Public Health Insights: Clinical Reporting Using {gtsummary}

Md. Jubayer Hossain

CHIRAL Bangladesh

Founder & Management Lead

Last Updated on September, 2023

Agenda

- Getting Started With R
- Data Types in R
- Operators in R
- gtsummary - Motivation
- tbl_summary()
- tbl_regression()
- tbl_merge()/tbl_stack()
- {gtsummary} themes
- {gtsummary} print engines
- inline_text()
- In Summary

Getting Started With R

Your Setup

If you can, we suggest working virtually with a **large monitor or two screens**. This setup allows you to follow along on Zoom while also doing the hands-on coding.

- Install the latest version from: https://posit.co/
- Install RStudio from : https://posit.co/download/rstudio-desktop/

RStudio is an **integrated development environment** (IDE) that makes it easier to work with R.

More on that soon!

RStudio - Major concepts

- **RStudio** - an Integrated Development Environment (IDE) for R - makes it easier to use R.
- **Source/Editor** - “Analysis” Script + Interactive Exploration - In a .R file (we call a script), code is saved on your disk
- **R Console** - Where code is executed (where things happen) - Code is not saved on your disk
- **Workspace/Environment** - Tells you what objects are in R. What exists in memory/what is loaded?/what did I read in?
- **R Markdown** - Files (.Rmd) help generate reports that include your code and output.

RStudio

- **Quarto** - An open-source scientific and technical publishing system. Files (.qmd) help generate reports that include your code and output. https://quarto.org/
- **R Project** - Helps you organize your work. Helps with working directories (discussed later). Allows you to easily know which project you're on.
- **Quarto Project** - Quarto projects are directories that provide: A way to render all or some of the files in a directory with a single command (e.g. quarto render myproject).
- RStudio Keyboard shortcuts: http://www.rstudio.com/ide/docs/using/keyboard_shortcuts

What is Reproducibility?

- **Reproducibility** - A different analyst re-performs the analysis with the same code and the same data and obtains the same result.
- **Repeatable** - keeping everything the same but repeating the analysis - do we get the same results
- **Reproducible** - using the same data and analysis but in the hands of another researcher - do we get the same results?
- **Replicable** - with new data do we obtain the same inferences?

Running Your First R Program

- Now that you have installed R and RStudio successfully, let's try to create your first R program. We will try to create a simple Hello World program.
- A Hello World program is a simple program that simply prints a `Hello World` message on the screen. It's generally used to introduce a new language to learners.

```
1 message <-"Hello World!"
2 print(message)
```

[1] "Hello World!"

Running Your First R Program

```
1 message <-"Hello World!"
2 print(message)
```

[1] “Hello World!”

- Here, we have created a simple variable called `message.` We have initialized this variable with a simple message string called `"Hello World!"`. On execution, this program prints the message stored inside the variable.
- Every output in R is preceded by a number (say n) in square brackets. This number means that the displayed value is the nth element printed.

R as a Calculator

```
1  2 + 2
```

[1] 4

```
1  2 * 4
```

[1] 8

```
1  2^3
```

[1] 8

Note: when you type your command, R inherently thinks you want to print the result.

R as a Calculator

- The R console is a full calculator
- Try to play around with it:
  - +, -, /, * are add, subtract, divide and multiply
  - ^ or ** is power
  - parentheses – (and) – work with order of operations
  - %% finds the remainder

R as a Calculator

```
2 + (2 * 3)^2
```

[1] 38

```
(1 + 3) / 2 + 45
```

[1] 47

```
6 / 2 * (1 + 2)
```

[1] 9

R as a Calculator

Try evaluating the following:

- `2 + 2 * 3 / 4 -3`
- `2 * 3 / 4 * 2`
- `2^4 - 1`

Variables (Identifiers) in R

- Variables are used to store data, whose value can be changed according to our need.
- A variable is a name given to a memory location, which is used to store values in a computer program.
- Variables in R programming can be used to store numbers (real and complex), words, matrices, and even tables.
- R is a dynamically programmed language which means that unlike other programming languages, we do not have to declare the data type of a variable before we can use it in our program.
- Unique name given to variable (function and objects as well) is identifier.

Rules for writing Identifiers in R

- Identifiers can be a combination of letters, digits, period (.) and underscore (_).
- It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
- It should not start with a number (e.g: 2x)
- It should not start with a dot followed by a number (e.g: .2x)
- It should not start with an underscore (e.g: _x)
- Reserved words in R cannot be used as identifiers(e.g: TRUE, FALSE)

Basically, there are 5 naming conventions

- alllowercase: e.g. `myname`
- period.separated: e.g. `new.name`
- underscore_separated: e.g. `my_name`
- lowerCamelCase: e.g. `myName`
- UpperCamelCase: e.g. `MyName`

Assigning Values to Objects

- You can create objects from within the R environment and from files on your computer
- R uses `<-` to assign values to an object name (you might also see `=` used, but this is not best practice)
- Object names are case-sensitive, i.e. X and x are different

```
1  x <- 2
2  x
```

[1] 2

```
1  x * 4
```

[1] 8

```
1  x + 2
```

[1] 4

Creating Variables

Using equal(=) operator

```
1  x = 10
```

Using leftward(<-) operator

```
1  y <- 15
```

Reserved Keaywords in R

- Don’t use any reserved keyword as variable name. List all of reserved words in R by using (?Reserved).

```
1  ?Reserved
```

Entering Input

At the R prompt/console we type expressions.

```
1  num <- 10
```

The `<-` symbol is the **assignment** operator. The grammar of the language determines whether an expression is complete or not.

Evaluation

When a complete expression is entered at the R console, it is evaluated and the result of evaluated expression is returned. The result may be auto-printed.

```
1 x <- 10
2 x
```

[1] 10

```
1 x <- 10
2 print(x)
```

[1] 10

```
1 x <- 10
2 cat(x)
```

10

R Comments

Comments are portions of a computer program that are used to describe a piece of code. For example,

```r
# declare variable
age = 24

# print variable
print(age)
```

```
[1] 24
```

Types of Comments in R

In general, all programming languages have the following types of comments:

- single-line comments
- multi-line comments

However, in R programming, there is no functionality for multi-line comments. Thus, you can only write single-line comments in R.

R Single-Line Comments

```
1  # this code prints Hello World
2  print("Hello World")
```

[1] “Hello World”

```
1  # check type of variables
2  age <- 30
3  class(age)
```

[1] “numeric”

R Multi-Line Comments

- As already mentioned, R does not have any syntax to create multi-line comments.
- However, you can use consecutive single-line comments to create a multi-line comment in R. For example,

```
# this is a print statement
# it prints Hello World

print("Hello World")
```

[1] “Hello World”

Purpose of Comments

As discussed above, R comments are used to just document pieces of code. This can help others to understand the working of our code.

Here are a few purposes of commenting on an R code:

- It increases readability of the program for users other than the developers.
- Comments in R provide metadata of the code or the overall project.
- Comments are generally used by programmers to ignore some pieces of code during testing.
- They are used to write a simple pseudo-code of the program.

How to Create Better Comments?

You should always keep in mind the following points while writing comments.

- Use comments only to describe what a particular block of code does, not how it does.
- Don’t overuse comments. Try to make your code self-explanatory.
- Try to create comments that are as precise as possible.
- Don’t use redundant comments.

Data Types in R

R Data Types

- A variable can store different types of values such as numbers, characters etc.
- These different types of data that we can use in our code are called data types. For example,

```
1  x <- 123L
```

- Here, `123L` is an integer data. So the data type of the variable `x` is `integer`.

