Statistical Analysis and Visualizations Using R

Okan Bulut

University of Alberta

Contents

Overview **1**

Course Description 1

Course Objectives 2

Instructor Information 2

Course Structure 2

Course Materials 3

Learning Process 3

Additional Resources 4

Introduction **5**

R and RStudio 5

Basics of the R Language 13

Data Wrangling **17**

Creating Datasets in **R** 17

Importing Data into **R** 19

Understanding the Data 22

Indexing 24

Subsetting 25

Other Data Manipulation Tools 28

Descriptive Statistics **31**

Quick Summary 31

Frequency Tables 31

Central Tendency and Dispersion 34

Data Visualizations in R **39**

Base **R** Graphics 39

`ggplot2` Graphics 47

Hypothesis Testing **55**

Some Theory 55

Types of Inferential Statistics 56

One-Sample *t* Test 56

Independent-Samples *t* Test 58

t-test with Paired Data 61

Analysis of Variance (ANOVA) 63

Exercise 9 66

Additional Packages to Consider 67

Correlation and Regression **69**

Correlation 69

Simple Linear Regression 73

Multiple Regression 78

Exercise 10 82

Additional Packages to Consider 83

Additional R Features **87**

Removing Objects from the Workspace 87

Saving Output in R 87

Unloading a Package 89

Updating **R** and **R** Packages 89

References **91**

Overview

Figure 1: Source: https://unsplash.com/photos/DErxVSSQNdM

Course Description

Welcome to Statistical Analysis and Visualizations Using R at the Technology Training Centre. **R** (R Core Team 2021) is a free and open-source programming language that allows users to access a wide range of statistical and graphical tools. Over the last decade, **R** has become one of the most widely used statistical software programs among researchers and practitioners around the world due to its growing capabilities through user-created, free packages.

This full-day course is intended to provide participants with a hands-on training in exploring, visualizing, and analyzing data using the R programming language. To control **R**, participants will use RStudio, which is a free, user-friendly program with a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting.

Course Objectives

Upon successfully completing this course, participants will be able to:

- understand the basics of the R programming language
- perform steps to manage different types of data
- execute data preparation steps
- visualize data with various types of variables
- compute descriptive statistics
- compute inferential statistics using **R**

Instructor Information

Okan Bulut – University of Alberta

- Associate Professor of educational measurement and psychometrics at the University of Alberta
- 10+ years using **R** for statistical data analysis and visualization
- Specialized in the analysis and visualization of big data (mostly from large-scale assessments)
- 8+ years teaching courses and workshops on statistics, psychometrics, and programming with R
- **Website:** https://sites.ualberta.ca/~bulut/
- **E-mail:** bulut@ualberta.ca

I also co-authored:

- Three **R** packages:
  - profileR for profile analysis of multivariate data
  - hemp for psychometric analysis of assessment data
  - eirm for explanatory item response modeling
- A recent book called Handbook of Educational Measurement and Psychometrics Using R

Course Structure

This course will introduce participants to statistical and data science procedures widely used in social sciences, public health, and other similar areas. Four aspects of statistical reasoning will be emphasized:

1. data wrangling
2. data visualization
3. univariate statistical methods
4. computer applications using **R**

During the course, we will use the following schedule:

Part	Description
1	**Introduction (9:00-9:30)**
	Overview of R and RStudio
	Basics of R language
2	**Data Wrangling (9:30-10:30)**
	Creating/importing and managing data
	Data manipulation
3	**Descriptive Statistics (10:30-12:00)**
	Frequency distributions, Graphical tools
	Central tendency and dispersion
Break (12:00-13:00)	
4	**Hypothesis Testing (13:00-14:30)**
	Overview of hypothesis testing, t-tests
	Analysis of variance (ANOVA)
5	**Correlation and Regression (14:30-16:00)**
	Correlations for different types of variables
	Simple and multiple linear regression

Course Materials

Participants will find copies of the course materials in the computers that they will be using. In addition, participants can access these materials online:

- To view the online course notes: https://okanbulut.github.io/rbook/
- To view and download other course materials (e.g., dataset, cheatsheets): https://github.com/okanbulut/rbook

Learning Process

Learning how to use **R** is just like learning a new language to speak. So, it might be a bit overwhelming at the beginning. Therefore, I strongly recommend you to ask all of your questions while we go over today's materials. Collaboration between the training participants is also highly recommended!

Additional Resources

There are **many** resources (e.g., websites and books) on statistical data analysis using **R** on the Internet. A brief list of such resources are shown below:

Websites:

- An Introduction to R: https://cran.r-project.org/doc/manuals/R-intro.pdf
- Using R for Introductory Statistics: https://goo.gl/owJbLg
- Quick R: http://www.statmethods.net/index.html
- R Cookbook: http://www.cookbook-r.com/

Online training:

- Coursera: https://www.coursera.org/learn/r-programming
- DataCamp: https://www.datacamp.com/courses/free-introduction-to-r
- And tons of free videos on YouTube!!!

Books:

- R for Data Science: https://r4ds.had.co.nz/
- OpenIntro Statistics: https://openintro-ims.netlify.app/

Introduction

R and RStudio

What is R?

R . . .

- is a free, open source program for statistical computing and data visualization.
- is cross-platform (e.g., available on Windows, Mac OS, and Linux).
- is maintained and regularly updated by the Comprehensive R Archive Network (CRAN).
- is capable of running all types of statistical analyses.
- has amazing visualization capabilities (high-quality, customizable figures).
- enables reproducible research.
- has many other capablities, such as web programming.
- supports user-created packages (currently, more than 10,000)

What is RStudio?

RStudio . . .

- is a free program available to control **R**.
- provides a more user-friendly interface for **R**.
- includes a set of tools to help you be more productive with **R**, such as:
  - A syntax-highlighting editor for highlighting your **R** codes
  - Functions for helping you type the **R** codes (auto-completion)
  - A variety of tools for creating and saving various plots (e.g., histograms, scatterplot)
  - A workspace management tool for importing or exporting data

Download and Install

To benefit from **RStudio**, both **R** and **RStudio** should be installed in your computer. **R** and **RStudio** are freely available from the following websites:

To download and install **R**:

1. Go to https://cran.r-project.org/
2. Click "Download R for Mac/Windows""
3. Download the appropriate file: • Windows users click Base, and download the installer for the latest R version • Mac users select the file R-3.X.X.pkg that aligns with your OS version
4. Follow the instructions of the installer.

To download and install **RStudio**:

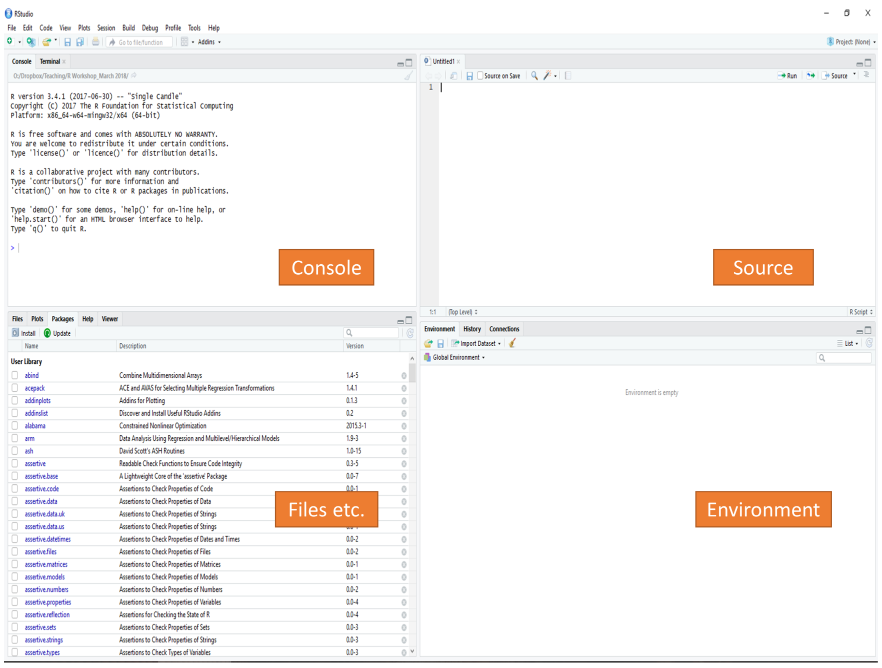
1. Go to https://www.rstudio.com/products/rstudio/download/
2. Click "Download" under *RStudio Desktop - Open Source License*
3. Select the install file for your OS
4. Follow the instructions of the installer.

Preview of RStudio

After you open RStudio, you should see the following screen:

I personally prefer console on the top-left, source on the top-right, files on the bottom-left, and environment on the bottom-right. The pane layout can be updated using *Global Options* under *Tools.*

We can also change the appearance (e.g., code highlighting, font type, font size, etc.):

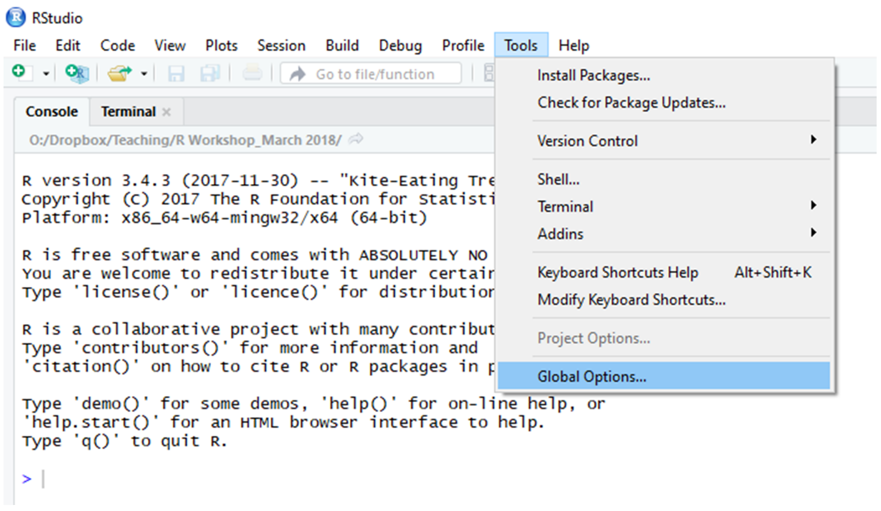


```
R version 3.4.1 (2017-06-30) -- "Single Candle"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

User Library

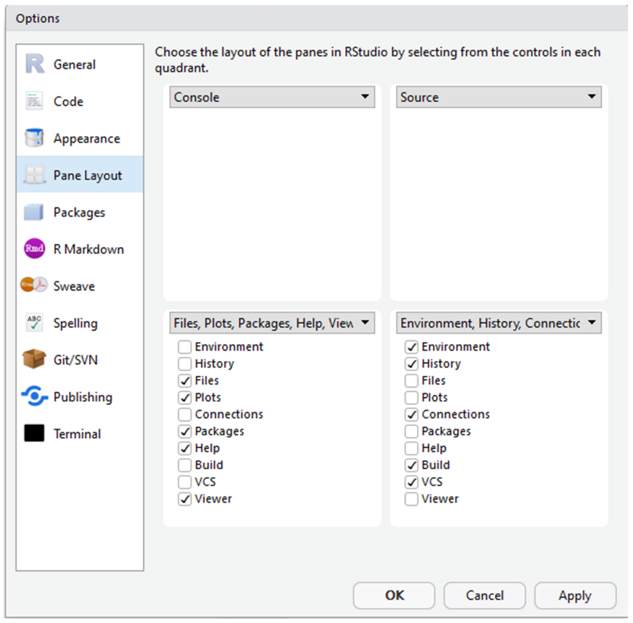
Name	Description	Version
abind	Combine Multidimensional Arrays	1.4-5
acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
addinplots	Addins for Plotting	0.1.3
addinslist	Discover and Install Useful RStudio Addins	0.2
alabama	Constrained Nonlinear Optimization	2015.3-1
arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.9-3
ash	David Scott's ASH Routines	1.0-15
assertive	Readable Check Functions to Ensure Code Integrity	0.3-5
assertive.base	A Lightweight Core of the 'assertive' Package	0.0-7
assertive.code	Assertions to Check Properties of Code	0.0-1
assertive.data	Assertions to Check Properties of Data	[illegible]
assertive.data.uk	Assertions to Check Properties of Strings	[illegible]
assertive.data.us	Assertions to Check Properties of Strings	[illegible]
assertive.datetimes	Assertions to Check Properties of Dates and Times	0.0-2
assertive.files	Assertions to Check Properties of Files	0.0-2
assertive.matrices	Assertions to Check Properties of Matrices	0.0-1
assertive.models	Assertions to Check Properties of Models	0.0-1
assertive.numbers	Assertions to Check Properties of Numbers	0.0-2
assertive.properties	Assertions to Check Properties of Variables	0.0-4
assertive.reflection	Assertions for Checking the State of R	0.0-4
assertive.sets	Assertions to Check Properties of Sets	0.0-3
assertive.strings	Assertions to Check Properties of Strings	0.0-3
assertive.types	Assertions to Check Types of Variables	0.0-3

Figure 2: Opening screen of RStudio


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Console Terminal
O:/Dropbox/Teaching/R Workshop_March 2018/
Install Packages...
Check for Package Updates...
Version Control
Shell...
Terminal
Addins
Keyboard Shortcuts Help Alt+Shift+K
Modify Keyboard Shortcuts...
Project Options...
Global Options...

```
R version 3.4.3 (2017-11-30) -- "Kite-Eating Tre
Copyright (C) 2017 The R Foundation for Statisti
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO
You are welcome to redistribute it under certair
Type 'license()' or 'licence()' for distributior

R is a collaborative project with many contribut
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in p

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Figure 3: Click Tools and then select Global Options



Choose the layout of the panes in RStudio by selecting from the controls in each quadrant.

Console

Source

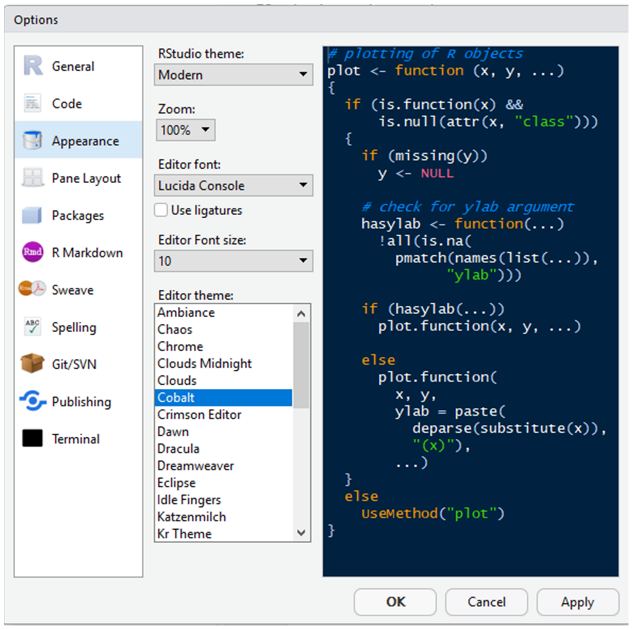
Files, Plots, Packages, Help, View

- [ ] Environment
- [ ] History
- [✓] Files
- [✓] Plots
- [ ] Connections
- [✓] Packages
- [✓] Help
- [ ] Build
- [ ] VCS
- [✓] Viewer

Environment, History, Connectic

- [✓] Environment
- [✓] History
- [ ] Files
- [ ] Plots
- [✓] Connections
- [ ] Packages
- [ ] Help
- [✓] Build
- [✓] VCS
- [ ] Viewer

Figure 4: Select console, source, environment, or files for each pane



RStudio theme: Modern

Zoom: 100%

Editor font: Lucida Console

- [ ] Use ligatures

Editor Font size: 10

Editor theme: Cobalt

```
# plotting of R objects
plot <- function (x, y, ...)
{
  if (is.function(x) &&
      is.null(attr(x, "class")))
  {
    if (missing(y))
      y <- NULL

    # check for ylab argument
    hasylab <- function(...)
      !all(is.na(
        pmatch(names(list(...)),
            "ylab")))

    if (hasylab(...))
      plot.function(x, y, ...)

    else
      plot.function(
        x, y,
        ylab = paste(
          deparse(substitute(x)),
          "(x)"),
        ...)
  }
  else
    UseMethod("plot")
}
```

Figure 5: Change the Appearance Settings

Note: To get yourself more familiar with **RStudio**, I recommend you to check out the **RStudio** cheatsheet and Oscar Torres-Reyna's nice tutorial (*Note:* You can click on these links to open and download the documents or see https://github.com/okanbulut/rbook/tree/master/cheatsheets).

Creating a New Script

In **R**, we can type our commands in the console; but once we close **R**, everything we have typed will be gone. Therefore, we should create an empty script, write the codes in the script, and save it for future use. We can replicate the exact same analysis and results by running the script again later on. The **R** script file has the .R extension, but it is essentially a text file. Thus, any text editor (e.g., Microsoft Word, Notepad, TextPad) can be used to open a script file for editing outside of the **R** environment.

