Introduction to R Programming

IBRAHIM KASSOUM Habibou

2025-02-03

Contents

Chapter 1

Introduction

Welcome to this introductory R programming course, specifically designed for Health Economics students to quickly give them a glimse of R. In today's data-driven healthcare environment, the ability to analyze and visualize health economic data effectively is crucial. This 8 hours course will equip you with the essential skills to use R for data analysis, and visualization using R Markdown.

1.1 Learning Objectives

By the end of this course, you will be able to:

- Master the fundamentals of R programming and RStudio environment
- Understand data wrangling techniques using tidyverse packages
- Create professional visualizations with ggplot2
- Apply these skills to real-world health economics scenarios througth exercices

1.2 Why R and R Markdown?

R has become an essential tool in health economics for several reasons:

- **Open-source and Free**: Access to powerful statistical and analytical tools without licensing costs
- **Reproducibility**: Your analyses can be easily shared and reproduced by others
- **Extensive Package Ecosystem**: Specialized packages for health economics and statistics
- **Data Visualization**: Professional-quality graphics and interactive visualizations

- **Integration with Other Tools**: Seamless integration with databases, other statistical software, and reporting tools

1.3 Course Structure

Our course is organized into several modules:

1.3.1 R basics and data manipulations (2h)

- Introduction to R and RStudio
- Data types and objects
- Importing data and basic data manipulations
- Exercices

1.3.2 Data Wrangling and basic statistics (2h)

- Data cleaning (Filtering, selecting, and transforming data) and preparation
- Combining datasets
- Computing statistics (mean, variance)
- Exercices

1.3.3 Data Visualization with ggplot (2h)

- Principles of effective visualization
- Creating plots with ggplot2
- Customizing graphics for healthcare data
- Exercices

1.4 Prerequisites

No prior programming experience is required Basic statistical knowledge Laptop with R and RStudio installed

1.5 Course Materials

All course materials will be provided at the end of the course in an R Markdown book format, allowing you to:

- Follow along with interactive examples
- Execute code in real-time
- Create your own annotations
- Test your leaning through exercices

1.6 Practical Applications in Health Economics

Throughout the course, we'll work with real-world examples relevant to health economics, such as:

- Covid 19
- Blood storage

using the **medicaldata** library.

1.7 Assessment

Your learning will be evaluated through exercises during sessions.

1.8 Getting Started

Before our first session, please:

- Install R from https://cran.r-project.org/
- Install RStudio Desktop from https://posit.co/download/rstudio-desktop/
- Test your installation by running basic R commands

RStudio is a free software tool that makes working with the R programming language much easier. Think of it as a specialized workspace for R - like having a well-organized desk with all your tools in the right place. While you can use R by itself, RStudio gives you helpful features like:

- A cleaner way to write and run your code
- Better organization of your data
- Simple ways to manage R add-ons (called packages)
- Easy viewing of your results

There's also a paid professional version available from the company **Posit**, but the free version has everything most people need. If you're planning to work with R, using RStudio is highly recommended since it's specifically designed to make R programming more user-friendly.

1.8.1 Understanding the RStudio Interface

When you first open RStudio, you'll encounter three main sections:

- **Console** :This is where the actual work happens in R. You can type R commands directly here and press Enter to execute them. Think of it as R's command center.
- **Environment**: Shows all your current R objects (variables, data, functions) like a workspace overview where you can see everything you've created.

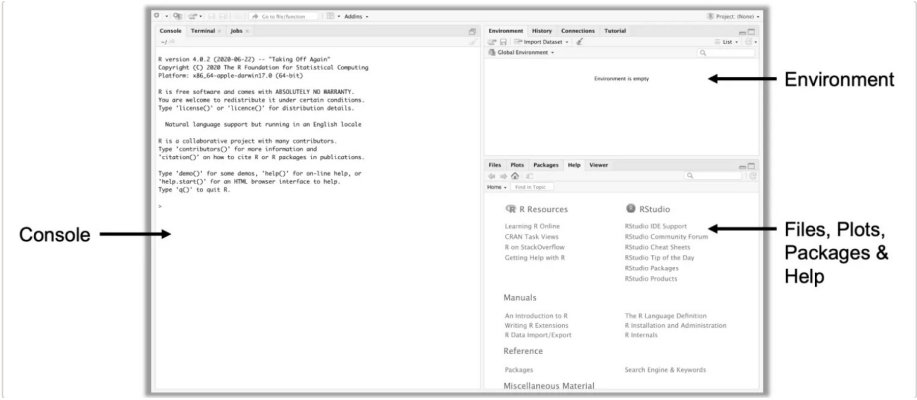


Figure 1.1: Harrer et al. [2023]

- **Files/Plots/Packages/Help**: a multi-purpose area that shows: your computer's files and folders, graphs and visualizations you create, installed R packages, help documentation.
- **Editor** : opens when you create a new R script (File > New File > R Script). This is where you write and save your R code Files are saved with a .R extension. *To run code from the editor: Select the code you want to run; Click the "Run" button ( ) in the top-right of the editor Or use the shortcut Ctrl+Enter (Windows) / Cmd+Enter (Mac)

Note: While you write code in the editor, it always executes in the console. Think of the editor as your notebook and the console as where the actual computation happens.

1.9 Support and Resources

- Office number: 404
- Office hours: 8AM-8PM
- Email: habibou.ibrahim_kassoum@doctorant.uca.fr
- Online forum for questions and discussions (stackoverflow)

Some online resources and reading materials: - Bookdown

- Evaluation of Randomized Controlled Trials With R
- Pierre Beaucoral's classes

Remember, learning to program is like learning a new language - it takes practice and patience. Don't hesitate to ask questions and collaborate with your peers throughout this journey.

Let's begin this exciting journey into the world of R programming for health

economics!

Chapter 2

R basic data manipulations

2.1 Introduction to R Syntax

R is a powerful programming language designed specifically for statistical computing and data analysis. Let's explore its fundamental concepts.

2.1.1 Variables and Data Types

Before diving into coding, it's essential to understand the basic data types in R. Think of variables as containers that store different types of information. R has three main types of data that you'll use frequently:

```
# Numeric (integers and decimals)
# Numbers can be whole (integers) or have decimal points
my_number <- 42.5
print(my_number)
```

```
## [1] 42.5
```

```
# Character (text strings)
# Any text data must be enclosed in quotes
my_text <- "Hello R Markdown!"
print(my_text)
```

```
## [1] "Hello R Markdown!"
```

```
# Logical (boolean values)
# TRUE/FALSE values are useful for conditional operations
my_logical <- TRUE
print(my_logical)
```

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

The <- symbol is the assignment operator in R. While you can use =, <- is preferred in the R community. Let's practice creating meaningful variables:

```
# Create and assign variables
age <- 25         # Numeric
name <- "Alice"   # Character
is_student <- TRUE  # Logical

# Display our variables
print(age)
```

```
## [1] 25
```

```
print(name)
```

```
## [1] "Alice"
```

```
print(is_student)
```

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

```
# Check variable types using class()
# This is useful to confirm what type of data you're working with
class(age)
```