R Data Types

We can verify this by printing the class of `x` using `class()` function.

```
x <- 123L
# print value of x
x
```

[1] 123

```
# print type of x
class(x)
```

[1] “integer”

Different Types of Data Types

Data Type	Example	Description
Logical	True, False	It is a special data type for data with only two possible values which can be construed as true/false.
Numeric	12,32,112,5432	Decimal value is called numeric in R, and it is the default computational data type.
Integer	3L, 66L, 2346L	Here, L tells R to store the value as an integer,

Different Types of Data Types

Data Type	Example	Description
Complex	Z=1+2i, t=7+3i	A complex value in R is defined as the pure imaginary value i.
Character	‘a’, ‘“good”’, “TRUE”, ’35.4’	In R programming, a character is used to represent string values. We convert objects into character values with the help of `as.character()` function.
Raw		A raw data type is used to holds raw bytes.

Logical Data Type

The `logical` data type in R is also known as `boolean` data type. It can only have two values: `TRUE` and `FALSE`. For example,

```
bool1 <- TRUE
# print bool1
bool1
```

[1] TRUE

```
# print type of bool1
class(bool1)
```

[1] “logical”

```
# print bool2
bool2 <- FALSE
bool2
```

[1] FALSE

```
# print type of bool2
class(bool2)
```

[1] “logical”

Logical Data Type

You can also define `logical` variables with a single letter - `T` for `TRUE` or `F` for `FALSE.` For example,

```
1 is_weekend <- F
2 class(is_weekend)  # "logical"
```

[1] “logical”

```
1 is_weekday <- T
2 class(is_weekday)  # "logical"
```

[1] “logical”

Numeric Data Type

In R, the `numeric` data type represents all real numbers with or without decimal values. For example,

```
# floating point values
weight <- 63.5
weight
```

[1] 63.5

```
# check variable types
class(weight)
```

[1] “numeric”

```
# real numbers
height <- 182
height
```

[1] 182

```
# check variable types
class(height)
```

[1] “numeric”

Integer Data Type

The integer data type specifies real values without decimal points. We use the suffix L to specify integer data. For example,

```
integer_variable <- 186L
integer_variable
```

[1] 186

```
# check variable types
class(integer_variable)
```

[1] “integer”

Complex Data Type

The `complex` data type is used to specify purely imaginary values in R. We use the suffix `i` to specify the imaginary part. For example,

```
# 2i represents imaginary part
complex_value <- 3 + 2i

# print class of complex_value
class(complex_value)
```

[1] “complex”

Character Data Type

- The `character` data type is used to specify character or string values in a variable.
- In programming, a string is a set of characters. For example, `'A'` is a single character and `"Apple"` is a string.
- You can use single quotes `''` or double quotes `""` to represent strings. In general, we use:
  - `''` for character variables
  - `""` for string variables

Character Data Type

For example,

```
1 # create a string variable
2 fruit <- "Apple"
3 class(fruit)
```

[1] “character”

```
1 # create a character variable
2 my_char <- 'A'
3 class(my_char)
```

[1] “character”

Operators in R

Operators in R

- In R, operators are symbols or characters that perform specific operations on variables, values, or expressions.
- R provides various types of operators, including arithmetic operators, assignment operators, comparison operators, logical operators, and more.
- Operators in R can mainly be classified into the following categories.
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators

Arithmetic Operators

Operator	Operation	Example
+	Addition	5 + 2 = 7
-	Subtraction	4 - 2 = 2
*	Multiplication	2 * 3 = 6
/	Division	4 / 2 = 2
%%	Modulo	5 %% 2 = 1
^	Power	4 ^ 2 = 16

Example: Arithmetic Operators

```
1  x <- 10
2  y <- 2
3
4  # Addition
5  x+y
```

[1] 12

```
1  # Subtraction
2  2-5
```

[1] -3

```
1  # Multiplication
2  2 * 5
```

[1] 10

```
1  # Division
2  2 / 5
```

[1] 0.4

Example: Arithmetic Operators

```
1  x <- 10
2  y <- 2
3
4  # Exponent
5  2 ^ 5
```

[1] 32

```
1  # Modulus(Remainder from division)
2  2 %% 5
```

[1] 2

Relational Operators

Operator	Operation	Example
>	Greater that	`5 > 6` returns `FALSE`
<	Less than	`5 < 6` returns `TRUE`
==	Equals to	`10 == 10` returns `TRUE`
!=	Not equal to	`10 != 10` returns `FALSE`
>=	Greater than or equal to	`5 >= 6` returns `FALSE`
<=	Less than or equal to	`6 <= 6` returns `TRUE`

Example: Relational Operators

The `output` of a comparison is a `boolean value`. For example, to check if two numbers are equal, you can use the `==` operator.

```
1 x <- 10
2 y <- 23
3
4 # compare x and y
5 x == y # FALSE
```

[1] FALSE

Similarly, to check if x is less than y, you can use the `<` operator.

```
1 x <- 10
2 y <- 23
3
4 # compare x and y
5 x < y # TRUE
```

[1] TRUE

Logical Operators

Logical operators are used to compare the output of two comparisons. There are `three` types of logical operators in R. They are:

- AND operator (&)
- OR operator (|)
- NOT operator (!)

AND Operator (&)

- The AND operator `&` takes as input two logical values and returns the output as another logical value.
- The output of the operator is `TRUE` only when `both` the input logical values are either `TRUE` or evaluated to `TRUE`.
- Let a and b represent two operands. `0` represents FALSE and 1 represents TRUE. Then,

a	b	a & b
1	1	1
1	0	0
0	1	0
0	0	0

Example: AND Operator (&)

```
# print & of TRUE and FALSE combinations
TRUE & TRUE
```

[1] TRUE

```
TRUE & FALSE
```

[1] FALSE

```
FALSE & TRUE
```

[1] FALSE

```
FALSE & FALSE
```

[1] FALSE

```
# print & of TRUE and FALSE combinations
x <- 10
y <- 23
z <- 12

# compare
x<y & y>z
```

[1] TRUE

OR Operator (|)

The OR operator | returns TRUE if all or any one of the logical inputs is TRUE or evaluates to TRUE. If all of them are FALSE, then it returns FALSE. Consider the table below.

a	b	a | b
1	1	1
1	0	1
0	1	1
0	0	0

Example: OR Operator (|)

```
# print | of TRUE and FALSE combinations
TRUE | TRUE
```

[1] TRUE

```
TRUE | FALSE
```

[1] TRUE

```
FALSE | TRUE
```

[1] TRUE

```
FALSE | FALSE
```

[1] FALSE

```
# print | of TRUE and FALSE combinations
w <- 54
x <- 12
y <- 25
z <- 1

w>x | x>y | z>w
```

[1] TRUE

NOT (!) Operator

The NOT operator ! is used to negate the logical values it is used on. If the input value is TRUE, it will turn to FALSE and vice-versa.

a	!a
1	0
0	1

Example: NOT (!) Operator

```
1  # print ! of TRUE and FALSE
2  !TRUE
```

[1] FALSE

```
1  !FALSE
```

[1] TRUE

Here, the output is the negation of the input.

- We can use the ! operator with comparisons.
- For example, !(x > 12) is the same as x <= 12. This means that x is not greater than 12. Which means that x can be less than or equal to 12.

What is Data Structure?

- Data structures are ways of organizing and storing data in a computer system.
- They define the format, organization, and relationship between data elements.
- Data structures facilitate efficient operations such as insertion, deletion, searching, and sorting of data.
- They provide a foundation for building algorithms and designing efficient programs.

Data Structures in R

- Vectors
- Matrix
- Lists
- Data Frame
- Factor

Types of Data Structures in R

- One-Dimensional Data Structures
  - One-dimensional data structures in R are used to store and manipulate data along a single dimension.
  - The main one-dimensional data structure in R is the `vector`.
- Two-Dimensional Data Structures
  - Two-dimensional data structures in R are used to store and manipulate data in a tabular format with rows and columns.
  - The main two-dimensional data structures in R are `matrices` and `data frames`.

