526247	1.739131115	-1.573071221	-0.171024448	-2.354790072
[741]	0.780339087	1.979648040	-2.078231982	1.160484545	-0.404249188
[746]	-1.924292906	-2.134577218	3.341781843	-1.406609453	-3.049136257
[751]	-1.998625192	-2.778676105	0.516780870	1.026168289	-1.762717903
[756]	0.394844036	3.448701607	0.495167282	1.004592852	2.032246723
[761]	-2.772832875	0.509743728	-0.585750504	0.333727094	1.445327025
[766]	0.607264869	-2.379448491	-1.591229929	0.271504189	0.680193268
[771]	1.585018301	-0.650294784	-2.432106758	-1.375937689	0.571459522
[776]	-3.849895184	-0.176514501	4.027177366	0.221159063	-2.659753775
[781]	0.833033135	0.824641735	1.893853409	-2.722062989	0.326381583
[786]	0.060853343	-0.333872570	-2.360696108	3.363922378	3.534582970
[791]	3.501195307	-2.887524173	1.351616302	-0.371674303	0.865894240
[796]	3.083921087	0.477652235	1.560310954	0.131129758	3.196750509
[801]	-1.153932167	0.520720160	-0.117817974	-0.375026334	1.601508320
[806]	1.440379895	-1.272401633	1.676142226	-1.321040472	-2.888995896
[811]	-2.390195905	-0.677593138	1.328703650	-1.520775057	2.738398536
[816]	1.032341740	2.020022646	0.091185865	2.494792506	6.194371984
[821]	2.731295374	-1.155921239	2.433682294	-0.312448279	-1.154212013
[826]	-2.297687624	-1.534831370	-0.126907132	0.286744092	0.388044829

```

[831] -1.291978598 -1.670785501 1.830859627 -1.019078219 -3.225061320
[836] 1.893104690 2.479830881 1.509866192 -0.471874412 -4.025412970
[841] -1.562955717 -2.043702464 1.892711394 -0.389582627 -0.120856565
[846] 0.483899298 0.249698728 2.199697780 -0.885373028 1.482728928
[851] -0.022910978 -2.263399131 -0.052130498 -2.398126167 1.532248576
[856] -2.004547567 -0.676415794 -3.558150542 -3.164817878 0.119541531
[861] -4.449492069 3.665391976 -2.612959309 0.381207982 -0.786696361
[866] 2.863370853 -2.594243805 -1.940495032 -0.589374001 -0.298191918
[871] -0.135340146 -0.204579964 -1.913340209 0.911172509 -0.179120655
[876] -0.541046880 -0.815475377 5.106296347 -1.710698066 -0.265394758
[881] 2.775479048 3.703512579 1.515769299 -1.164231415 1.034445848
[886] -0.092645972 -0.473303566 -0.781080256 0.986499265 0.897167526
[891] -0.564675619 -1.358859363 -0.348687806 0.315832430 1.430309486
[896] -0.273500636 -0.811107410 2.501931562 0.687529603 1.759114147
[901] -2.975462419 -0.342135996 0.782610094 -0.105644338 1.243999103
[906] -3.904897326 -1.736959210 1.288067017 -2.974132583 0.102194760
[911] 4.703620958 -3.392527419 3.186820702 0.332373407 -2.995099563
[916] 0.178180907 -0.099288153 -1.934530454 1.599149434 1.254872622
[921] -0.902174591 -2.576276798 2.716228203 -0.669929243 -2.009314960
[926] 1.511390747 -2.435321437 -3.110538335 0.308311352 2.296788654
[931] -1.775547788 -2.907015384 -0.221122904 -1.181956818 0.833712313
[936] -0.669660156 -1.593131751 -3.154444273 -1.728467780 -1.426366080
[941] 0.852887818 0.731144418 -1.704778324 -1.454016780 1.862475991
[946] -0.330784540 -1.485186432 2.673186977 -0.819011006 -0.022540519
[951] -0.022812331 0.846883701 -0.060289357 0.132540185 1.483488344
[956] 0.513139055 -3.091761941 -3.832654606 0.057951437 0.904888104
[961] -0.422389518 -1.762656548 0.149445421 4.416790163 6.848147516
[966] 0.804629641 -3.896896158 -0.783773211 3.287654153 0.736352573
[971] -1.992619014 0.999567741 -1.915950775 3.668155395 3.716253678
[976] 2.966850793 -1.345574016 0.174268544 -4.097004552 -3.179054491
[981] 2.944256978 -2.953145880 1.279483042 -1.258234490 3.393800659
[986] 1.848274473 -0.351461193 0.575284806 3.369328714 -2.589742417
[991] -0.126840023 -2.958525990 -0.797386685 0.018463952 2.011056350
[996] -5.301364061 -0.816718941 1.074973240 -0.445043322 -2.419012161

```

```

A <- matrix(data = rnorm(n = 1000, mean = 0, sd = 2), nrow = 100, ncol = 10)

# get mean over columns
apply(A, 2, mean)

```

```

[1] -0.36492484 0.52755161 0.17617211 0.07450259 -0.02912477 -0.04724076
[7] 0.27689004 -0.22532661 -0.42319503 0.13025137

```

```
# get mean over rows  
apply(A, 1, mean)
```

```
[1] 0.653616763 -1.101064472 0.482864784 0.356993932 -0.329434770  
[6] 0.212031630 0.177551729 0.592728402 0.311099547 0.286580765  
[11] -0.614545343 -0.295665490 0.612910479 -0.204680094 -0.911160311  
[16] 0.471203011 0.308914548 -0.328937334 0.785716233 -0.410707845  
[21] 0.476029234 -0.528225966 0.393752924 -0.010919594 -0.695512425  
[26] 0.364688631 -1.108077178 0.374520183 -0.440901430 0.042362826  
[31] -0.624333979 -0.264206287 0.525846478 -1.070984600 -0.981221942  
[36] -0.807459482 0.079738301 0.335943035 0.219689349 0.937048249  
[41] 0.657513080 -0.669005864 -0.210862228 -1.019952283 0.389795777  
[46] 0.203267761 0.301200884 1.175607525 0.190145626 0.053422449  
[51] -0.272316858 -0.510551867 -0.639100274 0.161262690 -0.139938411  
[56] -0.003446521 0.824645651 1.239081105 0.145453498 -0.094537010  
[61] 1.099429888 0.386989906 0.945832048 -1.183243211 0.630720361  
[66] 0.406052485 0.453958472 -1.047023815 -0.269473581 -0.088844286  
[71] 0.163603475 -0.523086143 -0.097629991 -1.250516664 0.445169880  
[76] -0.572285041 -0.025714074 0.817483901 -0.013963828 -0.372807080  
[81] -0.111056168 -0.586758899 0.529044084 -0.463233593 0.446098319  
[86] 0.054248991 -0.072535665 0.643043154 0.562894244 -0.027826132  
[91] -0.246087706 0.188290745 -0.357853881 0.477906107 0.121906060  
[96] 0.007807119 -0.362800890 0.377681752 0.145292031 -0.330632469
```

```
# calculate standard deviation for each column  
apply(A, 2, sd)
```

```
[1] 1.931555 2.247890 1.832899 2.060627 1.818917 1.942830 2.040748 1.949930  
[9] 2.052880 1.721227
```

18 Data Frames

A data frame is a table where each column can contain elements of different types (e.g., numbers, strings). It's the most common structure used for data sets.

```
# Creating a data frame
my_data <- data.frame(
  Name = c("Alice", "Bob", "Charlie"),
  Age = c(23, 30, 25),
  Gender = c("F", "M", "M")
)
```

```
my_data
```

	Name	Age	Gender
1	Alice	23	F
2	Bob	30	M
3	Charlie	25	M

```
# Accessing columns
my_data$Name # Output: "Alice", "Bob", "Charlie"
```

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

```
# create data frame
df1 <- data.frame(col1 = 1:3,
                  col2 = c("a", "b", "c"),
                  col3 = c(T, F, T),
                  col4 = c(as.Date("2020-01-01"), as.Date("2020-01-03"), as.Date("2020-01-03"))

# create data frame - vectors
col1 <- seq(10,100,10)
col2 <- seq(as.Date("2020-01-01"), length = 10, by = "weeks")
col3 <- rep("word", 10)
```

```
df2 <- data.frame(num = col1,
                  date = col2,
                  string = col3)
```

```
# check DF structure
str(df2)
```

```
'data.frame':  10 obs. of  3 variables:
 $ num      : num  10 20 30 40 50 60 70 80 90 100
 $ date     : Date, format: "2020-01-01" "2020-01-08" ...
 $ string   : chr  "word" "word" "word" "word" ...
```

```
# create data frame - matrix
M <- matrix(data = 1:100, nrow = 10, ncol = 10, byrow = T)
rownames(M) <- paste("row", 1:10, sep = "")
colnames(M) <- paste("col", 1:10, sep = "")
M
```

	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10
row1	1	2	3	4	5	6	7	8	9	10
row2	11	12	13	14	15	16	17	18	19	20
row3	21	22	23	24	25	26	27	28	29	30
row4	31	32	33	34	35	36	37	38	39	40
row5	41	42	43	44	45	46	47	48	49	50
row6	51	52	53	54	55	56	57	58	59	60
row7	61	62	63	64	65	66	67	68	69	70
row8	71	72	73	74	75	76	77	78	79	80
row9	81	82	83	84	85	86	87	88	89	90
row10	91	92	93	94	95	96	97	98	99	100

```
df3 <- as.data.frame(M)
df3
```

	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10
row1	1	2	3	4	5	6	7	8	9	10
row2	11	12	13	14	15	16	17	18	19	20
row3	21	22	23	24	25	26	27	28	29	30
row4	31	32	33	34	35	36	37	38	39	40
row5	41	42	43	44	45	46	47	48	49	50
row6	51	52	53	54	55	56	57	58	59	60

row7	61	62	63	64	65	66	67	68	69	70
row8	71	72	73	74	75	76	77	78	79	80
row9	81	82	83	84	85	86	87	88	89	90
row10	91	92	93	94	95	96	97	98	99	100

```
# check DF dimensions
dim(df3)
```

```
[1] 10 10
```

```
nrow(df3)
```

```
[1] 10
```

```
ncol(df3)
```

```
[1] 10
```

```
# check DF type / class
class(df3)
```

```
[1] "data.frame"
```

```
typeof(df3)
```

```
[1] "list"
```

```
# Accessing DF
```

```
# let's create DF - employees
```

```
df_emp <- data.frame(id = 1:6,
  name = c("Max", "Jane", "John", "Tony", "Janis", "Helen"),
  surname = c("Gordon", "Smith", "Don", "Price", "Jett", "Dust"),
  age = c(55, 35, 46, 22, 60, 27),
  date_start_work = c(as.Date("1985-09-01"), as.Date("2010-10-01"), as.Date("2015-01-01"), as.Date("2018-05-01"), as.Date("2019-03-01"), as.Date("2020-07-01")),
  gender = c("M", "F", "M", "M", "F", "M"),
  manager_position = c(T, F, F, F, T, F)
)
```

```
# extract data as data frame (one column) - []  
df_extr <- df_emp["name"]  
df_extr
```

```
      name  
1      Max  
2      Jane  
3      John  
4      Tony  
5     Janis  
6     Helen
```

```
class(df_extr)
```

```
[1] "data.frame"
```

```
# extract data as vector (one column) [[]] $  
df_extr <- df_emp[["age"]]  
df_extr
```

```
[1] 55 35 46 22 60 27
```

```
class(df_extr) # vector factor
```

```
[1] "numeric"
```

```
df_extr <- df_emp$age  
df_extr
```

```
[1] 55 35 46 22 60 27
```

```
class(df_extr) # vector factor
```

```
[1] "numeric"
```



```
# extract multiple columns
df_extr <- df_emp[c("name", "age")]
df_extr
```

```
   name age
1  Max  55
2  Jane 35
3  John 46
4  Tony 22
5 Janis 60
6 Helen 27
```

```
# data frame slicing
df_emp
```

```
   id name surname age date_start_work gender manager_position
1  1  Max  Gordon  55    1985-09-01      M             TRUE
2  2  Jane   Smith  35    2010-10-01      F             FALSE
3  3  John    Don  46    1999-06-01      M             FALSE
4  4  Tony   Price  22    2019-03-01      M             FALSE
5  5 Janis    Jett  60    1980-04-15      F             TRUE
6  6 Helen   Dust  27    2015-02-20      M             FALSE
```

```
#extract second row in name column (1 cell)
df_emp[2,2]
```

```
[1] "Jane"
```

```
df_emp[2,"name"]
```

```
[1] "Jane"
```

```
# extract first 4 rows of last 2 columns
df_emp[1:4, 6:7]
```

```
   gender manager_position
1      M             TRUE
2      F             FALSE
3      M             FALSE
4      M             FALSE
```

```
df_emp[1:4, c("gender", "manager_position")]
```