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

```
class(name)
```

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

```
class(is_student)
```

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

2.1.2 Basic Operations

R can perform various arithmetic operations just like a calculator. These operations are fundamental to data analysis: addition, subtraction, multiplication, division

```
# Basic arithmetic operations are straightforward
addition <- 10 + 5       # Adding numbers
subtraction <- 10 - 5    # Subtracting numbers
multiplication <- 10 * 5 # Multiplying numbers
division <- 10 / 5       # Dividing numbers

# Display results
print(addition)
```

```
## [1] 15
```

```
print(subtraction)
```

```
## [1] 5
```

```
print(multiplication)
```

```
## [1] 50
```

```
print(division)
```

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

```
# More complex mathematical operations
power <- 2^3            # Exponentiation (2 to the power of 3)
square_root <- sqrt(16) # Square root function
print(power)
```

```
## [1] 8
```

```
print(square_root)
```

```
## [1] 4
```

Logical operations are crucial for data filtering and conditional statements:

```
# Comparison operators return TRUE or FALSE
equals <- 5 == 5        # Equality comparison
greater_than <- 10 > 5  # Greater than comparison
less_than <- 3 < 7     # Less than comparison

# Logical operators combine TRUE/FALSE values
and_operator <- TRUE & TRUE    # Both conditions must be TRUE
or_operator <- TRUE | FALSE    # At least one condition must be TRUE

# Print results
print(equals)
```

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

```
print(greater_than)
```

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

```
print(less_than)
```

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

```
print(and_operator)
```

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

```
print(or_operator)
```

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

2.2 Objects in R

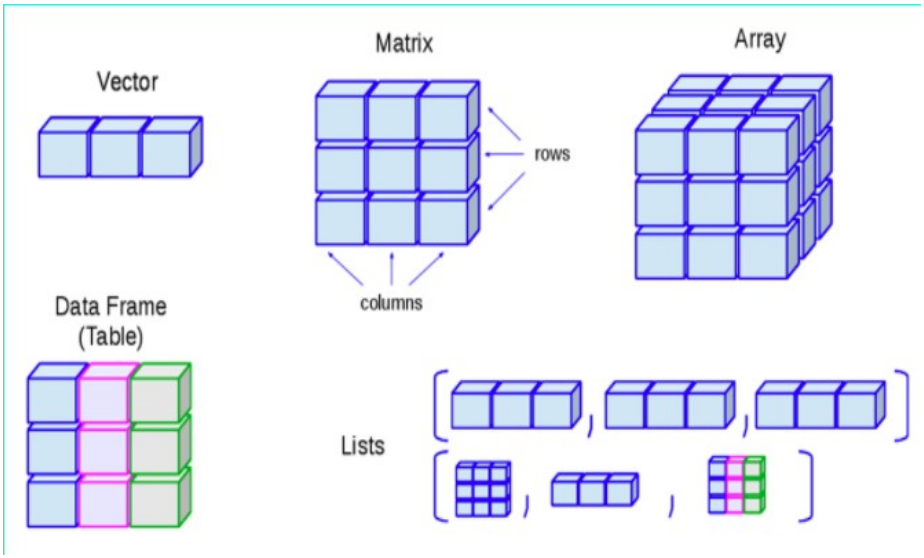


Figure 2.1: (image from http://venus.ifca.unican.es/Rintro/dataStruct.html)

2.2.1 Working with Vectors

Vectors are one of the most basic data structures in R. Think of them as a collection of elements of the same type, like a list of numbers or strings:

```
# Creating vectors using the combine function c()
numeric_vector <- c(1, 2, 3, 4, 5)           # Vector of numbers
character_vector <- c("apple", "banana", "cherry", "avocado", "mango")  # Vector of st
logical_vector <- c(TRUE, FALSE, TRUE)       # Vector of logical values

# Display vectors
print(numeric_vector)
```

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

```
print(character_vector)
```

```
## [1] "apple"   "banana"  "cherry"  "avocado" "mango"
```

```
print(logical_vector)
```

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

```
# Vector operations - R can perform operations on entire vectors at once
# This is called vectorization and is very efficient
print(length(numeric_vector)) # length of the vector
```

```
## [1] 5
```

```
doubled <- numeric_vector * 2  # Multiply each element by 2
print(doubled)
```

```
## [1]  2  4  6  8 10
```

```
# Accessing elements using indexing
# R uses 1-based indexing (first element is at position 1, not 0)
first_element <- numeric_vector[1]  # Get first element
selected_elements <- numeric_vector[c(1, 3, 5)]  # Get specific elements
print(first_element)
```

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

```
print(selected_elements)
```

```
## [1] 1 3 5
```

2.2.2 Creating Sequences

R provides several convenient ways to create sequences of numbers, which is particularly useful for data analysis and plotting:

```
# Using seq() for more control over sequence generation
sequence1 <- seq(1, 10)          # Basic sequence from 1 to 10
sequence2 <- seq(0, 20, by = 2)  # Even numbers from 0 to 20

# Using : operator for simple sequences
sequence3 <- 1:10                # Another way to create sequence from 1 to 10

# Using rep() to repeat values
repeated <- rep(5, times = 3)    # Repeat the number 5 three times

print(sequence1)
```

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

```
print(sequence2)
```

```
##  [1]  0  2  4  6  8 10 12 14 16 18 20
```

```
print(sequence3)
```

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

```
print(repeated)
```

```
## [1] 5 5 5
```

2.3 Exercise 1

Understanding Data Types

Create three variables and assign them values of different data types:

- A numeric variable representing your height in centimeters.
- A character variable storing your favorite fruit.
- A logical variable indicating whether you like R programming.

Then, print each variable and use class() to check its data type.

Basic Arithmetic Operations

Perform the following calculations and store the results in variables:

- Multiply 15 by 3.
- Subtract 7 from 100.
- Compute the square root of 64.
- Raise 3 to the power of 4.

Print all results.

Vector Manipulation

- Create a numeric vector containing the numbers 2, 4, 6, 8, 10. Multiply all elements of the vector by 3. Extract the second and fourth elements of the vector.
- Create a character vector with three country names of your choice. Multiply it by 3.

Creating Sequences

- Create a sequence of numbers from 5 to 50 with a step size of 5.
- Generate a sequence of odd numbers from 1 to 15 using seq().
- Use rep() to create a vector that repeats the number 7 five times.

2.3.1 Working with Data Frames

Data frames are the most common way to work with structured data in R. They're similar to Excel spreadsheets or database tables:

```r
# Create a simple data frame
# Each column can have a different data type
students_df <- data.frame(
  name = c("Monelson", "Noemie", "Alphonse", "Aichatou", "Laurene", "Anonkoua"),     # Character c
  age = c(25, 20, 23, 22, 22, 26),                    # Numeric column
  note = c(10, 15, 13, 15, 16.5, 9),                  # Numeric column
  is_graduate = c(FALSE, TRUE, TRUE, TRUE, T, F)      # Logical column
)

# Display the data frame
print(students_df)
```

```
##       name age note is_graduate
## 1 Monelson  25 10.0       FALSE
## 2   Noemie  20 15.0        TRUE
## 3 Alphonse  23 13.0        TRUE
## 4 Aichatou  22 15.0        TRUE
## 5  Laurene  22 16.5        TRUE
## 6 Anonkoua  26  9.0       FALSE
```

```r
# Basic data frame operations
# Access a column using $ notation
print(students_df$age)
```

```
## [1] 25 20 23 22 22 26
```

```r
# Access a row using index
print(students_df[5, ])
```

```
##      name age note is_graduate
## 5 Laurene  22 16.5        TRUE
```

```r
# Add a new column - must match the number of rows
students_df$height <- c(175, 168, 182, 150, 160, 155)
print(students_df)
```

```
##       name age note is_graduate height
## 1 Monelson  25 10.0       FALSE    175
## 2   Noemie  20 15.0        TRUE    168
## 3 Alphonse  23 13.0        TRUE    182
## 4 Aichatou  22 15.0        TRUE    150
## 5  Laurene  22 16.5        TRUE    160
## 6 Anonkoua  26  9.0       FALSE    155
```