Strings

- A string is a sequence of characters. For example, `"Programming"` is a string that includes characters: P, r, o, g, r, a, m, m, i, n, g.
- In R, we represent strings using quotation marks (double quotes, " " or single quotes, ' '). For example,

```
1 # string value using single quotes
2 'Hello'
```

[1] "Hello"

```
1 # string value using double quotes
2 "Hello"
```

[1] "Hello"

String Operations in R

R provides us various built-in functions that allow us to perform different operations on strings. Here, we will look at some of the commonly used string functions.

- Find the length of a string
- Join two strings
- Compare two strings
- Change the string case

Find Length of String

We use the `nchar()` method to find the length of a string. For example,

```
message1 <- "CHIRAL Bangladesh"
# use of nchar() to find length of message1
nchar(message1)
```

[1] 17

Here, `nchar()` returns the number of characters present inside the string.

Join Strings Together

In R, we can use the `paste()` function to join two or more strings together. For example,

```
message1 <- "CHIRAL"
message2 <- "Bangladesh"

# use paste() to join two strings
paste(message1, message2)
```

[1] “CHIRAL Bangladesh”

Here, we have used the `paste()` function to join two strings: `message1` and `message2`.

Compare Two Strings in R Programming

We use the `==` operator to compare two strings. If two strings are equal, the operator returns `TRUE`. Otherwise, it returns `FALSE`. For example,

```
message1 <- "Hello, World!"
message2 <- "Hi, Bangladesh!"
message3 <- "Hello, CHIRAL!"
# `message1 == message2` - returns FALSE because two strings are not equal
print(message1 == message2)
```

[1] FALSE

```
#`message1 == message3` - returns TRUE because both strings are equal
print(message1 == message3)
```

[1] FALSE

Change Case of R String

In R, we can change the case of a string using

- `toupper()` - convert string to uppercase
- `tolower()` - convert string to lowercase

```
message <- "R Programming"

# change string to uppercase
message_upper <- toupper(message)
message_upper
```

[1] “R PROGRAMMING”

```
# change string to lowercase
message_lower <- tolower(message)
message_lower
```

[1] “r programming”

Vector

- Vector is a basic data structure in R.
- It contains element of the same type.
- The data types can be logical, integer, double, character, and complex.
- A vector's type can be checked with the `typeof()` function.

Creating Vectors - Using the c() Function

The `c()` function is used to concatenate or combine elements into a vector.

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

# Character vector
character_vector <- c("apple", "banana", "orange")

# Logical vector
logical_vector <- c(TRUE, FALSE, TRUE)
```

Creating Vectors - Using the : Operator

The : operator generates a sequence of numbers from the starting value to the ending value.

```
1  # Numeric sequence vector
2  numeric_sequence <- 1:10
```

Creating Vectors - Using Sequence Generation Functions

R provides functions like `seq()`, `rep()`, and `seq_len()` to generate sequences of numbers.

```r
# Numeric sequence vector using seq()
numeric_sequence <- seq(from = 1, to = 10, by = 2)

# Repeated values vector using rep()
repeated_values <- rep(0, times = 5)

# Index sequence vector using seq_len()
index_sequence <- seq_len(10)
```

Creating Vectors - Using Vectorized Operations

Vectors can be created by performing operations on existing vectors or values.

```
# Vector created using vectorized operation
new_vector <- numeric_vector * 2
```

Creating Vectors - Mixing Objects

```r
# Character
x <- c(1.7, "a")
# Numeric
y <- c(TRUE, 2)
# Character
z <- c("a", TRUE)
```

Matrix

- Matrix is a two dimensional data structure in R programming.
- Matrix is similar to vector but additionally contains the dimension attributes.
- All attributes of an object can be checked by `attributes()` function.
- Dimension can be checked by directly with the `dim()`function. We can check if a variable is a matrix or not with the `class()` function.

Creatinig Matrix

- Matrix can be created using the `matrix()` function. Here's the general syntax:

```
matrix(data, nrow, ncol, byrow, dimnames)
```

- `data:` The data elements used to fill the matrix. It can be a vector or a combination of vectors.
- `nrow:` The number of rows in the matrix.
- `ncol:` The number of columns in the matrix.
- `byrow:` A logical value specifying whether the matrix should be filled by row (TRUE) or by column (FALSE) (default).
- `dimnames:` Optional names for the rows and columns of the matrix.

Creatinig Matrix

```r
# Create a matrix using matrix function
mat1 <- matrix(1:9, nrow = 3, ncol = 3)

# Create a matrix using matrix function: only one dimension
mat2 <- matrix(1:9, nrow = 3)

# Create a matrix using matrix function: filling by row-wise
mat3 <- matrix(1:9, nrow = 3, byrow = TRUE)

# Create a matrix using matrix function: dimension names
mat4 <- matrix(1:9, nrow = 3, dimnames = list(c("X", "Y", "Z"),
                                              c("A","B","C")))
```

Matrix Properties

```
# Create a matrix using matrix function
mat <- matrix(1:9, nrow = 3, dimnames = list(c("X", "Y", "Z"),
                                             c("A","B","C")))
# Column Names
colnames(mat)
```

[1] “A” “B” “C”

```
# Row Names
rownames(mat)
```

[1] “X” “Y” “Z”

```
# Dimension
dim(mat)
```

[1] 3 3

List

- List is a data structure having components of mixed data types.
- A vector having all elements of the same type is called atomic vector but a vector having elements of different type is called list.
- We can check if it's a list with `typeof()` function and find its length using `length()` function.

Creating List

List can be created using the list() function. Here's the general syntax:

```
1 list(..., recursive = FALSE)
```

- `...`: The elements to be included in the list, separated by commas.
- `recursive`: A logical value specifying whether the list should allow nested lists (TRUE) or not (FALSE) (default).

Creating List

```r
# Create a list
L = list(1, "a", TRUE, 1+3i)

# Create a list with different elements
my_list <- list(
  name = "John Doe",  # Character value
  age = 30,  # Numeric value
  is_student = TRUE,  # Logical value
  scores = c(90, 85, 92),  # Numeric vector
  matrix_data = matrix(1:6, nrow = 2),  # Matrix
  sub_list = list("a", "b", "c")  # Nested list
)
```

Factors

- In R, factors are used to represent categorical or discrete data with predefined levels or categories.
- Factors are useful when working with data that has distinct categories or when performing statistical analysis.
- Factors are used to represent categorical data and can be ordered and unordered.

Creating Factors

Factors are created using the factor() function in R. Here's the general syntax:

```
1  factor(x, levels, labels, ordered = FALSE)
```

- `x`: A vector or column of data that represents the categorical variable.
- `levels`: An optional argument specifying the unique levels or categories of the factor. If not provided, the distinct values in x are used as levels.
- `labels`: An optional argument specifying the labels for the levels. If not provided, the levels themselves are used as labels.
- `ordered`: A logical value indicating whether the factor should be treated as ordered (TRUE) or unordered (FALSE) (default).

Creating Factors

```
# Create a factor using factor() function
f <- factor(c("yes", "no", "yes", "no"))

# Check levels
levels(f)
```

[1] “no” “yes”

Data Frame

- In R, a data frame is a two-dimensional tabular data structure similar to a table in a relational database.
- It consists of rows and columns, where each column can have a different data type.
- Data frames are commonly used for storing and manipulating structured data, and they provide a convenient way to work with datasets.
- Data frames can be created using the `data.frame()` function or by importing data from external sources.

Create Data Frame

Data frames can be created using the data.frame() function or by importing data from external sources. Here's an example of creating a data frame in R:

```r
# Create a data frame
df <- data.frame(
  name = c("John", "Alice", "Bob"),
  age = c(25, 30, 35),
  city = c("New York", "London", "Paris"),
  stringsAsFactors = FALSE
)
```

Data Conversion Functions in R

- Conversion functions in R help transform data between different types and formats.
- `as.character()`, `as.numeric()`, `as.integer()`, `as.logical()`, and `as.factor()` are commonly used conversion functions.
- These functions are essential for data preprocessing, ensuring data compatibility, and performing operations on different data types.

as.character()

- as.character() function converts an object to a character string representation.
- Syntax: as.character(x)
- x: The object to be converted.

