R Course: Beginner to Expert

Sri Ram

2025-10-31

Table of contents

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

```r
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:

```r
# 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:

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

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

```r
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:

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

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

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

11 Working Directory

```r
getwd()
```

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

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

12 Vectors (Atomic)

```r
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"
```

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

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

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

```
[1] TRUE
```

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

```
[1] TRUE
```

```r
typeof(v_integer)
```

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

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

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

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

```
[1] TRUE
```

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

```
[1] TRUE
```

```r
typeof(v_double)
```

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

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

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

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

```
[1] TRUE
```

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

```
[1] TRUE
```

```r
typeof(v_character)
```

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

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

```
NULL
```

```r
typeof(v_NULL)
```

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

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

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

```r
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" 
```

```r
# 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
```

```r
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
```

```r
# 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
```

```r
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 orfer 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,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
```

```
[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)
```

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

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

```
  [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
 [38] 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
 [75] 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
[112] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
[149] 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
[186] 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
[223] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
[260] 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
[297] 3 1 2 3
```

```
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
```

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

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

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

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

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

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

```r
log(v)
```

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

```r
sum(v)
```

```
[1] 15
```

```r
# 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
```

```r
# 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",

# 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
```

```r
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
```

```r
# 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(el1 = c(1,5,10), el2 = 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
```

```r
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
```

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

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

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

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

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

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

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

```
[1] 1
```

```r
# 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
```

```r
# 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
```

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

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

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

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

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

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

```r
# 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
```

```r
rownames(M)
```

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

```r
colnames(M)
```

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

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

```
[1] 4 3
```

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

```
$dim
[1] 4 3

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

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

```r
# 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
```