2.4 Importing an Manipulating data in R

2.4.1 Importing data

```
# Installing and Loading the Required Package

## First, install and load the `medicaldata` package to access the `covid_testing` data


# Install the package (only needs to be done once)
## install.packages("medicaldata")

# Load the package into the R session
library("medicaldata")

# Load the COVID-19 testing dataset from the medicaldata package
covid <- medicaldata::covid_testing

# Display the first few rows of the dataset
head(covid)
```

```
##   subject_id fake_first_name fake_last_name gender pan_day test_id
## 1       1412         jhezane     westerling female       4   covid
## 2        533           penny      targaryen female       7   covid
## 3       9134           grunt         rivers   male       7   covid
## 4       8518      melisandre          swyft female       8   covid
## 5       8967          rolley       karstark   male       8   covid
## 6      11048           megga       karstark female       8   covid
##         clinic_name   result demo_group age drive_thru_ind ct_result orderset
## 1  inpatient ward a negative    patient 0.0             0        45        0
## 2      clinical lab negative    patient 0.0             1        45        0
## 3      clinical lab negative    patient 0.8             1        45        1
## 4      clinical lab negative    patient 0.8             1        45        1
## 5    emergency dept negative    patient 0.8             0        45        1
## 6 oncology day hosp negative    patient 0.8             0        45        0
##   payor_group        patient_class col_rec_tat rec_ver_tat
## 1  government            inpatient         1.4         5.2
## 2  commercial       not applicable         2.3         5.8
## 3        <NA>                 <NA>         7.3         4.7
## 4        <NA>                 <NA>         5.8         5.0
## 5  government            emergency         1.2         6.4
## 6  commercial recurring outpatient         1.4         7.0
```

```
# Show the column names of the dataset
colnames(covid)
```

```
##  [1] "subject_id"      "fake_first_name" "fake_last_name"  "gender"
```

```
##  [5] "pan_day"         "test_id"         "clinic_name"     "result"
##  [9] "demo_group"      "age"             "drive_thru_ind"  "ct_result"
## [13] "orderset"        "payor_group"     "patient_class"   "col_rec_tat"
## [17] "rec_ver_tat"
```

```
# Show the number of columns in the dataset
ncol(covid)
```

```
## [1] 17
```

```
# Show the number of rows in the dataset
nrow(covid)
```

```
## [1] 15524
```

2.4.2 Data Frame Manipulation

Data frames support powerful operations for data analysis:

```
# Filter data based on conditions
high_gpa <- students_df[students_df$gpa > 3.5, ]  # Select rows where GPA > 3.5
print(high_gpa)
```

```
## [1] name        age         note        is_graduate height
## <0 lignes> (ou 'row.names' de longueur nulle)
```

```
# Sort data using order()
sorted_by_age <- students_df[order(students_df$age), ]  # Sort by age
print(sorted_by_age)
```

```
##       name age note is_graduate height
## 2   Noemie  20 15.0        TRUE    168
## 4 Aichatou  22 15.0        TRUE    150
## 5  Laurene  22 16.5        TRUE    160
## 3 Alphonse  23 13.0        TRUE    182
## 1 Monelson  25 10.0       FALSE    175
## 6 Anonkoua  26  9.0       FALSE    155
```

```
# Or

sort(students_df$age)
```

```
## [1] 20 22 22 23 25 26
```

2.5 Exercise 2

- Import the **"blood_storage"** database from the package **medicaldata**.
- Log-transform the variable *age* in data and save the result as *age.log*.

- Square all values in *PVol* (Prostate volume) and save the result as *PVol.squared* within the dataset.
- Check whether the *AA* (African American race) is of class factor (0 = "non-African-American"; 1 = "African American").
- Filter out the records for which *(PreopPSA >= 10)* and *(Recurrence == 0)*.
- In the *fifth* and *sixth* rows of the data, change the value of *Age* to NA (missing).
- Remove the variables from the dataset: *AA*,*FamHx*,*OrganConfined*.

Remember that R is case-sensitive and very particular about syntax. Pay attention to brackets, commas, and quotation marks. The best way to learn is by experimenting with the code and modifying it to see what happens!