```r
# Convert numeric values to character strings
numbers <- c(1, 2, 3)
character_numbers <- as.character(numbers)
```

as.numeric()

- as.numeric() function converts an object to numeric (floating-point) values.
- Syntax: as.numeric(x)
- x: The object to be converted.

```r
# Convert character strings to numeric values
character_numbers <- c("1", "2", "3")
numeric_numbers <- as.numeric(character_numbers)
```

as.integer()

- as.integer() function converts an object to integer values.
- Syntax: as.integer(x)
- x: The object to be converted.

```r
# Convert numeric values to integer values
numbers <- c(1.5, 2.7, 3.9)
integer_numbers <- as.integer(numbers)
```

as.logical()

- as.logical() function converts an object to logical (boolean) values.
- Syntax: as.logical(x)
- x: The object to be converted.

```r
# Convert numeric values to logical values
numbers <- c(0, 1, 2)
logical_values <- as.logical(numbers)
```

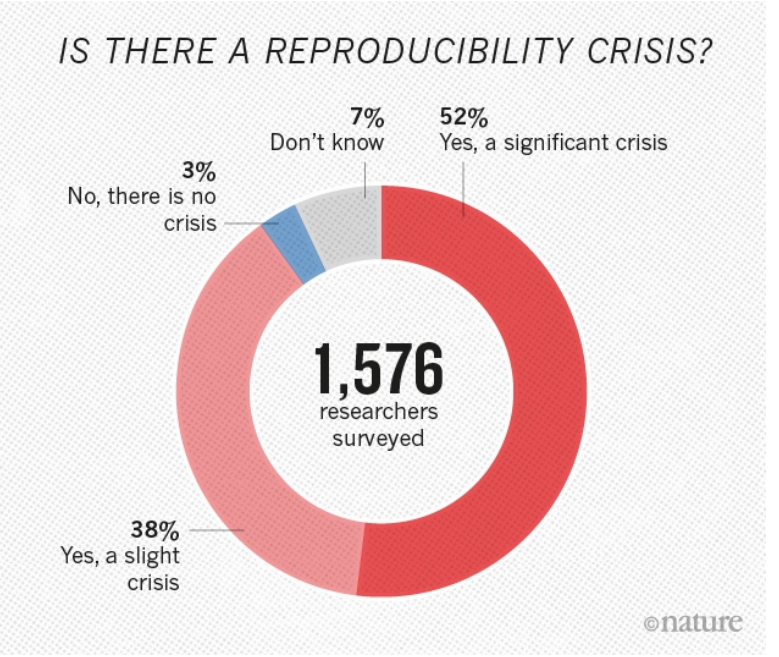
as.factor()

- as.factor() function converts an object to a factor, which represents categorical data.
- Syntax: as.factor(x)
- x: The object to be converted.

gtsummary - Motivation

Reproducibility Crisis

- Quality of medical research is often low
- **Low quality code** in medical research part of the problem
- Low quality code is more **likely to contain errors**
- Reproducibility is often **cumbersome** and **time-consuming**



https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970

{gtsummary} overview

- Create **tabular summaries** with sensible defaults but highly customizable
- Types of summaries:
  - “Table 1”-types
  - Cross-tabulation
  - Regression models
  - Survival data
  - Survey data
  - Custom tables
- Report statistics from {gtsummary} tables **inline** in R Markdown
- Stack and/or merge any table type
- Use **themes** to standardize across tables
- Choose from different **print engines**

Example Dataset

- The `trial` data set is included with {gtsummary}
- Simulated data set of baseline characteristics for 200 patients who receive Drug A or Drug B
- Variables were assigned labels using the `labelled` package

```
1 library(gtsummary)
2 library(tidyverse)
3 head(trial) |> gt::gt()
```

trt	age	marker	stage	grade	response	death	ttdeath
Drug A	23	0.160	T1	II	0	0	24.00
Drug B	9	1.107	T2	I	1	0	24.00
Drug A	31	0.277	T1	II	0	0	24.00
Drug A	NA	2.067	T3	III	1	1	17.64
Drug A	51	2.767	T4	III	1	1	16.43
Drug B	39	0.613	T4	I	0	1	15.64

Example Dataset

This presentation will use a subset of the variables.

```
1  sm_trial <-
2    trial |>
3    select(trt, age, grade, response)
```

Variable	Label
trt	Chemotherapy Treatment
age	Age
grade	Grade
response	Tumor Response

tbl_summary()

Basic tbl_summary()

```
sm_trial |>
  select(-trt) |>
  tbl_summary()
```

Characteristic	N = 200[1]
Age	47 (38, 57)
Unknown	11
Grade	
I	68 (34%)
II	68 (34%)
III	64 (32%)
Tumor Response	61 (32%)
Unknown	7

[1] Median (IQR); n (%)

- Four types of summaries: `continuous`, `continuous2`, `categorical`, and `dichotomous`
- Statistics are `median (IQR)` for continuous, `n (%)` for categorical/dichotomous
- Variables coded `0/1`, `TRUE/FALSE`, `Yes/No` treated as dichotomous
- Lists `NA` values under “Unknown”
- Label attributes are printed automatically

Customize tbl_summary() output

```
1 tbl_summary(
2   sm_trial,
3   by = trt,
4 )
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]
Age	46 (37, 59)	48 (39, 56)
Unknown	7	4
Grade		
I	35 (36%)	33 (32%)
II	32 (33%)	36 (35%)
III	31 (32%)	33 (32%)
Tumor Response	28 (29%)	33 (34%)
Unknown	3	4

[1] Median (IQR); n (%)

- by: specify a column variable for cross-tabulation

Customize tbl_summary() output

```
tbl_summary(
  sm_trial,
  by = trt,
  type = age ~ "continuous2",
)
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]
Age		
Median (IQR)	46 (37, 59)	48 (39, 56)
Unknown	7	4
Grade		
I	35 (36%)	33 (32%)
II	32 (33%)	36 (35%)
III	31 (32%)	33 (32%)
Tumor Response	28 (29%)	33 (34%)
Unknown	3	4

[1] n (%)

- by: specify a column variable for cross-tabulation
- type: specify the summary type

Customize tbl_summary() output

```
tbl_summary(
  sm_trial,
  by = trt,
  type = age ~ "continuous2",
  statistic =
    list(
      age ~ c("{mean} ({sd})",
              "{min}, {max}"),
      response ~ "{n} / {N} ({p}%)"
    ),
)
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]
Age		
Mean (SD)	47 (15)	47 (14)
Range	6, 78	9, 83
Unknown	7	4
Grade		
I	35 (36%)	33 (32%)
II	32 (33%)	36 (35%)
III	31 (32%)	33 (32%)
Tumor Response	28 / 95 (29%)	33 / 98 (34%)
Unknown	3	4

[1] n (%); n / N (%)

- `by`: specify a column variable for cross-tabulation
- `type`: specify the summary type
- `statistic`: customize the reported statistics

Customize tbl_summary() output

```
tbl_summary(
  sm_trial,
  by = trt,
  type = age ~ "continuous2",
  statistic =
    list(
      age ~ c("{mean} ({sd})",
              "{min}, {max}"),
      response ~ "{n} / {N} ({p}%)"
    ),
  label =
    grade ~ "Pathologic tumor grade",
)
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]
Age		
Mean (SD)	47 (15)	47 (14)
Range	6, 78	9, 83
Unknown	7	4
Pathologic tumor grade		
I	35 (36%)	33 (32%)
II	32 (33%)	36 (35%)
III	31 (32%)	33 (32%)
Tumor Response	28 / 95 (29%)	33 / 98 (34%)
Unknown	3	4

[1] n (%); n / N (%)

- `by`: specify a column variable for cross-tabulation
- `type`: specify the summary type
- `statistic`: customize the reported statistics
- `label`: change or customize variable labels

Customize tbl_summary() output