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

```
   [1]  1.3198283389 -2.5497505944 -3.7405413132  0.7216255361  1.7068175250
   [6] -0.2064872157 -0.3056290127  1.7331292428 -0.1004070992  0.5726041532
  [11] -0.8432493331  2.3807010320  0.7610791374 -1.5844661470  1.6188202796
  [16]  2.0756699311  1.5754814523  0.3935967701 -0.4534006754 -0.4973164733
  [21] -1.6859805352 -1.5922095392  1.9809001644  0.2593890744  0.6214955011
  [26] -0.8518929493  0.1559644252 -1.5564633100 -0.1880865139 -0.6428860793
  [31] -1.6155200313  0.5419334452  0.9306486514  1.5934771394 -0.8415760853
  [36] -1.8026778942  0.2074328655  3.7565170924  0.0634139067 -0.7208593004
  [41]  1.0166517645  1.2195172356 -1.4335037395  2.2221288317  3.8026979148
  [46] -1.7131710550  2.0972206206  0.9675433920  1.7440455315  4.2367108741
  [51] -1.7711667608 -0.7377434472  2.9068294706  1.7573588573 -1.7653162839
  [56]  3.9737807795  1.7275148563  1.7973980906 -3.0520717271 -0.5936226004
  [61]  1.2608787842  0.5971400387 -0.9585053016  1.9504175115  4.0033803889
  [66]  1.6236540437 -1.9586620534  0.7353823162 -1.5665187120  3.3593544492
  [71]  3.4456302394  5.2804972399  0.6631676925  2.5880117730 -0.6749693466
  [76]  1.1938661737  1.8537811927 -1.6628560603  1.1593594247  1.0397347668
  [81] -3.9368672504 -2.3476835490 -0.2978607045  2.3532275105  1.8994406810
  [86] -2.4909147454 -1.1543047741 -2.5008061683  0.3399895554  0.5143545668
  [91] -1.2086921100  0.1536838528 -0.5285611434  3.7822726255  3.6361682916
  [96]  4.0435897682  2.8470968634 -2.4530722824  0.6963953189 -0.7976531444
 [101]  1.3830132123  1.6953332445  3.9428556983 -0.6154140034 -2.3482847843
 [106] -1.6613423207 -2.2991623977  0.9827514483  0.7842262533 -2.2583244115
 [111] -1.5814742811 -0.8846487169 -3.1423532832  1.9494243892 -0.1982704095
 [116] -4.2129880044  4.8331335411 -1.2410977097 -0.8130370525 -0.6566117852
 [121]  1.5965642405 -4.8605416787  1.0707157915  2.1486730858 -2.5607370900
 [126] -1.4804794737  2.4765381026 -0.0995124641  1.1579259456  0.0943085272
 [131] -1.8349434508  3.2187153827 -0.4201110027  4.6567013266  1.4591796973
 [136]  3.7499669068  1.5470685860 -0.7375719969 -2.9971267835  4.1404762150
 [141] -1.5739434282  3.0008306039 -1.8884641769 -1.5739153172 -1.0086788788
 [146]  1.1418773441  1.4415396782 -2.3571448740  2.8862945801  1.5555410875
 [151]  0.0803959075  1.3251333144  4.9435910503  1.4720188913  1.3749734346
 [156]  1.4931400744  1.5305410017  3.8455505740  1.5068563662  1.0487061173
 [161]  2.2038788694  0.5571931874 -3.1004206982 -5.1162245425 -1.3252277076
 [166] -0.3350731319  1.8683172457 -3.3456455361 -2.6001857602 -2.0901571929
 [171] -0.4692153891 -1.3535468776  3.2141210746 -0.7068091587 -2.6494783998
 [176]  0.2899991667  0.5758584155 -2.0399735857 -1.4972853511  1.0773818945
 [181] -4.9830904477 -1.2629047124  0.6594820473  5.1336237729  0.4339336885
```

```
[186] -1.1774425010 -1.4712570506 -1.0742637681 -2.0428201206  3.2696876431
[191]  0.2682590180 -0.3605600527 -3.7404680284  0.7124163587 -3.1972700748
[196]  0.7653415491 -2.0335056388 -0.9285389055  0.3108445819 -1.3344427822
[201]  0.7092146422  2.1017251741 -0.9613201754  1.7805569440 -5.5911538046
[206]  1.3590552214  0.9380133075  1.4581905696  1.6298432359  1.9877819063
[211]  1.3192892836 -1.2671894905  2.1624467528 -1.1470056497 -1.8092678769
[216]  0.6520034888 -1.1339082179  0.1189147254  1.5715869057  5.1850995299
[221]  1.9945068253 -0.8369761548 -2.7078036182 -3.3386944529 -0.7072286382
[226]  2.6286338803  0.5502471548  0.4286681012  0.7367881214 -2.6957225433
[231]  0.6163435193 -2.0035699480 -1.6852903933 -2.2256445467  0.3227497859
[236] -0.9365579546  0.6109465989 -4.2408398171  1.2333475694  1.1202412191
[241] -3.1318844146 -1.0618031304 -2.3645649976  0.2171837589 -1.8168304701
[246]  6.0655855295  1.2248215149  2.2584194101  2.2470066405  3.5205899027
[251] -1.5013200086 -2.4461804237  4.3935620587  1.6540493381  1.4587372275
[256] -0.7305156500  3.0711204266  1.4854698381 -3.5747062878 -3.0545384463
[261]  1.6036341500  1.2572752010 -3.8733976465  0.0160946364  0.5526200244
[266] -2.7951039336  0.1018963750  3.4653948691 -0.8253312215  4.3433090775
[271]  0.4088245021  1.5337003262 -0.3892430569 -1.8896111425  3.9631424967
[276] -1.5120124898  2.6330402827 -0.8804319167 -1.7500531788  3.4395967845
[281] -0.2082471876  0.9348067216  0.7933782717  0.1547161925 -1.3390554565
[286] -2.6456665840  2.1525283135  0.3502829485 -0.2012412175  2.4108746069
[291]  1.6036543670  1.2535039988 -0.3286039086 -2.7353943582  2.5337045143
[296]  0.8825429740  1.3670741268 -3.0544227825  0.5072211967 -0.8136168455
[301]  3.4345111075  1.2070162387  0.6635614703 -0.2124942499  0.0975184627
[306] -2.1975475320 -1.6612992760 -2.4428519602  3.1236067919  0.3486160005
[311] -0.2981437898  0.2355943710  3.0697418359 -1.2608421669  1.1676551827
[316]  2.1483651803 -1.4644641203  1.5228645131 -1.5789803031 -1.4706324713
[321] -0.1102049128  2.7211575783 -1.6329178637 -1.3772822310 -2.0753984229
[326]  0.9353974825  0.9583562074 -2.9237501144 -0.8600837337  0.0515766319
[331] -0.4822839000  0.4655967120 -3.8698325172 -0.1799233863 -2.8965477543
[336] -0.9693511580  1.1059707246  2.2708348494 -0.0005851903  0.3773028163
[341]  0.0421754708  1.0315398745  2.1134145004 -0.5373266980 -0.2976942554
[346] -1.1196994973 -0.1722572473  0.3436538105  2.0908435052 -0.0432949153
[351] -0.2823763299  0.6069213324 -0.1740557687  3.6131447188 -1.6922366454
[356]  2.2992746643 -0.1186029785 -2.2177930119 -0.7110597939 -0.5716499249
[361]  0.2554604954 -2.7410878100  0.8196112713  0.4368076350  0.5193335976
[366] -0.7716689548  1.8496269842  3.6588462219 -0.2221727803 -2.4673272716
[371]  2.4643668099  0.5779554447  0.9303945179 -1.0435176694  2.0553796383
[376]  0.1288615944 -0.1089389497  1.4501197792  0.9060234857 -2.2409259558
[381] -1.6853201219 -1.1799553480  0.7523379162  1.4542834303  2.9590765834
[386] -0.6795371708  1.4041528529  3.4575379622  0.3573281589  3.4659399450
[391] -2.0570902287 -0.9819559396 -2.1056432557 -1.5910219649 -0.4244800501
[396]  3.0028191636 -3.8936900075 -0.3007653199  1.7573679538  2.6930939769
```

```
[401] -1.4046670247  2.1986835760 -0.9376937812 -0.4502023486 -3.1187215840
[406] -4.0299021827  2.5643512291  3.2927644251  1.0137654675 -0.0046640654
[411]  0.6493379579 -1.1419062852 -1.9747919322 -2.0052987856  0.9853041491
[416]  0.6558294466  0.1793909583  1.9674783727  0.4566307855 -2.0812951779
[421] -1.9941102283 -0.5913056542 -0.8395249866  0.0136766750 -0.9851075633
[426]  0.3205058864  1.7784224922 -0.8817036096 -1.8917102038 -0.8888332581
[431] -0.6721750981  3.0336342798  1.6801285796 -1.3488538181  1.7011336687
[436]  1.3279576038 -0.6092606371  0.5217279330 -2.1799168018  0.0895613236
[441] -0.3264496869  0.9551164430  1.9695264686  0.3276641155 -2.9838379859
[446]  1.0898758897  2.0707599062  1.3416541455  1.5763219272  1.1592824382
[451] -3.7665678617 -2.5587203797  0.0315259088  1.3469616994 -1.7737524432
[456]  0.0545321572  0.5493730964  1.1496535628 -0.3033832104 -0.3445266939
[461]  2.7762535716  1.4976517246 -2.0048095802  0.9694313837  1.9864184454
[466]  1.6583080819 -0.4540963955  2.3498509736  2.2201791138 -1.0178336296
[471] -0.2167698271  1.3754531412 -0.3109686449 -2.0039382205  2.3585789160
[476]  0.8859003599  0.5061261667 -0.3668239930 -1.7111405322  0.9946953922
[481]  0.0248171150 -1.1576630057  0.9889617691  0.9361039791  1.0708703313
[486] -2.7521028001  2.4569472066 -3.4666434243 -1.2505365900 -0.0929187948
[491]  4.2318030224 -0.8152540838  0.7418536268 -2.7473185994  0.4517318463