Chapter 3

Data Wrangling and basic statistics

3.1 1. Introduction to dplyr

The `dplyr` package is a powerful tool in R for data manipulation. It provides functions for filtering, selecting, arranging, summarizing, and mutating data in an easy and readable way. The `%>%` (pipe operator) is commonly used to link functions together.

```
# Install and load necessary packages
#install.packages("dplyr")
#install.packages("medicaldata")

library(dplyr)
```

```
##
## Attachement du package : 'dplyr'
```

```
## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag
```

```
## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(medicaldata)

# Print all the function in the package dplyr
ls("package:dplyr")
```

```
##   [1] "%>%"                  "across"                "add_count"
##   [4] "add_count_"           "add_row"               "add_rownames"
##   [7] "add_tally"            "add_tally_"            "all_equal"
##  [10] "all_of"               "all_vars"              "anti_join"
##  [13] "any_of"               "any_vars"              "arrange"
##  [16] "arrange_"             "arrange_all"           "arrange_at"
##  [19] "arrange_if"           "as.tbl"                "as_data_frame"
##  [22] "as_label"             "as_tibble"             "auto_copy"
##  [25] "band_instruments"     "band_instruments2"     "band_members"
##  [28] "bench_tbls"           "between"               "bind_cols"
##  [31] "bind_rows"            "c_across"              "case_match"
##  [34] "case_when"            "changes"               "check_dbplyr"
##  [37] "coalesce"             "collapse"              "collect"
##  [40] "combine"              "common_by"             "compare_tbls"
##  [43] "compare_tbls2"        "compute"               "consecutive_id"
##  [46] "contains"             "copy_to"               "count"
##  [49] "count_"               "cross_join"            "cumall"
##  [52] "cumany"               "cume_dist"             "cummean"
##  [55] "cur_column"           "cur_data"              "cur_data_all"
##  [58] "cur_group"            "cur_group_id"          "cur_group_rows"
##  [61] "current_vars"         "data_frame"            "db_analyze"
##  [64] "db_begin"             "db_commit"             "db_create_index"
##  [67] "db_create_indexes"    "db_create_table"       "db_data_type"
##  [70] "db_desc"              "db_drop_table"         "db_explain"
##  [73] "db_has_table"         "db_insert_into"        "db_list_tables"
##  [76] "db_query_fields"      "db_query_rows"         "db_rollback"
##  [79] "db_save_query"        "db_write_table"        "dense_rank"
##  [82] "desc"                 "dim_desc"              "distinct"
##  [85] "distinct_"            "distinct_all"          "distinct_at"
##  [88] "distinct_if"          "distinct_prepare"      "do"
##  [91] "do_"                  "dplyr_col_modify"      "dplyr_reconstruct"
##  [94] "dplyr_row_slice"      "ends_with"             "enexpr"
##  [97] "enexprs"              "enquo"                 "enquos"
## [100] "ensym"                "ensyms"                "eval_tbls"
## [103] "eval_tbls2"           "everything"            "explain"
## [106] "expr"                 "failwith"              "filter"
## [109] "filter_"              "filter_all"            "filter_at"
## [112] "filter_if"            "first"                 "full_join"
## [115] "funs"                 "funs_"                 "glimpse"
## [118] "group_by"             "group_by_"             "group_by_all"
## [121] "group_by_at"          "group_by_drop_default" "group_by_if"
## [124] "group_by_prepare"     "group_cols"            "group_data"
## [127] "group_indices"        "group_indices_"        "group_keys"
## [130] "group_map"            "group_modify"          "group_nest"
## [133] "group_rows"           "group_size"            "group_split"
## [136] "group_trim"           "group_vars"            "group_walk"
```

```
## [139] "grouped_df"            "groups"               "id"
## [142] "ident"                 "if_all"               "if_any"
## [145] "if_else"               "inner_join"           "intersect"
## [148] "is.grouped_df"         "is.src"               "is.tbl"
## [151] "is_grouped_df"         "join_by"              "lag"
## [154] "last"                  "last_col"             "last_dplyr_warnings"
## [157] "lead"                  "left_join"            "location"
## [160] "lst"                   "make_tbl"             "matches"
## [163] "min_rank"              "mutate"               "mutate_"
## [166] "mutate_all"            "mutate_at"            "mutate_each"
## [169] "mutate_each_"          "mutate_if"            "n"
## [172] "n_distinct"            "n_groups"             "na_if"
## [175] "near"                  "nest_by"              "nest_join"
## [178] "new_grouped_df"        "new_rowwise_df"       "nth"
## [181] "ntile"                 "num_range"            "one_of"
## [184] "order_by"              "percent_rank"         "pick"
## [187] "progress_estimated"    "pull"                 "quo"
## [190] "quo_name"              "quos"                 "recode"
## [193] "recode_factor"         "reframe"              "relocate"
## [196] "rename"                "rename_"              "rename_all"
## [199] "rename_at"             "rename_if"            "rename_vars"
## [202] "rename_vars_"          "rename_with"          "right_join"
## [205] "row_number"            "rows_append"          "rows_delete"
## [208] "rows_insert"           "rows_patch"           "rows_update"
## [211] "rows_upsert"           "rowwise"              "same_src"
## [214] "sample_frac"           "sample_n"             "select"
## [217] "select_"               "select_all"           "select_at"
## [220] "select_if"             "select_var"           "select_vars"
## [223] "select_vars_"          "semi_join"            "setdiff"
## [226] "setequal"              "show_query"           "slice"
## [229] "slice_"                "slice_head"           "slice_max"
## [232] "slice_min"             "slice_sample"         "slice_tail"
## [235] "sql"                   "sql_escape_ident"     "sql_escape_string"
## [238] "sql_join"              "sql_select"           "sql_semi_join"
## [241] "sql_set_op"            "sql_subquery"         "sql_translate_env"
## [244] "src"                   "src_df"               "src_local"
## [247] "src_mysql"             "src_postgres"         "src_sqlite"
## [250] "src_tbls"              "starts_with"          "starwars"
## [253] "storms"                "summarise"            "summarise_"
## [256] "summarise_all"         "summarise_at"         "summarise_each"
## [259] "summarise_each_"       "summarise_if"         "summarize"
## [262] "summarize_"            "summarize_all"        "summarize_at"
## [265] "summarize_each"        "summarize_each_"      "summarize_if"
## [268] "sym"                   "symdiff"              "syms"
## [271] "tally"                 "tally_"               "tbl"
## [274] "tbl_df"                "tbl_nongroup_vars"    "tbl_ptype"
```

```
## [277] "tbl_vars"              "tibble"               "top_frac"
## [280] "top_n"                 "transmute"            "transmute_"
## [283] "transmute_all"         "transmute_at"         "transmute_if"
## [286] "tribble"               "type_sum"             "ungroup"
## [289] "union"                 "union_all"            "validate_grouped_df"
## [292] "validate_rowwise_df"   "vars"                 "where"
## [295] "with_groups"           "with_order"           "wrap_dbplyr_obj"
```

```
# Load the COVID-19 dataset
covid <- medicaldata::covid_testing
```

3.2 Filtering, Selecting, and Arranging Data

3.2.1 Filtering Data

3.2.1.1 Filtering with Multiple Conditions

We can filter the dataset to focus on specific conditions. For example, let's filter cases where patients tested positive for COVID-19 (result == "positive").

```
positive_cases <- covid %>%
  filter(result == "positive")

# Show the first few rows of the filtered data
head(positive_cases)
```

```
## # A tibble: 6 x 17
##   subject_id fake_first_name fake_last_name gender pan_day test_id clinic_name
##        <dbl> <chr>           <chr>          <chr>    <dbl> <chr>   <chr>
## 1       2114 azzak           tully          male        10 covid   inpatient wa~
## 2       7240 arryk           mormont        male        11 covid   clinical lab
## 3      11391 zei             umber          female      11 covid   s  care ntwk
## 4        902 owen            seaworth       male        12 covid   emergency de~
## 5       2573 glendon         lannister      male        12 covid   emergency de~
## 6       5771 janna           lannister      female      12 covid   hem onc day ~
## # i 10 more variables: result <chr>, demo_group <chr>, age <dbl>,
## #   drive_thru_ind <dbl>, ct_result <dbl>, orderset <dbl>, payor_group <chr>,
## #   patient_class <chr>, col_rec_tat <dbl>, rec_ver_tat <dbl>
```

3.2.1.2 Filtering with multiple conditions

We can filter patients who are female (gender == "female") and tested positive for COVID-19.

```
female_positive <- covid %>%
  filter(gender == "female", result == "positive")
```

```
# Show the first few rows of the filtered data
head(female_positive)
```

```
## # A tibble: 6 x 17
##   subject_id fake_first_name fake_last_name gender pan_day test_id clinic_name
##        <dbl> <chr>           <chr>          <chr>    <dbl> <chr>   <chr>
## 1      11391 zei             umber          female      11 covid   s  care ntwk
## 2       5771 janna           lannister      female      12 covid   hem onc day ~
## 3      11381 sansa           martell        female      13 covid   clinical lab
## 4        864 meera           westerling     female      14 covid   clinical lab
## 5       5023 chataya         mormont        female      15 covid   emergency de~
## 6       6493 sybelle         karstark       female      16 covid   emergency de~
## # i 10 more variables: result <chr>, demo_group <chr>, age <dbl>,
## #   drive_thru_ind <dbl>, ct_result <dbl>, orderset <dbl>, payor_group <chr>,
## #   patient_class <chr>, col_rec_tat <dbl>, rec_ver_tat <dbl>
```