```
tbl_summary(
  sm_trial,
  by = trt,
  type = age ~ "continuous2",
  statistic =
    list(
      age ~ c("{mean} ({sd})",
              "{min}, {max}"),
      response ~ "{n} / {N} ({p}%)"
    ),
  label =
    grade ~ "Pathologic tumor grade",
  digits = age ~ 1
)
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]
Age		
Mean (SD)	47.0 (14.7)	47.4 (14.0)
Range	6.0, 78.0	9.0, 83.0
Unknown	7	4
Pathologic tumor grade		
I	35 (36%)	33 (32%)
II	32 (33%)	36 (35%)
III	31 (32%)	33 (32%)
Tumor Response	28 / 95 (29%)	33 / 98 (34%)
Unknown	3	4

[1] n (%); n / N (%)

- `by`: specify a column variable for cross-tabulation
- `type`: specify the summary type
- `statistic`: customize the reported statistics
- `label`: change or customize variable labels
- `digits`: specify the number of decimal places for rounding

{gtsummary} + formulas

select variables

give instructions

```
sm_trial %>%
tbl_summary(
  label      =          age          ~ "Patient Age",
  type       =    c(age, marker)     ~ "continuous",
  digits     =  starts_with("age")   ~      0,
  statistic  =   all_continuous()    ~ "{mean} ({sd})"
)
```

Use `lists` to pass ≥2 sets of instruction:

```
label = list(age ~ "Patient Age", marker ~ "Marker Level")
```

Named list are OK too! `label = list(age = "Patient Age")`

Add-on functions in {gtsummary}

`tbl_summary()` objects can also be updated using related functions.

- `add_*()` add **additional column** of statistics or information, e.g. p-values, q-values, overall statistics, treatment differences, N obs., and more
- `modify_*()` **modify** table headers, spanning headers, footnotes, and more
- `bold_*()/italicize_*()` **style** labels, variable levels, significant p-values

Update tbl_summary() with add_*()

```
sm_trial |>
  tbl_summary(
    by = trt
  ) |>
  add_p() |>
  add_q(method = "fdr")
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]	p-value[2]	q-value[3]
Age	46 (37, 59)	48 (39, 56)	0.7	0.9
Unknown	7	4		
Grade			0.9	0.9
I	35 (36%)	33 (32%)		
II	32 (33%)	36 (35%)		
III	31 (32%)	33 (32%)		
Tumor Response	28 (29%)	33 (34%)	0.5	0.9
Unknown	3	4		

[1] Median (IQR); n (%)

[2] Wilcoxon rank sum test; Pearson's Chi-squared test

[3] False discovery rate correction for multiple testing

- `add_p()`: adds a column of p-values
- `add_q()`: adds a column of p-values adjusted for multiple comparisons through a call to `p.adjust()`

Update tbl_summary() with add_*()

```
sm_trial |>
  tbl_summary(
    by = trt,
    missing = "no"
  ) |>
  add_overall()
```

Characteristic	**Overall**, N = 200[1]	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]
Age	47 (38, 57)	46 (37, 59)	48 (39, 56)
Grade			
I	68 (34%)	35 (36%)	33 (32%)
II	68 (34%)	32 (33%)	36 (35%)
III	64 (32%)	31 (32%)	33 (32%)
Tumor Response	61 (32%)	28 (29%)	33 (34%)

[1] Median (IQR); n (%)

- `add_overall()`: adds a column of overall statistics

Update tbl_summary() with add_*()

```
sm_trial |>
  tbl_summary(
    by = trt,
    missing = "no"
  ) |>
  add_overall() |>
  add_n()
```

Characteristic	N	**Overall**, N = 200[1]	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]
Age	189	47 (38, 57)	46 (37, 59)	48 (39, 56)
Grade	200			
I		68 (34%)	35 (36%)	33 (32%)
II		68 (34%)	32 (33%)	36 (35%)
III		64 (32%)	31 (32%)	33 (32%)
Tumor Response	193	61 (32%)	28 (29%)	33 (34%)

[1] Median (IQR); n (%)

- `add_overall()`: adds a column of overall statistics
- `add_n()`: adds a column with the sample size

Update tbl_summary() with add_*()

```
sm_trial |>
  tbl_summary(
    by = trt,
    missing = "no"
  ) |>
  add_overall() |>
  add_n() |>
  add_stat_label(
    label = all_categorical() ~ "No. (%)"
  )
```

Characteristic	**N**	**Overall**, N = 200	**Drug A**, N = 98	**Drug B**, N = 102
Age, Median (IQR)	189	47 (38, 57)	46 (37, 59)	48 (39, 56)
Grade, No. (%)	200			
I		68 (34%)	35 (36%)	33 (32%)
II		68 (34%)	32 (33%)	36 (35%)
III		64 (32%)	31 (32%)	33 (32%)
Tumor Response, No. (%)	193	61 (32%)	28 (29%)	33 (34%)

- `add_overall()`: adds a column of overall statistics
- `add_n()`: adds a column with the sample size
- `add_stat_label()`: adds a description of the reported statistic

Update with bold_*()/italicize_*()

```
sm_trial |>
  tbl_summary(
    by = trt
  ) |>
  add_p() |>
  bold_labels() |>
  italicize_levels() |>
  bold_p(t = 0.8)
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]	**p-value**[2]
Age	46 (37, 59)	48 (39, 56)	**0.7**
Unknown	7	4	
Grade			0.9
I	35 (36%)	33 (32%)	
II	32 (33%)	36 (35%)	
III	31 (32%)	33 (32%)	
Tumor Response	28 (29%)	33 (34%)	**0.5**
Unknown	3	4	

[1] Median (IQR); n (%)

[2] Wilcoxon rank sum test; Pearson's Chi-squared test

- `bold_labels()`: bold the variable labels
- `italicize_levels()`: italicize the variable levels
- `bold_p()`: bold p-values according a specified threshold

Update tbl_summary() with modify_*()

```
tbl <-
  sm_trial |>
  tbl_summary(by = trt,
              missing = "no") |>
  modify_header(
      stat_1 ~ "**Group A**",
      stat_2 ~ "**Group B**"
  ) |>
  modify_spanning_header(
    all_stat_cols() ~ "**Drug**") |>
  modify_footnote(
    all_stat_cols() ~
      paste("median (IQR) for continuous;",
            "n (%) for categorical")
  )
tbl
```

Characteristic	Drug	
	Group A[1]	**Group B**[1]
Age	46 (37, 59)	48 (39, 56)
Grade		
I	35 (36%)	33 (32%)
II	32 (33%)	36 (35%)
III	31 (32%)	33 (32%)
Tumor Response	28 (29%)	33 (34%)

[1] median (IQR) for continuous; n (%) for categorical

- Use `show_header_names()` to see the internal header names available for use in `modify_header()`

Column names

```
1  show_header_names(tbl)
```

Column Name	Column Header
label	**Characteristic**
stat_1	**Group A**
stat_2	**Group B**

`all_stat_cols()` selects columns `"stat_1"` and `"stat_2"`

Update tbl_summary() with add_*()

```
trial |>
  select(trt, marker, response) |>
  tbl_summary(
    by = trt,
    statistic = list(marker ~ "{mean} ({sd})",
                     response ~ "{p}%"),
    missing = "no"
  ) |>
  add_difference()
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]	**Difference**[2]	**95% CI**[2,3]	**p-value**[2]
Marker Level (ng/mL)	1.02 (0.89)	0.82 (0.83)	0.20	-0.05, 0.44	0.12
Tumor Response	29%	34%	-4.2%	-18%, 9.9%	0.6

[1] Mean (SD); %

[2] Welch Two Sample t-test; Two sample test for equality of proportions

[3] CI = Confidence Interval

- `add_difference()`: mean and rate differences between two groups. Can also be adjusted differences

Update tbl_summary() with add_*()