We can create a new script file in **R** as follows:

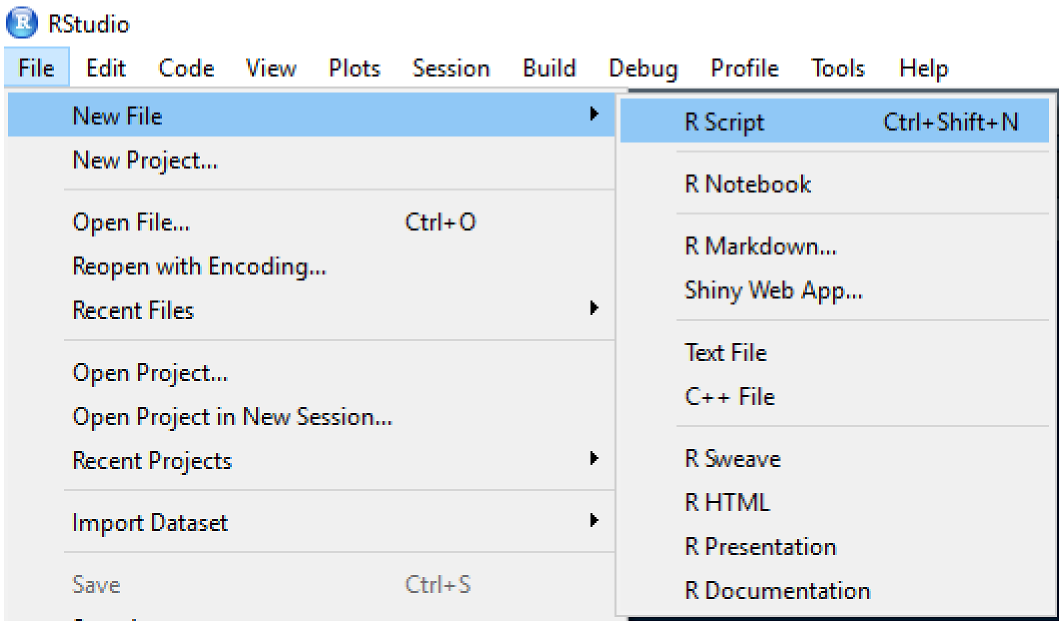
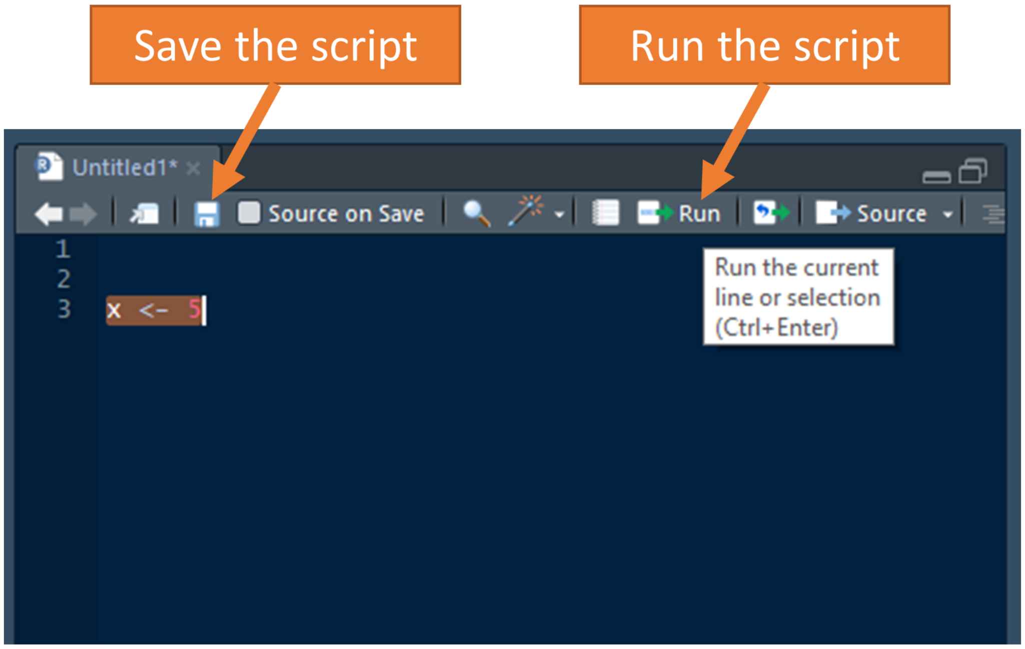


Figure 6: Creating a new script in **R** (using **RStudio**)

When we type some codes in the script, we can select the lines we want to run and then hit the run button. Alternatively, we can bring the cursor at the beginning of the line and hit the run button which runs one line at a time and moves to the next line.

Working Directory

An important feature of **R** is "working directory," which refers to a location or a folder in your computer where you keep your **R** script, your data files, etc. Once we define a working directory in **R**, any data file or script within that directory can be easily imported into **R**



```
x <- 5
```

Figure 7: Running **R** codes from a script file

without specifying where the file is located. By default, **R** chooses a particular location in your computer (typically Desktop or Documents) as your working director. To see our current working director, we need to run a `getwd()` command in the **R** console:

```
getwd()
```

This will return a path like this:

```
## [1] "C:/Users/bulut/Desktop"
```

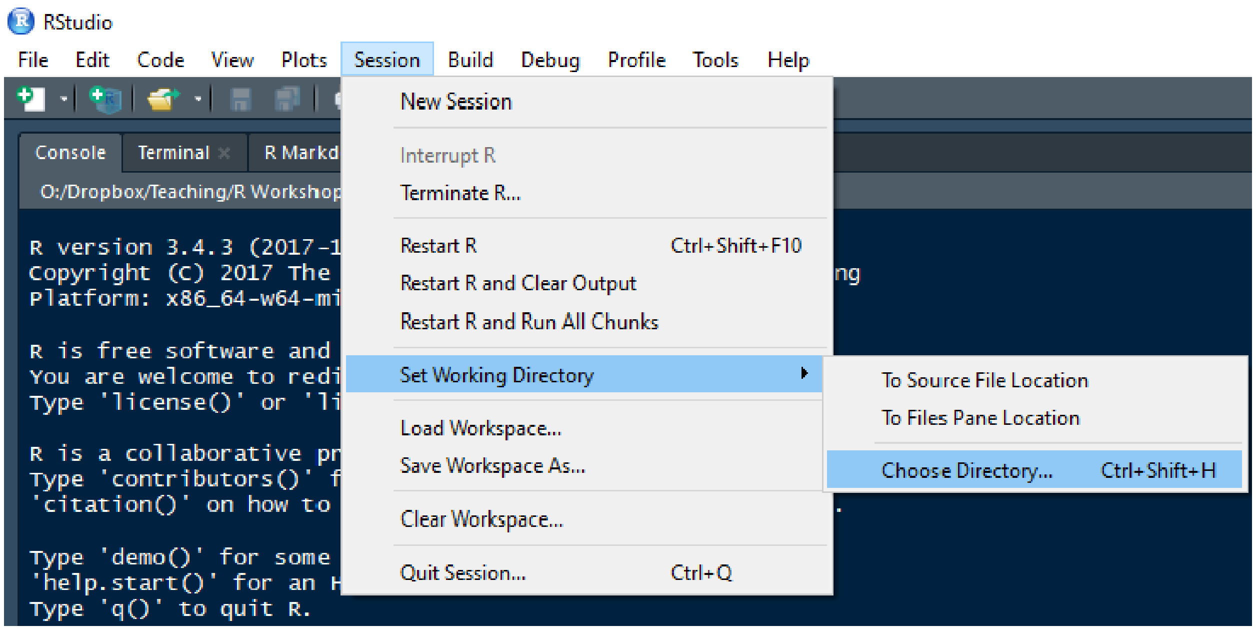
Once we decide to change the current working direcory into a different location, we can do it in two ways:

Method 1: Using the "Session" options menu in **RStudio**

We can select Session > Set Working Directory > Choose Directory to find a folder or location that we want to set as our current working directory.

Method 2: Using the `setwd` command in the console

Tpying the following code in the console will set the "R workshop" folder on my desktop as the working directory. If the folder path is correct, **R** changes the working directory without giving any error messages in the console.



```
R version 3.4.3 (2017-1
Copyright (C) 2017 The
Platform: x86_64-w64-mi

R is free software and
You are welcome to redi
Type 'license()' or 'li

R is a collaborative pr
Type 'contributors()' f
'citation()' on how to

Type 'demo()' for some
'help.start()' for an H
Type 'q()' to quit R.
```

Figure 8: **Method 1:** Setting the working directory in **R**

```
setwd("C:/Users/bulut/Desktop/R workshop")
```

To ensure that the working directory is properly set, we can use the `getwd()` command again:

```
getwd()
## [1] "C:/Users/bulut/Desktop/R workshop"
```

IMPORTANT: R does not accept any backslashes in the file path. Instead of a backslash, we need to use a frontslash. This is particulary important for Windows computers since the file paths involve backslashes (Mac OS X doesn't have this problem).

Downloading and Installing R Packages

The base **R** program comes with many built-in functions to compute a variety of statistics and to create graphics (e.g., histograms, scatterplots, etc.). However, what makes **R** more powerful than other software programs is that **R** users can write their own functions, put them in a package, and share it with other **R** users via the CRAN website.

For example, `ggplot2` (Wickham et al. 2020) is a well-known **R** package, created by Hadley Wickham and Winston Chang. This package allows **R** users to create elegant data visualizations. To download and install the `ggplot2` package, we need to use the `install.packages` command. Note that your computer has to be connected to the internet to be able to connect to the CRAN website and download the package.

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

Once a package is downloaded and installed, it is permanently in your **R** folder. That is, there is no need to re-install it, unless you remove the package or install a new version of R. These downloaded packages are not directly accessible until we activate them in your **R** session. Whenever we need to access a package in **R**, we need to use the `library` command to activate it. For example, to access the `ggplot2` package, we would use:

```r
library("ggplot2")
```

To get help on installed packages (e.g., what's inside this package):

```r
# To get details regarding contents of a package
help(package = "ggplot2")

# To list vignettes available for a specific package
vignette(package = "ggplot2")

# To view specific vignette
vignette("ggplot2-specs")
```

Exercise 1

1. Open **RStudio** and set the folder that you have the training materials (it's called *rtraining*) as your working directory using either the `setwd` command or the Session options menu in **RStudio**.

2. Open the **R** script file called *ttc-r-course* in the rtraining folder. You can open the script by either double-clicking on the file (so **RStudio** opens it automatically) or using "File" and "Open file" in **RStudio**.

3. Install and activate the `lattice` package using the `install.packages` and `library` commands. The `lattice` package (Sarkar 2020) is another well-known package for data visualization in **R**. You should type the following in your script file, choose all the lines, and hit the run button.

```r
install.packages("lattice")
library("lattice")
```

Basics of the R Language

Creating New Variables

To create a new variable in R, we use the assignment operator, `<-`. To create a variable `x` that equals 25, we need to type:

```
x <- 25
```

If we want to print `x`, we just type `x` in the console and hit enter. **R** returns the value assigned to `x`.

```
x
```

```
[1] 25
```

We can also create a variable that holds multiple values in it, using the `c` command (**c** stands for *combine*).

```
weight <- c(60, 72, 80, 84, 56)
weight
```

```
[1] 60 72 80 84 56
```

```
height <- c(1.7, 1.75, 1.8, 1.9, 1.6)
height
```

```
[1] 1.70 1.75 1.80 1.90 1.60
```

Once we create a variable, we can do further calculations with it. Let's say we want to transform the `weight` variable (in kg) to a new variable called weight2 (in lbs).

```
weight2 <- weight * 2.20462
weight2
```

```
[1] 132.3 158.7 176.4 185.2 123.5
```

Note that we named the variable as `weight2`. So, both `weight` and `weight2` exist in the active **R** session now. If we used the following, this would overwrite the existing `weight` variable.

```r
weight <- weight * 2.20462
```

We can also define a new variable based on existing variables.

```r
reading <- c(80, 75, 50, 44, 65)
math <- c(90, 65, 60, 38, 70)
total <- reading + math
total
```

```
[1] 170 140 110  82 135
```

Sometimes we need a variable that holds character strings rather than numerical values. If a value is not numerical, we need to use double quotation marks. In the example below, we create a new variable called `cities` that has four city names in it. Each city name is written with double quotation marks.

```r
cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")
cities
```

```
[1] "Edmonton"     "Calgary"      "Red Deer"     "Spruce Grove"
```

We can also treat numerical values as character strings. For example, assume that we have a `gender` variable where 1=Male and 2=Female. We want R to know that these values are not actual numbers; instead, they are just numerical labels for gender groups.

```r
gender <- c("1", "2", "2", "1", "2")
gender
```

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

Important Rules for the R Language

Here is a list of important rules for using the **R** language more effectively:

1. **Case-sensitivity:** **R** codes written in lowercase would NOT refer to the same codes written in uppercase.

```r
cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")
Cities
CITIES

Error: object 'Cities' not found
Error: object 'CITIES' not found
```

2. **Variable names:** A variable name **cannot** begin with a number or include a space.

```
4cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")
my cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")

Error: unexpected symbol in "4cities"
Error: unexpected symbol in "my cities"
```

3. **Naming conventions:** I recommend using consistent and clear naming conventions to keep the codes clear and organized. I personally prefer all lowercase with underscore (e.g., `my_variable`). The other naming conventions are:

- All lowercase: e.g. `mycities`
- Period.separated: e.g. `my.cities`
- Underscore_separated: e.g. `my_cities`
- Numbers at the end: e.g. `mycities2018`
- Combination of some of these rules: `my.cities.2018`

4. **Commenting:** The hashtag symbol (#) is used for commenting in **R**. Any words, codes, etc. coming after a hashtag are just ignored. I strongly recommend you to use comments throughout your codes. These annotations would remind you what you did in the codes and why you did it that way. You can easily comment out a line without having to remove it from your codes.

```
# Here I define four cities in Alberta
cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")
```

Self-Help

In the spirit of open-source, **R** is very much a self-guided tool. We can look for solutions to **R**-related problems in multiple ways:

1. Use the `?` to open help pages for functions or packages (e.g., try `?summary` in the console to see how the `summary` function works)

2. For tricky questions and funky error messages (there are many of these), use Google (include "in R" to the end of your query)

3. We can also use RSeek (https://rseek.org/) - a search engine just for **R**

4. StackOverflow (https://stackoverflow.com/) has become a great resource with many questions for many specific packages in **R**, and a rating system for answers

Exercise 2

1. Create two new variables `age` and `salary` for five persons:

- `age`: 21, 24, 32, 45, 52
- `salary`: 4500, 3500, 4100, 4700, 6000

2. Then, type the following code in your script and run it to find the correlation between `age` and `salary`:

```
cor(age, salary)
```

Data Wrangling

Nearly all datasets require some initial procedures (e.g., cleaning, reformatting, reshaping) to be applied before we start running any statistical analysis or creating visualizations. These procedures are often referred to as **data wrangling**. Here is a nice summary of the data wrangling process:

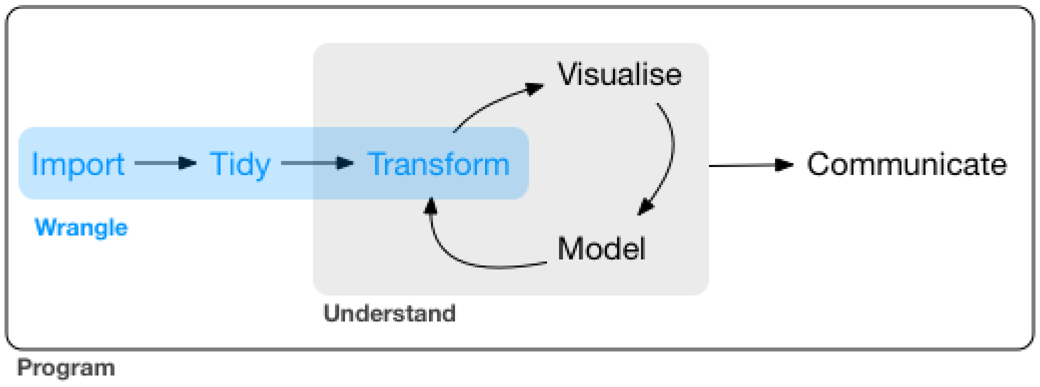


Figure 9: Data wrangling process [**Source:** Grolemund and Wickham (2018)]

In this section, we will follow the steps of data wrangling as shown above.

Creating Datasets in R

There are multiple ways of creating datasets in R. We can create individual variables and combine them using the `cbind` (column bind) command:

```
age <- c(21, 24, 32, 45, 52)
salary <- c(4500, 3500, 4100, 4700, 6000)
mydata <- cbind(age, salary)
mydata
```

```
     age salary
[1,]  21   4500
[2,]  24   3500
```

```
[3,]  32   4100
[4,]  45   4700
[5,]  52   6000
```

We can also create individual rows and combine them using the `rbind` (row bind) command (though this is not practical if there are many rows):

```r
person1 <- c(21, 4500)
person2 <- c(24, 3500)
person3 <- c(32, 4100)
person4 <- c(45, 4700)
person5 <- c(52, 6000)

mydata <- rbind(person1, person2, person3, person4, person5)
mydata
```

```
        [,1] [,2]
person1   21 4500
person2   24 3500
person3   32 4100
person4   45 4700
person5   52 6000
```

A better way to create datasets in **R** is to define variables within a data frame using the `data.frame` command.

```r
mydata <- data.frame(age = c(21, 24, 32, 45, 52),
                     salary = c(4500, 3500, 4100, 4700, 6000))
mydata
```

```
  age salary
1  21   4500
2  24   3500
3  32   4100
4  45   4700
5  52   6000
```

Data frames in **R** are very convenient because many mathematical operations can be directly applied to a data frame or some columns (or rows) of a data frame. Once a data frame is defined in **R**, we can see its content using the `View` command (which open the data window) or the `head` command (which prints the first six rows of the data):

```
# To print the first six rows of a data frame
head(mydata)

# To see the entire data in the view window
View(mydata)
```

Importing Data into R

We often save our data sets in convenient data formats, such as Excel, SPSS, or text files (.txt, .csv, .dat, etc.). **R** is capable of importing (i.e., reading) various data formats.