3.2.2 Selecting Specific Columns

If we only need a few columns, we can use **select()** to keep only the relevant ones.

```
selected_data <- covid %>%
  select(subject_id, age, gender, result)

# Display the selected columns
head(selected_data)
```

```
## # A tibble: 6 x 4
##   subject_id   age gender result
##        <dbl> <dbl> <chr>  <chr>
## 1       1412   0   female negative
## 2        533   0   female negative
## 3       9134   0.8 male   negative
## 4       8518   0.8 female negative
## 5       8967   0.8 male   negative
## 6      11048   0.8 female negative
```

We can use helper functions to select columns dynamically.

```
selected_columns <- covid %>%
  select(starts_with("fake"), contains("result"))

# Display the selected columns
head(selected_columns)
```

```
## # A tibble: 6 x 4
##   fake_first_name fake_last_name result   ct_result
##   <chr>           <chr>          <chr>        <dbl>
```

```
## 1 jhezane         westerling     negative        45
## 2 penny           targaryen      negative        45
## 3 grunt           rivers         negative        45
## 4 melisandre      swyft          negative        45
## 5 rolley          karstark       negative        45
## 6 megga           karstark       negative        45
```

3.2.3 Modifying Values and Missing

You can change a value in a column or even replace values with NA. For example, we set the *"fake_name_last"* to NA for the 5th row.

```
# First

covid[5, "fake_first_name"] <- "SAWADOGO"
covid[5, "fake_last_name"] <- "Laurent Benjamin"
covid[5, "age"] <- 26
covid[5, "result"] <- "positive"


covid[6, "fake_first_name"] <- "OUATTARA Siguissongui"
covid[6, "fake_last_name"] <- "Sarah"
covid[6, "age"] <- NA
covid[6, "result"] <- "positive"


# Display the selected columns
head(covid, 7)
```

```
## # A tibble: 7 x 17
##   subject_id fake_first_name   fake_last_name gender pan_day test_id clinic_name
##        <dbl> <chr>             <chr>          <chr>    <dbl> <chr>   <chr>
## 1       1412 jhezane           westerling     female       4 covid   inpatient ~
## 2        533 penny             targaryen      female       7 covid   clinical l~
## 3       9134 grunt             rivers         male         7 covid   clinical l~
## 4       8518 melisandre        swyft          female       8 covid   clinical l~
## 5       8967 SAWADOGO          Laurent Benja~ male         8 covid   emergency ~
## 6      11048 OUATTARA Siguiss~ Sarah          female       8 covid   oncology d~
## 7        663 ithoke            targaryen      male         9 covid   clinical l~
## # i 10 more variables: result <chr>, demo_group <chr>, age <dbl>,
## #   drive_thru_ind <dbl>, ct_result <dbl>, orderset <dbl>, payor_group <chr>,
## #   patient_class <chr>, col_rec_tat <dbl>, rec_ver_tat <dbl>
```

3.2.4 Arranging Data

We can arrange the dataset in descending order of age.

```
sorted_data <- covid %>%
  arrange(desc(age))

# Display the sorted columns
head(sorted_data)
```

```
## # A tibble: 6 x 17
##   subject_id fake_first_name fake_last_name gender pan_day test_id clinic_name
##        <dbl> <chr>           <chr>          <chr>    <dbl> <chr>   <chr>
## 1       3049 sansa           westerling     female     105 covid   line clinica~
## 2       4078 walda           harlaw         female      48 covid   emergency de~
## 3      12293 andrey          tyrell         male        87 covid   emergency de~
## 4      11177 harra           baratheon      female     100 covid   line clinica~
## 5        337 maerie          baratheon      female     105 covid   emergency de~
## 6       8426 missandei       tarly          female      94 covid   line clinica~
## # i 10 more variables: result <chr>, demo_group <chr>, age <dbl>,
## #   drive_thru_ind <dbl>, ct_result <dbl>, orderset <dbl>, payor_group <chr>,
## #   patient_class <chr>, col_rec_tat <dbl>, rec_ver_tat <dbl>
```

3.3 Adding and Modifying Columns

3.3.1 Creating a New Column

We can use mutate() to add new variables. Here, we create a new column to classify patients as “Young” (under 50) or “Elderly” (50 and above).

```
covid <- covid %>%
  mutate(age_group = case_when(
    age <= 5 ~ "Underfive",
    age > 5 & age <= 10 ~ "Children",
    age > 10 & age <= 15 ~ "Teenagers",
    age > 15 & age <= 30 ~ "Young Adults",
    age > 30 & age <= 60 ~ "Adults",
    age > 60 ~ "Elderly"
  ))

# Display results
head(covid %>% select(age, age_group), 10)
```

```
## # A tibble: 10 x 2
##      age age_group
##    <dbl> <chr>
##  1   0   Underfive
##  2   0   Underfive
##  3   0.8 Underfive
##  4   0.8 Underfive
```

```
##  5  26   Young Adults
##  6  NA   <NA>
##  7   0.8 Underfive
##  8   0   Underfive
##  9   0   Underfive
## 10   0.9 Underfive
```

3.3.2 Transforming a Column

We can apply transformations to existing columns. For example, we log-transform the age variable.

```
covid <- covid %>%
  mutate(age_log = log(age),
         age_squarred = age^2,
         age_square_root = sqrt(age))
# Display results
head(covid %>% select(age_log, age_squarred, age_square_root), 10)
```

```
## # A tibble: 10 x 3
##     age_log age_squarred age_square_root
##       <dbl>        <dbl>           <dbl>
##  1 -Inf             0               0
##  2 -Inf             0               0
##  3   -0.223         0.64            0.894
##  4   -0.223         0.64            0.894
##  5    3.26        676               5.10
##  6   NA            NA              NA
##  7   -0.223         0.64            0.894
##  8 -Inf             0               0
##  9 -Inf             0               0
## 10   -0.105         0.81            0.949
```

3.4 Summarizing Data

3.4.1 Simple summarizing

We can use summarise() to calculate statistics. Here, we find the average and standard deviation of patients' ages.

```
age_stats <- covid %>%
  summarise(mean_age = mean(age),
            median_age = median(age),
            sd_age = sd(age),
            min_age = min(age),
            max_age = max(age))
```

```
age_stats_without_na <- covid %>%
  summarise(mean_age = mean(age, na.rm=TRUE),
            median_age = median(age, na.rm=TRUE),
            sd_age = sd(age, na.rm=TRUE),
            min_age = min(age, na.rm=TRUE),
            max_age = max(age, na.rm=TRUE))

# Display results
age_stats
```

```
## # A tibble: 1 x 5
##   mean_age median_age sd_age min_age max_age
##      <dbl>      <dbl>  <dbl>   <dbl>   <dbl>
## 1       NA         NA     NA      NA      NA
```

```
age_stats_without_na
```

```
## # A tibble: 1 x 5
##   mean_age median_age sd_age min_age max_age
##      <dbl>      <dbl>  <dbl>   <dbl>   <dbl>
## 1     14.2          9   16.5       0     138
```

3.4.2 Grouping and Summarizing

Grouping allows us to calculate statistics for specific subgroups. Let's calculate the average age for each test result category.

```
age_by_result <- covid %>%
  group_by(result) %>%
  summarise(n=n(),
            mean_age = mean(age, na.rm = TRUE),
            sd_age = sd(age, na.rm = TRUE))%>%
  mutate(freq = n / sum(n), freq_in_percent = freq * 100)

# Display results
age_by_result
```

```
## # A tibble: 3 x 6
##   result       n mean_age sd_age   freq freq_in_percent
##   <chr>    <int>    <dbl>  <dbl>  <dbl>           <dbl>
## 1 invalid    301     14.9   15.9 0.0194            1.94
## 2 negative 14356     13.9   16.2 0.925            92.5
## 3 positive   867     19.2   19.4 0.0558            5.58
```

3.5 Joining Data Frames

Sometimes, we need to combine data from different sources. Here's how to join two datasets.