	gender	manager_position
1	M	TRUE
2	F	FALSE
3	M	FALSE
4	M	FALSE

```
# extract first column (all rows)  
df_emp[,1]
```

```
[1] 1 2 3 4 5 6
```

```
df_emp[, "id"]
```

```
[1] 1 2 3 4 5 6
```

```
df_emp$id
```

```
[1] 1 2 3 4 5 6
```

```
# extract last 2 rows (all columns)  
df_emp[5:6,]
```

	id	name	surname	age	date_start_work	gender	manager_position
5	5	Janis	Jett	60	1980-04-15	F	TRUE
6	6	Helen	Dust	27	2015-02-20	M	FALSE

```
cols <- colnames(df_emp)  
df_emp[5:6, cols]
```

	id	name	surname	age	date_start_work	gender	manager_position
5	5	Janis	Jett	60	1980-04-15	F	TRUE
6	6	Helen	Dust	27	2015-02-20	M	FALSE

```

# Modifying data frame

# append column
df_emp <- cbind(df_emp, role = c("director", "secretary", "analyst", "researcher", "CEO", "an
df_emp$new_col <- 1

# append rows
df_emp <- rbind(df_emp, list(7, "Mark", "Jax", 32, as.Date("2020-01-01"), "M", F, "researcher

# problem with factor variables (new values not in factor levels)
# easy solution - append new row as data frame (rbind 2 data frames)!!!
# will show few rows later

# remove column
df_emp$new_col <- NULL

# remove row
df_emp <- df_emp[-7,]

# merge two data frames (row wise)
df_new_emp <- data.frame(id = 7,
                        name = "Mark",
                        surname = "Jax",
                        age = 32,
                        date_start_work = as.Date("2020-01-01"),
                        gender = "M",
                        manager_position = F,
                        role = "researcher")

df_emp <- rbind(df_emp, df_new_emp)

# merge two data frames (column wise)
df_attr <- data.frame(eye_color = c("blue", "green", "brown", "hazel", "blue", "brown", "bro
                        hair_color = c("blonde", "light brown", "black", "brown", "blonde", "d
df_emp <- cbind(df_emp, df_attr)

# Tips

# Df summary
summary(df_emp)

```

id		name		surname		age	
Min.	:1.0	Length:7		Length:7		Min.	:22.00
1st Qu.	:2.5	Class :character		Class :character		1st Qu.	:29.50
Median	:4.0	Mode :character		Mode :character		Median	:35.00
Mean	:4.0					Mean	:39.57
3rd Qu.	:5.5					3rd Qu.	:50.50
Max.	:7.0					Max.	:60.00

date_start_work		gender		manager_position		role	
Min.	:1980-04-15	Length:7		Mode :logical		Length:7	
1st Qu.	:1992-07-16	Class :character		FALSE:5		Class :character	
Median	:2010-10-01	Mode :character		TRUE :2		Mode :character	
Mean	:2004-05-06						
3rd Qu.	:2017-02-24						
Max.	:2020-01-01						

eye_color		hair_color	
Length:7		Length:7	
Class :character		Class :character	
Mode :character		Mode :character	

```
# rows subsetting
subset(x = df_emp, gender == "M")
```

	id	name	surname	age	date_start_work	gender	manager_position	role
1	1	Max	Gordon	55	1985-09-01	M	TRUE	director
3	3	John	Don	46	1999-06-01	M	FALSE	analyst
4	4	Tony	Price	22	2019-03-01	M	FALSE	researcher
6	6	Helen	Dust	27	2015-02-20	M	FALSE	analyst
7	7	Mark	Jax	32	2020-01-01	M	FALSE	researcher

	eye_color	hair_color
1	blue	blonde
3	brown	black
4	hazel	brown
6	brown	dark brown
7	brown	brown

```
subset(x = df_emp, gender == "F" & manager_position == T)
```

	id	name	surname	age	date_start_work	gender	manager_position	role	eye_color
--	----	------	---------	-----	-----------------	--------	------------------	------	-----------

```

5 5 Janis Jett 60 1980-04-15 F TRUE CEO blue
   hair_color
5 blonde

```

```

rows <- which(df_emp[, "gender"] == "M")
df_emp[rows,]

```

```

   id name surname age date_start_work gender manager_position role
1  1  Max  Gordon  55 1985-09-01      M      TRUE  director
3  3  John   Don  46 1999-06-01      M     FALSE  analyst
4  4  Tony  Price  22 2019-03-01      M     FALSE researcher
6  6 Helen  Dust  27 2015-02-20      M     FALSE  analyst
7  7  Mark   Jax  32 2020-01-01      M     FALSE researcher
   eye_color hair_color
1    blue    blonde
3   brown    black
4   hazel    brown
6   brown dark brown
7   brown    brown

```

```

rows <- which(df_emp[, "gender"] == "F" & df_emp[, "manager_position"] == T)
df_emp[rows,]

```

```

   id name surname age date_start_work gender manager_position role eye_color
5  5 Janis  Jett  60 1980-04-15      F      TRUE  CEO    blue
   hair_color
5 blonde

```

```

# some calculations regarding data frames
nr_managers <- sum(df_emp$manager_position)
mean_age <- mean(df_emp$age)
df_emp$name_surname <- paste(df_emp$name, df_emp$surname, sep = " ") # merge name and surname

# use apply to sum over columns (age, manager_position)
apply(df_emp[, c("age", "manager_position")], 2, sum)

```

```

   age manager_position
277                2

```

19 Factors

Factors are used to represent categorical data. They store both the data values and the corresponding levels.

```
gender_factor <- factor(c("Male", "Female", "Male"))

# Display the factor and its levels
print(gender_factor)
levels(gender_factor)

# create factor variable (gender)
gender <- factor(x = c("male", "female", "female"))

# check new variable
gender
str(gender)
class(gender)
typeof(gender)

# create with ordering
gender <- factor(x = c("male", "female", "female"), ordered = T)
is.ordered(gender)

# check levels
levels(gender) # order of levels based on variable (string alphabetic order)

# we can define our own levels (custom levels order)
gender <- factor(x = c("male", "female", "female"), levels = c("male", "female"), ordered = T)
gender
levels(gender)

# factor properties
levels(gender)
is.factor(gender)
is.ordered(gender)
```

```
# create other object to factor
strings <- c("a", "b", "a", "c")
f_strings <- factor(strings)

#f_string
```

20 Arrays

Arrays are similar to matrices but can have more than two dimensions.

```
# Creating a 3-dimensional array
my_array <- array(1:24, dim = c(3, 4, 2))

# Accessing elements
my_array[1, 2, 1] # Access the element in the first dimension, second row, and first slice
```

```
[1] 4
```


21 Summary

- **Vector:** One-dimensional, homogeneous.
- **List:** One-dimensional, heterogeneous.
- **Matrix:** Two-dimensional, homogeneous.
- **Data Frame:** Two-dimensional, heterogeneous (columns can be different types).
- **Factor:** Categorical data representation.
- **Array:** Multi-dimensional, homogeneous.

22 Manipulating Vectors, Data Frames, and Lists

```
library(haven)  
library(dplyr)
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

```
library(tidyr)
```

23 Vectors: Indexing & Vectorized Ops

```
#Creating vector
x=1:10
x=10:1
x=-5:10
x=c(1:10)
x=c(-5:10)
x=c(1,2,3,4,5,6,7,8,9,10)

# Naming a vector
a=c(1:3)
names(a) #Returns null
```

NULL

```
names(a)=c("one","two","three")
names(a)
```

```
[1] "one"  "two"  "three"
```

```
a
```

```
  one  two three
    1    2    3
```

```
#2)Accessing vector element
x=c(1,3,5,7)
x[2]
```

```
[1] 3
```

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

```
[1] 1 5
```

```
x[-2]
```

```
[1] 1 5 7
```

```
x[-c(1,3)]
```

```
[1] 3 7
```

```
x[0]
```

```
numeric(0)
```

```
y=x[10]  
class(y)
```

```
[1] "numeric"
```

```
#Note: X[0],x[10],output is numeric class only  
#*****  
#3)Modification of Vector elements  
x=c(9,3,5,7)  
x[2]=13  
x[-c(2,3)]=c(11,17)  
x[-1]=c(110,170,70)  
  
#*****  
x=c(11,3,5,7)  
x[9]=x[7]  
#*****  
x=c(11,3,5,7)  
x[9]=x[2]  
  
x=c(1,3,5,7)  
x[2]=x[11]
```

```
#####
x=c(1,3,5,7)
x[c(2,5)]=x[c(4,4)] # It assign 4 element of to 2nd element and 4th element to 5th element
##### from x=1,3,5,7
x=c(11,3,5,7)
x[c(2,7)]=x[c(1,3)]

x=c(1,3,5,7)
x[c(2,3)]=x[c(1,10)]

#4)Airthematic Operations on Vector
x=c(1,3,5,7)
x+10
```

```
[1] 11 13 15 17
```

```
x-5
```

```
[1] -4 -2 0 2
```

```
x*10
```

```
[1] 10 30 50 70
```

```
x/10
```

```
[1] 0.1 0.3 0.5 0.7
```

```
x=c(1,3,5,7)
x%%2
```

```
[1] 0 1 2 3
```

```
x=c(1,3,5,7)
x%%2
```

```
[1] 1 1 1 1
```

```
min(x)
```

```
[1] 1
```

```
max(x)
```

```
[1] 7
```

```
median(x)
```

```
[1] 4
```

```
mean(x)
```

```
[1] 4
```

```
range(x)
```

```
[1] 1 7
```

```
var(x)
```

```
[1] 6.666667
```

```
sd(x)
```

```
[1] 2.581989
```

```
quantile(x)
```

```
 0%  25%  50%  75% 100%  
1.0  2.5  4.0  5.5  7.0
```

```
quantile(1:20,probs=c(.25,0.9))
```

```
 25%   90%  
5.75 18.10
```

```
IQR(x)
```

```
[1] 3
```

```
#5) "WHICH" function  
x=c(2,3,4,5,11,112,133,33)  
x>5
```

```
[1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE
```

```
x=c(2,3,4,5,11,112,133,33)  
y=which(x>5) #Returns the position, not values  
y=x[which(x>5)]  
y=x[x>5]  
  
x=c(2,3,4,5,11,112,133,33)  
min(x)
```

```
[1] 2
```

```
which(x==min(x))
```

```
[1] 1
```

```
x[which(x==min(x))] #Returns the value
```

```
[1] 2
```

```
which.min(x) #Returns the position, not values
```

```
[1] 1
```

```
x=c(2,3,4,5,11,112,133,33)  
max(x)
```

```
[1] 133
```

```
which(x==max(x)) #Returns the position,not values
```

```
[1] 7
```

```
x[which(x==max(x))]#Returns the value
```

```
[1] 133
```

```
which.max(x)#Returns the position,not values
```

```
[1] 7
```

```
x=c(8,7,4,5,11,112,133,33)
which(x>2 & x<5)
```

```
[1] 3
```

```
x[which(x>2 & x<5)]
```

```
[1] 4
```

```
x=c(8,7,4,5,11,112,133,33)
which(x>7 | x<12)
```

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

```
x[which(x>7 | x<12)]
```

```
[1] 8 7 4 5 11 112 133 33
```

```
#6)"REP" function
x=rep(1:5,times=10)
x=rep(100,times=10)
x=rep(c(3,6),times=4)
x=rep("Kummam",times=5)
x=rep(c("Ramesh","Kummam"),times=3)
```