There are two ways to import a data set into **R**:

1. By using the "Import Dataset" menu option in **RStudio**
2. By using a particular **R** command

Method 1: Using RStudio

Importing Excel Files

- Browse for the file that you want to import
- Give a name for the data set
- Choose the sheet to be imported
- "First Row as Names" if the variable names are in the first row of the file.

Importing SPSS Files

- Browse the file that you want to import
- Give a name for the data set
- Choose the SPSS data format (SAV)

Method 2: Using R Commands

R has some built-in functions, such as `read.csv` and `read.table`. Also, there are **R** packages for importing specific data formats. For example, `foreign` for SPSS files and `xlsx` for Excel files. Here are some examples:

Excel Files:

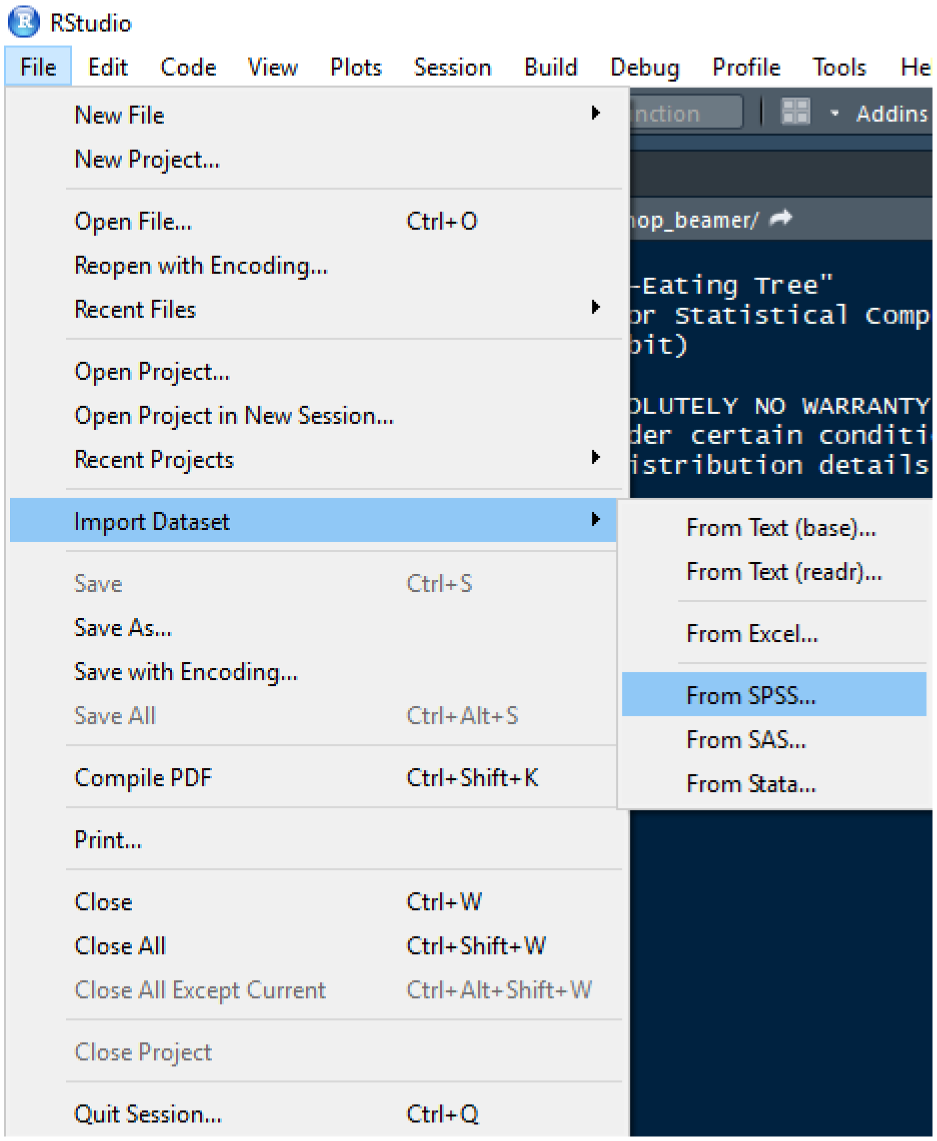
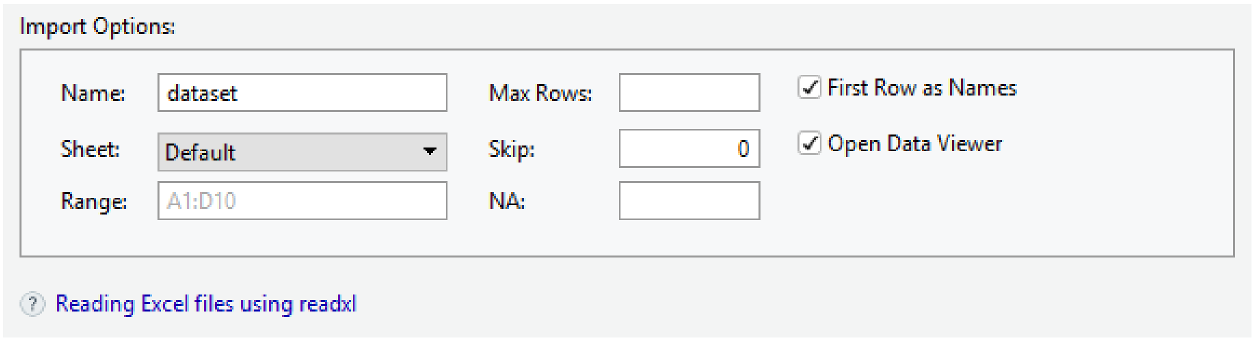


Figure 10: Importing a dataset using the **RStudio** menu

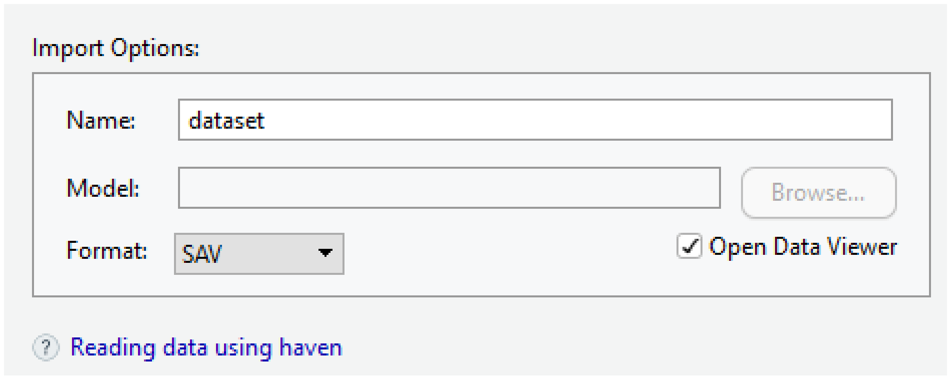


Import Options:

Name:	dataset	Max Rows:		✓ First Row as Names
Sheet:	Default	Skip:	0	✓ Open Data Viewer
Range:	A1:D10	NA:		

Reading Excel files using readxl

Figure 11: Importing Excel files


Import Options:

Name: dataset

Model:

Format: SAV

✓ Open Data Viewer

Figure 12: Importing SPSS files

```
# Install and activate the package first
install.packages("xlsx")
library("xlsx")

# Use read.xlsx to import an Excel file
my_excel_file <- read.xlsx("path to the file/filename.xlsx", sheetName = "sheetname")
```

SPSS Files:

```
# Install and activate the package first
install.packages("foreign")
library("foreign")

# Use read.spss to import an SPSS file
my_spss_file <- read.spss("path to the file/filename.sav", to.data.frame = TRUE)
```

Text Files:

```
# No need to install any packages
# R has many built-in functions already

# A comma-separated-values file with a .csv extension
my_csv_file <- read.csv("path to the file/filename.csv", header = TRUE)

# A tab delimited text file with .txt extension
my_txt_file <- read.table("path to the file/filename.txt", header = TRUE, sep = "\t")
```

Here we should note that:

- `header = TRUE` if the variable names are in the first row; otherwise, use `header = FALSE`
- `sep="\t"` for tab-separated files; `sep=","` for comma-separated files

Exercise 3

Now we will import the `medical` dataset into **R**. The dataset comes from a clinical study. Patients with no primary care physician were randomized to receive a multidisciplinary assessment and a brief motivational intervention, with the goal of linking them to primary medical care. You can find the details in "Codebook for the medical Dataset" in your folder. Our dataset is in a .csv format (`medical.csv`).

1. Import the file `medical.csv` by using the `read.csv` command and save it as `medical` using the following code (assuming that the file is in your working directory):

```
medical <- read.csv("medical.csv", header = TRUE)
```

2. Once the data file is successfully imported, run the following to see the first six rows of the data:

```
head(medical)
```

You should be able to see the output below:

```
  id age    sex  race homeless substance avg_drinks max_drinks suicidal treat
1  1  37   male black  housed   cocaine          13         26      yes   yes
2  2  37   male white homeless  alcohol          56         62      yes   yes
3  3  26   male black  housed    heroin           0          0       no    no
4  4  39 female white  housed    heroin           5          5       no    no
5  5  32   male black homeless  cocaine          10         13       no    no
6  6  47 female black  housed   cocaine           4          4       no   yes
  physical1 mental1 depression1 physical2 mental2 depression2
1     58.41  25.112          49     54.23   52.23           7
2     36.04  26.670          30     59.56   41.73          11
3     74.81   6.763          39     58.46   56.77          14
4     61.93  43.968          15     46.61   14.66          44
5     37.35  21.676          39     31.42   40.67          26
6     46.48  55.509           6     43.20   50.06          23
```

Understanding the Data

After we import a dataset into **R**, we can quickly check a few things to understand our dataset better:

- To see the number of rows in the data:

```
nrow(medical)
```

```
[1] 246
```

- To see the number of columns in the data:

```
ncol(medical)
```

```
[1] 16
```

- To see its dimensions all together:

```
dim(medical)
```

```
[1] 246  16
```

- To see all of the variable names in the data:

```
names(medical)
```

```
 [1] "id"           "age"          "sex"          "race"         "homeless"
 [6] "substance"    "avg_drinks"   "max_drinks"   "suicidal"     "treat"
[11] "physical1"    "mental1"      "depression1" "physical2"    "mental2"
[16] "depression2"
```

- To see the structure of the entire dataset:

```
str(medical)
```

```
'data.frame':   246 obs. of  16 variables:
 $ id         : int  1 2 3 4 5 6 8 9 10 12 ...
 $ age        : int  37 37 26 39 32 47 28 50 39 58 ...
 $ sex        : chr  "male" "male" "male" "female" ...
 $ race       : chr  "black" "white" "black" "white" ...
 $ homeless   : chr  "housed" "homeless" "housed" "housed" ...
 $ substance  : chr  "cocaine" "alcohol" "heroin" "heroin" ...
 $ avg_drinks : int  13 56 0 5 10 4 12 71 20 13 ...
 $ max_drinks : int  26 62 0 5 13 4 24 129 27 13 ...
 $ suicidal   : chr  "yes" "yes" "no" "no" ...
 $ treat      : chr  "yes" "yes" "no" "no" ...
 $ physical1  : num  58.4 36 74.8 61.9 37.3 ...
 $ mental1    : num  25.11 26.67 6.76 43.97 21.68 ...
 $ depression1: int  49 30 39 15 39 6 32 50 46 49 ...
 $ physical2  : num  54.2 59.6 58.5 46.6 31.4 ...
 $ mental2    : num  52.2 41.7 56.8 14.7 40.7 ...
 $ depression2: int  7 11 14 44 26 23 18 33 37 8 ...
```

Indexing

In **R**, each row and column is indexed by the position they appear in the data. **R** uses square brackets for indexing. Within the square brackets, the first number shows the row number(s) and the second number shows the column(s). To call a particular column (i.e., variables) or a particular row (i.e., persons), we can use the following structure: `data[row, col]`

For example, if we want to see the second variable for the fifth person in the `medical` dataset:

```
medical[5, 2]
```

```
[1] 32
```

Or, if we want to see the first three variables for the first five persons:

```
medical[1:5, 1:3]
```

```
  id age    sex
1  1  37   male
2  2  37   male
3  3  26   male
4  4  39 female
5  5  32   male
```

Instead of `medical[1:5, 1:3]`, we could also do:

```
medical[c(1, 2, 3, 4, 5), c(1, 2, 3)]
```

```
  id age    sex
1  1  37   male
2  2  37   male
3  3  26   male
4  4  39 female
5  5  32   male
```

or

```
medical[c(1, 2, 3, 4, 5), c("id", "age", "sex")]
```

```
  id age    sex
1  1  37   male
2  2  37   male
3  3  26   male
4  4  39 female
5  5  32   male
```

A common way of indexing variables (i.e., columns) in **R** is to use the dollar sign with a variable name from a data frame. For example, we can select the age variable as follows:

```
medical$age
```

This would print all the values for age in the `medical` dataset. We can also preview a particular variable using the `head` function.

```
head(medical$age)
```

```
[1] 37 37 26 39 32 47
```

Using a particular variable, we can also see the values for some rows in the data. For example, let's print the age variable for the 10th to 15th rows in the `medical` dataset.

```
medical$age[10:15]

# or

medical$age[c(10, 11, 12, 13, 14, 15)]
```

Note that now the brackets don't need a comma inside as we had before. This is because we have already selected a variable (age) and so **R** knows that we now refer to rows when we type any values inside the brackets.

Subsetting

Now assume that we want to create a new dataset with only females from the `medical` dataset. Although we can subset the data in many ways, the following two are the easiest:

1. Using the `subset` function in base **R**:

```
medical_female <- subset(medical, sex == "female")
head(medical_female)
```

```
   id age    sex     race homeless substance avg_drinks max_drinks suicidal
4   4  39 female    white   housed    heroin          5          5       no
6   6  47 female    black   housed   cocaine          4          4       no
8   9  50 female    white homeless   alcohol         71        129       no
10 12  58 female    black   housed   alcohol         13         13       no
14 17  28 female hispanic homeless    heroin          0          0      yes
17 20  27 female    white   housed    heroin          9         24      yes
   treat physical1 mental1 depression1 physical2 mental2 depression2
4     no     61.93   43.97          15     46.61   14.66          44
6    yes     46.48   55.51           6     43.20   50.06          23
8     no     38.27   22.03          50     45.56   28.88          33
10    no     41.93   13.38          49     52.96   51.45           8
14   yes     44.78   29.80          35     52.69   46.59          19
17   yes     37.45   15.46          52     61.40   41.53          15
```

Here we use a single selection criterion as `sex == "female"`. We can also subset the data based on multiple criteria and select only some variables from our original data. Let's assume that we want to select participants who are female and 40 years old or older. Also, we only want to keep the following variables in the dataset: id, age, sex, substance. Using the `subset` function, we can do this selection as follows:

```
medical_f40 <- subset(medical, sex == "female" & age >= 40,
                      select = c("id", "age", "sex", "substance"))

head(medical_f40)
```

```
   id age    sex substance
6   6  47 female   cocaine
8   9  50 female   alcohol
10 12  58 female   alcohol
21 27  48 female   cocaine
51 65  41 female   alcohol
56 71  40 female   alcohol
```

2. Using the `filter` function from the `dplyr` package (an amazing package for data wrangling):

```
# Install and activate the package first
install.packages("dplyr")
library("dplyr")

medical_female <- filter(medical, sex == "female")
head(medical_female)
```

```
  id age    sex     race homeless substance avg_drinks max_drinks suicidal
1  4  39 female    white   housed    heroin          5          5       no
2  6  47 female    black   housed   cocaine          4          4       no
3  9  50 female    white homeless   alcohol         71        129       no
4 12  58 female    black   housed   alcohol         13         13       no
5 17  28 female hispanic homeless    heroin          0          0      yes
6 20  27 female    white   housed    heroin          9         24      yes
  treat physical1 mental1 depression1 physical2 mental2 depression2
1    no     61.93   43.97          15     46.61   14.66          44
2   yes     46.48   55.51           6     43.20   50.06          23
3    no     38.27   22.03          50     45.56   28.88          33
4    no     41.93   13.38          49     52.96   51.45           8
5   yes     44.78   29.80          35     52.69   46.59          19
6   yes     37.45   15.46          52     61.40   41.53          15
```

Using the `filter` and `select` functions from the `dplyr` package, we can also subset the dataset based on multiple criteria and select some variables from the dataset:

```
# Filter the data first
medical_f40 <- filter(medical, sex == "female", age >= 40)

# Select the variables to be keep
medical_f40 <- select(medical_f40, id, age, sex, substance)

# Preview the data
head(medical_f40)
```

There is also a more practical way to accomplish the same task. Here I will demonstrate %>%, which is called the **pipe** operator. This operator forwards the result of a function to the next function. This way we can simplify the code without creating many intermediate datasets (see https://uc-r.github.io/pipe for more details on the pipe).

```
medical_f40 <- medical %>% # Send the medical data to filter
  filter(sex == "female", age >= 40) %>% # Filter the data and send it to select
  select(id, age, sex, substance) # Finally select the data
```

```
# Preview the data
head(medical_f40)
```

```
  id age    sex substance
1  6  47 female   cocaine
2  9  50 female   alcohol
3 12  58 female   alcohol
4 27  48 female   cocaine
5 65  41 female   alcohol
6 71  40 female   alcohol
```

Here are most common operators for subsetting:

- `<` Less than
- `>` Greater than
- `==` Equal to
- `<=` Less than or equal to
- `>=` Greater than or equal to
- `!=` Not equal to
- `%in%` Group membership
- `&` And
- `|` Or
- `is.na` Is missing (NA).
- `!is.na` Is not missing (NA)

Other Data Manipulation Tools

Here I will mention the other key functions from the `dplyr` package. These functions solve the vast majority of data manipulation challenges:

- `arrange`: Reorder data based on values of variables
- `mutate`: Create new variables
- `summarise`: Summarize data by functions of choice

Arrange:

```
# Reorder the data by age
medical_f40 <- arrange(medical_f40, age)

# Let's see if the ordering worked
head(medical_f40)
```

```
   id age    sex substance
1  71  40 female   alcohol
2  65  41 female   alcohol
3  75  41 female    heroin
4 121  42 female   cocaine
5 465  42 female   alcohol
6 364  43 female    heroin
```

```
# Reorder the data by age in descending order
medical_f40 <- arrange(medical_f40, desc(age))

# Let's see if the ordering worked
head(medical_f40)
```

```
   id age    sex substance
1  12  58 female   alcohol
2 181  57 female   alcohol
3 264  55 female    heroin
4   9  50 female   alcohol
5 134  50 female   alcohol
6  27  48 female   cocaine
```

Mutate:

```
# Create a new variable based on age
medical_f40 <- medical_f40 %>%
  mutate(age2 = ifelse(age < 45, "Younger than 45", "45 or older"))

# Let's see if the ordering worked
head(medical_f40)
```

```
   id age    sex substance        age2
1  12  58 female   alcohol 45 or older
2 181  57 female   alcohol 45 or older
3 264  55 female    heroin 45 or older
4   9  50 female   alcohol 45 or older
5 134  50 female   alcohol 45 or older
6  27  48 female   cocaine 45 or older
```

We will use the `summarise` function in the next section.