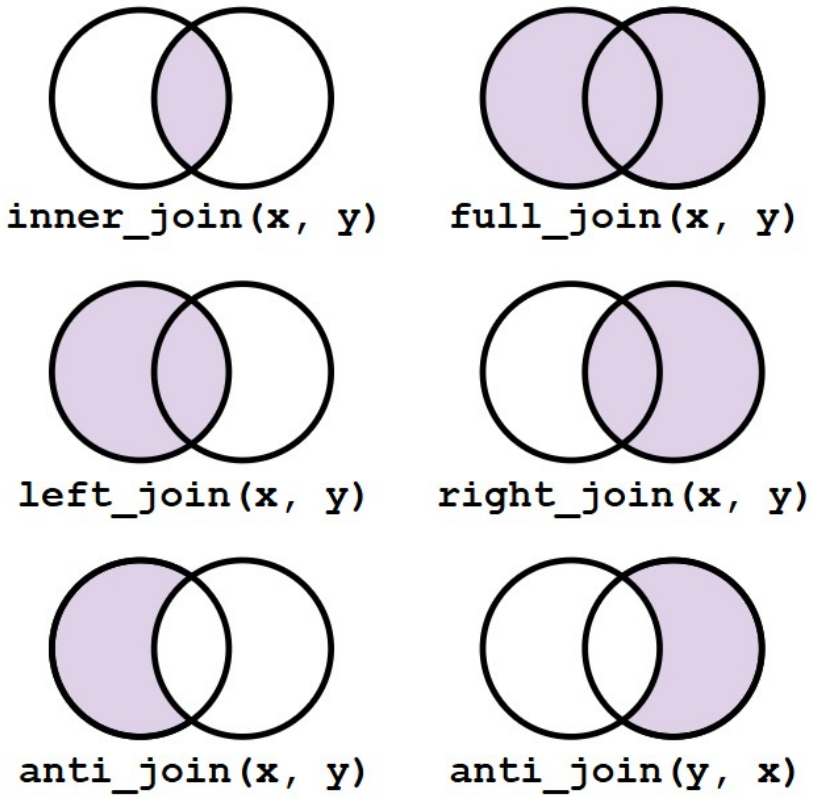


Figure 3.1: Join

```
# Example datasets
data_grade <- data.frame(
  subject_id = seq(1:nrow(covid)),     # subject id
  note = sample(1:20, nrow(covid), replace=TRUE)                  # Numeric
)

data_grade <- data_grade %>%
  mutate(is_graduate = ifelse(note>=10, TRUE, FALSE))

# Inner Join: Only matching subject_id rows
join_data <- full_join(covid, data_grade, by = "subject_id")
```

```
# Shown results
head(join_data)
```

```
## # A tibble: 6 x 23
##   subject_id fake_first_name   fake_last_name gender pan_day test_id clinic_name
##        <dbl> <chr>             <chr>          <chr>    <dbl> <chr>   <chr>
## 1       1412 jhezane           westerling     female       4 covid   inpatient ~
## 2        533 penny             targaryen      female       7 covid   clinical l~
## 3       9134 grunt             rivers         male         7 covid   clinical l~
## 4       8518 melisandre        swyft          female       8 covid   clinical l~
## 5       8967 SAWADOGO          Laurent Benja~ male         8 covid   emergency ~
## 6      11048 OUATTARA Siguiss~ Sarah          female       8 covid   oncology d~
## # i 16 more variables: result <chr>, demo_group <chr>, age <dbl>,
## #   drive_thru_ind <dbl>, ct_result <dbl>, orderset <dbl>, payor_group <chr>,
## #   patient_class <chr>, col_rec_tat <dbl>, rec_ver_tat <dbl>, age_group <chr>,
## #   age_log <dbl>, age_squarred <dbl>, age_square_root <dbl>, note <int>,
## #   is_graduate <lgl>
```

3.6 Exercises 3

Try these exercises to practice dplyr functions:

- Import the **"blood_storage"** database from the package **medicaldata**.
- Log-transform the variable *age* in data and save the result as *age.log*in the dataframe.
- Arranging the dataframe using the *age* column.
- Create a new variable called *PVol.squared* with values equal to the square all values in *PVol* (Prostate volume).
- Create a new variable called *PVol.squared_root* with values equal to the square root all values in *PVol* (Prostate volume).
- Compute the proportion of African American using the column *AA* (0 = "non-African-American"; 1 = "African American").
- Compute the proportion of African American by Tumor volume (using the variable *TVol*) and the mean and variance of the column *age*.
- Filter out the records for which *(PreopPSA >= 10)* and *(Recurrence == 0)* using the **filter** function.
- Create a new column called *subject_id* and merge the **"covid"** database with the **"blood_storage"** using the newly created column.

This lesson introduces essential dplyr functions for data manipulation in R. Try the exercises and explore the dataset further!

Chapter 4

Data Visualization with ggplot

4.1 Overview

The `ggplot2` package is one of the most powerful and flexible tools for creating complex, multi-layered graphics in **R**. It implements the Grammar of Graphics, a framework that breaks down plots into semantic components such as layers, scales, and themes.

4.2 Basic Concepts

- **Aesthetic Mappings (`aes()`)**: Defines how data variables are mapped to visual properties like color, size, and shape.
- **Geometries (`geom_*`)**: Defines the type of plot, such as points (`geom_point`), lines (`geom_line`), and bars (`geom_bar`).
- **Layers**: Multiple geometries can be added to a plot.
- **Scales and Coordinate Systems**: Allows control over appearance.
- **Themes**: Adjust non-data elements like background and grid lines.