```

x=rep(1:4,5:8)
#x=rep(1:4,1:2) #output as invalid argument
x=rep(1:4,c(2,3,5,7))
x=rep(1:4,each=3)
x=rep(1:4,each=2,times=3)

#7) "SEQ" function
x=seq(from=1,to=10,by=3)
#x=seq(from=1,to=10,by=-3) # wrong arguments
x=seq(from=10,to=1,by=-3)
x=seq(from=1,to=10,length=100)
x=seq(from=1,by=2,length=100)
y=seq(from=1,by=3,length=50)
z=c(x,y)

#8) seq_len() & seq_along() functions
x=c(8,7,4,5,11,112,133,33)
length(x)

```

```
[1] 8
```

```
seq_len(length(x))
```

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

```
seq_along(x) #Returns length of the object
```

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

```

#9) Dealing with missing values
x=c(11,3,5,7)
x[2]=NA
x[c(2,3)]=NA
is.na(x) #Output is a logical vector

```

```
[1] FALSE TRUE TRUE FALSE
```

```
x[!is.na(x)]
```

```
[1] 11 7
```

```
na.omit(x)
```

```
[1] 11 7  
attr(,"na.action")  
[1] 2 3  
attr(,"class")  
[1] "omit"
```

```
#Note: Observe the below operations carefully
```

```
x=c(1,NA,5,NA)  
#x==NA # not followed it so far  
#x[x==NA]# not followed it so far
```

```
#10) Naming a vector
```

```
x=c(1:3)  
names(x)=c("a","b","c")  
x[c("a","b")]
```

```
a b  
1 2
```

```
y=c("Ramesh","Kummam")  
names(y)=c("First","Last")  
y[c("Last","First")]
```

```
      Last      First  
"Kummam" "Ramesh"
```

```
#9) Checking the availability of elements in a vector
```

```
a=c(1:10)  
b=c(5:15)  
1 %in% a
```

```
[1] TRUE
```

```
1 %in% b
```

```
[1] FALSE
```

```
a %in% b
```

```
[1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
b %in% a
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
```

```
is.element(a,b)
```

```
[1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
is.element(b,a)
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
```

```
# print strings  
print("string")
```

```
[1] "string"
```

```
# concatenate strings "a" + "b" = "ab"  
paste("a", "b", sep = "")
```

```
[1] "ab"
```

```
# paste objects of different length  
paste("i", 1:10, sep = ".")
```

```
[1] "i.1" "i.2" "i.3" "i.4" "i.5" "i.6" "i.7" "i.8" "i.9" "i.10"
```

```
paste(c("i","j", "k"), 1:10, sep=".")
```

```
[1] "i.1" "j.2" "k.3" "i.4" "j.5" "k.6" "i.7" "j.8" "k.9" "i.10"
```

```
# paste with collapsing
```

```
paste(c("i","j", "k"), 1:3, sep = "", collapse = "")
```

```
[1] "i1j2k3"
```

```
# paste without collapsing
```

```
paste(c("i","j", "k"), 1:3, sep = "")
```

```
[1] "i1" "j2" "k3"
```

```
# paste0() shorter version of paste(..., sep="")
```

```
paste0("Hello", "world", ",", "I", "use", "R")
```

```
[1] "Helloworld,IuseR"
```

```
paste("Hello", "world", ",", "I", "use", "R", sep=" ")
```

```
[1] "Hello world , I use R"
```

```
# concatenate strings with cat()
```

```
cat("Hello", "world", "!")
```

```
Hello world !
```

```
# it prints without quotes !!!
```

```
cat("Hello", "world", "!", sep = "/")
```

```
Hello/world/!
```

```
# counting number of characters nchar()
```

```
nchar("Hello world")
```

```
[1] 11
```

```
# load US states names from data frame regarding crime rate
df <- USArrests
head(df)
```

	Murder	Assault	UrbanPop	Rape
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0
Arkansas	8.8	190	50	19.5
California	9.0	276	91	40.6
Colorado	7.9	204	78	38.7

```
rownames(df)
```

[1] "Alabama"	"Alaska"	"Arizona"	"Arkansas"
[5] "California"	"Colorado"	"Connecticut"	"Delaware"
[9] "Florida"	"Georgia"	"Hawaii"	"Idaho"
[13] "Illinois"	"Indiana"	"Iowa"	"Kansas"
[17] "Kentucky"	"Louisiana"	"Maine"	"Maryland"
[21] "Massachusetts"	"Michigan"	"Minnesota"	"Mississippi"
[25] "Missouri"	"Montana"	"Nebraska"	"Nevada"
[29] "New Hampshire"	"New Jersey"	"New Mexico"	"New York"
[33] "North Carolina"	"North Dakota"	"Ohio"	"Oklahoma"
[37] "Oregon"	"Pennsylvania"	"Rhode Island"	"South Carolina"
[41] "South Dakota"	"Tennessee"	"Texas"	"Utah"
[45] "Vermont"	"Virginia"	"Washington"	"West Virginia"
[49] "Wisconsin"	"Wyoming"		

```
states <- rownames(df)

# convert all states names to upper case
states_upper <- toupper(states)

# convert all names to lower
states_lower <- tolower(states)

# or select which to apply with function casefold
casefold(x = states, upper = T)
```

[1] "ALABAMA"	"ALASKA"	"ARIZONA"	"ARKANSAS"
---------------	----------	-----------	------------

[5]	"CALIFORNIA"	"COLORADO"	"CONNECTICUT"	"DELAWARE"
[9]	"FLORIDA"	"GEORGIA"	"HAWAII"	"IDAHO"
[13]	"ILLINOIS"	"INDIANA"	"IOWA"	"KANSAS"
[17]	"KENTUCKY"	"LOUISIANA"	"MAINE"	"MARYLAND"
[21]	"MASSACHUSETTS"	"MICHIGAN"	"MINNESOTA"	"MISSISSIPPI"
[25]	"MISSOURI"	"MONTANA"	"NEBRASKA"	"NEVADA"
[29]	"NEW HAMPSHIRE"	"NEW JERSEY"	"NEW MEXICO"	"NEW YORK"
[33]	"NORTH CAROLINA"	"NORTH DAKOTA"	"OHIO"	"OKLAHOMA"
[37]	"OREGON"	"PENNSYLVANIA"	"RHODE ISLAND"	"SOUTH CAROLINA"
[41]	"SOUTH DAKOTA"	"TENNESSEE"	"TEXAS"	"UTAH"
[45]	"VERMONT"	"VIRGINIA"	"WASHINGTON"	"WEST VIRGINIA"
[49]	"WISCONSIN"	"WYOMING"		

```
casefold(x = states, upper = F)
```

[1]	"alabama"	"alaska"	"arizona"	"arkansas"
[5]	"california"	"colorado"	"connecticut"	"delaware"
[9]	"florida"	"georgia"	"hawaii"	"idaho"
[13]	"illinois"	"indiana"	"iowa"	"kansas"
[17]	"kentucky"	"louisiana"	"maine"	"maryland"
[21]	"massachusetts"	"michigan"	"minnesota"	"mississippi"
[25]	"missouri"	"montana"	"nebraska"	"nevada"
[29]	"new hampshire"	"new jersey"	"new mexico"	"new york"
[33]	"north carolina"	"north dakota"	"ohio"	"oklahoma"
[37]	"oregon"	"pennsylvania"	"rhode island"	"south carolina"
[41]	"south dakota"	"tennessee"	"texas"	"utah"
[45]	"vermont"	"virginia"	"washington"	"west virginia"
[49]	"wisconsin"	"wyoming"		

```
# character translation
chartr(old = "o", new = "0", x = "Hello World")
```

```
[1] "Hello W0rld"
```

```
# sorting strings
sort(states, decreasing = F) #ascending order
```

[1]	"Alabama"	"Alaska"	"Arizona"	"Arkansas"
[5]	"California"	"Colorado"	"Connecticut"	"Delaware"
[9]	"Florida"	"Georgia"	"Hawaii"	"Idaho"

[13]	"Illinois"	"Indiana"	"Iowa"	"Kansas"
[17]	"Kentucky"	"Louisiana"	"Maine"	"Maryland"
[21]	"Massachusetts"	"Michigan"	"Minnesota"	"Mississippi"
[25]	"Missouri"	"Montana"	"Nebraska"	"Nevada"
[29]	"New Hampshire"	"New Jersey"	"New Mexico"	"New York"
[33]	"North Carolina"	"North Dakota"	"Ohio"	"Oklahoma"
[37]	"Oregon"	"Pennsylvania"	"Rhode Island"	"South Carolina"
[41]	"South Dakota"	"Tennessee"	"Texas"	"Utah"
[45]	"Vermont"	"Virginia"	"Washington"	"West Virginia"
[49]	"Wisconsin"	"Wyoming"		

```
sort(states, decreasing = T) #descending order
```

[1]	"Wyoming"	"Wisconsin"	"West Virginia"	"Washington"
[5]	"Virginia"	"Vermont"	"Utah"	"Texas"
[9]	"Tennessee"	"South Dakota"	"South Carolina"	"Rhode Island"
[13]	"Pennsylvania"	"Oregon"	"Oklahoma"	"Ohio"
[17]	"North Dakota"	"North Carolina"	"New York"	"New Mexico"
[21]	"New Jersey"	"New Hampshire"	"Nevada"	"Nebraska"
[25]	"Montana"	"Missouri"	"Mississippi"	"Minnesota"
[29]	"Michigan"	"Massachusetts"	"Maryland"	"Maine"
[33]	"Louisiana"	"Kentucky"	"Kansas"	"Iowa"
[37]	"Indiana"	"Illinois"	"Idaho"	"Hawaii"
[41]	"Georgia"	"Florida"	"Delaware"	"Connecticut"
[45]	"Colorado"	"California"	"Arkansas"	"Arizona"
[49]	"Alaska"	"Alabama"		

```
# extracting parts of string
# sub string first 3 letters from state name Alabama
substr(x = "Alabama", start = 1, stop = 3)
```

```
[1] "Ala"
```

```
# String matching - back to toy example
help(regex)