[496]  1.3564547631  0.4168272292 -2.0907354303  2.8228216998  0.7339674435
[501] -1.6347181535 -1.5589950069  0.6698153656  0.5769381226  0.4402270137
[506]  0.9253171466 -0.9499134208  2.0417886393 -4.3729922039 -2.6439790324
[511] -0.2373907664  1.1905984982  2.7605068808 -2.1917917435  1.3593443279
[516]  0.1830529137 -3.2324733853  3.5995097719  2.0819764566  0.0479657139
[521] -3.3365810550 -0.5998617290 -0.0026531806  1.7382414761 -1.8557728221
[526] -4.1434505607 -2.3167274433  1.1692596614 -0.7089921585  1.4852661781
[531]  2.9658408967  1.3955431148 -0.3791863293  0.7513273588 -1.8167734841
[536]  0.5383409903  0.2380772835 -0.8508777655 -1.1378258551 -0.8193273785
[541] -1.1856430319  1.6128489356 -2.2985695179  1.1945104833 -0.9604349205
[546]  2.6697184862  5.1548542586  1.7005128867 -1.1823026925 -0.6515701946
[551] -1.2746176586  1.7322961043  1.4238815062  0.8603388026 -0.1731636351
[556]  0.4132324226 -2.1346961219  0.0692863139  0.0733819372  1.8882511398
[561] -1.2520269377  0.2333378527  0.5483900564 -1.0397856566 -1.0177186297
[566]  1.1015789254  0.8156882890 -1.2321437471 -0.6779371302 -1.9931120410
[571] -2.2236867813 -0.0882132275 -0.2781046083 -0.5272087631  1.4370464820
[576]  0.9215736903 -2.7076283779 -2.1538600133 -3.6063295845 -2.1712891979
[581]  2.3292455982 -0.0727862551  2.6238214441  2.4531505271  1.2488758840
[586] -1.4315071132 -3.6559238248  1.2252915922 -0.2421943866 -2.7458807499
[591] -3.4398279958 -1.7678680311 -1.1092343405  1.8697088640 -3.6571526804
[596]  1.1625100247  1.0743859673 -2.6473191293 -0.6223061152 -0.7170926808
[601]  0.9865561863 -1.0696619771  0.3175302596 -0.3750945464  1.8886816973
[606] -5.1167891142  2.2925852780  1.1212595640 -1.5917603188  1.6656510109
[611]  1.3321761060 -1.3623416264  3.4698294937 -0.6490763698  0.8609981227
```

```
[616]  0.2934586628  1.8245273432  1.7464659954 -1.2595534421 -0.8136025596
[621]  0.3116158285  0.2518867813 -1.5493809484 -1.0623098538  1.7838126296
[626]  3.7092984903 -0.1596686675  2.6059367641 -2.4115940837  0.7604406148
[631]  2.4981138593  0.9564153278  1.7674682368  1.0197740839  0.5263089489
[636]  1.4262157357 -0.9272977352  1.6071922084 -1.0200511883  2.7082956425
[641]  0.0025486590  1.6776063946 -0.0896797094  0.1544998577  0.0811461396
[646] -0.8557383386  1.0557708093  2.4732948583 -0.8615998373  2.9702916465
[651] -2.1666547831  0.8679267522  0.7734226566  4.2439807356 -0.2271893435
[656]  0.3505460261 -0.7745086856 -1.1342947789 -0.8718945469  0.6424769215
[661]  3.1841045821  0.4150405518  3.3004080485 -1.4018522812  1.0107419379
[666]  0.0020053095  0.6194809389  1.4568265298 -2.0057971306  4.3545460644
[671]  0.1213144134 -1.7363645099  3.7283877218 -0.6103639127  2.3685273497
[676]  1.1442484005 -2.7335873552 -0.2340865224 -2.7478059611 -1.7390583372
[681]  0.4822850756 -1.2880037641  0.9343378012  1.1139643438 -0.1081677850
[686] -0.7122315559  2.0469138814 -0.2659639247  0.1803614577  0.4215160128
[691] -1.4152936442  0.8230128852  0.1378751454  1.1419818046  2.9258265216
[696] -0.3104303262 -0.2446584943 -0.8878646806  1.5311027558  0.7678052600
[701] -2.7204101578 -0.0541038956 -0.9893367368 -1.0228413752  0.6402591377
[706]  1.8788908141 -3.8603069154  2.4408801522 -2.5600976159  1.9108572132
[711]  0.7517217729  0.3207137967 -0.0257559959 -0.7307235812  0.1800629378
[716] -1.0677337218  2.0678618286  0.5941677156  3.3155577754 -2.6298134359
[721] -3.6556684574  1.0230965507 -0.9555113484 -1.3080485324  0.1293068671
[726] -0.0036227462  0.1222516010  0.7719766535 -0.9022448759  3.6209370727
[731] -2.4810219062  0.7013971045  5.8409247956 -3.1390342678  0.4283695399
[736] -2.3158299187 -2.7463510611  0.4512508601  0.1140393083 -0.3979149988
[741] -0.6879103253 -4.4334378979 -3.5498012670 -1.3743827745  0.5520230189
[746]  0.7557026249  1.5386371883 -3.1762697487 -1.3935911972  2.2912915104
[751]  0.2409713969  2.5767948810 -0.0180945743 -1.1992261994 -2.0383474702
[756]  0.6429836821  0.3534376749 -2.3352247321 -2.2773156300  2.6896487436
[761]  1.3036604208  3.6939342384 -0.9860958515  1.2398266315 -2.3160459463
[766]  0.7843060919  1.3630962263 -1.0020340164  0.6176732485 -0.1735856334
[771]  0.0414339060  5.0082931457 -1.6090424185 -2.4244061851 -3.4464673301
[776]  2.3738421073  0.7952495895  0.0694765317  0.7561935114  2.1197336037
[781] -0.9751529648 -4.9301107876  0.4376418604  2.1095618442  2.8074936273
[786] -1.5838173540  1.1903676575 -0.8461043924 -1.8221893809 -0.8345545120
[791] -1.8342221827  1.1923421392  0.5023426560 -0.9895095414  0.9197508159
[796] -4.3527874114 -1.6912178186  0.1073989119  1.4567130154 -1.7805709629
[801] -0.2136593629  1.1424579791  5.3572371473 -1.0083033866  0.1738627195
[806]  0.8866774707  1.6993465025  2.9998083374  1.9111190720 -1.2218514160
[811]  2.2096930428  0.9482090462  0.4980139649 -2.3078674492  1.2916037408
[816]  3.5160226658  0.3628375074  1.8553863676  0.7308659902  0.5875306266
[821]  1.7922324396 -0.4037576290 -1.3597613959  1.2340901161  0.3627696504
[826]  1.0403534233  0.5235215389 -0.6265382786  0.3772927983 -2.8034340813
```

```
[831]  3.0577283708  2.1295260292  0.0900435975 -1.6565142448  0.9245171189
[836]  1.6427563076  0.1843438884 -3.6310123147 -0.6981985692  0.3899747081
[841]  0.3492374619  0.1514043792 -2.3790114168 -1.4577351846  0.5815022306
[846]  2.5334538325  0.0854379310 -0.1053849327 -0.9935330858 -0.0575904717
[851] -0.7310301282 -0.7839963341 -0.6722308680  2.4839842141  1.3648216571
[856] -2.1074232016  0.5305324824 -2.6647195449 -2.5895619621 -0.5660598953
[861]  1.8988656688 -1.2321482240 -0.2997346246 -0.3917033537  0.2636984233
[866]  0.5875066090  0.5132412657 -2.5174111107 -1.6032803534 -1.8484212558
[871] -3.4570539096 -1.5507468274 -3.2572000094  2.2049215333  3.7880347197
[876]  1.1387913724 -0.2806984338 -0.3801715109 -3.2119653867  5.2637728186
[881] -1.1285707554  0.9713228633 -1.1012434113 -1.1681460137 -2.1795903234
[886]  1.8511310272  1.7704245062  2.3331348465  1.6459022641  0.1295939785
[891] -0.5821306161  0.3165956255 -2.5623819927  3.6293919002 -3.0044384793
[896]  1.5712368662  0.4237771776  2.5865717080  0.9441385133  2.7313870026
[901] -2.6253021115  1.0739938698 -1.9150809327  1.0113771424  1.5819941385
[906]  0.0888468862 -3.5545149152 -1.5024554496 -0.3669619290  1.3005276036
[911] -1.0688339700  3.7981136199  2.4510890225 -1.5498320132  3.4076327362
[916]  2.8161236968  2.5450342406  0.3244623313  2.7919197206  0.6579846326
[921]  1.5277228546  3.1080571831 -0.6567163514  0.5973448271  1.1295875054
[926]  1.5708612916  0.0457622606  1.6037434506 -0.8322443951  0.8699524509
[931] -0.3302479634 -0.0336797923  1.8898267483  0.3576721500 -0.7137163964
[936] -3.1771943000 -3.5122126136 -0.0388557490  2.9971007153  3.5031192895
[941]  2.8463617718  3.5843579510 -2.7030753338  1.1799088419 -1.4055797915
[946] -3.9308117148  1.6021094881  1.8903281786 -1.0079702410 -2.4338051514
[951]  0.1374814556  3.1986709724 -0.5297300081 -2.4353499870 -0.5588593146
[956]  1.1143357065 -4.0125851089 -1.6251362215  3.6959821543 -1.2640223284
[961]  0.1096410025 -1.6292929393  3.9869315149  3.1222775454  0.9303912989
[966]  0.9230763823 -1.0318546228 -0.2575261928  1.3776519038 -0.2430776875
[971] -5.6179810713 -2.4611520169 -3.0187688239  2.2071769682 -0.6685195458
[976] -0.5174305573 -0.2551943457 -0.6237478300 -3.3782683770 -2.7989810282
[981] -2.6696212506 -0.2060206367  2.8455349034  1.3198211812 -2.4157381091
[986]  3.8994885401 -0.7822550140 -0.7779757631 -1.6858462078 -0.7759917544
[991] -0.8672490700 -0.3211456187  1.6598379759 -2.4876168582 -1.6935015014
[996]  1.5579553552  6.5824501789  3.1579549835  0.8558636477 -1.4489784849
```