4.3 Scatter Plot

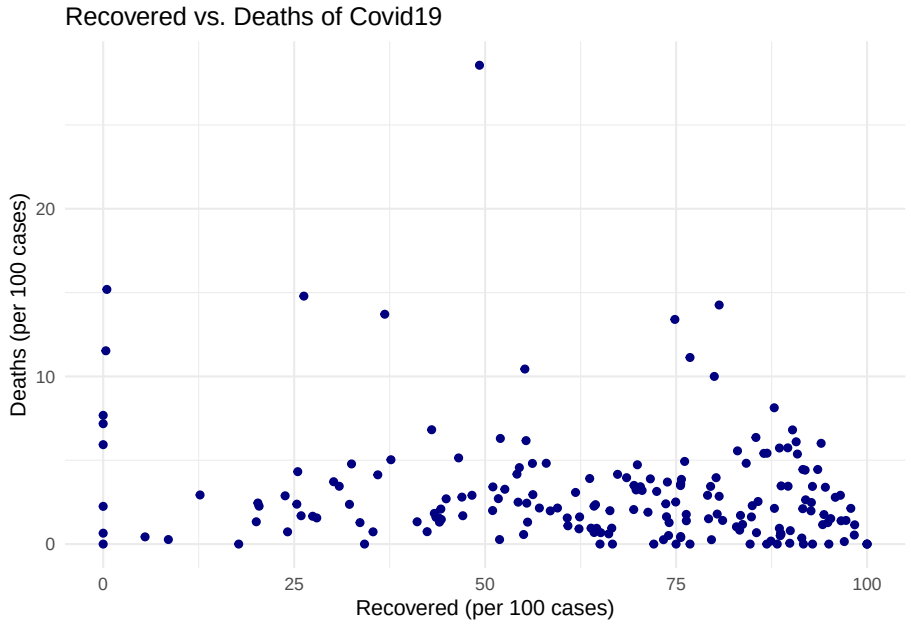
4.3.1 Basic Scatter Plot

```
# Load libraries
library(dplyr)
library(ggplot2)
library(readr)
```

```
# Load the COVID-19 dataset
covid_data <- read_csv("country_wise_latest.csv")
```

```
## Rows: 187 Columns: 15
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): Country/Region, WHO Region
## dbl (13): Confirmed, Deaths, Recovered, Active, New cases, New deaths, New r...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

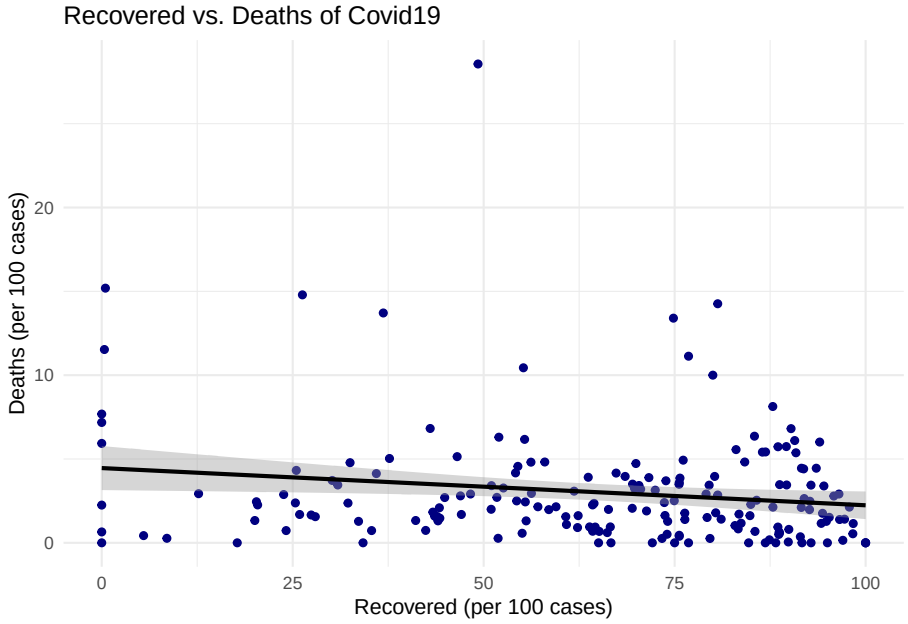
```
#View(covid_data)

# Create the scatter plot
ggplot(covid_data, # The data frame containing the COVID-19 data
       aes(x = `Recovered / 100 Cases`, # Variable for the x-axis (Recovered cases per
           y = `Deaths / 100 Cases`)) + # Variable for the y-axis (Deaths per 100 case
  geom_point(color = 'navy') + # Add points to the plot, colored navy
  labs(title = "Recovered vs. Deaths of Covid19", # Set the plot title
       x = "Recovered (per 100 cases)", # Set the x-axis label
       y = "Deaths (per 100 cases)") + # Set the y-axis label
  theme_minimal() # Use a minimal theme for a cleaner look
```



4.3.2 Scatter Plots and Line Plots

```
# Create the scatter plot with a smooth line
ggplot(covid_data, # The data frame containing the COVID-19 data
       aes(x = `Recovered / 100 Cases`, # Variable for the x-axis (Recovered cases per 100)
           y = `Deaths / 100 Cases`)) + # Variable for the y-axis (Deaths per 100 cases)
  geom_point(color = 'navy') + # Add points to the plot, colored navy
  geom_smooth(method = "lm", # Add a smooth line, using a linear model ("lm")
              color = "black", # Set the line color to black
              se = TRUE) + # Display the standard error around the line
  labs(title = "Recovered vs. Deaths of Covid19", # Set the plot title
       x = "Recovered (per 100 cases)", # Set the x-axis label
       y = "Deaths (per 100 cases)") + # Set the y-axis label
  theme_minimal() # Use a minimal theme for a cleaner look
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



4.4 Exercise 1: Customizing Your First Plot

Objective: Create a scatter plot showing the relationship between `proportion_of_active` and `proportion_of_recovery`. Customize the plot: - Create a new variable (using the function `mutate`of the package `dplyr`) named `proportion_of_active` = `Active`/`Confirmed`. - Create a new variable (using the function `mutate`of the package `dplyr`) named `proportion_of_recovery` = `Recovered`/`Confirmed`. - Change the color of

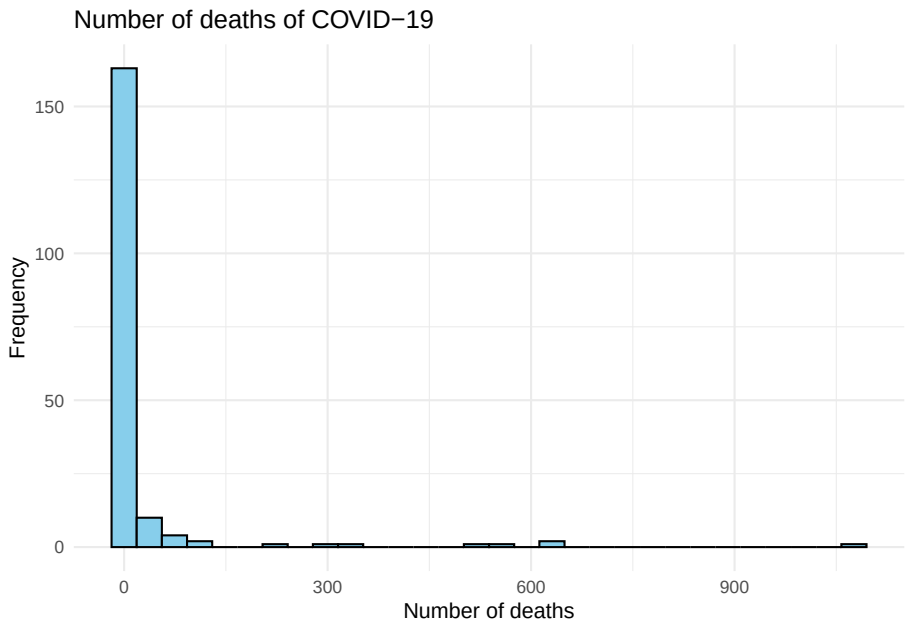
the points to “darkturquoise” (see list of color names) - Adjust the size of the points to 3. - Use triangles for the point shapes. - Add the regression line with the confidence interval estimate.