# get all country names
#install.packages("countrycode")
require(countrycode)
```

```
Loading required package: countrycode
```

```
countries <- as.vector(countrycode::codelist$country.name.en)
```

```
# countries beginning with letter "A"
grep(pattern = "^A", x = countries)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

```
countries[grep(pattern = "^A", x = countries)]
```

```
[1] "Afghanistan"      "Albania"           "Algeria"
[4] "American Samoa"   "Andorra"           "Angola"
[7] "Anguilla"         "Antarctica"        "Antigua & Barbuda"
[10] "Argentina"        "Armenia"           "Aruba"
[13] "Australia"        "Austria"           "Austria-Hungary"
[16] "Azerbaijan"
```

```
countries[grep(pattern = "^A", x = countries)]
```

```
[1] "Afghanistan"      "Albania"           "Algeria"
[4] "American Samoa"   "Andorra"           "Angola"
[7] "Anguilla"         "Antarctica"        "Antigua & Barbuda"
[10] "Argentina"        "Armenia"           "Aruba"
[13] "Australia"        "Austria"           "Austria-Hungary"
[16] "Azerbaijan"
```

```
# all country names that end with letter "y"
rez <- grep(pattern = "*y$", x = countries)
countries[rez]
```

```
[1] "Austria-Hungary"      "British Indian Ocean Territory"
[3] "Germany"             "Guernsey"
[5] "Hungary"             "Italy"
[7] "Jersey"              "Norway"
[9] "Paraguay"            "Saxony"
[11] "St. Barthélemy"      "Turkey"
[13] "Tuscany"             "Uruguay"
[15] "Vatican City"
```



```
# all country with 2 words or more for a country name
rez <- grep(pattern = "\\w\\s\\w", x = countries)
countries[rez]
```

```
[1] "American Samoa"
[2] "Bouvet Island"
[3] "British Indian Ocean Territory"
[4] "British Virgin Islands"
[5] "Burkina Faso"
[6] "Cape Verde"
[7] "Caribbean Netherlands"
[8] "Cayman Islands"
[9] "Central African Republic"
[10] "Channel Islands"
[11] "Christmas Island"
[12] "Cook Islands"
[13] "Costa Rica"
[14] "Côte d'Ivoire"
[15] "Dominican Republic"
[16] "El Salvador"
[17] "Equatorial Guinea"
[18] "Falkland Islands"
[19] "Faroe Islands"
[20] "French Guiana"
[21] "French Polynesia"
[22] "French Southern Territories"
[23] "German Democratic Republic"
[24] "Heard & McDonald Islands"
[25] "Hesse Electoral"
[26] "Hesse Grand Ducal"
[27] "Hong Kong SAR China"
[28] "Isle of Man"
[29] "Macao SAR China"
[30] "Marshall Islands"
[31] "Mecklenburg Schwerin"
[32] "Micronesia (Federated States of)"
[33] "Netherlands Antilles"
[34] "New Caledonia"
[35] "New Zealand"
[36] "Norfolk Island"
[37] "North Korea"
[38] "North Macedonia"
```

```

[39] "Northern Mariana Islands"
[40] "Orange Free State"
[41] "Palestinian Territories"
[42] "Papua New Guinea"
[43] "Pitcairn Islands"
[44] "Puerto Rico"
[45] "Republic of Vietnam"
[46] "Saint Martin (French part)"
[47] "San Marino"
[48] "Saudi Arabia"
[49] "Serbia and Montenegro"
[50] "Sierra Leone"
[51] "Sint Maarten"
[52] "Solomon Islands"
[53] "South Africa"
[54] "South Georgia & South Sandwich Islands"
[55] "South Korea"
[56] "South Sudan"
[57] "Sri Lanka"
[58] "Svalbard & Jan Mayen"
[59] "São Tomé & Príncipe"
[60] "Turks & Caicos Islands"
[61] "Two Sicilies"
[62] "U.S. Virgin Islands"
[63] "United Arab Emirates"
[64] "United Arab Republic"
[65] "United Kingdom"
[66] "United Province CA"
[67] "United States"
[68] "United States Minor Outlying Islands (the)"
[69] "Vatican City"
[70] "Western Sahara"
[71] "Yemen Arab Republic"
[72] "Yemen People's Republic"
[73] "Åland Islands"

```

```

# all country names that end with letter "e" or "i"
rez <- grep(pattern = "*e$|*i$", x = countries)
countries[rez]

```

```

[1] "Belize"           "Brunei"           "Burundi"
[4] "Cape Verde"       "Chile"             "Congo - Brazzaville"

```

```

[7] "Côte d'Ivoire"      "Djibouti"          "Eswatini"
[10] "Fiji"               "France"            "Greece"
[13] "Guadeloupe"         "Haiti"             "Kiribati"
[16] "Malawi"             "Mali"              "Martinique"
[19] "Mayotte"           "Mozambique"        "Niue"
[22] "Orange Free State"  "Sierra Leone"     "Singapore"
[25] "Suriname"          "São Tomé & Príncipe" "Timor-Leste"
[28] "Ukraine"           "Zimbabwe"

```

```

# all country names which contain combination of letters "gin"
rez <- grep(pattern = "*(gin)", x = countries)
countries[rez]

```

```

[1] "British Virgin Islands" "U.S. Virgin Islands"

```

```

# metacharacters and double backslash sign
strings <- c("dollar $", "dollar", "US dollar")

# how to escape metacharacters in R
# we would like to match a word with "$" dollar sign
strings

```

```

[1] "dollar $" "dollar" "US dollar"

```

```

rez1 <- grep(pattern = "$", x = strings) # wrong way all are find $ indicating end of string
strings[rez1]

```

```

[1] "dollar $" "dollar" "US dollar"

```

```

# we need to escape $ with \ backslash
#rez2 <- grep(pattern = "\$", x = strings) # wrong way error
#strings[rez2]

# right way $ escape with \\ double backslash
#rez3 <- grep(pattern = "\\$", x = strings) # wrong way error
#strings[rez3]

# Escape dot .
strings <-c("word.word", "word word")
rez <- grep(pattern = "\\.", x = strings)
strings[rez]

```

```
[1] "word.word"
```

```
# the hat sign beginning of the string and dollar sign end of string
strings <- c("Word", "worD")
strings[grepl("^W",strings)]
```

```
[1] "Word"
```

```
strings[grepl("D$",strings)]
```

```
[1] "worD"
```

```
# example of an anchor sequences in R
strings <- c("123", "onetwothree", "1twothree")
strings[grepl("\\d", strings)] # digit character
```

```
[1] "123"          "1twothree"
```

```
strings[grepl("\\D", strings)] # non-digit character
```

```
[1] "onetwothree" "1twothree"
```

```
# example of a character classes
strings <- c("123", "dollar", "shkjl")
strings[grepl("[aeiou]", strings)] # match word with any vowel
```

```
[1] "dollar"
```

```
strings[grepl("[0-9]", strings)] # match word with digits
```

```
[1] "123"
```

```
# find and replace sub() & gsub()
string <- "I have started to learn R, and I already love R."

# replace "R" with "X"
sub(pattern = "R", replacement = "X", x = string) #only first occurrence replaced
```

```
[1] "I have started to learn X, and I already love R."
```

```
gsub(pattern = "R", replacement = "X", x = string) #all occurrences replaced
```

```
[1] "I have started to learn X, and I already love X."
```

```
#replace white space with blank space  
gsub(pattern = "\\s", replacement = "", x = string)
```

```
[1] "IhavestartedtolearnR,andIalreadyloveR."
```

```
# string split, split given sentence by comma ","  
strsplit(x = string, split = ",")
```

```
[[1]]  
[1] "I have started to learn R" " and I already love R."
```

```
# split phone numbers by digits  
numbers <- c("310-555-123", "311-444-456")  
strsplit(x = numbers, split = "-")
```

```
[[1]]  
[1] "310" "555" "123"
```

```
[[2]]  
[1] "311" "444" "456"
```

24 Data Frames with dplyr

```
# create data frame
df1 <- data.frame(col1 = 1:3,
                  col2 = c("a", "b", "c"),
                  col3 = c(T, F, T),
                  col4 = c(as.Date("2020-01-01"), as.Date("2020-01-03"), as.Date("2020-01-03")))

# create data frame - vectors
col1 <- seq(10,100,10)
col2 <- seq(as.Date("2020-01-01"), length = 10, by = "weeks")
col3 <- rep("word", 10)

df2 <- data.frame(num = col1,
                  date = col2,
                  string = col3)

# check DF structure
str(df2)
```

```
'data.frame':  10 obs. of  3 variables:
 $ num      : num  10 20 30 40 50 60 70 80 90 100
 $ date     : Date, format: "2020-01-01" "2020-01-08" ...
 $ string   : chr  "word" "word" "word" "word" ...
```

```
# create data frame - matrix
M <- matrix(data = 1:100, nrow = 10, ncol = 10, byrow = T)
rownames(M) <- paste("row", 1:10, sep = "")
colnames(M) <- paste("col", 1:10, sep = "")
M
```

	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10
row1	1	2	3	4	5	6	7	8	9	10
row2	11	12	13	14	15	16	17	18	19	20

row3	21	22	23	24	25	26	27	28	29	30
row4	31	32	33	34	35	36	37	38	39	40
row5	41	42	43	44	45	46	47	48	49	50
row6	51	52	53	54	55	56	57	58	59	60
row7	61	62	63	64	65	66	67	68	69	70
row8	71	72	73	74	75	76	77	78	79	80
row9	81	82	83	84	85	86	87	88	89	90
row10	91	92	93	94	95	96	97	98	99	100

```
df3 <- as.data.frame(M)
df3
```

	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10
row1	1	2	3	4	5	6	7	8	9	10
row2	11	12	13	14	15	16	17	18	19	20
row3	21	22	23	24	25	26	27	28	29	30
row4	31	32	33	34	35	36	37	38	39	40
row5	41	42	43	44	45	46	47	48	49	50
row6	51	52	53	54	55	56	57	58	59	60
row7	61	62	63	64	65	66	67	68	69	70
row8	71	72	73	74	75	76	77	78	79	80
row9	81	82	83	84	85	86	87	88	89	90
row10	91	92	93	94	95	96	97	98	99	100

```
# check DF dimensions
dim(df3)
```

```
[1] 10 10
```

```
nrow(df3)
```

```
[1] 10
```

```
ncol(df3)
```

```
[1] 10
```

```
# check DF type / class
class(df3)
```

```
[1] "data.frame"
```

```
typeof(df3)
```

```
[1] "list"
```

```
# 4.3.2 Accessing DF
```

```
# let's create DF - employees
```

```
df_emp <- data.frame(id = 1:6,  
                     name = c("Max", "Jane", "John", "Tony", "Janis", "Helen"),  
                     surname = c("Gordon", "Smith", "Don", "Price", "Jett", "Dust"),  
                     age = c(55, 35, 46, 22, 60, 27),  
                     date_start_work = c(as.Date("1985-09-01"), as.Date("2010-10-01"), as.Date("2015-01-01"),  
                                          as.Date("2012-03-01"), as.Date("2018-05-01"), as.Date("2019-08-01")),  
                     gender = c("M", "F", "M", "M", "F", "M"),  
                     manager_position = c(T, F, F, F, T, F)  
                     )
```

```
# extract data as data frame (one column) - []
```

```
df_extr <- df_emp["name"]
```

```
df_extr
```

```
   name  
1   Max  
2  Jane  
3  John  
4  Tony  
5 Janis  
6 Helen
```

```
class(df_extr)
```

```
[1] "data.frame"
```

```
# extract data as vector (one column) [[]] $
```

```
df_extr <- df_emp[["age"]]
```

```
df_extr
```

```
[1] 55 35 46 22 60 27
```



```
class(df_extr) # vector factor
```

```
[1] "numeric"
```

```
df_extr <- df_emp$age  
df_extr
```

```
[1] 55 35 46 22 60 27
```

```
class(df_extr) # vector factor
```

```
[1] "numeric"
```

```
# extract multiple columns  
df_extr <- df_emp[c("name", "age")]  
df_extr
```

```
      name age  
1    Max  55  
2   Jane  35  
3   John  46  
4   Tony  22  
5  Janis  60  
6  Helen  27
```

```
# data frame slicing  
df_emp
```

```
   id  name surname age date_start_work gender manager_position  
1  1   Max  Gordon  55    1985-09-01      M             TRUE  
2  2  Jane   Smith  35    2010-10-01      F             FALSE  
3  3  John    Don   46    1999-06-01      M             FALSE  
4  4  Tony   Price  22    2019-03-01      M             FALSE  
5  5 Janis    Jett  60    1980-04-15      F             TRUE  
6  6 Helen   Dust  27    2015-02-20      M             FALSE
```

```
#extract second row in name column (1 cell)
df_emp[2,2]
```

```
[1] "Jane"
```

```
df_emp[2,"name"]
```

```
[1] "Jane"
```

```
# extract first 4 rows of last 2 columns
df_emp[1:4, 6:7]
```

	gender	manager_position
1	M	TRUE
2	F	FALSE
3	M	FALSE
4	M	FALSE

```
df_emp[1:4, c("gender", "manager_position")]
```

	gender	manager_position
1	M	TRUE
2	F	FALSE
3	M	FALSE
4	M	FALSE

```
# extract first column (all rows)
df_emp[,1]
```

```
[1] 1 2 3 4 5 6
```

```
df_emp[,"id"]
```

```
[1] 1 2 3 4 5 6
```

```
df_emp$id
```

```
[1] 1 2 3 4 5 6
```

```
# extract last 2 rows (all columns)
df_emp[5:6,]
```

	id	name	surname	age	date_start_work	gender	manager_position
5	5	Janis	Jett	60	1980-04-15	F	TRUE
6	6	Helen	Dust	27	2015-02-20	M	FALSE

```
cols <- colnames(df_emp)
df_emp[5:6, cols]
```

	id	name	surname	age	date_start_work	gender	manager_position
5	5	Janis	Jett	60	1980-04-15	F	TRUE
6	6	Helen	Dust	27	2015-02-20	M	FALSE

```
# 4.3.3 Modifying data frame
```

```
# append column
```

```
df_emp <- cbind(df_emp, role = c("director", "secretary", "analyst", "researcher", "CEO", "analyst"))
df_emp$new_col <- 1
```

```
# append rows
```

```
df_emp <- rbind(df_emp, list(7, "Mark", "Jax", 32, as.Date("2020-01-01"), "M", F, "researcher"))
```

```
# problem with factor variables (new values not in factor levels)
```

```
# easy solution - append new row as data frame (rbind 2 data frames)!!!
```

```
# will show few rows later
```

```
# remove column
```

```
df_emp$new_col <- NULL
```

```
# remove row
```

```
df_emp <- df_emp[-7,]
```

```
# merge two data frames (row wise)
```

```
df_new_emp <- data.frame(id = 7,
```