Exercise 4

1. Using the medical dataset, create a subset where the patients:

   - are older than 30 years old: `age > 30`
   - are female: `sex == "female"`
   - are not homeless: `homeless != "homeless"`

and save this data as `medical_example`. You can use either `subset` or `filter` for this task.

2. Use the `dim` function to see how many rows you have in the new data
3. Sort this new dataset by age in descending order
4. Use the `head` function to preview the final dataset

Descriptive Statistics

Quick Summary

The easiest way to get a quick summary of a dataset in **R** is to the `summary( )` function. This function provides the min and max, mean, median, and first and third quartiles for the entire dataset or variables that we select. Let's take a look at the summary table for the `medical` dataset for a few variables.

```
summary(medical[,c("sex", "race", "age", "avg_drinks")])
```

```
     sex                race                age          avg_drinks    
 Length:246         Length:246         Min.   :20.0   Min.   :  0.0  
 Class :character   Class :character   1st Qu.:31.0   1st Qu.:  2.0  
 Mode  :character   Mode  :character   Median :35.0   Median : 12.0  
                                       Mean   :36.3   Mean   : 17.1  
                                       3rd Qu.:41.0   3rd Qu.: 24.0  
                                       Max.   :60.0   Max.   :142.0  
```

R often knows which variables are numerical (i.e., quantitative) and which variables are characters (i.e., qualitative). In the output above, we see a bunch of summary statistics for each variable. For character variables, we only see the length value – which is the number of rows for these variables.

We can see the summary table for the entire dataset using:

```
summary(medical)
```

Frequency Tables

Another handy function in base **R** is `table` which tabulates the data and creates frequency tables for variables. If the variable is a character, it will show the frequency of each level; if the variable is numerical, it will show the frequency of each value.

```
# Frequency tables for homeless status and sex
table(medical$homeless)
```

```

homeless   housed 
     118      128 
```

```
table(medical$sex)
```

```

female   male 
    57    189 
```

```
# Frequency table for age
table(medical$age)
```

```

20 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 
 2  7  3  4  2  6  8  9  4 12  7 19 14 11 16 11 16  6 16  5  8  7 10  3  6  2 
47 48 49 50 51 53 54 55 56 57 58 60 
10  5  4  2  1  2  1  2  1  1  2  1 
```

The output is kind of messy. The first row shows the age values and the second row shows how many times those values appear in the dataset. Let's reformat our frequency table into a more readable format.

```
age_table <- as.data.frame(table(medical$age))
head(age_table)
```

```
  Var1 Freq
1   20    2
2   22    7
3   23    3
4   24    4
5   25    2
6   26    6
```

We can rename the columns with better names using the `colnames( )` function.

```
colnames(age_table) <- c("Age", "Frequency")
head(age_table)
```

```
  Age Frequency
1  20         2
2  22         7
3  23         3
4  24         4
5  25         2
6  26         6
```

Sometimes these raw freqency values are hard to interpret. Therefore, we may prefer to have proportions or percentages rather than actual frequency values. For this task, we need to use `prop.table`. Let's see the proportion of male and female participants in the data.

```
# Proportions
prop.table(table(medical$sex))
```

```
female   male
0.2317 0.7683
```

```
# Percentages
prop.table(table(medical$sex))*100
```

```
female   male
 23.17  76.83
```

```
# Percentages rounded (no decimal points)
round(prop.table(table(medical$sex))*100, 0)
```

```
female   male
    23     77
```

We can also use the `table` function for cross-tabulation. For example, if we want to see the number of homeless and housed patients by sex:

```
# Frequencies
table(medical$homeless, medical$sex)
```

```
           female male
  homeless     22   96
  housed       35   93
```

```r
# Proportions
prop.table(table(medical$homeless, medical$sex))
```

```
            female    male
  homeless 0.08943 0.39024
  housed   0.14228 0.37805
```

```r
# Percentages
prop.table(table(medical$homeless, medical$sex))*100
```

```
           female   male
  homeless  8.943 39.024
  housed   14.228 37.805
```

```r
# Percentages rounded to the 2nd decimal point
round(prop.table(table(medical$homeless, medical$sex))*100, 2)
```

```
           female  male
  homeless   8.94 39.02
  housed    14.23 37.80
```

Exercise 5

Use the `table` and `prop.table` functions to create a cross-tabulation for `race` and `substance`. We want to see the percentages. Which group is the largest and which group is the smallest based on their percentages?

Central Tendency and Dispersion

Central tendency refers to indices or measures that gives us an idea about the center of the data. Typical central tendency measures are:

- **Mean**: The sum of the values for a given variable divided by the number of values (n). Mean is typically denoted by $\bar{X}$ (read as "x bar") or simply M. We can find the mean as:

$$\bar{X} = \frac{X_1 + X_2 + X_3 + ... + X_n}{n}.$$

- **Median**: The middle value for a given variable when the values are sorted from smallest to largest.

Dispersion refers to statistics that tell us how dispersed or spread out the values of a variable are. Typical dispersion measures are:

- **Standard deviation**: A typical difference (deviation) between a particular value and the mean of a variable. Standard deviation is denoted by σ (read as "sigma"). We can find the standard deviation as:

$$\sigma = \sqrt{\frac{(X_1 - \bar{X})^2 + (X_2 - \bar{X})^2 + ... + (X_n - \bar{X})^2}{n}}$$

- **Variance**: Variance is the squared value of standard deviation, σ^2.
- **Quantiles**: If we divide a cumulative frequency curve into quarters, the value at the lower quarter is referred to as the lower quartile, the value at the middle gives the median and the value at the upper quarter is the upper quartile.
- **Range**: Difference between the biggest value and the smallest value of a variable.
- **Interquartile range (IQR)**: Like the range, but instead of calculating the difference between the biggest and smallest values, it calculates the difference between the 25th quantile and the 75th quantile.

In **R**, we can calculate all of these statistics very easily. A critical point is that if the variable has missing values, then these statistics cannot be computed. Therefore, we need to add `na.rm = TRUE` inside the functions to remove missing values before calculations begin. Let's try the variable `age`.

```
# Mean
mean(medical$age, na.rm = TRUE)
```

```
[1] 36.31
```

```
# Median
median(medical$age, na.rm = TRUE)
```

```
[1] 35
```

```
# Standard deviation
sd(medical$age, na.rm = TRUE)
```

```
[1] 7.984
```

```
# Variance
var(medical$age, na.rm = TRUE)
```

```
[1] 63.75
```

```
# Quantile
quantile(medical$age, na.rm = TRUE)
```

```
  0%  25%  50%  75% 100% 
  20   31   35   41   60 
```

```
# 95th percentile
quantile(medical$age, 0.95)
```

```
  95% 
49.75 
```

```
# Range
range(medical$age, na.rm = TRUE)
```

```
[1] 20 60
```

```
# Min and max values
min(medical$age, na.rm = TRUE)
```

```
[1] 20
```

```
max(medical$age, na.rm = TRUE)
```

```
[1] 60
```

We can also calculate central tendency and dispersion by grouping variables, using the `tapply` function. Let's take a look at average and median age by sex.

```
tapply(medical$age, medical$sex, mean)
```

```
female   male
 37.07  36.08
```

```
tapply(medical$age, medical$sex, median)
```

```
female   male
    35     36
```

We can combine these functions using the `summarise` function from the `dplyr` package.

```
medical %>%
  summarise(mean_age = mean(age, na.rm = TRUE),
            median_age = median(age, na.rm = TRUE),
            sd_age = sd(age, na.rm = TRUE),
            var_age = var(age, na.rm = TRUE))
```

```
  mean_age median_age sd_age var_age
1    36.31         35  7.984   63.75
```

We can also create summaries by grouping variables using the `group_by` function from the `dplyr` package. Let's take a look at the summary of age by sex.

```
medical %>%
  group_by(sex) %>%
  summarise(n = n(), # Count by sex
            mean_age = mean(age, na.rm = TRUE), # Mean
            median_age = median(age, na.rm = TRUE), # Median
            sd_age = sd(age, na.rm = TRUE), # Standard deviation
            var_age = var(age, na.rm = TRUE)) # Variance
```

```
# A tibble: 2 x 6
  sex        n mean_age median_age sd_age var_age
  <chr>  <int>    <dbl>      <int>  <dbl>   <dbl>
1 female    57     37.1         35   8.51    72.4
2 male     189     36.1         36   7.83    61.3
```

Another conventient way to summarize a dataset descriptively is to use the `skim` function from the `skimr` package (Waring et al. 2020). Let's try it with our medical dataset.

```
# Let's install and activate the package
install.packages("skimr")
library("skimr")
```

To summarize the entire dataset:

```
skim(medical)
```

To summarize some variables:

```
skim(medical, mental1, mental2, avg_drinks, max_drinks)
```

To summarize the dataset by grouping variables:

```
medical %>%
  group_by(sex) %>%
  select(sex, mental1, mental2, avg_drinks, max_drinks) %>%
  skim()
```

Exercise 6

Using the `summarise` function from the `dplyr` package or the `skim` function from the `skimr` package, create a summary of the variable `depression1` by `race`. If you decide to use `summarise`, you need to include count, mean, standard deviation, minimum, and maximum values.

Data Visualizations in R

Base R Graphics

When it comes to data visualization, **R** is a wonderful software program. We can create a wide range of visualizations, from simple scatterplots and histograms to animated or interactive graphics. Let's start by drawing a few very simple graphs just to get a feel for what it's like to draw pictures using base **R** functions. In each plot, there are several elements that we can modify:

- `main`: Title for the figure
- `xlab`: Label for the x-axis
- `ylab`: Label for the y-axis

There are also a bunch of graphical parameters that we can use to customise the font style:

- *Font styles:* `font.main`, `font.sub`, `font.lab`, `font.axis`. These four parameters control the font style used for the plot title (`font.main`), the subtitle (`font.sub`), the axis labels (`font.lab`: note that you can't specify separate styles for the x-axis and y-axis without using low level commands), and the numbers next to the tick marks on the axis (`font.axis`). Somewhat irritatingly, these arguments are numbers instead of meaningful names: a value of 1 corresponds to plain text, 2 means boldface, 3 means italic and 4 means bold italic.

- *Font colours:* `col.main`, `col.sub`, `col.lab`, `col.axis`. These parameters do pretty much what the name says: each one specifies a **col**our in which to type each of the different bits of text. Conveniently, **R** has a very large number of named colours (type `colours()` to see a list of over 650 colour names that **R** knows), so you can use the English language name of the colour to select it. Thus, the parameter value here string like `"red"`, `"gray25"` or `"springgreen4"`.

- *Font size:* `cex.main`, `cex.sub`, `cex.lab`, `cex.axis`. Font size is handled in a slightly curious way in **R**. The "cex" part here is short for "**c**haracter **ex**pansion," and it's essentially a magnification value. By default, all of these are set to a value of 1, except for the font title: `cex.main` has a default magnification of 1.2, which is why the title font is 20% bigger than the others.

- *Font family:* `family`. This argument specifies a font family to use: the simplest way to use it is to set it to `"sans"`, `"serif"`, or `"mono"`, corresponding to a san serif font, a serif font, or a monospaced font. If you want to, you can give the name of a specific font, but keep in mind that different operating systems use different fonts, so it's probably safest to keep it simple. Better yet, unless you have some deep objections to the **R** defaults, just ignore this parameter entirely.

Boxplots

```
boxplot(medical$depression1,
        main = "Depression Scores")
```

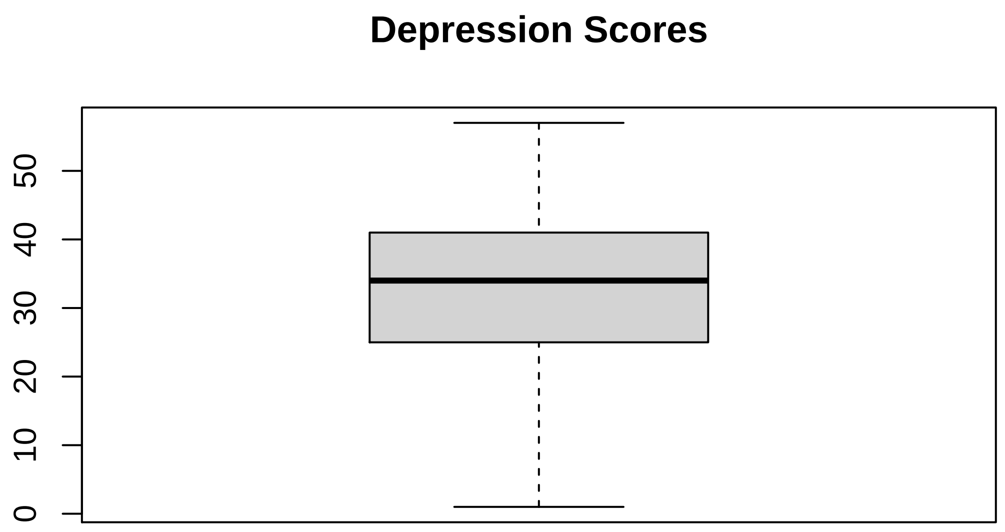


Figure 13: A boxplot example

What **R** draws is shown in the figure, the most basic boxplot possible. When we look at this plot, this is how we should interpret it: the thick line in the middle of the box is the median; the box itself spans the range from the 25th percentile to the 75th percentile; and the "whiskers" cover the full range from the minimum value to the maximum value.

We can also create the boxplots by a grouping variable. For this, we have to use a `formula` rather than a single variable. Let's create a boxplot of depression scores by sex.

```
boxplot(formula = depression1 ~ sex,
        data = medical,
        main = "Depression Scores by Sex",
        ylab = "Depression at the baseline",
        names = c("Female", "Male"))
```

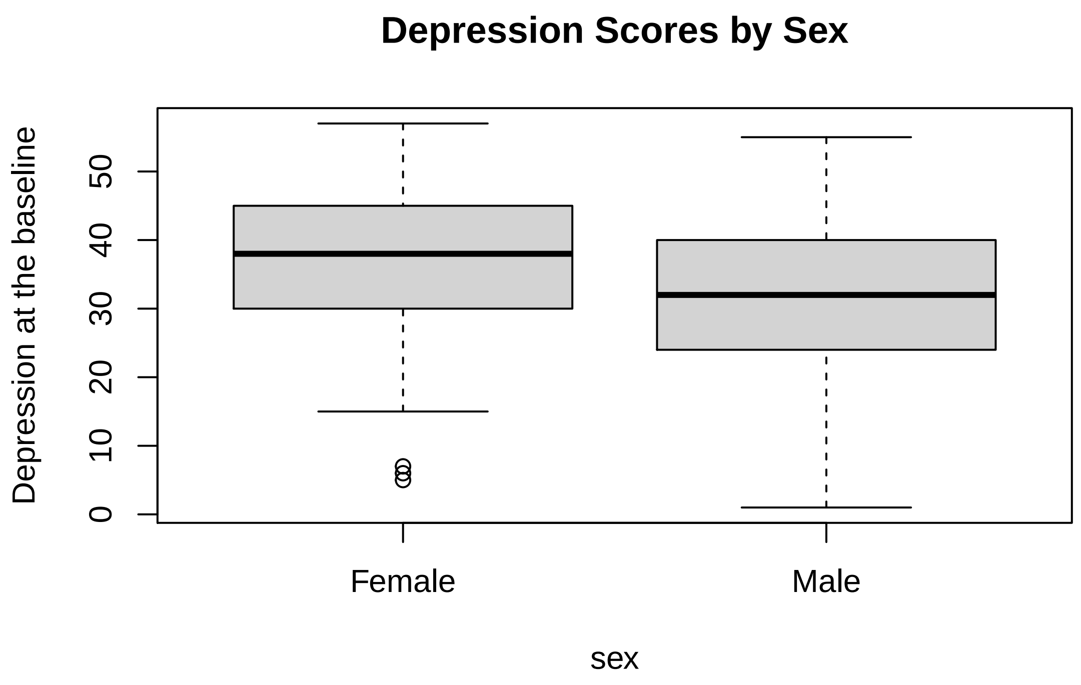


Figure 14: A boxplot by a grouping variable

Histograms

```
hist(medical$depression1,
     main = "Depression Scores at the Baseline",
     xlab = "Depression")
```

Bar Graphs

Bar plots are essentially histograms for categorical variables (e.g., sex, race, etc.). Before we create a bar plot, we need to make sure that our categorical variables are "factors." Otherwise, **R** attempts to treat such variables as quantative and thus fails to return a plot.