4.5 Bar Plots and Histograms

4.5.1 Histogram:

```
# Create the histogram
ggplot(covid_data, # The data frame containing the COVID-19 data
       aes(x = `New deaths`)) + # Variable for the x-axis (New deaths)
  geom_histogram(fill = "skyblue", # Fill the bars with skyblue
                 color = "black") + # Set the bar outline color to black
  labs(title = "Number of deaths of COVID-19", # Set the plot title
       x = "Number of deaths", # Set the x-axis label
       y = "Frequency") + # Set the y-axis label
  theme_minimal() # Use a minimal theme for a cleaner look
```

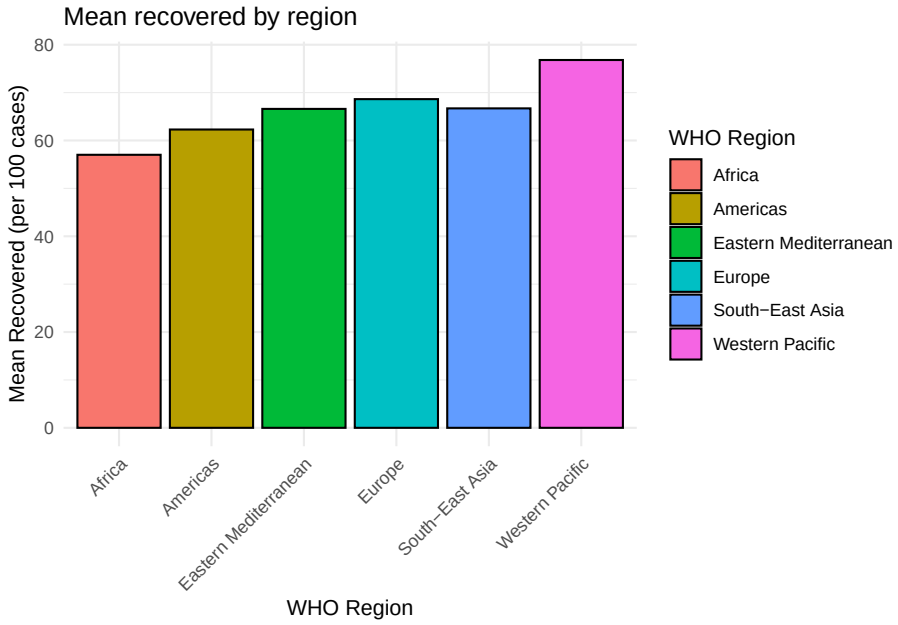
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



4.5.2 Bar Plot

```
# Calculate mean recovery rates by region and create bar plot
covid_data %>% # Start with the COVID-19 data
```

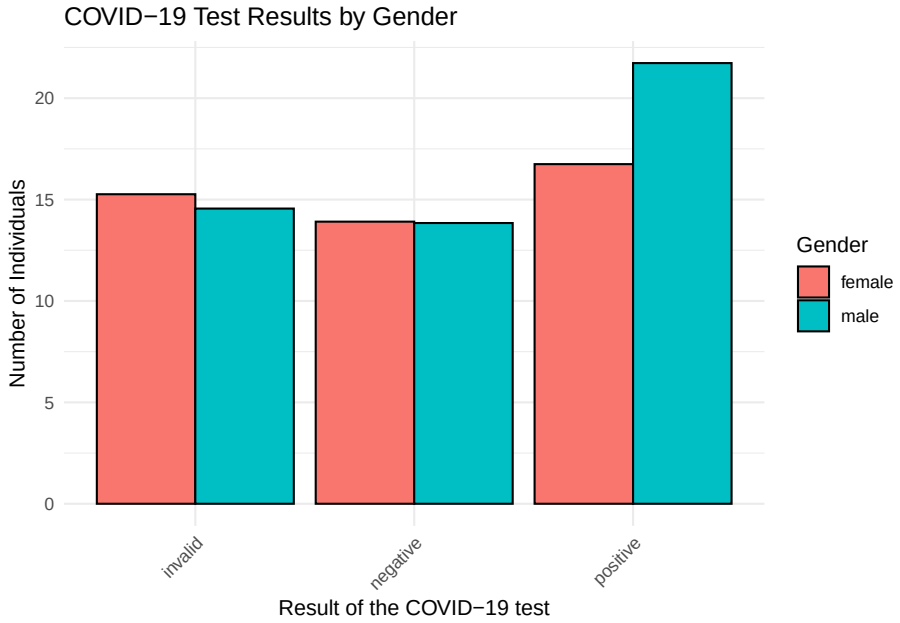
```
  group_by(`WHO Region`) %>% # Group the data by WHO Region
  summarise(mean_recovery = mean(`Recovered / 100 Cases`, na.rm = TRUE)) %>% # Calculate the mean
  ggplot(aes(y = mean_recovery, # Set the y-axis to the mean recovery rate
             x = `WHO Region`, # Set the x-axis to the WHO Region
             fill = `WHO Region`)) + # Fill the bars with different colors based on the region
  geom_bar(stat = "identity", # Use "identity" because we've already calculated the means
           color = "black") + # Set the bar outline color to black
  labs(title = "Mean recovered by region", # Set the plot title
       x = "WHO Region", # Set the x-axis label
       y = "Mean Recovered (per 100 cases)") + # Set the y-axis label
  theme_minimal() + # Use a minimal theme for a cleaner look
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) # Rotate x-axis labels for better read
```




```
library(medicaldata) # For the covid_testing dataset

# Load the covid_testing dataset
testing_data <- medicaldata::covid_testing

# Analyze test results by gender and create bar plot
testing_data %>%
  group_by(gender, result) %>% # Group the data by gender and test result
  summarise(age_moyen = mean(age, na.rm=TRUE)) %>% # Count the number of individuals in each grou
  ungroup() %>% # Ungroup the data for plotting
  ggplot(aes(y = age_moyen, # Set the y-axis to the mean age
             x = result, # Set the x-axis to the test result
```

```
            fill = gender)) + # Fill the bars with different colors based on gender
  geom_bar(stat = "identity", # Use "identity" because we've already calculated the co
           color = "black", # Set the bar outline color to black
           position = "dodge") + # Place bars for different genders side-by-side
  labs(title = "COVID-19 Test Results by Gender", # Set the plot title
       x = "Result of the COVID-19 test", # Set the x-axis label
       y = "Number of Individuals", # Set the y-axis label
       fill = "Gender") + # Set the legend title
  theme_minimal() + # Use a minimal theme for a cleaner look
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) # Rotate x-axis labels for
```

```
## `summarise()` has grouped output by 'gender'. You can override using the
## `.groups` argument.
```



4.6 Exercise 3: Creating Histograms and Bar Plots

- Import the **blood_storage** database from the **medicaldata** package.
- Log-transform the variable **age** in the data and save the result as **age.log** in the dataframe.
- Compare the histograms of the **age** and **age.log** variables and comment on the distribution.
- Create a bar plot of the mean **PVol** by aggregating by race (use the **AA** variable).

- Create a bar plot of the mean **PVol** by aggregating by the volume of the tumor (use the **TVol} variable).

Bibliography

Mathias Harrer, Pim Cuijpers, Lea Schuurmans, Tim Kaiser, Claudia Buntrock, Annemieke van Straten, and David Daniel Ebert. *Evaluation of randomized controlled trials: a primer and tutorial for mental health researchers.*, volume 24. Trials, 2023. doi: 10.1186/s13063-023-07596-3. URL https://rct-tutorial.mharrer.dev/#introduction.