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

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

```
 [1]  0.32635666 -0.11691961  0.28528400  0.14473270  0.13879530  0.32578750
 [7]  0.10990659  0.03121288  0.38779619 -0.10646494
```

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

```
  [1]  0.512193626  0.647124912 -0.462956309 -0.535843455  1.639269779
  [6]  0.805390497  0.536570059  0.280204220  0.148720971 -0.735161071
 [11] -0.380720199  0.489304022  0.527105254 -1.079687819  0.264962820
 [16] -0.131735462  0.874146179 -0.264588722 -0.530658291  0.587623096
 [21] -0.060426707  1.023340336  0.093606691  0.398146510 -0.139674625
 [26]  0.423124442  0.682592541  0.862212363 -0.178278790 -0.730981224
 [31]  0.923748634  1.340431719 -0.380314636  0.372918961 -0.196386261
 [36] -0.076411741  0.349348746 -0.775684496 -0.301455300 -0.363389444
 [41] -0.047323541  0.608990208 -0.399866577 -0.130582450  0.323363824
 [46] -1.039732506 -0.453506397  0.872114940  0.400586797 -0.897882491
 [51]  0.202584512  1.588560499 -0.047491682  0.278596656 -0.637463422
 [56] -0.032854902  0.726261159 -0.369826151  0.547216843  0.003209608
 [61] -0.650771946  0.074665752  1.030307564  0.104961175 -0.101149745
 [66]  0.173050089 -0.021703075  0.590657024 -0.111534449  0.542047885
 [71] -0.326778539 -0.449411081  0.463998256  0.774439553  0.516827120
 [76] -0.587968699  0.272420972  0.986541675  0.479711051  0.605656976
 [81] -0.979840554 -0.337372723 -1.270208766  0.469686244 -0.374441632
 [86] -0.064211310  0.914586878  0.516967694  0.021853817  0.407853817
 [91]  0.046923936  0.232284207  0.103583597 -0.059214216  0.673522548
 [96]  1.069255705 -0.471548789  1.337678458 -0.498702522  1.207562023
```

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

```
 [1] 1.706799 2.037718 1.953685 2.121048 2.092075 1.824241 2.152782 2.107679
 [9] 2.116132 1.664882
```

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)
```

```r
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" ...
```

```r
# 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
```

```r
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.Da
                     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, "researche

# 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", "brow
                      hair_color = c("blonde", "light brown", "black", "brown", "blonde", "da
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
```

```r
# 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
```

```r
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.

```r
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 =
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.

```r
# 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

```r
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
```

```r
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
```

```r
min(x)
```

```
[1] 1
```

```r
max(x)
```

```
[1] 7
```

```r
median(x)
```

```
[1] 4
```

```r
mean(x)
```

```
[1] 4
```

```r
range(x)
```

```
[1] 1 7
```

```r
var(x)
```

```
[1] 6.666667
```

```r
sd(x)
```

```
[1] 2.581989
```

```r
quantile(x)
```

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

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

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

```r
IQR(x)
```

```
[1] 3
```

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

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

```r
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
```

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

```
[1] 1
```

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

```
[1] 2
```

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

```
[1] 1
```

```r
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 withour 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 withoute "" quotes !!!
cat("Hello", "world", "!", sep = "/")
```

```
Hello/world/!
```

```
# counting number of characters nchar()
nchar("Hello world")
```

```
[1] 11
```

```r
# 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
```

```r
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"       
```

```r
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 casefol
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 = "O", x = "Hello World")
```

```
[1] "HellO WOrld"
```

```
# 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
```

```r
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
```

```r
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"       
```

```r
countries[grepl(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"       
```

```r
# 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"                  
```

```r
# 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[grep("^W",strings)]
```

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

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

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

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

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

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

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

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

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

```
strings[grep("[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 occurence replaced
```

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

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

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

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

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

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

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

```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
```

```r
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
```

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

```
[1] 10 10
```

```r
nrow(df3)
```

```
[1] 10
```

```r
ncol(df3)
```

```
[1] 10
```

```r
# 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.Da
                     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", "ar
df_emp$new_col <- 1

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

# 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", "brow
                      hair_color = c("blonde", "light brown", "black", "brown", "blonde", "da
df_emp <- cbind(df_emp, df_attr)


# 4.3.4 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  
```

```r
# 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
```

```r
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
```

```r
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
```

```r
X %in% Y
```

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

```r
# 5.1.3 Logical operators

X | Y
```

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

```r
#X || Y
X & Y
```

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

```r
#X && Y
!X
```

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

```r
# 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$"
```

```r
# 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

```r
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

```r
adsl <- haven::read_sas("data/adam/adsl.sas7bdat")
adsl2 <- dplyr::select(adsl, -c(AGE, SEX, TRT01A))
adsl3 <- dplyr::select(adsl, -c(STUDYID:ARM))
adsl4 <- dplyr::select(adsl, -c(1:6))
adsl4 <- dplyr::select(adsl, -c(1:6, 43))
adsl5 <- dplyr::select(adsl, -starts_with("A"))
adsl6 <- dplyr::select(adsl, -ends_with("DTC"))
adsl7 <- dplyr::select(adsl, -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”:

```r
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))
```

```r
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(

adsl2 <- adsl |>
  dplyr::select(USUBJID,TRT01A,SEX,AGE) |>
  dplyr::filter(TRT01A %in% c("Placebo","Xanomeline High Dose") & !SEX == "M" & AGE >= 70)
```

28 Rename

```r
adsl1 <- adsl |>
    dplyr::rename(usubjid=USUBJID)

adsl1 <- adsl |>
  dplyr::rename(usubjid=USUBJID,
                trt=TRT01P)
```

29 Arrange (sorting)

```r
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(adsl))
```

```
 [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" 
```

```
adsl1 <- adsl

names(adsl1) <- tolower(names(adsl1))
```

31 Creating new variables

```r
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 innter 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
righjointab <- right_join(x=tab_a,y=tab_b,by="id")
```

```r
#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

```r
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:

```r
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)
```

```r
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.

```r
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

```r
lst <- list(a=1:3, b=10:12)
lapply(lst, mean)
```

```
$a
[1] 2

$b
[1] 11
```

41 Exercises

1. Using `across()`, standardize (`(x-mean)/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 String Functions

43 String Functions in Base R

Strings (or character vectors) are a fundamental data type in R, and base R provides a variety of functions to manipulate and analyze strings. Here are some commonly used string functions in base R:

```r
# Create a character vector
text <- c("Hello, World!", "R is great", "Data Science")
```

43.1 Basic String Functions

```r
# nchar(): Returns the number of characters in a string
nchar(text)
```

```
[1] 13 10 12
```

```r
# tolower(): Converts a string to lowercase
tolower(text)
```

```
[1] "hello, world!" "r is great"    "data science"
```

```r
# toupper(): Converts a string to uppercase
toupper(text)
```

```
[1] "HELLO, WORLD!" "R IS GREAT"    "DATA SCIENCE"
```

```r
# substr(): Extracts a substring from a string
substr(text, start = 1, stop = 5)
```

```
[1] "Hello" "R is " "Data "
```

```
# paste(): Concatenates strings together
paste("Hello", "R", sep = " ")
```

```
[1] "Hello R"
```

```
# paste0(): Concatenates strings without any separator
paste0("Hello", "R")
```

```
[1] "HelloR"
```

```
# trimws(): Trims leading and trailing whitespace from a string
trimws("   Hello, World!   ")
```

```
[1] "Hello, World!"
```

43.2 Pattern Matching and Replacement

```
# grep(): Searches for patterns in a character vector and returns the indices of matches
grep("R", text)
```

```
[1] 2
```

```
# grepl(): Searches for patterns and returns a logical vector indicating matches
grepl("R", text)
```

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

```
# sub(): Replaces the first occurrence of a pattern in a string
sub("great", "awesome", text)
```

```
[1] "Hello, World!" "R is awesome"  "Data Science"
```

```
# gsub(): Replaces all occurrences of a pattern in a string
gsub(" ", "_", text)
```

```
[1] "Hello,_World!" "R_is_great"    "Data_Science"
```

43.3 String Splitting and Joining

```r
# strsplit(): Splits a string into substrings based on a specified delimiter
strsplit(text, split = " ")
```

```
[[1]]
[1] "Hello," "World!"