```
sm_trial |>
  tbl_summary(
    by = trt,
    missing = "no"
  ) |>
  add_stat(...)
```

- Customize statistics presented with `add_stat()`
- Added statistics can be placed on the label or the level rows
- Added statistics may be a single column or multiple

Add-on functions in {gtsummary}

And many more!

See the documentation at
http://www.danieldsjoberg.com/gtsummary/reference/index.html

And a detailed `tbl_summary()` vignette at
http://www.danieldsjoberg.com/gtsummary/articles/tbl_summary.html

Cross-tabulation with tbl_cross()

`tbl_cross()` is a wrapper for `tbl_summary()` for **n x m** tables

```
sm_trial |>
  tbl_cross(
    row = trt,
    col = grade,
    percent = "row",
    margin = "row"
  ) |>
  add_p(source_note = TRUE) |>
  bold_labels()
```

	Grade		
	I	II	III
Chemotherapy Treatment			
Drug A	35 (36%)	32 (33%)	31 (32%)
Drug B	33 (32%)	36 (35%)	33 (32%)
Total	68 (34%)	68 (34%)	64 (32%)

Pearson's Chi-squared test, p=0.9

Continuous Summaries with tbl_continuous()

`tbl_continuous()` summarizes a continuous variable by 1, 2, or more categorical variables

```
sm_trial |>
  tbl_continuous(
    variable = age,
    by = trt,
    include = grade
  )
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]
Grade		
I	46 (36, 60)	48 (42, 55)
II	44 (31, 54)	50 (43, 57)
III	52 (42, 60)	45 (36, 52)

[1] Age: Median (IQR)

Survey data with tbl_svysummary()

```r
survey::svydesign(
  ids = ~1,
  data = as.data.frame(Titanic),
  weights = ~Freq
) |>
  tbl_svysummary(
    by = Survived,
    include = c(Class, Sex)
  ) |>
  add_p() |>
  modify_spanning_header(
    all_stat_cols() ~ "**Survived**")
```

Characteristic	Survived		p-value[2]
	No, N = 1,490[1]	**Yes**, N = 711[1]	
Class			0.7
1st	122 (8.2%)	203 (29%)	
2nd	167 (11%)	118 (17%)	
3rd	528 (35%)	178 (25%)	
Crew	673 (45%)	212 (30%)	
Sex			0.048
Male	1,364 (92%)	367 (52%)	
Female	126 (8.5%)	344 (48%)	

[1] n (%)

[2] chi-squared test with Rao & Scott's second-order correction

Survival outcomes with tbl_survfit()

```
library(survival)
fit <- survfit(Surv(ttdeath, death) ~ trt, trial)

tbl_survfit(
  fit,
  times = c(12, 24),
  label_header = "**{time} Month**"
) |>
  add_p()
```

Characteristic	12 Month	24 Month	p-value[1]
Chemotherapy Treatment			0.2
Drug A	91% (85%, 97%)	47% (38%, 58%)	
Drug B	86% (80%, 93%)	41% (33%, 52%)	

[1] Log-rank test

tbl_regression()

Traditional model summary()

```r
m1 <-
  glm(
    response ~ age + stage,
    data = trial,
    family = binomial(link = "logit")
  )
```

```
Call:
glm(formula = response ~ age + stage, family = binomial(link = "logit"),
    data = trial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.2302  -0.8911  -0.7720   1.3563   1.9994

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.48622    0.62023  -2.396   0.0166 *
age          0.01939    0.01147   1.691   0.0909 .
stageT2     -0.54143    0.44000  -1.231   0.2185
stageT3     -0.05953    0.45042  -0.132   0.8948
stageT4     -0.23109    0.44823  -0.516   0.6062
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 228.58  on 182  degrees of freedom
Residual deviance: 223.93  on 178  degrees of freedom
  (17 observations deleted due to missingness)
AIC: 233.93

Number of Fisher Scoring iterations: 4
```

Looks **messy** and it’s not easy to digest

Basic tbl_regression()

```
1  tbl_regression(m1)
```

Characteristic	log(OR)[1]	95% CI[1]	p-value
Age	0.02	0.00, 0.04	0.091
T Stage			
T1	—	—	
T2	-0.54	-1.4, 0.31	0.2
T3	-0.06	-0.95, 0.82	0.9
T4	-0.23	-1.1, 0.64	0.6

[1] OR = Odds Ratio, CI = Confidence Interval

- Displays **p-values** for covariates
- Shows **reference levels** for categorical variables
- **Model type recognized** as logistic regression with odds ratio appearing in header

Customize tbl_regression() output

```
tbl_regression(
  m1,
  exponentiate = TRUE
) |>
  add_global_p() |>
  add_glance_table(
    include = c(nobs,
                logLik,
                AIC,
                BIC)
  )
```

Characteristic	OR[1]	95% CI[1]	p-value
Age	1.02	1.00, 1.04	0.087
T Stage			0.6
T1	—	—	
T2	0.58	0.24, 1.37	
T3	0.94	0.39, 2.28	
T4	0.79	0.33, 1.90	
No. Obs.	183		
Log-likelihood	-112		
AIC	234		
BIC	250		

[1] OR = Odds Ratio, CI = Confidence Interval

- Display odds ratio estimates and confidence intervals
- Add global p-values
- Add various model statistics

Supported models in tbl_regression()

- `biglm::bigglm()`
- `biglmm::bigglm()`
- `brms::brm()`
- `cmprsk::crr()`
- `fixest::feglm()`
- `fixest::femlm()`
- `fixest::feNmlm()`
- `fixest::feols()`
- `gam::gam()`
- `geepack::geeglm()`
- `glmmTMB::glmmTMB()`
- `lavaan::lavaan()`
- `lfe::felm()`
- `lme4::glmer.nb()`
- `lme4::glmer()`
- `lme4::lmer()`
- `logitr::logitr()`
- `MASS::glm.nb()`
- `MASS::polr()`
- `mgcv::gam()`
- `mice::mira`
- `multgee::nomLORgee()`
- `multgee::ordLORgee()`
- `nnet::multinom()`
- `ordinal::clm()`
- `ordinal::clmm()`
- `parsnip::model_fit`
- `plm::plm()`
- `rstanarm::stan_glm()`
- `stats::aov()`
- `stats::glm()`
- `stats::lm()`
- `stats::nls()`
- `survey::svycoxph()`
- `survey::svyglm()`
- `survey::svyolr()`
- `survival::clogit()`
- `survival::coxph()`
- `survival::survreg()`
- `tidycmprsk::crr()`
- `VGAM::vglm()`

Custom tidiers can be written and passed to `tbl_regression()` using the `tidy_fun=` argument.

Univariate models with tbl_uvregression()

```r
tbl_uvreg <-
  sm_trial |>
  tbl_uvregression(
    method = glm,
    y = response,
    method.args =
      list(family = binomial),
    exponentiate = TRUE
  )
tbl_uvreg
```

Characteristic	N	OR[1]	95% CI[1]	p-value
Chemotherapy Treatment	193			
Drug A		—	—	
Drug B		1.21	0.66, 2.24	0.5
Age	183	1.02	1.00, 1.04	0.10
Grade	193			
I		—	—	
II		0.95	0.45, 2.00	0.9
III		1.10	0.52, 2.29	0.8

[1] OR = Odds Ratio, CI = Confidence Interval

- Specify model `method`, `method.args`, and the `response` variable
- Arguments and helper functions like `exponentiate`, `bold_*()`, `add_global_p()` can also be used with `tbl_uvregression()`

tbl_merge()/tbl_stack()

tbl_merge() for side-by-side tables

A **univariable** table:

```
tbl_uvsurv <-
  trial |>
  select(age, grade, death, ttdeath) |>
  tbl_uvregression(
    method = coxph,
    y = Surv(ttdeath, death),
    exponentiate = TRUE
  ) |>
  add_global_p()
tbl_uvsurv
```