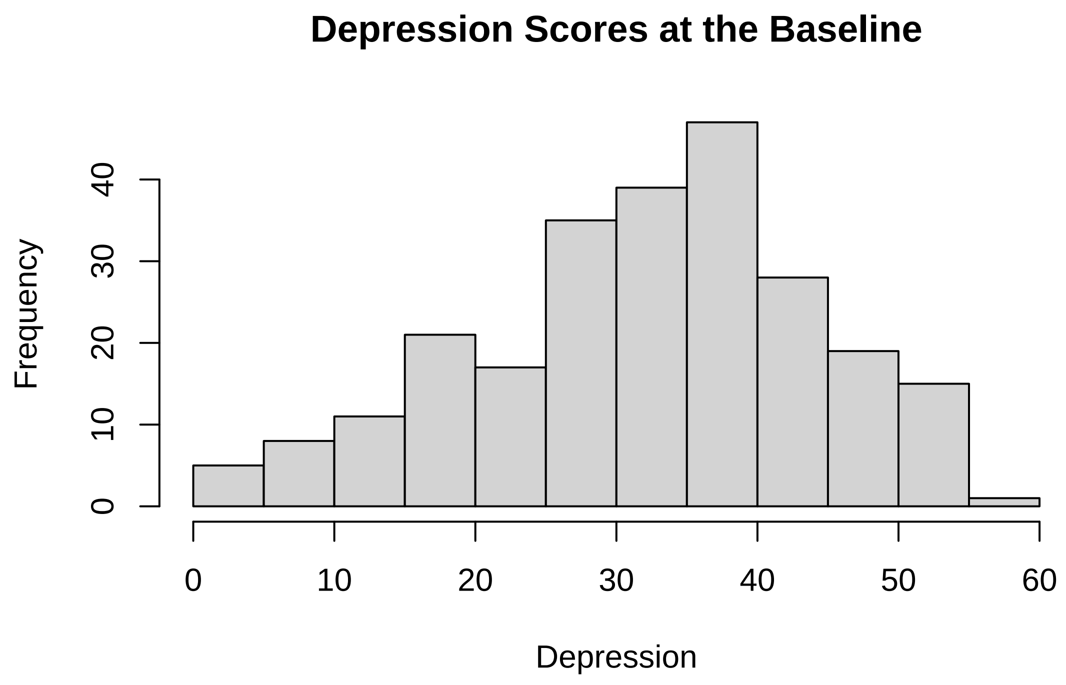


Figure 15: A histogram example

```
# Let's save race as a factor
medical$race <- as.factor(medical$race)

# Create a bar graph for race
plot(medical$race,
     main = "Race Groups in the medical Dataset",
     xlab = "Race",
     ylab = "Count")
```

Scatterplots

A scatterplot of depression scores at the baseline (`depression1`) against depression scores after 6 months (`depression2`):

```
plot(medical$depression1, medical$depression2,
     xlab = "Depression at the baseline",
     ylab = "Depression after 6 months",
     main = "Scatterplot of Depression Scores")
```

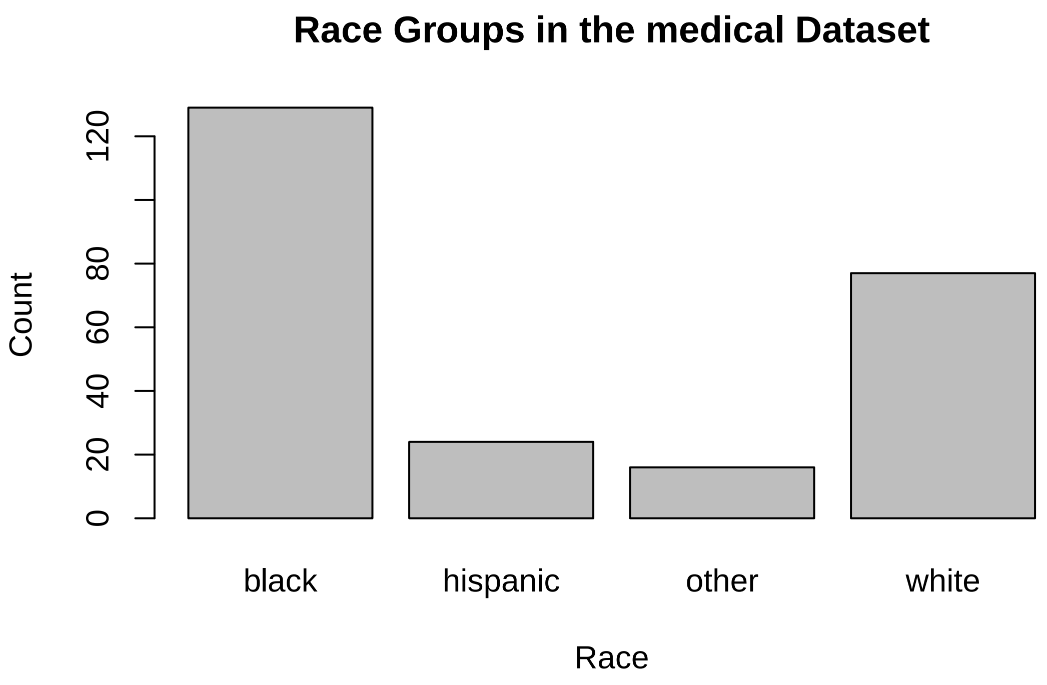


Figure 16: A bar graph example

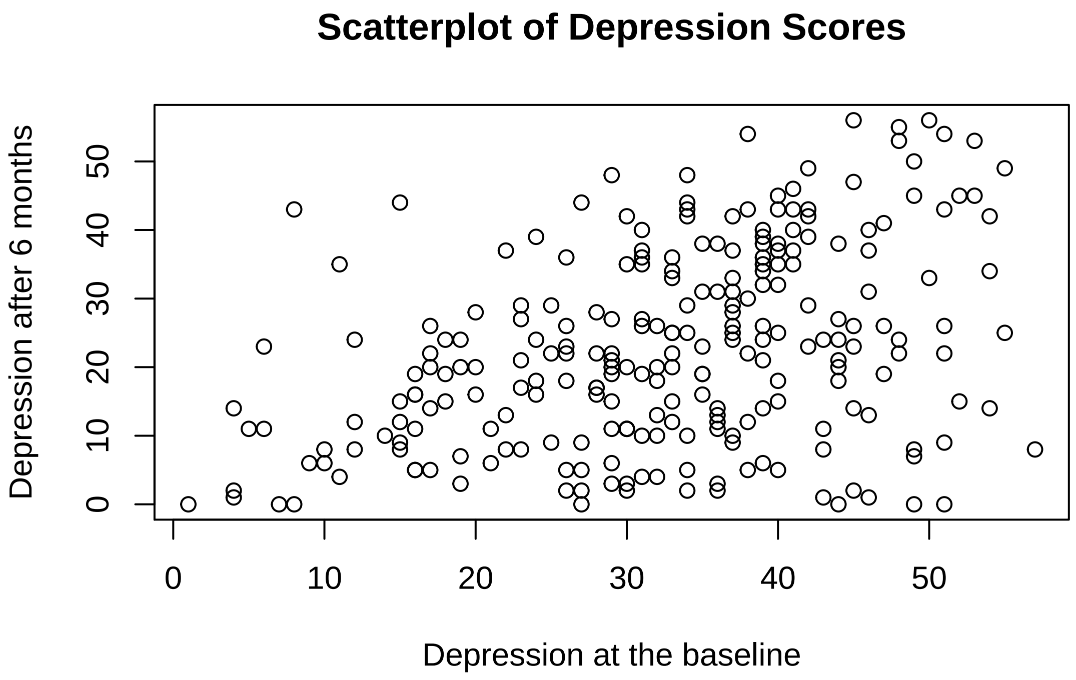


Figure 17: A scatterplot example

We can customise the appearance of the actual plot. To start with, let's look at the single most important options that the `plot()` function provides for us to use, which is the `type` argument. The type argument specifies the visual style of the plot. The possible values for this are:

- `type = "p"`. Draw the **p**oints only
- `type = "l"`. Draw a **l**ine through the points
- `type = "o"`. Draw the line **o**ver the top of the points
- `type = "b"`. Draw **b**oth points and lines, but don't overplot
- `type = "h"`. Draw "**h**istogram-like" vertical bars
- `type = "s"`. Draw a **s**taircase, going horizontally then vertically
- `type = "S"`. Draw a **S**taircase, going vertically then horizontally
- `type = "c"`. Draw only the **c**onnecting lines from the "b" version
- `type = "n"`. Draw nothing

The simplest way to illustrate what each of these really looks like is just to draw them. Figure 18 shows a scatterplot using six different `types` of plot. As you can see, by altering the type argument we can get a qualitatively different appearance to our plot.

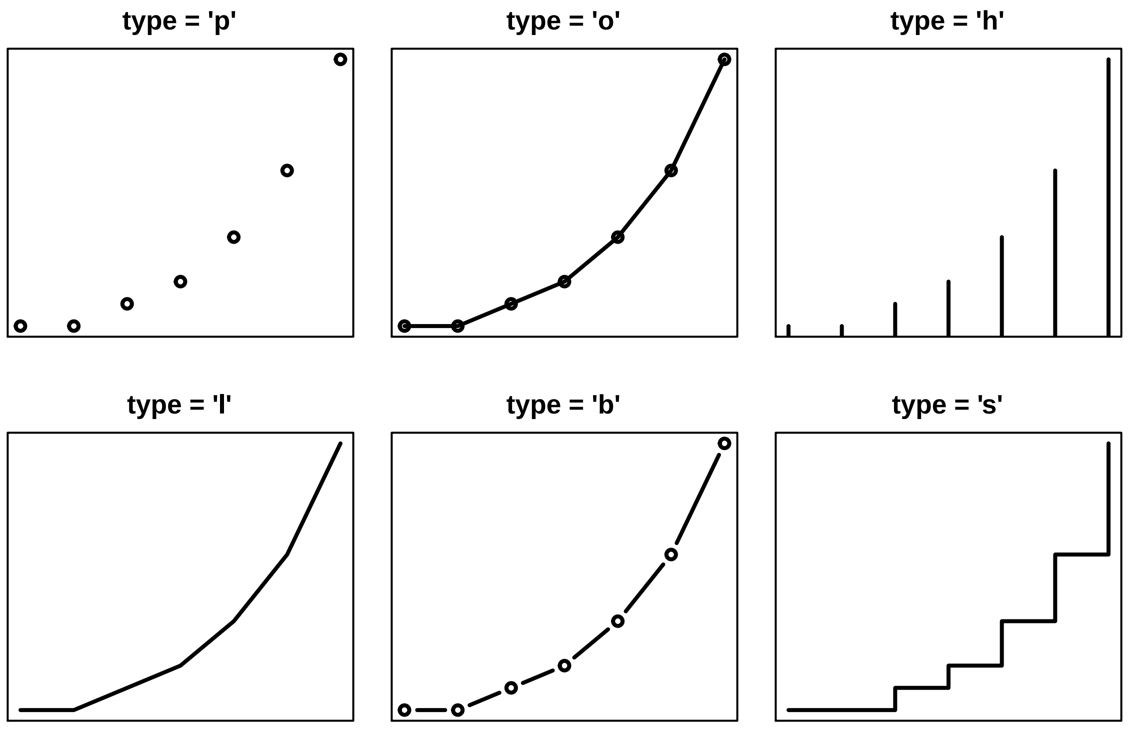


Figure 18: Changing the 'type' of the plot.

Scatterplots + Boxplots

The `scatterplot` function from the `car` package (Fox, Weisberg, and Price 2020) gives a nice plot that includes boxplots for individual variables and a scatterplot of the two variables together.

```
# Install and activate the car package
install.packages("car")
library("car")

scatterplot(depression1 ~ depression2,
            data = medical,
            smooth = FALSE)
```

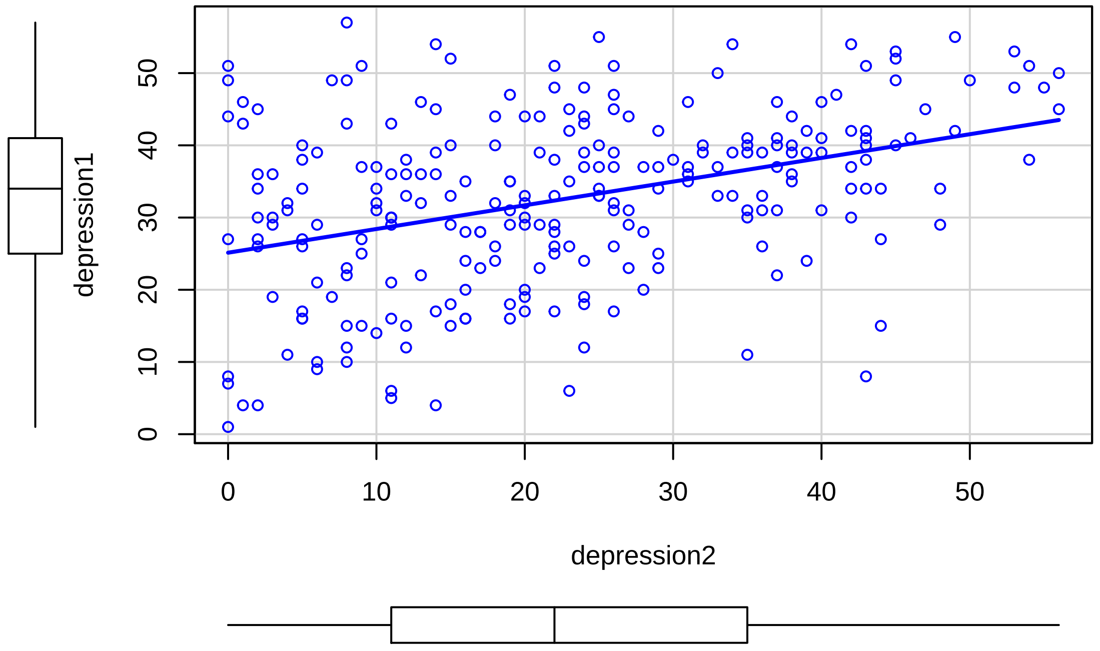


Figure 19: A scatterplot along with boxplots

Scatterplot Matrix

Often we find yourself wanting to look at the relationships between several variables at once. One useful tool for doing so is to produce a "scatterplot matrix," analogous to the correlation matrix. We can create a scatterplot matrix using the `pairs` function in base **R**. Let's take a look at the following variables: depression1, mental1, and physical1.

```
pairs(formula = ~ depression1 + mental1 + physical1,
      data = medical,
      main = "Scatterplot Matrix with Three Scores")
```

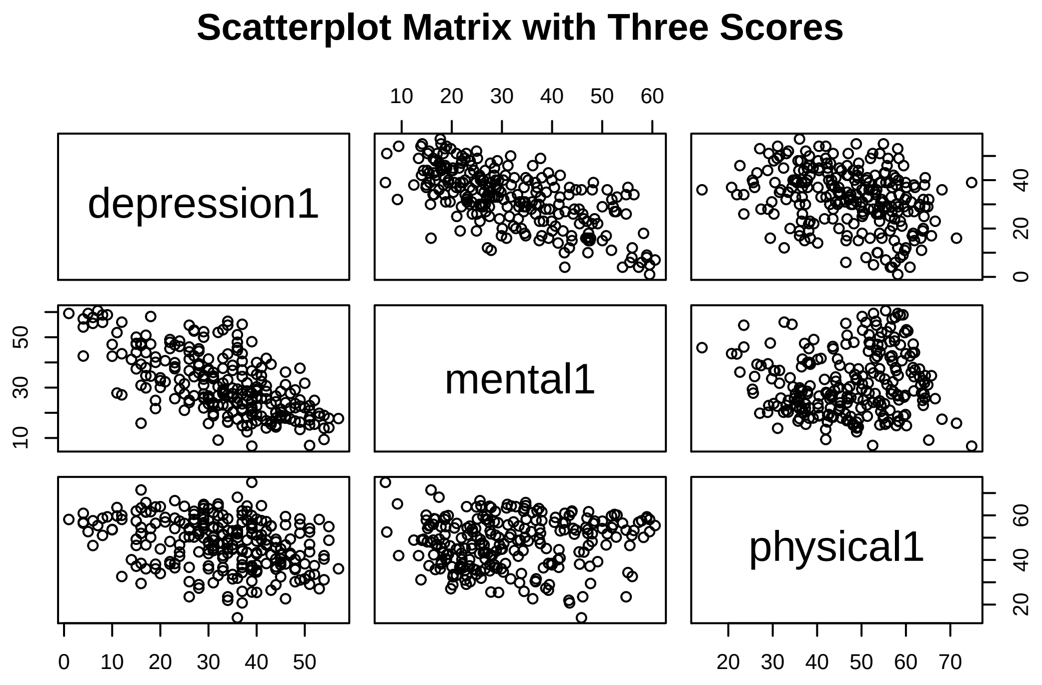


Figure 20: A scatterplot matrix from the ‘pairs()‘ function

Saving Base R Figures

We can save figures generated by base **R** functions in several ways:

- `jpeg("filename.jpg")`
- `png("filename.jpng")`
- `pdf("filename.pdf")`
- `tiff("filename.tif")`

For example, to save our plot using .jpg format, we would do:

```
jpeg("myplot.jpg", width = 8, height = 4, units = "in", res = 300)
plot(medical$depression1, medical$depression2)
dev.off()
```

where `width` and `height` are dimensions in inches (`units = "in"`) and resolution is 300 dpi.