[[2]]
[1] "R"     "is"    "great"

[[3]]
[1] "Data"    "Science"
```

```r
# unlist(): Converts a list to a vector (useful after strsplit)
unlist(strsplit(text, split = " "))
```

```
[1] "Hello,"  "World!"  "R"       "is"      "great"   "Data"    "Science"
```

43.4 Regular Expressions

```r
# regexpr(): Finds the position of the first match of a pattern in a string
regexpr("R", text)
```

```
[1] -1  1 -1
attr(,"match.length")
[1] -1  1 -1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

```r
# gregexpr(): Finds the positions of all matches of a pattern in a string
gregexpr(" ", text)
```

```
[[1]]
[1] 7
attr(,"match.length")
[1] 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE

[[2]]
[1] 2 5
attr(,"match.length")
[1] 1 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE

[[3]]
[1] 5
attr(,"match.length")
[1] 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

```
# regmatches(): Extracts the matched substrings based on the positions returned by regexpr o
regmatches(text, gregexpr(" ", text))
```

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

[[2]]
[1] " " " "

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

43.5 Example Usage

```r
# Example: Convert text to lowercase and replace spaces with underscores
cleaned_text <- tolower(gsub(" ", "_", text))
cleaned_text
```

```
[1] "hello,_world!" "r_is_great"    "data_science"
```

These functions provide a solid foundation for string manipulation in R. For more advanced string operations, you # using the `stringr` package, which offers a more consistent and user-friendly interface for string handling.

44 Overall differences

We'll begin with a lookup table between the most important stringr functions and their base R equivalents.

```r
library(stringr)
data_stringr_base_diff <- tibble::tribble(
  ~stringr,                                        ~base_r,
  "str_detect(string, pattern)",                   "grepl(pattern, x)",
  "str_dup(string, times)",                        "strrep(x, times)",
  "str_extract(string, pattern)",                  "regmatches(x, m = regexpr(pattern, text)
  "str_extract_all(string, pattern)",              "regmatches(x, m = gregexpr(pattern, text
  "str_length(string)",                            "nchar(x)",
  "str_locate(string, pattern)",                   "regexpr(pattern, text)",
  "str_locate_all(string, pattern)",               "gregexpr(pattern, text)",
  "str_match(string, pattern)",                    "regmatches(x, m = regexec(pattern, text)
  "str_order(string)",                             "order(...)",
  "str_replace(string, pattern, replacement)",     "sub(pattern, replacement, x)",
  "str_replace_all(string, pattern, replacement)", "gsub(pattern, replacement, x)",
  "str_sort(string)",                              "sort(x)",
  "str_split(string, pattern)",                    "strsplit(x, split)",
  "str_sub(string, start, end)",                   "substr(x, start, stop)",
  "str_subset(string, pattern)",                   "grep(pattern, x, value = TRUE)",
  "str_to_lower(string)",                          "tolower(x)",
  "str_to_title(string)",                          "tools::toTitleCase(text)",
  "str_to_upper(string)",                          "toupper(x)",
  "str_trim(string)",                              "trimws(x)",
  "str_which(string, pattern)",                    "grep(pattern, x)",
  "str_wrap(string)",                              "strwrap(x)"
)

# create MD table, arranged alphabetically by stringr fn name
data_stringr_base_diff |>
  dplyr::mutate(dplyr::across(
      .cols = everything(),
      .fns = ~ paste0("`", .x, "`"))
  ) |>
```

stringr	base R
str_detect(string, pattern)	grepl(pattern, x)
str_dup(string, times)	strrep(x, times)
str_extract(string, pattern)	regmatches(x, m = regexpr(pattern, text
str_extract_all(string, pattern)	regmatches(x, m = gregexpr(pattern, tex
str_length(string)	nchar(x)
str_locate(string, pattern)	regexpr(pattern, text)
str_locate_all(string, pattern)	gregexpr(pattern, text)
str_match(string, pattern)	regmatches(x, m = regexec(pattern, text
str_order(string)	order(...)
str_replace(string, pattern, replacement)	sub(pattern, replacement, x)
str_replace_all(string, pattern, replacement)	gsub(pattern, replacement, x)
str_sort(string)	sort(x)
str_split(string, pattern)	strsplit(x, split)
str_sub(string, start, end)	substr(x, start, stop)
str_subset(string, pattern)	grep(pattern, x, value = TRUE)
str_to_lower(string)	tolower(x)
str_to_title(string)	tools::toTitleCase(text)
str_to_upper(string)	toupper(x)
str_trim(string)	trimws(x)
str_which(string, pattern)	grep(pattern, x)
str_wrap(string)	strwrap(x)

```r
dplyr::arrange(stringr) |>
dplyr::rename(`base R` = base_r) |>
gt::gt() |>
gt::fmt_markdown(columns = everything()) |>
gt::tab_options(column_labels.font.weight = "bold")
```

Overall the main differences between base R and stringr are:

1. stringr functions start with `str_` prefix; base R string functions have no consistent naming scheme.
2. The order of inputs is usually different between base R and stringr. In base R, the `pattern` to match usually comes first; in stringr, the `string` to manupulate always comes first. This makes stringr easier to use in pipes, and with `lapply()` or `purrr::map()`.
3. Functions in stringr tend to do less, where many of the string processing functions in base R have multiple purposes.

4. The output and input of stringr functions has been carefully designed. For example, the output of `str_locate()` can be fed directly into `str_sub()`; the same is not true of `regexpr()` and `substr()`.

5. Base functions use arguments (like `perl`, `fixed`, and `ignore.case`) to control how the pattern is interpreted. To avoid dependence between arguments, stringr instead uses helper functions (like `fixed()`, `regex()`, and `coll()`).

Next we'll walk through each of the functions, noting the similarities and important differences. These examples are adapted from the stringr documentation and here they are contrasted with the analogous base R operations.

45 Detect matches

45.1 `str_detect()`: Detect the presence or absence of a pattern in a string

Suppose you want to know whether each word in a vector of fruit names contains an "a".

```
fruit <- c("apple", "banana", "pear", "pineapple")

# base
grepl(pattern = "a", x = fruit)
```

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

```
#stringr
stringr::str_detect(fruit, pattern = "a")
```

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

In base you would use `grepl()` (see the "l" and think logical) while in stringr you use `str_detect()` (see the verb "detect" and think of a yes/no action).

45.2 `str_which()`: Find positions matching a pattern

Now you want to identify the positions of the words in a vector of fruit names that contain an "a".

```
# base
grep(pattern = "a", x = fruit)
```

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

```r
# stringr
str_which(fruit, pattern = "a")
```

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

In base you would use `grep()` while in stringr you use `str_which()` (by analogy to `which()`).

45.3 `str_count()`: Count the number of matches in a string

How many "a"s are in each fruit?