[illegible]

id		name		surname		age	
Min.	:1.0	Length:7		Length:7		Min.	:22.00
1st Qu.:	2.5	Class :character		Class :character		1st Qu.:	29.50
Median	:4.0	Mode :character		Mode :character		Median	:35.00
Mean	:4.0					Mean	:39.57
3rd Qu.:	5.5					3rd Qu.:	50.50
Max.	:7.0					Max.	:60.00
date_start_work		gender		manager_position		role	
Min.	:1980-04-15	Length:7		Mode :logical		Length:7	
1st Qu.:	1992-07-16	Class :character		FALSE:5		Class :character	
Median	:2010-10-01	Mode :character		TRUE :2		Mode :character	
Mean	:2004-05-06						
3rd Qu.:	2017-02-24						
Max.	:2020-01-01						
eye_color		hair_color					
Length:7		Length:7					
Class :character		Class :character					
Mode :character		Mode :character					

```
# rows subsetting
subset(x = df_emp, gender == "M")
```

	id	name	surname	age	date_start_work	gender	manager_position	role
1	1	Max	Gordon	55	1985-09-01	M	TRUE	director
3	3	John	Don	46	1999-06-01	M	FALSE	analyst
4	4	Tony	Price	22	2019-03-01	M	FALSE	researcher
6	6	Helen	Dust	27	2015-02-20	M	FALSE	analyst
7	7	Mark	Jax	32	2020-01-01	M	FALSE	researcher

	eye_color	hair_color
1	blue	blonde
3	brown	black
4	hazel	brown
6	brown	dark brown
7	brown	brown

```
subset(x = df_emp, gender == "F" & manager_position == T)
```

	id	name	surname	age	date_start_work	gender	manager_position	role	eye_color
5	5	Janis	Jett	60	1980-04-15	F	TRUE	CEO	blue

	hair_color
5	blonde

```
rows <- which(df_emp[, "gender"] == "M")
df_emp[rows,]
```

	id	name	surname	age	date_start_work	gender	manager_position	role
1	1	Max	Gordon	55	1985-09-01	M	TRUE	director
3	3	John	Don	46	1999-06-01	M	FALSE	analyst
4	4	Tony	Price	22	2019-03-01	M	FALSE	researcher
6	6	Helen	Dust	27	2015-02-20	M	FALSE	analyst
7	7	Mark	Jax	32	2020-01-01	M	FALSE	researcher

	eye_color	hair_color
1	blue	blonde
3	brown	black
4	hazel	brown
6	brown	dark brown
7	brown	brown

```
rows <- which(df_emp[, "gender"] == "F" & df_emp[, "manager_position"] == T)
df_emp[rows,]
```

```
   id name surname age date_start_work gender manager_position role eye_color
5  5 Janis      Jett  60      1980-04-15      F              TRUE   CEO      blue
   hair_color
5      blonde
```

```
# some calculations regarding data frames
nr_managers <- sum(df_emp$manager_position)
mean_age <- mean(df_emp$age)
df_emp$name_surname <- paste(df_emp$name, df_emp$surname, sep = " ") # merge name and surname

# use apply to sum over columns (age, manager_position)
apply(df_emp[, c("age", "manager_position")], 2, sum)
```

```
      age manager_position
      277                2
```

```
# if statement

x <- 3
y <- 14

if(x < y){
  print("x is less than y")
}
```

```
[1] "x is less than y"
```

```
# if-else statement

x <- 3
y <- 14

if(x > y){
  print("x is greater than y")
} else{
  print("x is less or equal to y")
}
```

```
[1] "x is less or equal to y"
```

```
# if-else if-else statement

x <- 14
y <- 14

if(x > y){
  print("x is greater than y")
} else if (x < y){
  print("x is less than y")
} else{
  print("x is equal to y")
}
```

```
[1] "x is equal to y"
```

```
# 5.1.2 relational operators

# scalar
x <- 3

x == 4
```

```
[1] FALSE
```

```
x > 0
```

```
[1] TRUE
```

```
x < 5
```

```
[1] TRUE
```

```
x <= 3
```

```
[1] TRUE
```

```
x >= 10
```

```
[1] FALSE
```

```
x %in% c(1,2,3,4,5)
```

```
[1] TRUE
```

```
# vectors
X <- c(F,T,0,10)
Y <- c(T,F,F,T)

X == Y # element-wise comparison
```

```
[1] FALSE FALSE TRUE FALSE
```

```
X > Y
```

```
[1] FALSE TRUE FALSE TRUE
```

```
X < Y
```

```
[1] TRUE FALSE FALSE FALSE
```

```
X >= Y
```

```
[1] FALSE TRUE TRUE TRUE
```

```
X <= Y
```

```
[1] TRUE FALSE TRUE FALSE
```

```
X != Y
```

```
[1] TRUE TRUE FALSE TRUE
```



```
X %in% Y
```

```
[1] TRUE TRUE TRUE FALSE
```

```
# 5.1.3 Logical operators
```

```
X | Y
```

```
[1] TRUE TRUE FALSE TRUE
```

```
#X || Y
```

```
X & Y
```

```
[1] FALSE FALSE FALSE TRUE
```

```
#X && Y
```

```
!X
```

```
[1] TRUE FALSE TRUE FALSE
```

```
# logical operators in if statement (flip coin twice)
```

```
c1 <- "H"
```

```
c2 <- "H"
```

```
if(c1 == "H" & c2 == "H"){  
  paste("You win 10$")  
} else if((c1 == "H" & c2 == "T") | (c1 == "T" & c2 == "H")){  
  paste("You win 2$")  
} else{  
  paste("You lose all the money")  
}
```

```
[1] "You win 10$"
```

```
# loop over iterations and print number of iteration
```

```
for(it in 1:10){  
  print(paste("iteration ", it, sep = ""))  
}
```

```
[1] "iteration 1"
[1] "iteration 2"
[1] "iteration 3"
[1] "iteration 4"
[1] "iteration 5"
[1] "iteration 6"
[1] "iteration 7"
[1] "iteration 8"
[1] "iteration 9"
[1] "iteration 10"
```

```
# sum of numbers in a sequence
sequence <- c(1,2,3,4,5)
s <- 0 # initial sum

for(val in sequence){
  s <- s + val
  print(paste("value = ", val, sep = ""))
  print(paste("s = ", s, sep = ""))
  print("-----")
}
```

```
[1] "value = 1"
[1] "s = 1"
[1] "-----"
[1] "value = 2"
[1] "s = 3"
[1] "-----"
[1] "value = 3"
[1] "s = 6"
[1] "-----"
[1] "value = 4"
[1] "s = 10"
[1] "-----"
[1] "value = 5"
[1] "s = 15"
[1] "-----"
```

```
# for loop with conditional statement
# (count number of even numbers in a vector)
x <- c(1,3,2,4,5,10,22,21,100) # given numbers
count <- 0 # counter for even numbers
```

```
for(val in x){  
  if(val %% 2 == 0){ # if number is divisible with 2  
    count <- count + 1  
  }  
}  
  
print(count)
```

```
[1] 5
```

25 Read sas dataset

```
dm <- haven::read_sas("data/sdtm/dm.sas7bdat")  
ae <- haven::read_sas("data/sdtm/ae.sas7bdat")
```

26 keeping selected columns

```
dm_1 <- dm |>
  dplyr::select(USUBJID,ARM) |>
  dplyr::filter(ARM == "Placebo")

dm_2 <- dm |>
  dplyr::select(USUBJID:ARM)

dm_2 <- dm |>
  dplyr::select(1,2,3)

dm_2 <- dm |>
  dplyr::select(1:3)

dm_2 <- dm |>
  dplyr::select(starts_with("AR"))

dm_2 <- dm |>
  dplyr::select(contains("DT"))

dm2 <- dplyr::select(dm,USUBJID,ARM)
```

27 dropping columns

To select all columns except certain ones, put a “-” in front of the variable to exclude it. For multiple variables, you can use the function `c()` to combine values into a vector or list. This will select all the variables in surveys except ARM, SITEID

```
adsl1 <- haven::read_sas("data/adam/adsl.sas7bdat")
adsl2 <- dplyr::select(adsl1, -c(AGE, SEX, TRT01A))
adsl3 <- dplyr::select(adsl1, -c(STUDYID:ARM))
adsl4 <- dplyr::select(adsl1, -c(1:6))
adsl4 <- dplyr::select(adsl1, -c(1:6, 43))
adsl5 <- dplyr::select(adsl1, -starts_with("A"))
adsl6 <- dplyr::select(adsl1, -ends_with("DTC"))
adsl7 <- dplyr::select(adsl1, -contains("TRT"))
```

There are many helper functions available with `select()` like: `starts_with()`, `ends_with()`, `contains()` among others. These were imported from the `tidyselect` package. You can put a “-” in front of the helper function to negate it. Here is an example using `contains()`: # Filtering rows To choose rows based on a specific criteria, use `filter()`: To select rows where the planned treatment code equals “Placebo”:

```
adsl1 <- adsl |>
  dplyr::select(USUBJID, TRT01A) |>
  dplyr::filter(TRT01A == "Placebo" )

adsl2 <- adsl |>
  dplyr::select(USUBJID, TRT01A, SEX, AGE) |>
  dplyr::filter(TRT01A == "Placebo" & SEX == "M" & AGE == 70)

adsl2 <- adsl |>
  dplyr::select(USUBJID, TRT01A, SEX, AGE) |>
  dplyr::filter(TRT01A == "Placebo" & (SEX == "M" | AGE == 70))

adsl2 <- adsl |>
  dplyr::select(USUBJID, TRT01A, SEX, AGE) |>
  dplyr::filter(TRT01A == "Placebo" & SEX == "M" & AGE %in% c(70, 80))
```

```

adsl2 <- adsl |>
  dplyr::select(USUBJID, TRT01A, SEX, AGE) |>
  dplyr::filter(TRT01A=="Placebo" & SEX == "M" & !AGE %in% c(70,80))

adsl2 <- adsl |>
  dplyr::select(USUBJID, TRT01A, SEX, AGE) |>
  dplyr::filter(TRT01A=="Placebo" & !SEX == "M" & !AGE %in% c(70,80))

adsl2 <- adsl |>
  dplyr::select(USUBJID, TRT01A, SEX, AGE) |>
  dplyr::filter(TRT01A %in% c("Placebo", "Xanomeline High Dose") & !SEX == "M" & !AGE %in% c(70,80))

adsl2 <- adsl |>
  dplyr::select(USUBJID, TRT01A, SEX, AGE) |>
  dplyr::filter(TRT01A %in% c("Placebo", "Xanomeline High Dose") & !SEX == "M" & AGE >= 70)

```

28 Rename

```
ads11 <- ads1 |>
  dplyr::rename(usubjid=USUBJID)

ads11 <- ads1 |>
  dplyr::rename(usubjid=USUBJID,
               trt=TRT01P)
```


29 Arrange (sorting)

```
adsl1 <- adsl |>
  dplyr::select(USUBJID,SEX,AGE) |>
  dplyr::arrange(AGE,SEX)

adsl1 <- adsl |>
  dplyr::select(USUBJID,SEX,AGE) |>
  dplyr::arrange(AGE,desc(SEX))
```