Exercise 7

Here you will create two plots:

1. A boxplot of `mental1` (i.e., mental test scores at the baseline) by `substance` (i.e., type of substance being used). Do you see any differences between the mental test scores of the three substance groups?

2. A scatterplot of `depression1` against `mental1`. You need to see the depression scores on the x-axis and the mental test scores on the y-axis. What type of relationship do you see between the two variables (e.g., negative, positive, or no relationship)?

`ggplot2` Graphics

What is `ggplot2`?

- A comprehensive data visualization package in **R**
- Popular method for creating explanatory graphics
- Simpler than base **R** graphics due its multi-layer approach
- Many other supplementary packages using the `ggplot2` platform

How `ggplot2` works?

The `ggplot2` package (Wickham et al. 2020) follows data visualization rules known as "The Grammar of Graphics." The grammar tells us that a statistical graphic is a **mapping** of data variables to **aesthetic** attributes of **geometric** objects.

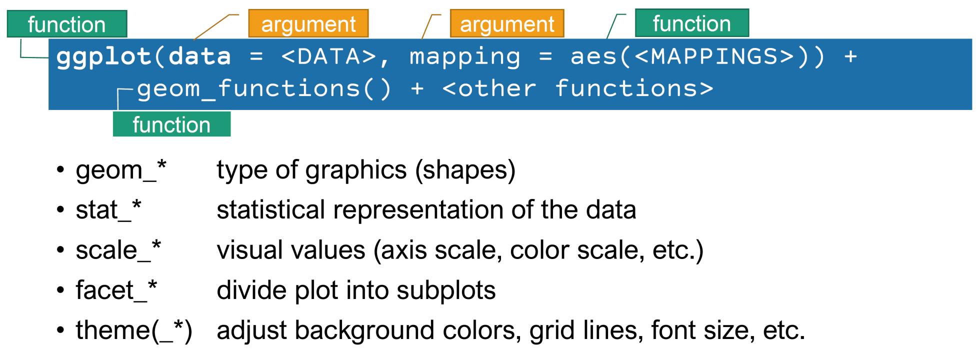
Specifically, we can break a graphic into the following three essential components:

- `data`: the data set composed of variables that we map.
- `geom`: the geometric object in question. This refers to the type of object we can observe in a plot. For example: points, lines, and bars.
- `aes`: aesthetic attributes of the geometric object. For example, color, shape, and size. Each assigned aesthetic attribute can be mapped to a variable in our data set.

Figure 21 shows these three components are laid out in a typical `ggplot2` function. As we can see, each part (e.g., `geom_function`) is added to the plot using a plus sign. That is, each layer like that brings an additional functionality into the plot we are drawing.

In order to keep things simple, we will only take a look at the following types of graphics in `ggplot2`:

- scatterplots



```
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) +
    geom_functions() + <other functions>
```

• geom_*	type of graphics (shapes)
• stat_*	statistical representation of the data
• scale_*	visual values (axis scale, color scale, etc.)
• facet_*	divide plot into subplots
• theme(_*)	adjust background colors, grid lines, font size, etc.

Figure 21: How the elements of ‘ggplot2‘ work

- boxplots
- histograms
- bar plots

For more information on `ggplot2`, check out http://ggplot2.tidyverse.org/.

Scatterplots

```r
# Activate the package first
library("ggplot2")

ggplot(data = medical,
       mapping = aes(depression1, depression2)) +
  geom_point(size = 3) +
  labs(x = "Depression (Baseline)",
       y = "Depression (6 months)") +
  theme_bw() # for black & white theme
```

```r
ggplot(data = medical,
       mapping = aes(depression1, depression2, colour = sex)) +
  geom_point(size = 3) +
  geom_smooth(method = lm, color = "red", se = TRUE) +
  labs(colour = "Sex",
       x = "Depression (Baseline)",
       y = "Depression (6 months)") +
  theme_bw() # for black & white theme
```

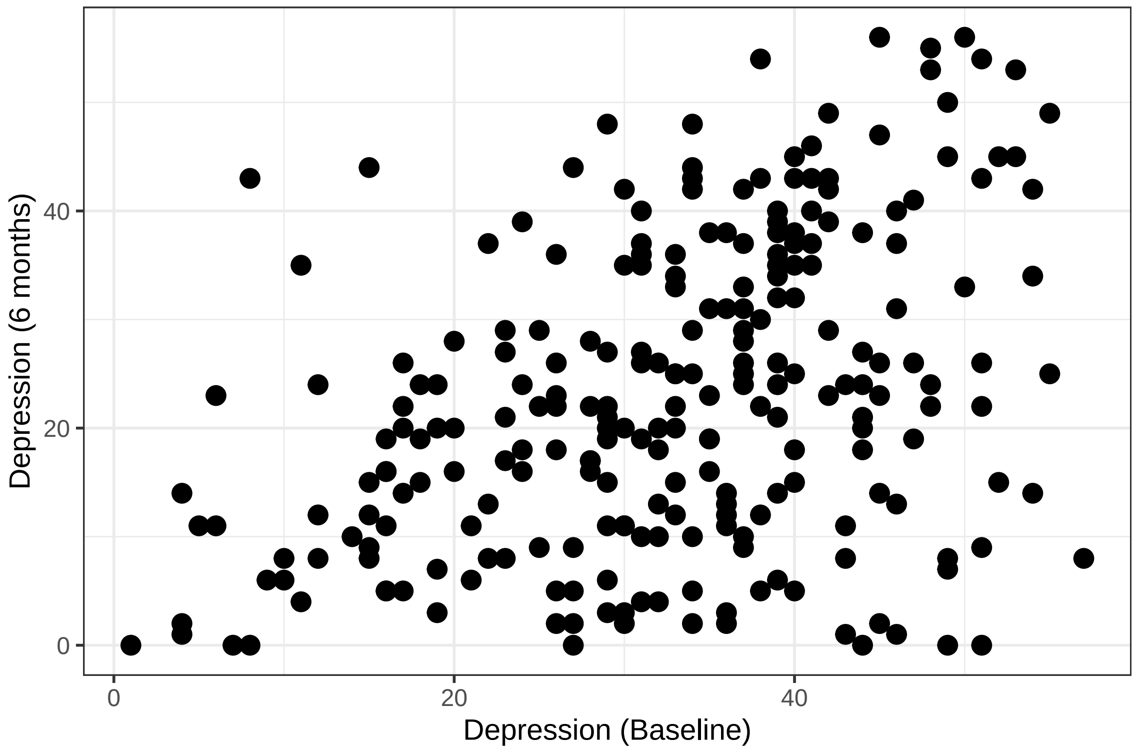


Figure 22: A scatterplot example with ‘ggplot2‘

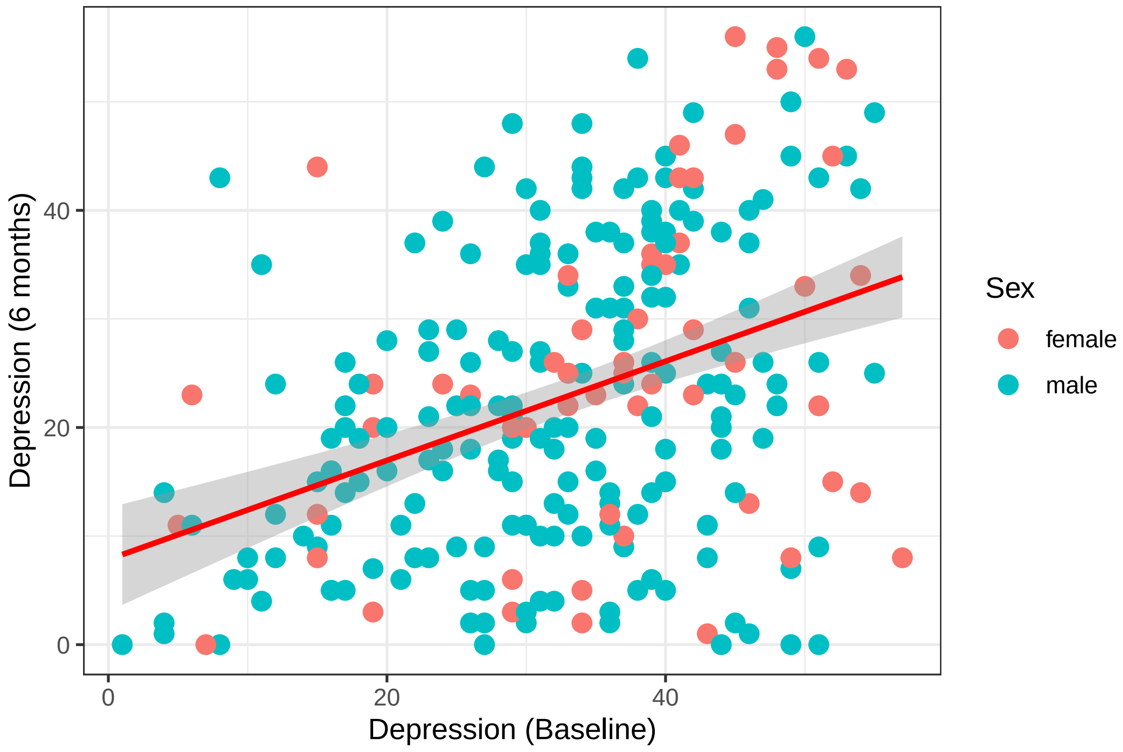


Figure 23: A scatterplot example with ‘ggplot2‘ (With regression line)

Boxplots

```
ggplot(data = medical,
       mapping = aes(x = sex, y = depression1, fill = race)) +
  labs(x = "Sex",
       y = "Depression at the baseline",
       fill = "Race") +
  geom_boxplot() +
  theme_bw()
```

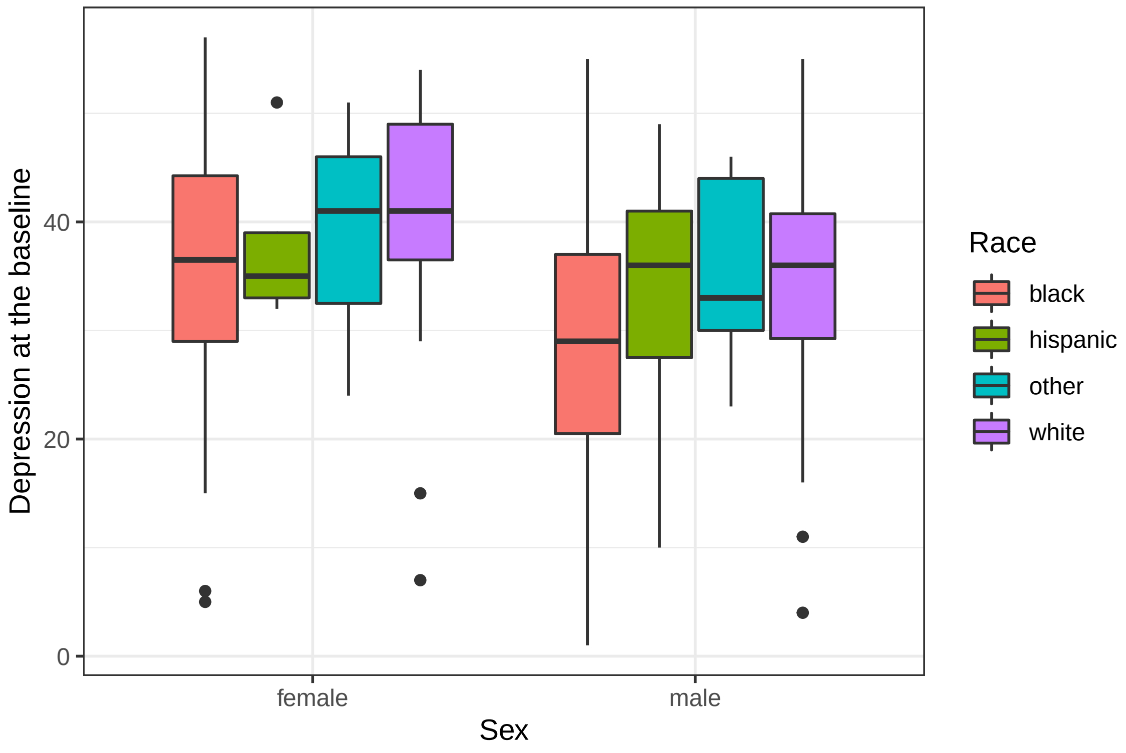


Figure 24: A boxplot example with ‘ggplot2‘

Histograms

```
ggplot(data = medical,
       mapping = aes(x = depression1)) +
  labs(x = "Depression at the baseline",
       y = "Frequency",
       title = "Depression Scores at the the Baseline") +
  geom_histogram(color = "white", # color of bar lines
                 fill = "steelblue", # filling color
                 bins = 40) + # number of bins
  theme_bw()
```

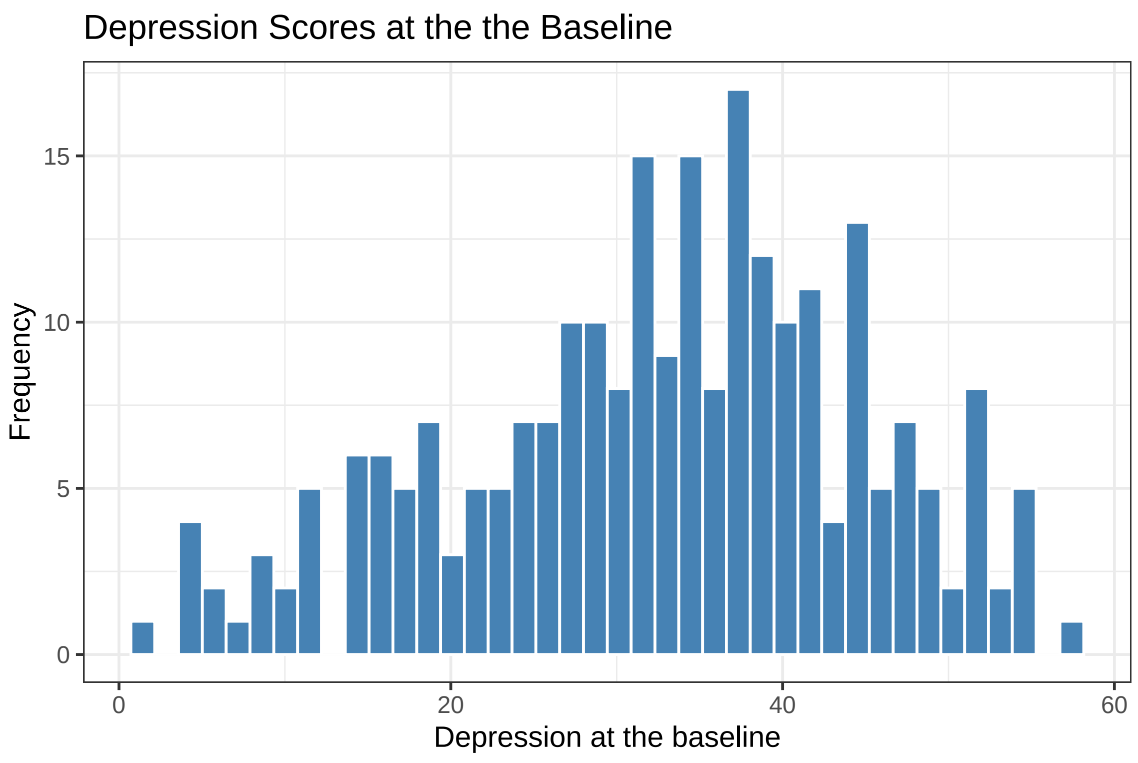


Figure 25: A histogram example with ‘ggplot2‘

Bar Plots

```
ggplot(data = medical,
       mapping = aes(x = race)) +
  labs(x = "Race",
       y = "Frequency") +
  geom_bar(color = "white",
           fill = "orange") +
  theme_bw()
```

```
ggplot(data = medical,
       mapping = aes(x = race, fill = sex)) +
  labs(x = "Race",
```