Characteristic	N	HR[1]	95% CI[1]	p-value
Age	189	1.01	0.99, 1.02	0.3
Grade	200			0.075
I		—	—	
II		1.28	0.80, 2.05	
III		1.69	1.07, 2.66	

[1] HR = Hazard Ratio, CI = Confidence Interval

A **multivariable** table:

```
tbl_mvsurv <-
  coxph(
    Surv(ttdeath, death) ~ age + grade,
    data = trial
  ) |>
  tbl_regression(
    exponentiate = TRUE
  ) |>
  add_global_p()
tbl_mvsurv
```

Characteristic	HR[1]	95% CI[1]	p-value
Age	1.01	0.99, 1.02	0.3
Grade			0.041
I	—	—	
II	1.20	0.73, 1.97	
III	1.80	1.13, 2.87	

[1] HR = Hazard Ratio, CI = Confidence Interval

tbl_merge() for side-by-side tables

```
tbl_merge(
  list(tbl_uvsurv, tbl_mvsurv),
  tab_spanner = c("**Univariable**", "**Multivariable**")
)
```

	Univariable				Multivariable		
Characteristic	**N**	**HR**[1]	**95% CI**[1]	**p-value**	**HR**[1]	**95% CI**[1]	**p-value**
Age	189	1.01	0.99, 1.02	0.3	1.01	0.99, 1.02	0.3
Grade	200			0.075			0.041
I		—	—		—	—	
II		1.28	0.80, 2.05		1.20	0.73, 1.97	
III		1.69	1.07, 2.66		1.80	1.13, 2.87	

[1] HR = Hazard Ratio, CI = Confidence Interval

tbl_stack() to combine vertically

A **univariable** table:

```
tbl_uvsurv2 <-
  coxph(Surv(ttdeath, death) ~ trt,
        data = trial) |>
  tbl_regression(
    show_single_row = trt,
    label = trt ~ "Drug B vs A",
    exponentiate = TRUE
  )
tbl_uvsurv2
```

Characteristic	HR[1]	95% CI[1]	p-value
Drug B vs A	1.25	0.86, 1.81	0.2

[1] HR = Hazard Ratio, CI = Confidence Interval

A **multivariable** table:

```
tbl_mvsurv2 <-
  coxph(Surv(ttdeath, death) ~
          trt + grade + stage + marker,
        data = trial) |>
  tbl_regression(
    show_single_row = trt,
    label = trt ~ "Drug B vs A",
    exponentiate = TRUE,
    include = "trt"
  )
tbl_mvsurv2
```

Characteristic	HR[1]	95% CI[1]	p-value
Drug B vs A	1.30	0.88, 1.92	0.2

[1] HR = Hazard Ratio, CI = Confidence Interval

tbl_stack() to combine vertically

```
list(tbl_uvsurv2, tbl_mvsurv2) |>
  tbl_stack(
    group_header =
      c("Unadjusted", "Adjusted")
  )
```

Characteristic	HR[1]	95% CI[1]	p-value
Unadjusted			
Drug B vs A	1.25	0.86, 1.81	0.2
Adjusted			
Drug B vs A	1.30	0.88, 1.92	0.2

[1] HR = Hazard Ratio, CI = Confidence Interval

tbl_strata() for stratified tables

```
sm_trial |>
  mutate(grade = paste("Grade", grade)) |>
  tbl_strata(
    strata = grade,
    ~tbl_summary(.x, by = trt, missing = "no") |>
      modify_header(all_stat_cols() ~ "**{level}**")
  )
```

	Grade I		Grade II		Grade III	
Characteristic	**Drug A**[1]	**Drug B**[1]	**Drug A**[1]	**Drug B**[1]	**Drug A**[1]	**Drug B**[1]
Age	46 (36, 60)	48 (42, 55)	45 (31, 55)	51 (43, 57)	52 (42, 60)	45 (36, 52)
Tumor Response	8 (23%)	13 (41%)	7 (23%)	12 (36%)	13 (43%)	8 (24%)

[1] Median (IQR); n (%)

Define custom function `tbl_cmh()`

	Control		Case				
Characteristic	**Not Exposed**	**Exposed**	**Not Exposed**	**Exposed**	**Odds Ratio**	**CMH Odds Ratio**	**p-value**
T Stage							
Crude	46	42	52	60	1.26 (0.72, 2.21)	1.23 (0.69, 2.18)	0.6
T1	16	13	12	12	1.23 (0.42, 3.64)		
T2	14	13	11	16	1.57 (0.53, 4.60)		
T3	9	12	13	9	0.52 (0.15, 1.74)		
T4	7	4	16	23	2.52 (0.63, 10.0)		
Grade							
Crude	46	42	52	60	1.26 (0.72, 2.21)	1.26 (0.71, 2.22)	0.5
I	19	16	16	17	1.26 (0.49, 3.27)		
II	16	16	16	20	1.25 (0.48, 3.25)		
III	11	10	20	23	1.27 (0.44, 3.60)		

Define custom function `tbl_cmh()`

Characteristi	Control Not Exposed	Control Exposed	Case Not Exposed	Case Exposed	Odds Ratio	CMH Odds Ratio	p-value
T Stage							
Crude	46	42	52	60	1.26 (0.72, 2.21)	1.23 (0.69, 2.18)	0.6
T1	16	13	12	12	1.23 (0.42, 3.64)		
T2	14	13	11	16	1.57 (0.53, 4.60)		
T3	9	12	13	9	0.52 (0.15, 1.74)		
T4	7	4	16	23	2.52 (0.63, 10.0)		
Grade							
Crude	46	42	52	60	1.26 (0.72, 2.21)	1.26 (0.71, 2.22)	0.5
I	19	16	16	17	1.26 (0.49, 3.27)		
II	16	16	16	20	1.25 (0.48, 3.25)		
III	11	10	20	23	1.27 (0.44, 3.60)		

tbl_merge()

tbl_cross()

tbl_stack()

add_stat()

{gtreg} for regulatory submissions

- The {gtreg} package uses {gtsummary} to construct tables for regulatory agencies.
- https://shannonpileggi.github.io/gtreg/


```
gtreg::df_adverse_events |>
  gtreg::tbl_ae(
    id_df = gtreg::df_patient_characteristics,
    id = patient_id,
    ae = adverse_event,
    soc = system_organ_class,
    by = grade,
    strata = trt
  ) |>
  modify_header(gtreg::all_ae_cols() ~ "**Grade {by}**") |>
  bold_labels()
```

{gtreg} for regulatory submissions

	Drug A, N = 44					**Drug B**, N = 56				
Adverse Event	**Grade 1**	**Grade 2**	**Grade 3**	**Grade 4**	**Grade 5**	**Grade 1**	**Grade 2**	**Grade 3**	**Grade 4**	**Grade 5**
Blood and lymphatic system disorders	—	1 (2.3)	—	1 (2.3)	1 (2.3)	—	—	—	1 (1.8)	6 (11)
Anaemia	—	—	1 (2.3)	1 (2.3)	—	—	—	1 (1.8)	1 (1.8)	3 (5.4)
Increased tendency to bruise	—	—	—	1 (2.3)	—	—	—	—	3 (5.4)	2 (3.6)
Iron deficiency anaemia	—	—	—	1 (2.3)	1 (2.3)	1 (1.8)	2 (3.6)	—	1 (1.8)	1 (1.8)
Thrombocytopenia	—	1 (2.3)	—	1 (2.3)	—	—	—	3 (5.4)	—	4 (7.1)
Gastrointestinal disorders	—	—	—	2 (4.5)	1 (2.3)	—	—	—	2 (3.6)	5 (8.9)
Difficult digestion	—	—	—	3 (6.8)	—	1 (1.8)	—	—	—	1 (1.8)
Intestinal dilatation	1 (2.3)	—	—	—	—	1 (1.8)	1 (1.8)	—	—	1 (1.8)
Myochosis	—	2 (4.5)	1 (2.3)	—	—	—	1 (1.8)	—	1 (1.8)	3 (5.4)
Non-erosive reflux disease	3 (6.8)	—	—	—	—	1 (1.8)	—	—	3 (5.4)	3 (5.4)
Pancreatic enzyme abnormality	—	—	1 (2.3)	1 (2.3)	1 (2.3)	2 (3.6)	1 (1.8)	1 (1.8)	1 (1.8)	—