30 convert var names to lower case

```
tolower(names(ads1))
```

```
[1] "studyid" "usubjid" "subjid" "siteid" "sitegr1" "arm"  
[7] "trt01p" "trt01pn" "trt01a" "trt01an" "trtsdt" "trtedt"  
[13] "trtdur" "avgdd" "cumdose" "age" "agegr1" "agegr1n"  
[19] "ageu" "race" "racen" "sex" "ethnic" "saffl"  
[25] "ittfl" "efffl" "comp8fl" "comp16fl" "comp24fl" "disconfl"  
[31] "dsraefl" "dthfl" "bmibl" "bmiblgr1" "heightbl" "weightbl"  
[37] "educlvl" "disonsdt" "durdis" "durdsgr1" "visit1dt" "rfstdtc"  
[43] "rfendtc" "visnumen" "rfendt" "dcdecod" "dcreascd" "mmsetot"
```

```
ads11 <- ads1
```

```
names(ads11) <- tolower(names(ads11))
```

31 Creating new variables

```
adsl1 <- adsl |>
  dplyr::mutate(name="Sri Ram")

adsl_test <- adsl |>
  dplyr::mutate(
    AGEGR1_t = dplyr::if_else(AGE < 65, "<65", ">=65"),           # character
    SAFFL_t  = dplyr::if_else(!is.na(TRTSDT), "Y", "N")         # character flag
  ) |>
  dplyr::select(USUBJID,AGE,AGEGR1_t,SAFFL_t,TRTSDT)
```

32 bind rows (set operator in SAS)

```
# bind rows
df_a <- data.frame(
  id = 1:3,
  value = c("A", "B", "C"),
  score = c(10, 20, 30)
)
df_b <- data.frame(
  id = 4:6,
  value = c("D", "E", "F"),
  score = c(40, 50, 60)
)

com <- dplyr::bind_rows(df_a,
                        df_b)

df_combined <- dplyr::bind_rows(df_a, df_b, df_a, .id = "source")

df_combined <- dplyr::bind_rows(df_a, df_b, df_a, .id = "source")

df_combined <- dplyr::bind_rows("dset1"=df_a,
                                "dset2"=df_b,
                                "test"=df_a, .id = "Source_var_name")

df_combined <- dplyr::bind_rows(select(df_a, id, value),
                                df_b)

adsl_tot <- dplyr::bind_rows(adsl |>
                             dplyr::select(USUBJID, TRT01P, TRT01PN),

                             adsl |>
                             dplyr::mutate(TRT01P="Total", TRT01PN=99) |>
                             dplyr::select(USUBJID, TRT01P, TRT01PN)
)
```

33 recode (similar to PROC FORMAT in SAS)

```
sex_fm <- c("M"="Male",
           "F"="Female"
)
adsl_test1 <- adsl |>
  dplyr::mutate(gender=dplyr::recode(SEX,!!!sex_fm))

adsl_test2 <- adsl |>
  dplyr::mutate(SEX=dplyr::recode(SEX,"M"="Male",
                                "F"="Female"))

adsl_test <- adsl |>
  dplyr::mutate(
    gn = dplyr::if_else(SEX=="M","Male","Female") # character
  ) |>
  dplyr::select(USUBJID,SEX,gn) |>
  head(5)
```

34 joins

The dplyr package provides functions to join tables based on shared keys, such as patient IDs. The main types of joins include:

- `inner_join()`
- `left_join()`
- `right_join()`
- `full_join()`
- `semi_join()`
- `anti_join()`

```
# Patient demographics data
patients <- data.frame(
  patient_id = c(101, 102, 103, 104),
  age = c(34, 45, 29, 56),
  gender = c("F", "M", "M", "F")
)

# Patient lab results data
lab_results <- data.frame(
  patient_id = c(103, 104, 105, 106),
  test = c("CBC", "Lipid Panel", "Blood Glucose", "CBC"),
  result = c("Normal", "High Cholesterol", "Elevated", "Normal")
)

# Display the data frames
patients
```

	patient_id	age	gender
1	101	34	F
2	102	45	M
3	103	29	M
4	104	56	F

```
lab_results
```

	patient_id	test	result
1	103	CBC	Normal
2	104	Lipid Panel	High Cholesterol
3	105	Blood Glucose	Elevated
4	106	CBC	Normal

34.1 inner join

`inner_join()`: returns only rows with matching patient IDs in both data frames.

```
# Inner join on patient_id
inner_join(patients, lab_results, by = "patient_id")
```

	patient_id	age	gender	test	result
1	103	29	M	CBC	Normal
2	104	56	F	Lipid Panel	High Cholesterol

In this example, only patients with `patient_ids` 103 and 104 are in both tables, so only their information appears in the result.

34.2 left join

`left_join()` returns all rows from the patients data frame and matches rows from `lab_results` where possible. Unmatched rows in `lab_results` are filled with NA.

```
# Left join on patient_id
left_join(patients, lab_results, by = "patient_id")
```

	patient_id	age	gender	test	result
1	101	34	F	<NA>	<NA>
2	102	45	M	<NA>	<NA>
3	103	29	M	CBC	Normal
4	104	56	F	Lipid Panel	High Cholesterol

Here, all patients from `patients` are included, with NA for lab results where no match is found.

34.3 right join

`right_join()` returns all rows from `lab_results`, matching rows from `patients` where possible. Unmatched rows in `patients` are filled with `NA`.

```
# Right join on patient_id
right_join(patients, lab_results, by = "patient_id")
```

	patient_id	age	gender	test	result
1	103	29	M	CBC	Normal
2	104	56	F	Lipid Panel	High Cholesterol
3	105	NA	<NA>	Blood Glucose	Elevated
4	106	NA	<NA>	CBC	Normal

In this example, all patients with lab results are included, with demographic data from patients where available.

34.4 Full Join

`full_join()` returns all rows from both tables, with `NA` for missing matches in either table.

```
# Full join on patient_id
full_join(patients, lab_results, by = "patient_id")
```

	patient_id	age	gender	test	result
1	101	34	F	<NA>	<NA>
2	102	45	M	<NA>	<NA>
3	103	29	M	CBC	Normal
4	104	56	F	Lipid Panel	High Cholesterol
5	105	NA	<NA>	Blood Glucose	Elevated
6	106	NA	<NA>	CBC	Normal

34.5 Semi Join

`semi_join()` returns only the rows in `patients` that have a match in `lab_results`, without bringing in columns from `lab_results`.


```
# Semi join on patient_id
semi_join(patients, lab_results, by = "patient_id")
```

	patient_id	age	gender
1	103	29	M
2	104	56	F

Only patients with lab results are included here (patients 103 and 104).

34.6 Anti join

`anti_join()` returns only the rows in `patients` that do not have a match in `lab_results`.

```
# Anti join on patient_id
anti_join(patients, lab_results, by = "patient_id")
```

	patient_id	age	gender
1	101	34	F
2	102	45	M

```
#example
tab_a <- data.frame(
  id=c(001,002,003,004),
  age=c(25,50,30,40)
)

tab_b <- data.frame(
  id=c(001,002,004,005),
  sex=c("M","F","M","F")
)

#inner join:
injointab <- dplyr::inner_join(tab_a,tab_b,by="id")
#full join:
fulljointab <- full_join(tab_a,tab_b,by="id")
#left join
leftjointab <- left_join(x=tab_a,y=tab_b,by="id")
#right join
rightjointab <- right_join(x=tab_a,y=tab_b,by="id")
```

```
#anti_join
antijointab <- anti_join(x=tab_a,y=tab_b,by="id")
antijointab1 <- anti_join(x=tab_b,y=tab_a,by="id")
```

35 Reshaping data with tidyr

```
test <- data.frame(
  visit=c("wk0","wk2","wk4","wk6","wk12"),
  drug_a=c(10,20,30,40,50),
  drug_b=c(50,40,30,20,10),
  drug_c=c(15,25,35,45,55))
#tidyr::pivot_longer
library(tidyr)
test1 <- pivot_longer(data=test,
                      cols=c(drug_a,drug_b,drug_c),
                      names_to="drug",
                      values_to = "dose")

test2 <- pivot_wider(data=test1,
                    id_cols =visit,
                    names_prefix = "d_", # it is optional
                    values_from = dose,
                    names_from = drug)
```

36 counting with n()

When working with data, we often want to know the number of observations found for each factor or combination of factors. For this task, dplyr provides `count()`. For example, if we wanted to count the number of rows of data for each sex, we would do:

```
cnt <- adsl |>
  dplyr::count(SEX)

# race count by treatment
cnt2 <- adsl |>
  dplyr::count(TRT01P, RACE)

#treatment count
cnt3 <- adsl |>
  dplyr::filter(!is.na(TRT01P)) |>
  dplyr::count(TRT01P)

adae <- haven::read_sas("data/adam/adae.sas7bdat") |>
  dplyr::mutate(dplyr::across(where(is.character),~ dplyr::na_if(.,"")))

adsl <- haven::read_sas("data/adam/adsl.sas7bdat") |>
  dplyr::mutate(dplyr::across(where(is.character),~ dplyr::na_if(.,"")))

bign <- adsl |>
  dplyr::filter(SAFFL == "Y") |>
  dplyr::count(TRT01AN, TRT01A) |>
  dplyr::rename(TRTA = TRT01A, TRTAN = TRT01AN, bign = n)

bign1 <- dplyr::count(adsl, TRT01AN, TRT01A) |>
  dplyr::rename(TRTA = TRT01A, TRTAN = TRT01AN, bign = n)

adae1 <- adae |>
  dplyr::distinct(USUBJID, AEBODSYS, AEDECOD, TRTA, TRTAN, SAFFL)
```

```

ae_t <- adae1 |>
  dplyr::filter(SAFFL == "Y") |>
  dplyr::group_by(TRTA) |>
  dplyr::count(AEBODSYS, AEDECOD)

ae_t1 <- ae_t |>
  dplyr::left_join(bign, by = "TRTA") |>
  dplyr::mutate(
    pct = paste0("(", round(n / bign * 100, 2), "%"),
    val = paste0(n, " ", pct)
  )

ae2 <- pivot_wider(
  data = ae_t1,
  id_cols = c(AEBODSYS, AEDECOD),
  names_from = TRTA,
  values_from = val,
  values_fill = "0 (0.0)"
)

```

37 dplyr distinct()

The `distinct()` function in `dplyr` is used to return unique/distinct rows from a data frame or tibble. It removes duplicate rows based on the values in specified columns or all columns if none are specified.

```
# distinct
distinct_trt <- adsl |>
  dplyr::distinct(TRT01P, TRT01PN)
print(distinct_trt)
```

```
# A tibble: 3 x 2
  TRT01P          TRT01PN
  <chr>          <dbl>
1 Placebo          0
2 Xanomeline High Dose 81
3 Xanomeline Low Dose  54
```

38 case_when()

The `case_when()` function in `dplyr` is a powerful tool for creating new variables based on multiple conditions. It allows you to specify a series of conditions and corresponding values to assign when those conditions are met.

```
adsl1 <- adsl |>
dplyr::mutate(
  agegrp1 = dplyr::case_when(
    AGE < 18 ~ "Child",
    AGE >= 18 & AGE < 65 ~ "Adult",
    AGE >= 65 ~ "Senior",
    TRUE ~ "Unknown" # Default case
  )
) |>
dplyr::select(USUBJID, AGE, agegrp1) |>
head(10)
```

39 create seq no.

```
class <- haven::read_sas("data/class.sas7bdat")

t <- dplyr::arrange(class, Age)
# Create t1 data set with first and last records for each unique value of 'age'

t1 <- class |>
  dplyr::arrange(Age) |>
  dplyr::group_by(Age) |>
  dplyr::mutate(seq = dplyr::row_number())

t2 <- class |>
  dplyr::mutate(seq = dplyr::row_number(), .by = Age) |>
  dplyr::arrange(Age)

t3 <- class |>
  dplyr::arrange(Age) |>
  dplyr::group_by(Age) |>
  dplyr::filter(dplyr::row_number() == 1 )

t4 <- class |>
  dplyr::arrange(Age) |>
  dplyr::group_by(Age) |>
  dplyr::filter(dplyr::row_number() == n())

t5 <- class |>
  dplyr::group_by(Age) |>
  dplyr::filter(row_number() == 1 & row_number() == n())

t6 <- class |>
  dplyr::group_by(Age) |>
  dplyr::filter(row_number() == 1|row_number() == n())
```


40 Lists: lapply, purrr

```
lst <- list(a=1:3, b=10:12)  
lapply(lst, mean)
```

```
$a  
[1] 2
```

```
$b  
[1] 11
```

41 Exercises

1. Using `across()`, standardize $(x - \text{mean}) / \text{sd}$ numeric columns.
2. Create a row-wise mean of `age` and `wt`.
3. Split `df` by `grp` and compute group means with `lapply` or `purrr::map`.