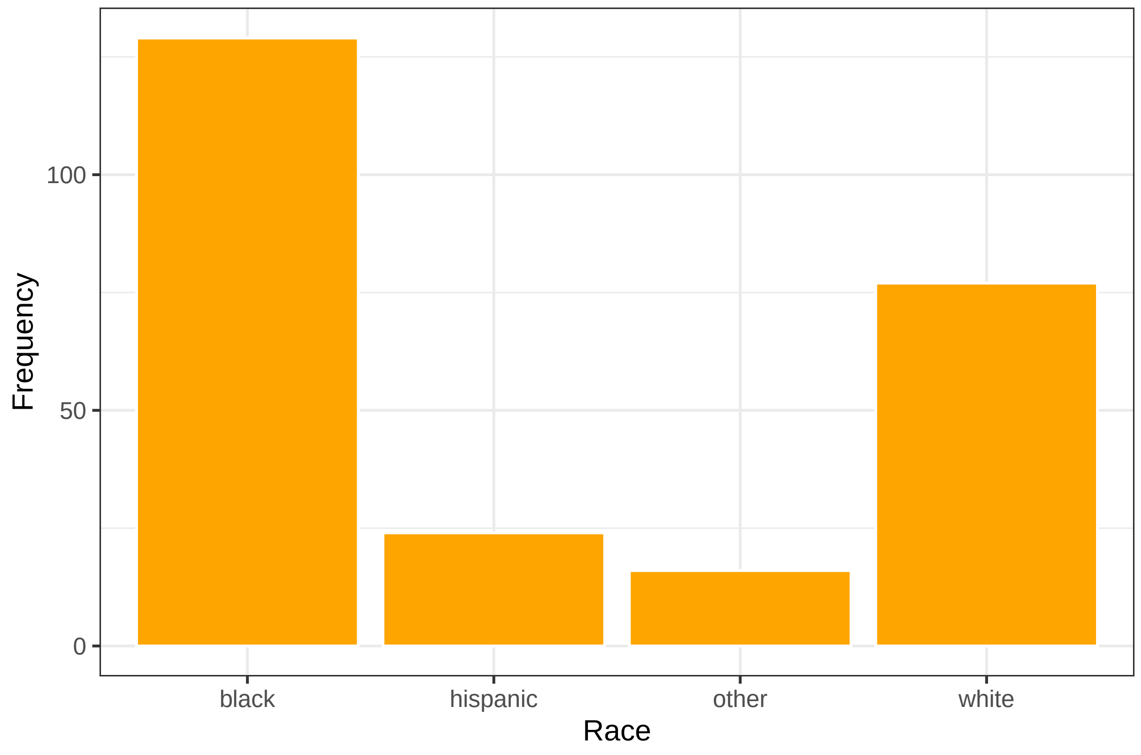


Figure 26: A bar plot example with ‘ggplot2‘

```
       y = "Frequency") +
  geom_bar() +
  theme_bw()
```

```
ggplot(data = medical,
       mapping = aes(x = race, fill = sex)) +
  labs(x = "Race",
       y = "Frequency") +
  geom_bar(position = "dodge") +
  theme_bw()
```

```
ggplot(data = medical,
       mapping = aes(x = race, fill = sex)) +
  labs(x = "Race",
       y = "Frequency") +
  geom_bar(position = "dodge") +
  facet_wrap(. ~ sex) +
  theme_bw()
```

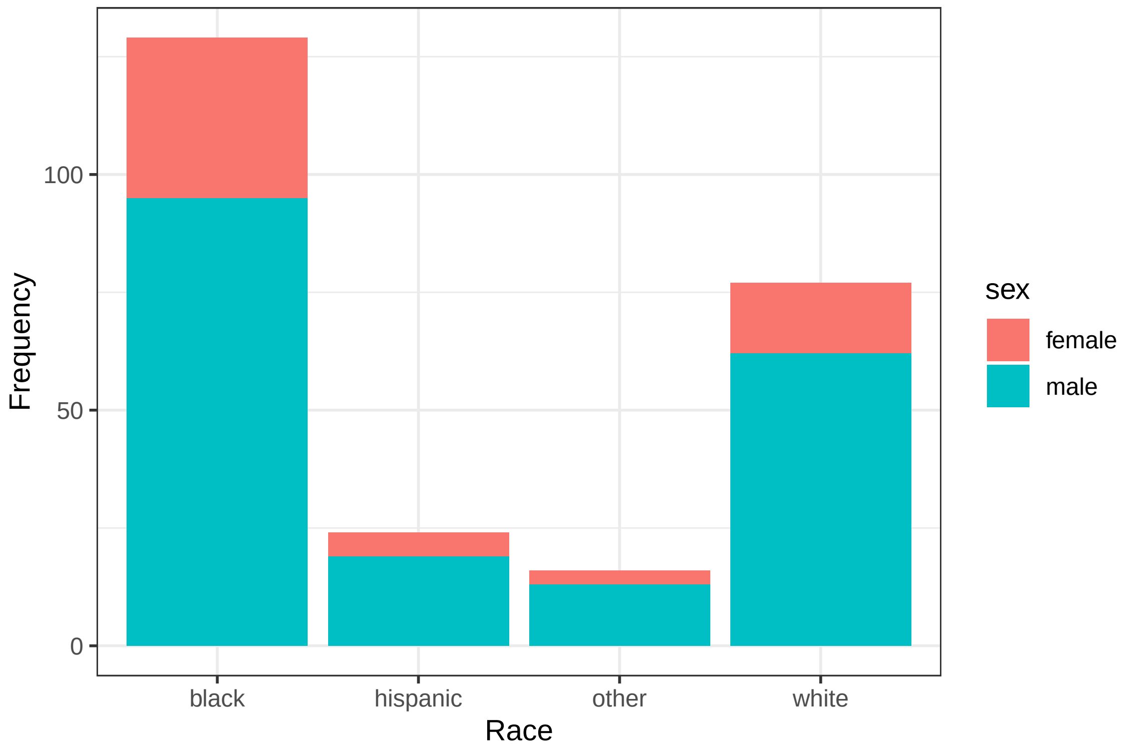


Figure 27: A bar plot example with ‘ggplot2‘ (stacked bar chart)

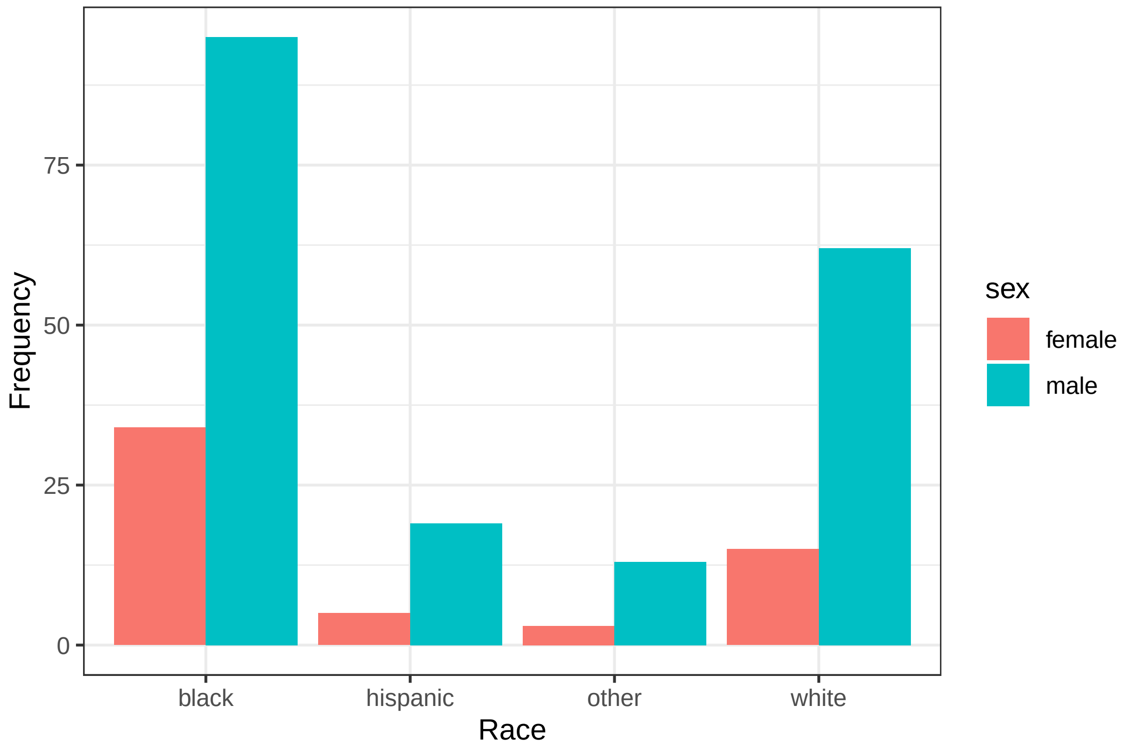


Figure 28: A bar plot example with ‘ggplot2‘ (side-by-side bars)

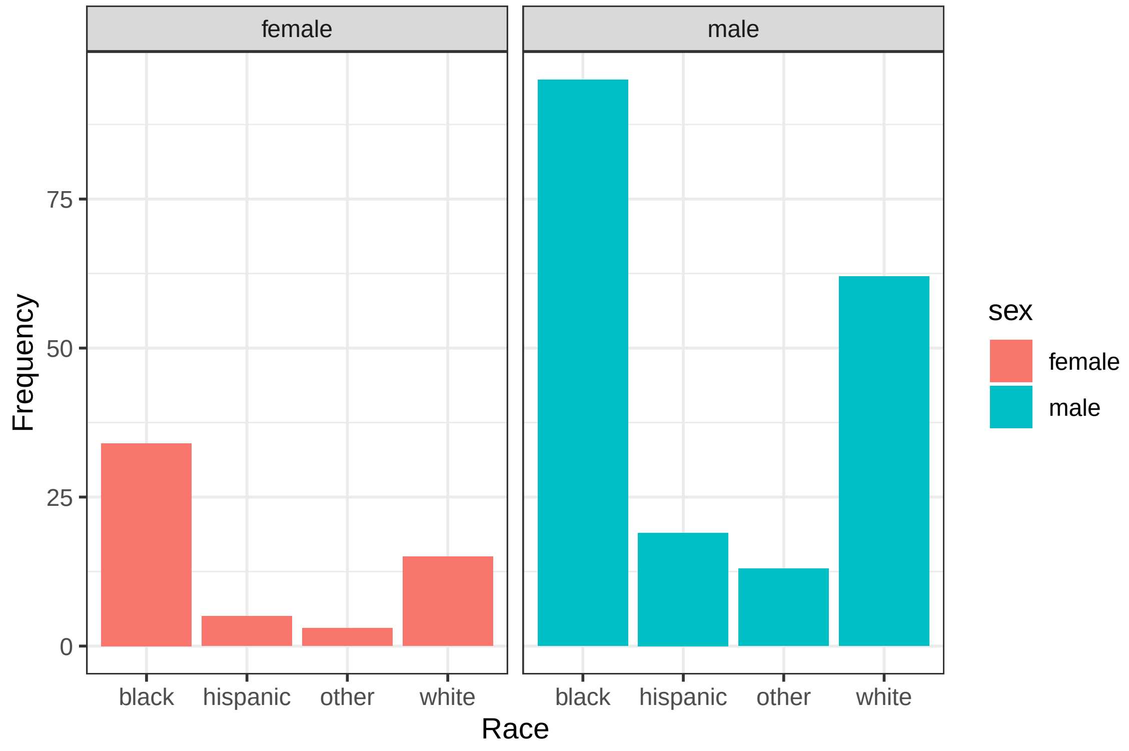


Figure 29: A bar plot example with ‘ggplot2‘ (faceted)

Exercise 8

Here you will create a scatterplot of `depression1` and `mental1` using `geom_point()`, with

- point colours set by `sex` (i.e., `colour = sex`)
- faceted by `substance` (i.e., `facet_wrap(. ~ substance)`)

Do `sex` or `substance` seem to affect the relationship between `depression1` and `mental1`?

Hypothesis Testing

Some Theory

> Statistics cannot prove anything with certainty. Instead, the power of statistical inference derives from observing some pattern or outcome and then using probability to determine the most likely explanation for that outcome (Wheelan 2013).

In its most abstract form, hypothesis testing has a very simple logic: the researcher has some theory about the world and wants to determine whether or not the data actually support that theory. In hypothesis testing, we want to:

- explore whether parameters in a model take specified values or fall in certain ranges
- detect significant differences, or differences that did not occur by random chance

To address these goals, we will use data from a sample to help us decide between two competing hypotheses about a population. These two completing hypotheses are:

- **a null hypothesis** (H_0) that corresponds to the exact opposite of what we want to prove
- **an alternative hypothesis** (H_1) that represents what we actually believe.

The claim for which we seek significant evidence is assigned to the alternative hypothesis. The alternative is usually what the researcher wants to establish or find evidence for. Usually, the null hypothesis is a claim that there really is "no effect" or "no difference." In many cases, the null hypothesis represents the status quo or that nothing interesting is happening.

For example, we can think of hypothesis testing in the same context as a criminal trial. A criminal trial is a situation in which a choice between two contradictory claims must be made.

- The accuser of the crime must be judged either guilty or not guilty.
- Under the rules of law, the individual on trial is initially presumed not guilty.

- Only *strong evidence* to the contrary causes the not guilty claim to be rejected in favor of a guilty verdict.

The phrase "beyond a reasonable doubt" is often used to set the cut-off value for when enough evidence has been given to convict. Theoretically, we should never say "The person is innocent" but instead "There is not sufficient evidence to show that the person is guilty." That is, technically it is **not** correct to say that we accept the null hypothesis. Accepting the null hypothesis is the same as saying that a person is innocent. We cannot show that a person is innocent; we can only say that there was not enough substantial evidence to find the person guilty.

Types of Inferential Statistics

We assess the strength of evidence by assuming the null hypothesis is true and determining how unlikely it would be to see sample statistics as extreme (or more extreme) as those in the original sample. Using inferential statistics allows us to make predictions or inferences about a population from observations based on a sample. We can calculate several inferential statistics:

1. Whether a sample mean is equal to a particular value:
    - One sample t-test ($H_0 : \mu = \text{value}$)
2. Whether two sample means are equal:
    - Independent samples t-test ($H_0 : \mu_1 = \mu_2$ or $H_0 : \mu_1 - \mu_2 = 0$)
    - Repeated measures t-test ($H_0 : \mu_D = 0$ or $H_0 : \mu_1 - \mu_2 = 0$)
3. Whether three or more groups have equal means:
    - ANOVA ($H_0 : \mu_1 = \mu_2 = \mu_3 = \cdots = \mu_k$)

One-Sample t Test

Suppose we wish to test for the population mean (μ) using a dataset of size (n), and the population standard deviation (σ) is not known. We want to test the null hypothesis $H_0 : \mu = \mu_0$ against some alternative hypothesis, with (α) level of significance.

The test statistic will be:

$$t = \frac{\bar{x} - \mu_0}{\frac{S}{\sqrt{n}}}$$

with $df = n - 1$ degrees of freedom. In the formula:

- $\bar{x}$ is the sample mean
- μ_0 is the population mean that we are comparing against our sample mean
- S is the sample standard deviation
- n is the sample size

If $t > t_{critical}$ at the α level of significance, we reject the null hypothesis; otherwise we *retain* the null hypothesis.

Example

Let's see one-sample t-test in action. All the patients in the `medical` dataset received a mental test at the baseline (when they were accepted to the study). The researcher who created the mental test reported that the mean score on this test for people with no mental issues should be around 35. Now we want to know whether the mean mental test for our sample of patients differs from the mean mental test score for the general population. Our hypotheses are:

- $H_0 : \mu = 35$
- $H_1 : \mu \neq 35$

```r
# Let's see the mean for mental1 in the data
mean(medical$mental1)
```

```
[1] 31.68
```

It looks like our sample mean is less than the population mean of 35. But, the question is whether it is small enough to conclude that there is a statistically significant difference between the sample mean (i.e., 31.68) and the population mean (i.e., 35).

```r
t.test(medical$mental1, mu = 35, conf.level = 0.95, alternative = "two.sided")
```

```
    One Sample t-test

data:  medical$mental1
t = -4.2, df = 245, p-value = 4e-05
alternative hypothesis: true mean is not equal to 35
95 percent confidence interval:
 30.11 33.25
sample estimates:
mean of x
    31.68
```

The `lsr` package (Navarro 2015) does the same analysis and it provides more organized output:

```
# Install the activate the package
install.packages("lsr")
library("lsr")

# Run one-sample t test
oneSampleTTest(x=medical$mental1, mu=35, conf.level=0.95, one.sided=FALSE)
```

```

   One sample t-test

Data variable:   medical$mental1

Descriptive statistics:
            mental1
   mean      31.680
   std dev.  12.486

Hypotheses:
   null:        population mean equals 35
   alternative: population mean not equal to 35

Test results:
   t-statistic:  -4.17
   degrees of freedom:  245
   p-value:  <.001

Other information:
   two-sided 95% confidence interval:  [30.112, 33.248]
   estimated effect size (Cohen's d):  0.266
```

Conclusion: With the significance level of $\alpha = .05$, we reject the null hypothesis that the average mental score in the `medical` dataset is the same as the average mental score in the population, $t(240) = -4.2$, $p < .001$, $CI_{95} = [30.11, 33.25]$.