```r
# base
loc <- gregexpr(pattern = "a", text = fruit, fixed = TRUE)
sapply(loc, function(x) length(attr(x, "match.length")))
```

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

```r
# stringr
str_count(fruit, pattern = "a")
```

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

This information can be gleaned from `gregexpr()` in base, but you need to look at the `match.length` attribute as the vector uses a length-1 integer vector (`-1`) to indicate no match.

45.4 `str_locate()`: Locate the position of patterns in a string

Within each fruit, where does the first "p" occur? Where are all of the "p"s?

```r
fruit3 <- c("papaya", "lime", "apple")

# base
str(gregexpr(pattern = "p", text = fruit3))
```

```
List of 3
 $ : int [1:2] 1 3
  ..- attr(*, "match.length")= int [1:2] 1 1
  ..- attr(*, "index.type")= chr "chars"
  ..- attr(*, "useBytes")= logi TRUE
 $ : int -1
  ..- attr(*, "match.length")= int -1
  ..- attr(*, "index.type")= chr "chars"
  ..- attr(*, "useBytes")= logi TRUE
 $ : int [1:2] 2 3
  ..- attr(*, "match.length")= int [1:2] 1 1
  ..- attr(*, "index.type")= chr "chars"
  ..- attr(*, "useBytes")= logi TRUE
```

```
# stringr
str_locate(fruit3, pattern = "p")
```

```
     start end
[1,]     1   1
[2,]    NA  NA
[3,]     2   2
```

```
str_locate_all(fruit3, pattern = "p")
```

```
[[1]]
     start end
[1,]     1   1
[2,]     3   3

[[2]]
     start end

[[3]]
     start end
[1,]     2   2
[2,]     3   3
```

45.5 `str_subset()`: Keep strings matching a pattern, or find positions

We may want to retrieve strings that contain a pattern of interest:

```
# base
grep(pattern = "g", x = fruit, value = TRUE)
```

```
character(0)
```

```
# stringr
str_subset(fruit, pattern = "g")
```

```
character(0)
```

45.6 `str_extract()`: Extract matching patterns from a string

We may want to pick out certain patterns from a string, for example, the digits in a shopping list:

```
shopping_list <- c("apples x4", "bag of flour", "10", "milk x2")

# base
matches <- regexpr(pattern = "\\d+", text = shopping_list) # digits
regmatches(shopping_list, m = matches)
```

```
[1] "4"  "10" "2"
```

```
matches <- gregexpr(pattern = "[a-z]+", text = shopping_list) # words
regmatches(shopping_list, m = matches)
```

```
[[1]]
[1] "apples" "x"

[[2]]
[1] "bag"   "of"    "flour"

[[3]]
character(0)

[[4]]
[1] "milk" "x"
```

```r
# stringr
str_extract(shopping_list, pattern = "\\d+")
```

```
[1] "4"  NA   "10" "2"
```

```r
str_extract_all(shopping_list, "[a-z]+")
```

```
[[1]]
[1] "apples" "x"

[[2]]
[1] "bag"   "of"    "flour"

[[3]]
character(0)

[[4]]
[1] "milk" "x"
```

Base R requires the combination of `regexpr()` with `regmatches()`; but note that the strings without matches are dropped from the output. stringr provides `str_extract()` and `str_extract_all()`, and the output is always the same length as the input.

45.7 `str_match()`: Extract matched groups from a string

We may also want to extract groups from a string. Here I'm going to use the scenario from Section 14.4.3 in R for Data Science.

```r
head(sentences)
```

```
[1] "The birch canoe slid on the smooth planks."
[2] "Glue the sheet to the dark blue background."
[3] "It's easy to tell the depth of a well."
[4] "These days a chicken leg is a rare dish."
[5] "Rice is often served in round bowls."
[6] "The juice of lemons makes fine punch."
```

```
noun <- "([A]a|[Tt]he) ([^ ]+)"

# base
matches <- regexec(pattern = noun, text = head(sentences))
do.call("rbind", regmatches(x = head(sentences), m = matches))
```

```
     [,1]        [,2]  [,3]
[1,] "The birch" "The" "birch"
[2,] "the sheet" "the" "sheet"
[3,] "the depth" "the" "depth"
[4,] "The juice" "The" "juice"
```

```
# stringr
str_match(head(sentences), pattern = noun)
```

```
     [,1]        [,2]  [,3]
[1,] "The birch" "The" "birch"
[2,] "the sheet" "the" "sheet"
[3,] "the depth" "the" "depth"
[4,] NA          NA    NA
[5,] NA          NA    NA
[6,] "The juice" "The" "juice"
```

As for extracting the full match base R requires the combination of two functions, and inputs with no matches are dropped from the output.

46 Manage lengths

46.1 str_length(): The length of a string

To determine the length of a string, base R uses `nchar()` (not to be confused with `length()` which gives the length of vectors, etc.) while stringr uses `str_length()`.

```r
# base
nchar(letters)
```

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

```r
# stringr
str_length(letters)
```

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

There are some subtle differences between base and stringr here. `nchar()` requires a character vector, so it will return an error if used on a factor. `str_length()` can handle a factor input.

```r
# base
nchar(factor("abc"))
```

```
Error in nchar(factor("abc")): 'nchar()' requires a character vector
```

```r
# stringr
str_length(factor("abc"))
```

```
[1] 3
```

Note that "characters" is a poorly defined concept, and technically both `nchar()` and `str_length()` returns the number of code points. This is usually the same as what you'd consider to be a charcter, but not always:

```r
x <- c("\u00fc", "u\u0308")
x
```

```
[1] "ü" "ü"
```

```r
nchar(x)
```

```
[1] 1 2
```

```r
str_length(x)
```

```
[1] 1 2
```

46.2 `str_pad()`: Pad a string

To pad a string to a certain width, use stringr's `str_pad()`. In base R you could use `sprintf()`, but unlike `str_pad()`, `sprintf()` has many other functionalities.

```r
# base
sprintf("%30s", "Sriram")
```

```
[1] "                        Sriram"
```

```r
sprintf("%-30s", "Sriram")
```

```
[1] "Sriram                        "
```

```r
# "both" is not as straightforward

# stringr
rbind(
  str_pad("Sriram", 30, "left"),
  str_pad("Sriram", 30, "right"),
  str_pad("Sriram", 30, "both")
)
```

```
     [,1]
[1,] "                        Sriram"
[2,] "Sriram                        "
[3,] "            Sriram            "
```

46.3 `str_trunc()`: Truncate a character string

The stringr package provides an easy way to truncate a character string: `str_trunc()`. Base R has no function to do this directly.

```
x <- "This string is moderately long"

# stringr
rbind(
  str_trunc(x, 20, "right"),
  str_trunc(x, 20, "left"),
  str_trunc(x, 20, "center")
)
```

```
     [,1]
[1,] "This string is mo..."
[2,] "...s moderately long"
[3,] "This stri...ely long"
```

46.4 `str_trim()`: Trim whitespace from a string

Similarly, stringr provides `str_trim()` to trim whitespace from a string. This is analogous to base R's `trimws()` added in R 3.3.0.

```
# base
trimws(" String with trailing and leading white space\t")
```

```
[1] "String with trailing and leading white space"
```

```
trimws("\n\nString with trailing and leading white space\n\n")
```

```
[1] "String with trailing and leading white space"
```

```
# stringr
str_trim(" String with trailing and leading white space\t")
```

```
[1] "String with trailing and leading white space"
```

```
str_trim("\n\nString with trailing and leading white space\n\n")
```

```
[1] "String with trailing and leading white space"
```

The stringr function `str_squish()` allows for extra whitespace within a string to be trimmed (in contrast to `str_trim()` which removes whitespace at the beginning and/or end of string). In base R, one might take advantage of `gsub()` to accomplish the same effect.