{gtsummary} themes

{gtsummary} theme basics

- A theme is a set of customization preferences that can be easily set and reused.
- Themes control default settings for existing functions
- Themes control more fine-grained customization not available via arguments or helper functions
- Easily use one of the available themes, or create your own

{gtsummary} default theme

```
reset_gtsummary_theme()
m1 |>
  tbl_regression(
    exponentiate = TRUE
  ) |>
  modify_caption(
    "Default Theme"
  )
```

Default Theme

Characteristic	OR[1]	95% CI[1]	p-value
Age	1.02	1.00, 1.04	0.091
T Stage			
T1	—	—	
T2	0.58	0.24, 1.37	0.2
T3	0.94	0.39, 2.28	0.9
T4	0.79	0.33, 1.90	0.6

[1] OR = Odds Ratio, CI = Confidence Interval

{gtsummary} theme_gtsummary_journal()

```r
reset_gtsummary_theme()
theme_gtsummary_journal(journal = "jama")

m1 |>
  tbl_regression(
    exponentiate = TRUE
  ) |>
  modify_caption(
    "Journal Theme (JAMA)"
  )
```

Journal Theme (JAMA)

Characteristic	OR (95% CI)[1]	p-value
Age	1.02 (1.00 to 1.04)	0.091
T Stage		
T1	—	
T2	0.58 (0.24 to 1.37)	0.22
T3	0.94 (0.39 to 2.28)	0.89
T4	0.79 (0.33 to 1.90)	0.61

[1] OR = Odds Ratio, CI = Confidence Interval

{gtsummary} theme_gtsummary_language()

```
reset_gtsummary_theme()
theme_gtsummary_language(language = "zh-tw"

m1 |>
  tbl_regression(
    exponentiate = TRUE
  ) |>
  modify_caption(
    "Language Theme (Chinese)"
  )
```

Language Theme (Chinese)

特色	OR[1]	95% CI[1]	P 值
Age	1.02	1.00, 1.04	0.091
T Stage			
T1	—	—	
T2	0.58	0.24, 1.37	0.2
T3	0.94	0.39, 2.28	0.9
T4	0.79	0.33, 1.90	0.6

[1] OR=勝算比, CI=信賴區間

Language options:

- German
- English
- Spanish
- French
- Gujarati
- Hindi
- Icelandic
- Japanese
- Korean
- Marathi
- Dutch
- Norwegian
- Portuguese
- Swedish
- Chinese Simplified
- Chinese Traditional

{gtsummary} theme_gtsummary_compact()

```
reset_gtsummary_theme()
theme_gtsummary_compact()

tbl_regression(m1, exponentiate = TRUE) |>
  modify_caption("Compact Theme")
```

Compact Theme

Characteristic	OR[1]	95% CI[1]	p-value
Age	1.02	1.00, 1.04	0.091
T Stage			
T1	—	—	
T2	0.58	0.24, 1.37	0.2
T3	0.94	0.39, 2.28	0.9
T4	0.79	0.33, 1.90	0.6

[1] OR = Odds Ratio, CI = Confidence Interval

Reduces padding and font size

{gtsummary} set_gtsummary_theme()

- `set_gtsummary_theme()` to use a custom theme.
- See the {gtsummary} + themes vignette for examples

http://www.danieldsjoberg.com/gtsummary/articles/themes.html

{gtsummary} print engines

{gtsummary} print engines

Print Engine	Function	HTML	PDF	RTF	Word
gt	`as_gt()`				
flextable	`as_flex_table()`				
huxtable	`as_hux_table()`				
kableExtra	`as_kable_extra()`				
kable	`as_kable()`				
tibble	`as_tibble()`				

{gtsummary} print engines

Use any print engine to customize table

::: {.cell output-location='column' hash='index_cache/revealjs/unnamed-chunk-61_334932da2615a3ce7c96f154d4c76d4b'}

```
library(gt)
trial |>
  select(age, grade) |>
  tbl_summary() |>
  as_gt() |>
  cols_width(label ~ px(300)) |>
  cols_align(columns = stat_0,
             align = "left")
```

Characteristic	N = 200[1]
Age	47 (38, 57)
Unknown	11
Grade	
I	68 (34%)
II	68 (34%)
III	64 (32%)

[1] Median (IQR); n (%)

:::

inline_text()

{gtsummary} reporting with inline_text()

- Tables are important, but we often need to **report results in-line**.
- Any statistic reported in a {gtsummary} table can be extracted and reported in-line in an R Markdown document with the `inline_text()` function.
- The pattern of what is reported can be modified with the `pattern=` argument.
- Default is `pattern = "{estimate} ({conf.level*100}% CI {conf.low}, {conf.high}; {p.value})"` for regression summaries.

{gtsummary} reporting with inline_text()

Characteristic	N	OR[1]	95% CI[1]	p-value
Chemotherapy Treatment	193			
Drug A		—	—	
Drug B		1.21	0.66, 2.24	0.5
Age	183	1.02	1.00, 1.04	0.10
Grade	193			
I		—	—	
II		0.95	0.45, 2.00	0.9
III		1.10	0.52, 2.29	0.8

[1] OR = Odds Ratio, CI = Confidence Interval

In Code: The odds ratio for age is `` `r inline_text(tbl_uvreg, variable = age)` ``

In Report: The odds ratio for age is 1.02 (95% CI 1.00, 1.04; p=0.10)

{gtsummary} reporting with inline_text()

```r
gts_small_summary <-
  trial %>%
  tbl_summary(
    by = trt,
    include = marker,
    missing = "no"
  ) %>%
  add_difference()
gts_small_summary
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]	Difference[2]	95% CI[2,3]	p-value[2]
Marker Level (ng/mL)	0.84 (0.24, 1.57)	0.52 (0.19, 1.20)	0.20	-0.05, 0.44	0.12

[1] Median (IQR)
[2] Welch Two Sample t-test
[3] CI = Confidence Interval

In Code:

- The median (IQR) marker among participants randomized to Drug A was `` `r inline_text(gts_small_summary, variable = marker, column = 'Drug A')` ``.
- The median (IQR) age among participants randomized to Drug A was `` `r inline_text(gts_small_summary, variable = marker, column = 'Drug A', pattern = '{median}')` ``.
- The difference in marker level was `` `r inline_text(gts_small_summary, variable = marker, pattern = '{estimate} (95% {ci})')` ``.

In Report:

- The median (IQR) marker among participants randomized to Drug A was 0.84 (0.24, 1.57).
- The median (IQR) age among participants randomized to Drug A was 0.84.
- The difference in marker level was 0.20 (95% -0.05, 0.44).

In Summary

Acknowledgements

- The original materials was licensed under a Creative Commons Attribution 4.0 International License.
- I extend my sincere appreciation to all the creators of the original works that we I adopted for teaching purposes.
- Please learn more about gtsummary from the author of the package - https://www.danieldsjoberg.com/clinical-reporting-gtsummary-rmed/material.html

{gtsummary} website

http://www.danieldsjoberg.com/gtsummary/

gtsummary

(https://github.com/ddsjoberg/gtsummary)

The {gtsummary} package provides an elegant and flexible way to create publication-ready analytical and summary tables using the **R** programming language. The {gtsummary} package summarizes data sets, regression models, and more, using sensible defaults with highly customizable capabilities.

- **Summarize data frames or tibbles** (https://www.danieldsjoberg.com/gtsummary/articles/tbl_summary.html) easily in **R**. Perfect for

Thank you