42 Reading SAS Datasets (+ Cleaning)

```
library(haven)
library(dplyr)
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

```
library(labelled)
```

We try to read a SAS dataset (e.g., SDTM DM). If not present, we **synthesize** an example.

```
dm_path <- "data/sdtm/dm.sas7bdat"

if (file.exists(dm_path)) {
  dm <- read_sas(dm_path)
} else {
  dm <- tibble::tibble(
    STUDYID = "XYZ123",
    USUBJID = sprintf("XYZ-%03d", 1:10),
    ARM = rep(c("Placebo", "Active"), length.out=10),
    AGE = c(55, 62, 47, 50, 71, 66, 45, 59, 53, 68),
    SEX = rep(c("M", "F"), length.out=10)
  )
  message("Synthesized `dm` since data/sdtm/dm.sas7bdat was not found.")
}
str(dm)
```

```

tibble [306 x 25] (S3: tbl_df/tbl/data.frame)
 $ STUDYID : chr [1:306] "CDISCPIL0T01" "CDISCPIL0T01" "CDISCPIL0T01" "CDISCPIL0T01" ...
  ..- attr(*, "label")= chr "Study Identifier"
 $ DOMAIN  : chr [1:306] "DM" "DM" "DM" "DM" ...
  ..- attr(*, "label")= chr "Domain Abbreviation"
 $ USUBJID : chr [1:306] "01-701-1015" "01-701-1023" "01-701-1028" "01-701-
1033" ...
  ..- attr(*, "label")= chr "Unique Subject Identifier"
 $ SUBJID  : chr [1:306] "1015" "1023" "1028" "1033" ...
  ..- attr(*, "label")= chr "Subject Identifier for the Study"
 $ RFSTDTC : chr [1:306] "2014-01-02" "2012-08-05" "2013-07-19" "2014-03-
18" ...
  ..- attr(*, "label")= chr "Subject Reference Start Date/Time"
 $ RFENDTC : chr [1:306] "2014-07-02" "2012-09-02" "2014-01-14" "2014-04-
14" ...
  ..- attr(*, "label")= chr "Subject Reference End Date/Time"
 $ RFXSTDTC: chr [1:306] "2014-01-02" "2012-08-05" "2013-07-19" "2014-03-
18" ...
  ..- attr(*, "label")= chr "Date/Time of First Study Treatment"
 $ RFXENDTC: chr [1:306] "2014-07-02" "2012-09-01" "2014-01-14" "2014-03-
31" ...
  ..- attr(*, "label")= chr "Date/Time of Last Study Treatment"
 $ RFICDTC : chr [1:306] "" "" "" "" ...
  ..- attr(*, "label")= chr "Date/Time of Informed Consent"
 $ RFPENDTC: chr [1:306] "2014-07-02T11:45" "2013-02-18" "2014-01-14T11:10" "2014-
09-15" ...
  ..- attr(*, "label")= chr "Date/Time of End of Participation"
 $ DTHDTC  : chr [1:306] "" "" "" "" ...
  ..- attr(*, "label")= chr "Date/Time of Death"
 $ DTHFL   : chr [1:306] "" "" "" "" ...
  ..- attr(*, "label")= chr "Subject Death Flag"
 $ SITEID  : chr [1:306] "701" "701" "701" "701" ...
  ..- attr(*, "label")= chr "Study Site Identifier"
 $ AGE      : num [1:306] 63 64 71 74 77 85 59 68 81 84 ...
  ..- attr(*, "label")= chr "Age"
 $ AGEU     : chr [1:306] "YEARS" "YEARS" "YEARS" "YEARS" ...
  ..- attr(*, "label")= chr "Age Units"
 $ SEX      : chr [1:306] "F" "M" "M" "M" ...
  ..- attr(*, "label")= chr "Sex"
 $ RACE     : chr [1:306] "WHITE" "WHITE" "WHITE" "WHITE" ...
  ..- attr(*, "label")= chr "Race"
 $ ETHNIC   : chr [1:306] "HISPANIC OR LATINO" "HISPANIC OR LATINO" "NOT HISPANIC OR LATINO" ...
  ..- attr(*, "label")= chr "Ethnicity"

```

```

$ ARMCD      : chr [1:306] "Pbo" "Pbo" "Xan_Hi" "Xan_Lo" ...
  ..- attr(*, "label")= chr "Planned Arm Code"
$ ARM        : chr [1:306] "Placebo" "Placebo" "Xanomeline High Dose" "Xanomeline Low Dose" ..
  ..- attr(*, "label")= chr "Description of Planned Arm"
$ ACTARMCD   : chr [1:306] "Pbo" "Pbo" "Xan_Hi" "Xan_Lo" ...
  ..- attr(*, "label")= chr "Actual Arm Code"
$ ACTARM     : chr [1:306] "Placebo" "Placebo" "Xanomeline High Dose" "Xanomeline Low Dose" ..
  ..- attr(*, "label")= chr "Description of Actual Arm"
$ COUNTRY    : chr [1:306] "USA" "USA" "USA" "USA" ...
  ..- attr(*, "label")= chr "Country"
$ DMDTC      : chr [1:306] "2013-12-26" "2012-07-22" "2013-07-11" "2014-03-
10" ...
  ..- attr(*, "label")= chr "Date/Time of Collection"
$ DMDY       : num [1:306] -7 -14 -8 -8 -7 -21 NA -9 -13 -7 ...
  ..- attr(*, "label")= chr "Study Day of Collection"

```

42.1 Handling Labels & Missing

```

# Example: Convert blank strings "" to NA for character columns
convert_blanks_to_na <- function(x) {
  if (is.character(x)) x[x == ""] <- NA_character_
  x
}
dm <- dm |> mutate(across(where(is.character), convert_blanks_to_na))

```

42.2 Labelled to Factor (if needed)

```

if (inherits(dm$SEX, "labelled")) {
  dm <- dm |> mutate(SEX = to_factor(SEX))
}

```

42.3 Common Cleaning

```
dm <- dm |>
  mutate(
    AGEGR1 = cut(AGE, breaks=c(-Inf, 50, 65, Inf),
                 labels=c("<50", "50-65", "65+"))
  )
```

43 Exercises

1. Read another SAS dataset (e.g., `sv.sas7bdat`) if available. If not, create a synthetic tibble.
2. Write a function to trim character whitespace for all character columns.
3. Make a clean factor for `ARM` with levels `Placebo < Active`.

44 Base R Functions & Apply Family

45 Common Utilities

```
x <- 1:10  
sum(x); mean(x); sd(x); var(x); quantile(x)
```

```
[1] 55
```

```
[1] 5.5
```

```
[1] 3.02765
```

```
[1] 9.166667
```

0%	25%	50%	75%	100%
1.00	3.25	5.50	7.75	10.00

```
seq(0, 1, by=0.1); rep(5, times=3)
```

```
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
[1] 5 5 5
```

46 Apply Family

```
m <- matrix(1:9, nrow=3)
apply(m, 1, mean) # row means
```

```
[1] 4 5 6
```

```
apply(m, 2, mean) # col means
```

```
[1] 2 5 8
```

```
lst <- list(a=1:3, b=10:12)
sapply(lst, mean) # simplifies result
```

```
  a  b
2 11
```

```
mapply(sum, 1:3, 10:12)
```

```
[1] 11 13 15
```

47 Subsetting Essentials

```
df <- data.frame(id=1:3, val=c(10,20,30))  
df[1, "val"]
```

```
[1] 10
```

```
df[df$val > 10, ]
```

```
  id val  
2  2  20  
3  3  30
```

48 Exercises

1. Use `apply` to get the max per column of a numeric matrix.
2. Write a base R snippet to compute IQR for each column of `mtcars`.
3. Compare `lapply` vs `sapply` in behavior on a list with mixed types.

49 Custom Functions & Validation

50 Writing Functions

```
safe_mean <- function(x, na.rm = TRUE) {  
  stopifnot(is.numeric(x))  
  mean(x, na.rm = na.rm)  
}  
safe_mean(c(1, 2, NA))
```

```
[1] 1.5
```

51 Error Handling

```
robust_divide <- function(a, b) {  
  tryCatch(a / b,  
    warning = function(w) NA_real_,  
    error   = function(e) NA_real_)  
}  
robust_divide(10, 0)
```

```
[1] Inf
```

52 Unit Testing with testthat

Install once: `install.packages(c("testthat","devtools","usethis","roxygen2"))`

```
usethis::use_testthat()
usethis::use_test("safe_mean")
```

Create `tests/testthat/test-safe_mean.R`:

```
testthat::test_that("safe_mean works", {
  x <- c(1,2,NA)
  testthat::expect_equal(safe_mean(x), 1.5)
  testthat::expect_error(safe_mean("oops"))
})
```

Test passed

53 Document with roxygen2

```
#' Compute a safe mean
#' @param x Numeric vector
#' @param na.rm Logical; remove NAs
#' @return Numeric scalar
#' @examples
#' safe_mean(c(1,2,NA))
#' @export
safe_mean <- function(x, na.rm = TRUE) {
  stopifnot(is.numeric(x))
  mean(x, na.rm = na.rm)
}
```

Run:

```
devtools::document()
```

54 Exercises

1. Write `winsorize(x, probs=c(0.05,0.95))` and test it.
2. Create `validate_columns(df, required=c("USUBJID","AGE"))` and add tests.
3. Add roxygen docs and build help pages.

55 R Package Development

55.1 Setup

```
install.packages(c("usethis", "devtools", "testthat", "roxygen2", "pkgdown"))
```

55.2 Create a Package

```
usethis::create_package("mypkg")
# In the new project:
usethis::use_mit_license("Your Name")
usethis::use_git()
usethis::use_github() # optional
usethis::use_roxygen_md()
usethis::use_testthat()
usethis::use_package("dplyr") # adds to DESCRIPTION
```

55.3 Add a Function

Create `R/safe_mean.R` and its tests (see previous chapter).

55.4 Build, Install, Check

```
devtools::document()
devtools::build()
devtools::install()
devtools::check()
```

55.5 Vignette & Website

```
usethis::use_vignette("intro")
usethis::use_pkgdown()
pkgdown::build_site()
```

Exercise: Package-ize a small utility set (`convert_blanks_to_na`, `validate_columns`, etc.) with docs and tests.

56 Git in RStudio (Setup & Auth)

56.1 One-Time Setup

- Install Git and ensure `git --version` works.
- In R:

```
usethis::use_git_config(user.name = "Your Name", user.email = "you@example.com")
```

56.2 Initialize Git for the Current Project

```
usethis::use_git()
```

56.3 Connect to GitHub

- Create a GitHub account.
- In R:

```
usethis::create_github_token()
gitcreds::gitcreds_set() # paste token when prompted
usethis::use_github(protocol = "https")
```

Or set up SSH keys via RStudio (Tools > Global Options > Git/SVN).

56.4 Typical Workflow

1. Stage changes (Git pane in RStudio).
2. Commit with a clear message.
3. Push to origin (GitHub).

56.5 Remove Git from a Project (macOS/RStudio)

- In Finder/Terminal, delete the hidden `.git` folder in the project root (careful!).
- Or from Terminal at project root:

```
rm -rf .git
```

- Reopen project in RStudio; Git pane will disappear.

Exercises - Create a new repo for this Quarto course and push it. - Branch, make a change, open a Pull Request on GitHub.

57 Creating ADaM: ADSL from SDTM-like Inputs