```
# stringr
str_squish(" String with trailing, middle,   and leading white space\t")
```

```
[1] "String with trailing, middle, and leading white space"
```

```
str_squish("\n\nString with excess, trailing and leading white space\n\n")
```

```
[1] "String with excess, trailing and leading white space"
```

46.5 `str_wrap()`: Wrap strings into nicely formatted paragraphs

`strwrap()` and `str_wrap()` use different algorithms. `str_wrap()` uses the famous Knuth-Plass algorithm.

```
gettysburg <- "Four score and seven years ago our fathers brought forth on this continent, a

# base
cat(strwrap(gettysburg, width = 60), sep = "\n")
```

```
Four score and seven years ago our fathers brought forth on
this continent, a new nation, conceived in Liberty, and
dedicated to the proposition that all men are created
equal.
```

```
# stringr
cat(str_wrap(gettysburg, width = 60), "\n")
```

```
Four score and seven years ago our fathers brought forth
on this continent, a new nation, conceived in Liberty, and
dedicated to the proposition that all men are created equal.
```

Note that `strwrap()` returns a character vector with one element for each line; `str_wrap()` returns a single string containing line breaks.

47 Mutate strings

47.1 `str_replace()`: Replace matched patterns in a string

To replace certain patterns within a string, stringr provides the functions `str_replace()` and `str_replace_all()`. The base R equivalents are `sub()` and `gsub()`. Note the difference in default input order again.

```r
fruits <- c("apple", "banana", "pear", "pineapple")

# base
sub("[aeiou]", "-", fruits)
```

```
[1] "-pple"     "b-nana"    "p-ar"      "p-neapple"
```

```r
gsub("[aeiou]", "-", fruits)
```

```
[1] "-ppl-"     "b-n-n-"    "p--r"      "p-n--ppl-"
```

```r
# stringr
str_replace(fruits, "[aeiou]", "-")
```

```
[1] "-pple"     "b-nana"    "p-ar"      "p-neapple"
```

```r
str_replace_all(fruits, "[aeiou]", "-")
```

```
[1] "-ppl-"     "b-n-n-"    "p--r"      "p-n--ppl-"
```

47.2 case: Convert case of a string

Both stringr and base R have functions to convert to upper and lower case. Title case is also provided in stringr.

```
dog <- "The quick brown dog"

# base
toupper(dog)
```

```
[1] "THE QUICK BROWN DOG"
```

```
tolower(dog)
```

```
[1] "the quick brown dog"
```

```
tools::toTitleCase(dog)
```

```
[1] "The Quick Brown Dog"
```

```
# stringr
str_to_upper(dog)
```

```
[1] "THE QUICK BROWN DOG"
```

```
str_to_lower(dog)
```

```
[1] "the quick brown dog"
```

```
str_to_title(dog)
```

```
[1] "The Quick Brown Dog"
```

In stringr we can control the locale, while in base R locale distinctions are controlled with global variables. Therefore, the output of your base R code may vary across different computers with different global settings.

```
# stringr
str_to_upper("i") # English
```

```
[1] "I"
```

```r
str_to_upper("i", locale = "tr") # Turkish
```

```
[1] "İ"
```

48 Join and split

48.1 `str_flatten()`: Flatten a string

If we want to take elements of a string vector and collapse them to a single string we can use the `collapse` argument in `paste()` or use stringr's `str_flatten()`.

```
# base
paste0(letters, collapse = "-")
```

```
[1] "a-b-c-d-e-f-g-h-i-j-k-l-m-n-o-p-q-r-s-t-u-v-w-x-y-z"
```

```
# stringr
str_flatten(letters, collapse = "-")
```

```
[1] "a-b-c-d-e-f-g-h-i-j-k-l-m-n-o-p-q-r-s-t-u-v-w-x-y-z"
```

The advantage of `str_flatten()` is that it always returns a vector the same length as its input; to predict the return length of `paste()` you must carefully read all arguments.

48.2 `str_dup()`: duplicate strings within a character vector

To duplicate strings within a character vector use `strrep()` (in R 3.3.0 or greater) or `str_dup()`:

```
fruit <- c("apple", "pear", "banana")

# base
strrep(fruit, 2)
```

```
[1] "appleapple"   "pearpear"     "bananabanana"
```

```
strrep(fruit, 1:3)
```

```
[1] "apple"              "pearpear"           "bananabananabanana"
```

```
# stringr
str_dup(fruit, 2)
```

```
[1] "appleapple"   "pearpear"     "bananabanana"
```

```
str_dup(fruit, 1:3)
```

```
[1] "apple"              "pearpear"           "bananabananabanana"
```

48.3 `str_split()`: Split up a string into pieces

To split a string into pieces with breaks based on a particular pattern match stringr uses `str_split()` and base R uses `strsplit()`. Unlike most other functions, `strsplit()` starts with the character vector to modify.

```
fruits <- c(
  "apples and oranges and pears and bananas",
  "pineapples and mangos and guavas"
)
# base
strsplit(fruits, " and ")
```

```
[[1]]
[1] "apples"  "oranges" "pears"   "bananas"

[[2]]
[1] "pineapples" "mangos"     "guavas"
```

```
# stringr
str_split(fruits, " and ")
```

```
[[1]]
[1] "apples"  "oranges" "pears"   "bananas"

[[2]]
[1] "pineapples" "mangos"     "guavas"
```

The stringr package's `str_split()` allows for more control over the split, including restricting the number of possible matches.

```
# stringr
str_split(fruits, " and ", n = 3)
```

```
[[1]]
[1] "apples"            "oranges"           "pears and bananas"

[[2]]
[1] "pineapples" "mangos"     "guavas"
```

```
str_split(fruits, " and ", n = 2)
```

```
[[1]]
[1] "apples"                        "oranges and pears and bananas"

[[2]]
[1] "pineapples"        "mangos and guavas"
```

48.4 `str_glue()`: Interpolate strings

It's often useful to interpolate varying values into a fixed string. In base R, you can use `sprintf()` for this purpose; stringr provides a wrapper for the more general purpose glue package.

```
name <- "Fred"
age <- 50
anniversary <- as.Date("1991-10-12")

# base
sprintf(
  "My name is %s my age next year is %s and my anniversary is %s.",
  name,
  age + 1,
  format(anniversary, "%A, %B %d, %Y")
)
```

```
[1] "My name is Fred my age next year is 51 and my anniversary is Saturday, October 12, 1991
```

```
# stringr
str_glue(
  "My name is {name}, ",
  "my age next year is {age + 1}, ",
  "and my anniversary is {format(anniversary, '%A, %B %d, %Y')}."
)
```

```
My name is Fred, my age next year is 51, and my anniversary is Saturday, October 12, 1991.
```

49 Order strings

49.1 `str_order()`: Order or sort a character vector

Both base R and stringr have separate functions to order and sort strings.

```
# base
order(letters)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

```
sort(letters)
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

```
# stringr
str_order(letters)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

```
str_sort(letters)
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

Some options in `str_order()` and `str_sort()` don't have analogous base R options. For example, the stringr functions have a `locale` argument to control how to order or sort. In base R the locale is a global setting, so the outputs of `sort()` and `order()` may differ across different computers. For example, in the Norwegian alphabet, å comes after z:

```
x <- c("å", "a", "z")
str_sort(x)
```

```
[1] "a" "å" "z"
```

```
str_sort(x, locale = "no")
```

```
[1] "a" "z" "å"
```

The stringr functions also have a `numeric` argument to sort digits numerically instead of treating them as strings.

```
# stringr
x <- c("100a10", "100a5", "2b", "2a")
str_sort(x)
```

```
[1] "100a10" "100a5"  "2a"     "2b"
```

```
str_sort(x, numeric = TRUE)
```

```
[1] "2a"     "2b"     "100a5"  "100a10"
```

```
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
```

```r
ae <- haven::read_sas("./data/sdtm/ae.sas7bdat")
ae1 <- ae |>
  select (USUBJID,AEDECOD,AEBODSYS)

#find function in sas, str_detect

ae2 <- ae1 |>
  filter(str_detect(AEDECOD,"HER"))

ae2 <- ae1 |>
  filter(str_detect(AEDECOD,"^HIATUS ")) # start of the string

ae2 <- ae1 |>
  filter(str_detect(AEDECOD,"HERNIA$")) # end of the string

#substr in sas

ae2 <- ae1 |>
  mutate(newvar1=str_sub(AEDECOD,1,6),
         newvar2=str_sub(AEDECOD,2,6),
         newvar3=str_sub(AEDECOD,-3),
         newvar4=str_length(AEDECOD))

#cat function in sas , Str_c in r

ae2 <- ae1 |>
  mutate(newvar1=str_c(AEDECOD,AEBODSYS,sep="/"),
         newvar2=paste(AEDECOD,AEBODSYS,sep="/"))

# scan function in sas, word in r

ae2 <- ae1 |>
  mutate(newvar1=word(AEDECOD,1),
         newvar2=word(AEDECOD,2))

# upper & lower case

ae2 <- ae1 |>
  mutate(newvar1=str_to_lower(AEDECOD),
         newvar2=str_to_upper(AEDECOD),
         newvar3=str_to_title(AEDECOD),
         newvar4=str_to_sentence(AEDECOD))
```

```r
#str_trim

a <- "  this is    my String   "
b <- str_trim(a)
c <- str_replace_all(a," "," ")
d <- str_squish(a)
```

50 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] "CDISCPILOT01" "CDISCPILOT01" "CDISCPILOT01" "CDISCPILOT01" ...
  ..- 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"
```

50.1 Handling Labels & Missing

```r
# 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))
```

50.2 Labelled to Factor (if needed)

```r
if (inherits(dm$SEX, "labelled")) {
  dm <- dm |> mutate(SEX = to_factor(SEX))
}
```

50.3 Common Cleaning

```r
dm <- dm |>
  mutate(
    AGEGR1 = cut(AGE, breaks=c(-Inf, 50, 65, Inf),
                 labels=c("<50","50-65", "65+"))
  )
```

51 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`.

52 Base R Functions & Apply Family

53 Common Utilities

```r
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 
```

```r
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
```

54 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
```

55 Subsetting Essentials

```r
df <- data.frame(id=1:3, val=c(10,20,30))
df[1, "val"]
```

```
[1] 10
```

```r
df[df$val > 10, ]
```

```
  id val
2  2  20
3  3  30
```

56 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.

57 Custom Functions & Validation

58 Writing Functions

```r
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
```

59 Error Handling

```r
robust_divide <- function(a, b) {
  tryCatch(a / b,
           warning = function(w) NA_real_,
           error   = function(e) NA_real_)
}
robust_divide(10, 0)
```

```
[1] Inf
```

60 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
```

61 Document with roxygen2

```r
#' 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:

```r
devtools::document()
```

62 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.

63 R Package Development

63.1 Setup

```
install.packages(c("usethis","devtools","testthat","roxygen2","pkgdown"))
```

63.2 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
```

63.3 Add a Function

Create `R/safe_mean.R` and its tests (see previous chapter).

63.4 Build, Install, Check

```
devtools::document()
devtools::build()
devtools::install()
devtools::check()
```

63.5 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.

64 Git in RStudio (Setup & Auth)

64.1 One-Time Setup

- Install Git and ensure `git --version` works.
- In R:

```r
usethis::use_git_config(user.name = "Your Name", user.email = "you@example.com")
```

64.2 Initialize Git for the Current Project

```r
usethis::use_git()
```

64.3 Connect to GitHub

- Create a GitHub account.
- In R:

```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`).

64.4 Typical Workflow

1. Stage changes (Git pane in RStudio).
2. Commit with a clear message.
3. Push to origin (GitHub).

64.5 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.

65 Creating ADaM: ADSL from SDTM-like Inputs

```r
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
```

```r
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)
)
```

65.1 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     2025-02-03 2025-02-03 Placebo
 2 XYZ123  XYZ-002 Active        2    62 F     2025-01-29 2025-02-01 Active
 3 XYZ123  XYZ-003 Placebo       1    47 M     2025-01-27 2025-01-28 Placebo
 4 XYZ123  XYZ-004 Active        2    50 F     2025-01-18 2025-01-18 Active
 5 XYZ123  XYZ-005 Placebo       1    71 M     2025-02-04 2025-02-04 Placebo
 6 XYZ123  XYZ-006 Active        2    66 F     2025-01-18 2025-01-20 Active
 7 XYZ123  XYZ-007 Placebo       1    45 M     2025-02-03 2025-02-04 Placebo
 8 XYZ123  XYZ-008 Active        2    59 F     2025-01-26 2025-01-28 Active
 9 XYZ123  XYZ-009 Placebo       1    53 M     2025-01-26 2025-01-29 Placebo
10 XYZ123  XYZ-010 Active        2    68 F     2025-01-18 2025-01-19 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.

66 TLFs: Table, Figure, Listing

```r
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
```

```r
library(gt)
library(ggplot2)
library(survival)
```

We reuse `adsl` from the previous chapter (or synthesize if missing).

```r
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]

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

66.1 Table 1: Baseline Characteristics by Treatment

```r
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)
```

66.2 Figure: (Toy) Survival Curve

We simulate time-to-event data for illustration only.

```r
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="Trea
    theme_minimal()
}
#ggsurv(fit)
```

66.3 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 (mean ± 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

67 Capstone: End-to-End Mini Workflow

This chapter ties everything together: **read data → derive ADSL → produce TLFs → render a report.**

67.1 Parameters

```
# You could parametrize paths via YAML; here we keep inline defaults.
dm_path <- "data/sdtm/dm.sas7bdat"
ex_path <- "data/sdtm/ex.sas7bdat"
```

67.2 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
```

```r
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)
  )
}
```

67.3 2) Derive ADSL (Minimal Demo)

```r
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, SAF
```

[

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

67.4 3) TLFs

```r
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
```

```r
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)
```

67.5 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.

68 Appendix: Tips, Profiles, .libPaths

68.1 Useful Profiles

Create `~/.Rprofile` to set options (be careful on shared systems):

```
options(
  repos = c(CRAN = "https://cloud.r-project.org"),
  scipen = 999
)
```

68.2 Custom Library Paths

```
# In .Rprofile or project-level .Rprofile
.libPaths(c("/path/to/Rlibs", .libPaths()))
```

68.3 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"
```

68.4 POSIXct vs POSIXlt

- **POSIXct**: seconds since epoch (numeric), compact, fast.
- **POSIXlt**: list-like with components (year, mon, mday…), easier to extract parts.

68.5 Recommended Packages

- `tidyverse`, `lubridate`, `janitor`, `gt`, `gtsummary`, `survival`, `broom`, `here`.
- Pharma/CDISC: `admiral`, `tlf/tern`, `pharmaverse` meta-packages (explore as you grow).

68.6 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.