```
library(dplyr)
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

```
library(tidyr)  
library(lubridate)
```

Attaching package: 'lubridate'

The following objects are masked from 'package:base':

date, intersect, setdiff, union

We simulate **minimal** SDTM-like DM and EX to illustrate ADSL creation. If available, replace with your own data/sdtm/*.sas7bdat.

```
# DM
dm <- tibble::tibble(
  STUDYID = "XYZ123",
  USUBJID = sprintf("XYZ-%03d", 1:10),
  ARM = rep(c("Placebo","Active"), length.out=10),
  AGE = c(55, 62, 47, 50, 71, 66, 45, 59, 53, 68),
  SEX = rep(c("M","F"), length.out=10),
  RANDDT = as.Date("2025-01-15") + sample(0:20, 10, replace=TRUE)
)

# EX (first dose date)
ex <- tibble::tibble(
  USUBJID = dm$USUBJID,
  EXSTDTC = dm$RANDDT + sample(0:3, 10, replace=TRUE)
)
```

57.1 Build ADSL

```
adsl <- dm |>
  left_join(ex, by="USUBJID") |>
  transmute(
    STUDYID, USUBJID,
    TRT01P = ARM,
    TRT01PN = as.integer(factor(ARM, levels=c("Placebo","Active"))),
    AGE, SEX,
    RANDDT,
    TRTSDT = EXSTDTC,
    TRT01A = TRT01P,          # assume planned == actual for demo
    TRT01AN = TRT01PN
  )
adsl
```

A tibble: 10 x 10

	STUDYID	USUBJID	TRT01P	TRT01PN	AGE	SEX	RANDDT	TRTSDT	TRT01A
	<chr>	<chr>	<chr>	<int>	<dbl>	<chr>	<date>	<date>	<chr>
1	XYZ123	XYZ-001	Placebo	1	55	M	2025-01-28	2025-01-30	Placebo
2	XYZ123	XYZ-002	Active	2	62	F	2025-01-24	2025-01-25	Active
3	XYZ123	XYZ-003	Placebo	1	47	M	2025-01-27	2025-01-29	Placebo
4	XYZ123	XYZ-004	Active	2	50	F	2025-01-19	2025-01-20	Active


```

5 XYZ123 XYZ-005 Placebo      1    71 M    2025-01-19 2025-01-19 Placebo
6 XYZ123 XYZ-006 Active      2    66 F    2025-01-15 2025-01-18 Active
7 XYZ123 XYZ-007 Placebo      1    45 M    2025-01-17 2025-01-19 Placebo
8 XYZ123 XYZ-008 Active      2    59 F    2025-01-19 2025-01-22 Active
9 XYZ123 XYZ-009 Placebo      1    53 M    2025-01-21 2025-01-23 Placebo
10 XYZ123 XYZ-010 Active      2    68 F    2025-02-01 2025-02-04 Active
# i 1 more variable: TRT01AN <int>

```

Note: Real ADSL creation must follow **ADaM IG** (derive flags, dates, imputations, populations). This example is educational only.

Exercises 1. Add analysis populations (e.g., SAFFL, FASFL) based on simple rules. 2. Derive AGEGR1 as <65 / 65 and use ordered factor. 3. Add a treatment end date TRTEDT and compute treatment duration.

58 TLFs: Table, Figure, Listing

```
library(dplyr)
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

```
library(gt)
library(ggplot2)
library(survival)
```

We reuse `adsl` from the previous chapter (or synthesize if missing).

```
if (!exists("adsl")) {
  set.seed(123)
  adsl <- tibble::tibble(
    USUBJID = sprintf("XYZ-%03d", 1:60),
    TRT01P = sample(c("Placebo", "Active"), 60, replace=TRUE),
    AGE = round(rnorm(60, 60, 8)),
    SEX = sample(c("M", "F"), 60, replace=TRUE)
  )
}
```

Table 1. Baseline Characteristics by Treatment

TRT01P	N	mean_age	sd_age	pct_female
Active	24	59.4	8.1	50.0
Placebo	36	61.7	5.6	61.1

58.1 Table 1: Baseline Characteristics by Treatment

```
tbl1 <- adsl |>
  group_by(TRT01P) |>
  summarise(
    N = dplyr::n(),
    mean_age = mean(AGE, na.rm=TRUE),
    sd_age = sd(AGE, na.rm=TRUE),
    pct_female = mean(SEX == "F")*100
  )

gt(tbl1) |>
  tab_header(title = "Table 1. Baseline Characteristics by Treatment") |>
  fmt_number(columns = c(mean_age, sd_age, pct_female), decimals = 1)
```

58.2 Figure: (Toy) Survival Curve

We simulate time-to-event data for illustration only.

```
set.seed(42)
n <- nrow(adsl)
adsl$time <- rexp(n, rate = ifelse(adsl$TRT01P=="Active", 0.08, 0.1))
adsl$status <- rbinom(n, 1, 0.7)
fit <- survival::survfit(survival::Surv(time, status) ~ TRT01P, data = adsl)

# Quick GGplot
ggsurv <- function(fit) {
  # rebuild data for plotting
  ss <- summary(fit)
```

```

dd <- data.frame(
  time = ss$time,
  surv = ss$surv,
  strata = rep(names(fit$strata), fit$strata)
)
ggplot(dd, aes(x=time, y=surv, linetype=strata)) +
  geom_step() +
  labs(title="Kaplan-Meier (Toy Data)", x="Time", y="Survival Probability", linetype="Treatment")
  theme_minimal()
}
#ggsurv(fit)

```

58.3 Listing: Subject-Level Listing

```

lst <- adsl |>
  arrange(USUBJID) |>
  select(USUBJID, TRT01P, AGE, SEX) |>
  head(20)

gt(lst) |>
  tab_header(title = "Listing: First 20 Subjects")

```

Exercises 1. Format Table 1 to N ($\text{mean} \pm \text{SD}$) for age. 2. Add risk table to the KM plot (use an extension like survminer outside of this minimal example). 3. Create a listing that includes population flags once you derive them.

[

Listing: First 20 Subjects | Listing: First 20 Subjects

USUBJID	TRT01P	AGE	SEX
XYZ-001	Placebo	63	F
XYZ-002	Placebo	58	M
XYZ-003	Placebo	67	M
XYZ-004	Active	67	M
XYZ-005	Placebo	67	M
XYZ-006	Active	66	F
XYZ-007	Active	64	F
XYZ-008	Active	60	M
XYZ-009	Placebo	58	M
XYZ-010	Placebo	57	F
XYZ-011	Active	54	F
XYZ-012	Active	58	M
XYZ-013	Active	50	M
XYZ-014	Placebo	77	F
XYZ-015	Active	70	F
XYZ-016	Placebo	51	M
XYZ-017	Active	57	M
XYZ-018	Placebo	56	F
XYZ-019	Placebo	66	M
XYZ-020	Placebo	59	F

59 Capstone: End-to-End Mini Workflow

This chapter ties everything together: **read data** → **derive ADSL** → **produce TLFs** → **render a report**.

59.1 Parameters

```
# You could parametrize paths via YAML; here we keep inline defaults.
dm_path <- "data/sdtm/dm.sas7bdat"
ex_path <- "data/sdtm/ex.sas7bdat"
```

59.2 1) Read (or Synthesize) SDTM

```
library(haven); library(dplyr); library(lubridate)
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

Attaching package: 'lubridate'

The following objects are masked from 'package:base':

date, intersect, setdiff, union

```
if (file.exists(dm_path)) {
  dm <- read_sas(dm_path)
} else {
  dm <- tibble::tibble(
    STUDYID = "XYZ123",
    USUBJID = sprintf("XYZ-%03d", 1:60),
    ARM = sample(c("Placebo", "Active"), 60, replace=TRUE),
    AGE = round(rnorm(60, 60, 8)),
    SEX = sample(c("M", "F"), 60, replace=TRUE),
    RANDDT = as.Date("2025-01-15") + sample(0:40, 60, replace=TRUE)
  )
}
if (file.exists(ex_path)) {
  ex <- read_sas(ex_path)
} else {
  ex <- tibble::tibble(
    USUBJID = dm$USUBJID,
    EXSTDTC = dm$RANDDT + sample(0:3, nrow(dm), replace=TRUE)
  )
}
```

59.3 2) Derive ADSL (Minimal Demo)

```
adsl <- dm |>
  left_join(ex, by="USUBJID") |>
  mutate(
    TRT01P = ARM,
    TRT01PN = as.integer(factor(ARM, levels=c("Placebo", "Active"))),
    TRT01A = TRT01P,
    TRT01AN = TRT01PN,
    SAFFL = "Y",          # demo only; define rules in real life
    FASFL = "Y"
  ) |>
  dplyr::select(STUDYID.x, USUBJID, TRT01P, TRT01PN, TRT01A, TRT01AN, AGE, SEX, EXSTDTC, SAFI
```

[
Table	1.	Baseline	by	Treatment
Table	1.	Baseline	by	Treatment
Description of Planned Arm	N	mean_age	sd_age	pct_female
Placebo	226	75.04867	8.503715	60.61947
Screen Failure	52	75.09615	9.699928	69.23077
Xanomeline High Dose	184	74.01087	7.939656	48.36957
Xanomeline Low Dose	181	75.29834	8.277778	60.77348

59.4 3) TLFs

```
library(gt); library(ggplot2); library(survival)
```

```
tbl1 <- adsl |>
  group_by(TRT01P) |>
  summarise(N=n(),
            mean_age = mean(AGE), sd_age = sd(AGE),
            pct_female = mean(SEX=="F")*100)
tbl1_gt <- gt(tbl1) |> tab_header(title="Table 1. Baseline by Treatment")
tbl1_gt
```

```
set.seed(123)
adsl$time <- rexp(nrow(adsl), rate=ifelse(adsl$TRT01P=="Active", 0.08, 0.1))
adsl$status <- rbinom(nrow(adsl), 1, 0.7)
fit <- survfit(Surv(time, status) ~ TRT01P, data=adsl)
# reuse plotting function from prior chapter
ggsurv <- function(fit) {
  ss <- summary(fit)
  dd <- data.frame(time=ss$time, surv=ss$surv, strata=rep(names(fit$strata), fit$strata))
  ggplot(dd, aes(x=time, y=surv, linetype=strata)) + geom_step() + theme_minimal() +
    labs(title="KM Curve (Toy)", x="Time", y="Survival", linetype="Treatment")
}
#ggsurv(fit)
```


59.5 4) Save Outputs

```
# Example: Save Table 1 as PNG
#gtsave(tbl1_gt, "tlf-table1.png")
```

Challenge: Convert this chapter into a **parameterized report** (e.g., treatment subset or different cohort) and render multiple outputs.

60 Appendix: Tips, Profiles, .libPaths

60.1 Useful Profiles

Create ~/.Rprofile to set options (be careful on shared systems):

```
options(  
  repos = c(CRAN = "https://cloud.r-project.org"),  
  scipen = 999  
)
```

60.2 Custom Library Paths

```
# In .Rprofile or project-level .Rprofile  
.libPaths(c("/path/to/Rlibs", .libPaths()))
```

60.3 Format vs formatC (quick recap)

```
x <- c(123.456, 0.00123456)  
format(x, digits = 4)
```

```
[1] "1.235e+02" "1.235e-03"
```

```
format(x, nsmall = 2)
```

```
[1] "1.23456e+02" "1.23456e-03"
```

```
formatC(x, digits = 3, format = "f")
```

```
[1] "123.456" "0.001"
```

60.4 POSIXct vs POSIXlt

- **POSIXct**: seconds since epoch (numeric), compact, fast.
- **POSIXlt**: list-like with components (year, mon, mday...), easier to extract parts.

60.5 Recommended Packages

- tidyverse, lubridate, janitor, gt, gtsummary, survival, broom, here.
- Pharma/CDISC: admiral, tlf/tern, pharmaverse meta-packages (explore as you grow).

60.6 Short Glossary

- **SDTM**: Study Data Tabulation Model (FDA submission standard for raw domains).
- **ADaM**: Analysis Data Model (derived analysis-ready datasets).
- **TLF**: Tables, Listings, Figures for reporting.