Independent-Samples t Test

Suppose we have data from two independent populations, $x_1 \sim N(\mu_{x_1}, \sigma_{x_1})$ and $x_2 \sim N(\mu_{x_2}, \sigma_{x_2})$. We wish to determine whether the two population means, μ_{x_1} and μ_{x_2}, are the same or different. We want to test the null hypothesis $H_0 : \mu_{X_1} = \mu_{X_2}$ against some alternative hypothesis, with α level of significance. The test statistic will be:

$$t = \frac{(\bar{x}_1 - \bar{x}_2)}{\sqrt{\frac{{S_p}^2}{n_1} + \frac{{S_p}^2}{n_2}}}$$

and

$$S_p = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2}$$

with $df = n_1 + n_2 - 2$ degrees of freedom. In the formula:

- $\bar{x}_1$ is the sample mean response of the first group
- $\bar{x}_2$ is the sample mean response of the second group
- S_1^2 is the sample variance of the response of the first group
- S_2^2 is the sample variance of the response of the second group
- S_p is the pooled variance
- n_1 is the sample size of the first group
- n_2 is the sample size of the second group

If $t > t_{critical}$ at the α level of significance, we reject the null hypothesis; otherwise we *retain* the null hypothesis.

Example

Using the `medical` dataset, we want to know whether there was a significant difference between male and female patients' depression levels at the baseline. We will use `sex` and `depression1` to investigate this question. Before we test this question, let's see the boxplot for these two groups:

```r
boxplot(formula = depression1 ~ sex,
        data = medical,
        main = "Depression Scores by Sex",
        ylab = "Depression at the baseline",
        names = c("Female", "Male"))
```

It seems that female patients have higher levels of depression on average, compared to male patients. Next, we will create two new small datasets (`male` and `female`) that consist of only depression scores for each gender group.

```r
male <- subset(medical, sex == "male", select = "depression1")
female <- subset(medical, sex == "female", select = "depression1")
t.test(male, female, conf.level = 0.95, alternative = "two.sided")
```

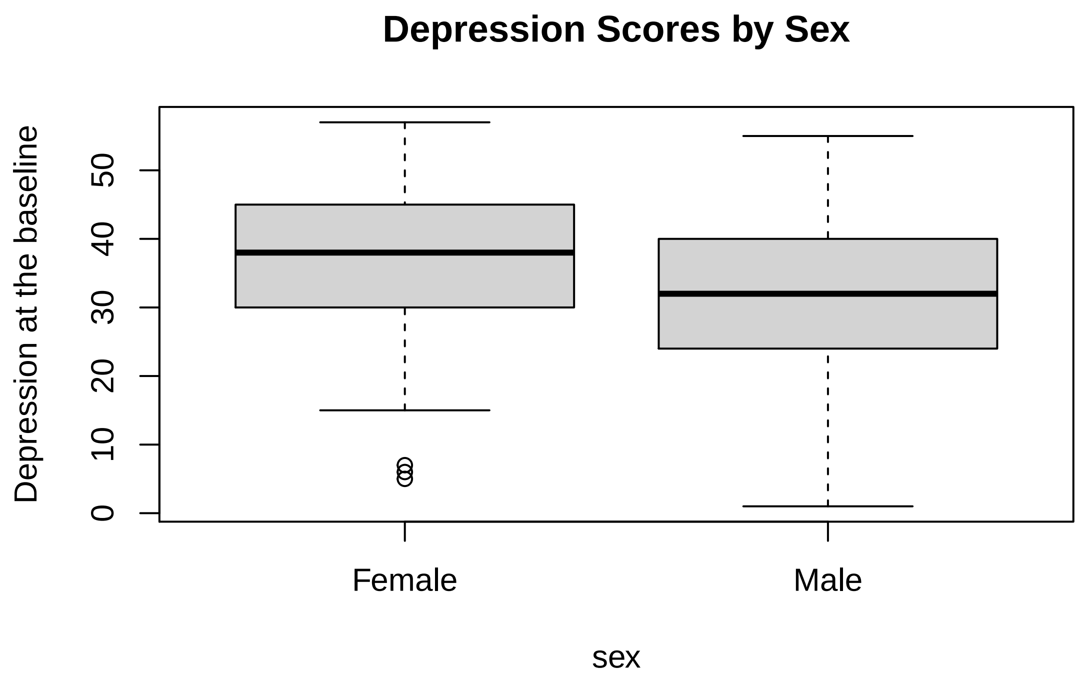


Figure 30: Boxplot of depression scores by sex

```
    Welch Two Sample t-test

data:  male and female
t = -2.5, df = 87, p-value = 0.02
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -8.418 -0.928
sample estimates:
mean of x mean of y
    31.50     36.18
```

We can again use the `lsr` package to get a better output. `independentSamplesTTest` function does not require us to separate the dataset for each group. We only need to specify the group variable, which is `sex` in our example. The only thing we need to make sure that the group variable is a factor.

```
medical$sex <- as.factor(medical$sex)

independentSamplesTTest(formula = depression1 ~ sex,
                        conf.level = 0.95,
                        one.sided = FALSE,
                        data = medical)
```

```
   Welch's independent samples t-test

Outcome variable:   depression1
Grouping variable:  sex

Descriptive statistics:
            female   male
   mean     36.175 31.503
   std dev. 12.675 11.758

Hypotheses:
   null:        population means equal for both groups
   alternative: different population means in each group

Test results:
   t-statistic:  2.48
   degrees of freedom:  87.09
   p-value:  0.015

Other information:
   two-sided 95% confidence interval:  [0.928, 8.418]
   estimated effect size (Cohen's d):  0.382
```

Conclusion: With the significance level of $\alpha = .05$, we reject the null hypothesis that the average depression score for male and female patients is the same in the population, $t(87) = -2.5$, $p < .05$, $CI_{95} = [-8.42, -0.93]$.

t-test with Paired Data

Suppose we have paired data, X_1 and X_2 with $D = X_1 - X_2 \sim N(\mu_D, \sigma)$. We wish to determine whether the two population means (or a single population across two time points), μ_{X_1} and μ_{X_2}, are the same (i.e., $\mu_D = 0$) or different (i.e., $\mu_D \neq 0$). We test the null hypothesis $H_0 : \mu_X = \mu_Y$ against some alternative hypothesis, with α level of significance. The test statistic will be:

$$t = \frac{\bar{D}}{\frac{S_D}{\sqrt{n}}}$$

with $df = n - 1$ degrees of freedom. In the formula:

- D is the difference between two populations (or two time points)
- S_D is the sample standard deviation of the difference
- n is the sample size of the second group

If $t > t_{critical}$ at the α level of significance, we reject the null hypothesis; otherwise we *retain* the null hypothesis.

Example

Using the `medical` dataset, this time we want to know whether patients' depression scores at the baseline (`depression1`) are the same as their depression scores after 6 months (`depression2`). First, let's see the means for the two variables.

```
mean(medical$depression1)
```

```
[1] 32.59
```

```
mean(medical$depression2)
```

```
[1] 22.72
```

It seems that the scores at month 6 are much lower. Let's see if the difference is statistically significant.

```
t.test(medical$depression1, medical$depression2,
       paired = TRUE, alternative = "two.sided",
       conf.level = 0.95)
```

```

    Paired t-test

data:  medical$depression1 and medical$depression2
t = 11, df = 245, p-value <2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  8.021 11.719
sample estimates:
mean of the differences
                   9.87
```

Let's repeat the same analysis with the `lsr` package.

```
pairedSamplesTTest(formula = ~ depression1 + depression2,
                   conf.level = 0.95,
                   one.sided = FALSE,
                   data = medical)
```

```
   Paired samples t-test

Variables:  depression1 , depression2

Descriptive statistics:
            depression1 depression2 difference
   mean          32.585      22.715      9.870
   std dev.      12.112      14.287     14.727

Hypotheses:
   null:        population means equal for both measurements
   alternative: different population means for each measurement

Test results:
   t-statistic:  10.51
   degrees of freedom:  245
   p-value:  <.001

Other information:
   two-sided 95% confidence interval:  [8.021, 11.719]
   estimated effect size (Cohen's d):  0.67
```

Conclusion: With the significance level of $\alpha = .05$, we reject the null hypothesis that the average depression scores for the baseline and 6th month are the same in the population, $t(245) = 10.51$, $p < .001$, $CI_{95} = [8.02, 11.72]$.

Analysis of Variance (ANOVA)

Independent-samples t-test that we have seen earlier is suitable for comparing the means of two independent groups. But, what if there are more than two groups to compare? One could suggest that we run multiple t-tests to compare all possible pairs and make a decision at the end. However, each statistical test that we run involves a certain level of error (known as Type I error) that leads to incorrect conclusions on the results. Repeating several t-tests to compare the groups would increase the likelihood of making incorrect conclusions. Therefore, when there are three or more groups to be compared, we follow a procedure called Analysis of Variance – or shortly ANOVA.

Suppose we have K number of populations. Collect a random sample of size n_1 from population 1, n_2 from population 2, ..., n_K from population k. We assume all populations have the same standard deviation (and they are normally distributed). We wish to test the following null hypothesis:

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \cdots = \mu_K$$

against

$$H_1 : H_0 \text{ is false}$$

which means that all of the groups would have equal means in their populations. If at least one of the groups has a significantly different mean, then we would reject the null hypothesis and run post-hoc tests to find out which group(s) are different.

Let $N = \sum_{k=1}^{K} n_k$ be the grand total, $(\overline{x}_k = \frac{1}{n_k} \sum_{i=1}^{n_k} x_{ki})$ be the sample mean for sample k, and $\overline{x}. = \frac{1}{N} \sum_{k=1}^{K} \sum_{i=1}^{n_k} x_{ki}$ be the grand mean. Then, the F-statistic is

$$F = \frac{\sum_{k=1}^{K} \left(\overline{x}_k - \overline{x}.\right)^2 / (K-1)}{\sum_{k=1}^{K} \sum_{i=1}^{n_k} \left(x_{ki} - \overline{x}_k\right)^2 / (N-K)}$$

with degrees of freedom of $df_1 = K - 1$ and $df_2 = N - K$.

If $F > F_{critical}$ at the α level of significance, we reject the null hypothesis; otherwise we *retain* the null hypothesis.

Example

Using the `medical` dataset, we want to investigate whether patients with different types of substance addition had the same level of depression at the baseline. We will use `depression1` and `substance` for this analysis. Before we begin the analysis, let's visualize the distribution of `depression1` by `substance` to get a sense of potential group differences.

```r
boxplot(formula = depression1 ~ substance,
        data = medical,
        main = "Depression Scores by Substance Type",
        ylab = "Depression at the baseline")
```

The boxplot shows that the means are close across the three groups but we cannot tell for sure if the differences are negligible to ignore. We will use the `aov` function – which is part of base **R**.

```r
# Compute the analysis of variance
aov_model1 <- aov(depression1 ~ substance, data = medical)

# Summary of the analysis
summary(aov_model1)
```

```
             Df Sum Sq Mean Sq F value Pr(>F)
substance     2   1209     605    4.23  0.016 *
Residuals   243  34733     143
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

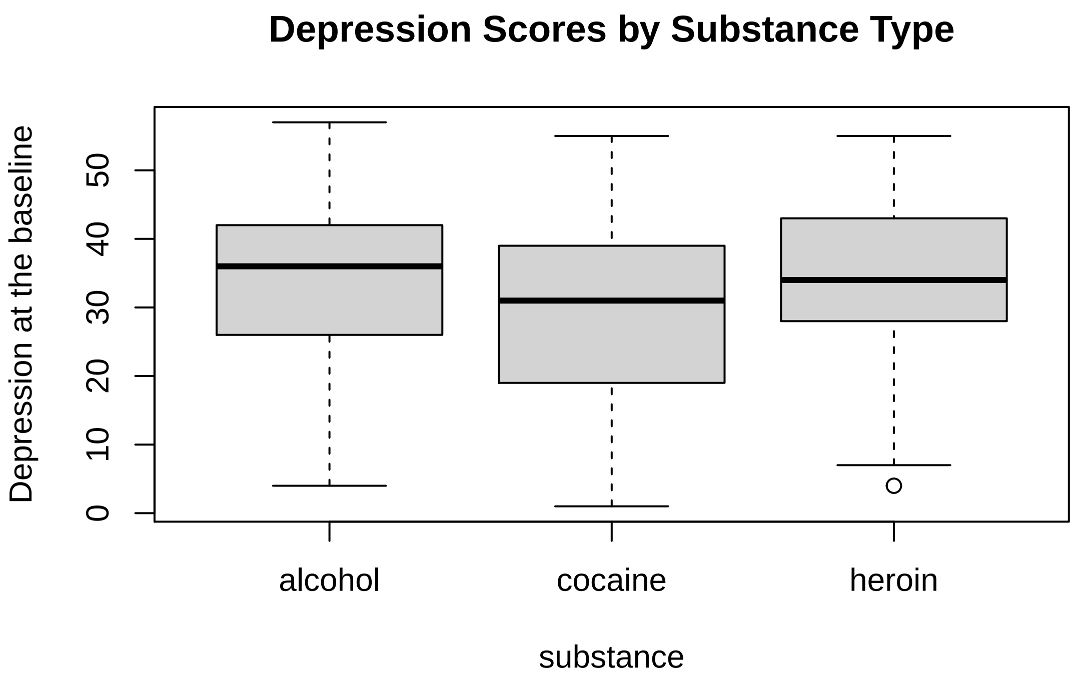


Figure 31: Boxplot of depression scores by substance type

Conclusion: With the significance level of $\alpha = .05$, the output shows that $F(2, 243) = 4.23, p < .05$, indicating that the test is statistically significant and thus we need to reject the null hypothesis of equal group means. This finding also suggests that at least one of the groups is different from the others.

As the ANOVA test is significant, now we can compute Tukey HSD (Tukey Honest Significant Differences) for performing multiple pairwise-comparison between the means of groups. The function `TukeyHD()` takes the fitted ANOVA as an argument and gives us the pairwise-comparison results.

```
TukeyHSD(aov_model1)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = depression1 ~ substance, data = medical)

$substance
                    diff     lwr     upr  p adj
cocaine-alcohol -4.62684 -8.7730 -0.4807 0.0245
heroin-alcohol  -0.08964 -4.7249  4.5456 0.9989
heroin-cocaine   4.53720 -0.1281  9.2025 0.0586
```

In the output above,

- **diff:** difference between means of the two groups
- **lwr, upr:** the lower and the upper end point of the confidence interval at 95%
- **p adj:** p-value after adjustment for the multiple comparisons

It can be seen from the output that only the difference between cocaine and alcohol is significant with an adjusted p-value of 0.0245.

We can add other variables into the ANOVA model and continue testing. Let's include `sex` as a second variable.

```
# Substance + Sex
aov_model2 <- aov(depression1 ~ substance + sex, data = medical)
summary(aov_model2)
```

```
             Df Sum Sq Mean Sq F value Pr(>F)
substance     2   1209     605    4.36 0.0138 *
sex           1   1199    1199    8.65 0.0036 **
Residuals   242  33534     139
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Substance + Sex + Substance x Sex
aov_model3 <- aov(depression1 ~ substance*sex, data = medical)
summary(aov_model3)
```

```
               Df Sum Sq Mean Sq F value Pr(>F)
substance       2   1209     605    4.38 0.0135 *
sex             1   1199    1199    8.69 0.0035 **
substance:sex   2    435     218    1.58 0.2083
Residuals     240  33098     138
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The output shows that `sex` is also statistically significant in the model; but the interaction of `sex` and `substance` is not statisticall significant.

Exercise 9

Now you will run two hypothesis tests using the variables in the `medical` dataset:

1. Run an independent-samples *t*-test where you will investigate whether the average depression scores at the baseline (i.e., `depression1`) are the same for suicidal patients (i.e., `suicidal == "yes"`) and non-suicidal patients (i.e., `suicidal == "no"`).
2. You will conduct an ANOVA to investigate whether the patients' physical scores at the baseline (i.e., `physical1`) differ depending on their race (i.e., `race`).

Additional Packages to Consider

There are also additional **R** packages that might be useful when conducting *t*-tests. One of these packages is **report** (https://github.com/easystats/report). This package automatically produces reports of models and data frames according to best practices guidelines (e.g., APA's style), ensuring standardization and quality in results reporting.

```
# Install the activate the package
install.packages("report")
library("report")

# Run an independent-samples t test and create a report
report(t.test(male, female, conf.level = 0.95, alternative = "two.sided"))

# Run ANOVA and create a report
report(aov(depression1 ~ substance + sex, data = medical))
```

```
Effect sizes were labelled following Cohen's (1988) recommendations.

The Welch Two Sample t-test testing the difference between male and female (mean of x =

The ANOVA (formula: depression1 ~ substance + sex) suggests that:

  - The main effect of substance is significant and small (F(2, 242) = 4.36, p = 0.014;
  - The main effect of sex is significant and small (F(1, 242) = 8.65, p = 0.004; Eta2 (

Effect sizes were labelled following Field's (2013) recommendations.
```

Finally, **report** also includes some functions to help you write the data analysis paragraph about the tools used.

```
report(sessionInfo())
```

```
Analyses were conducted using the R Statistical language (version 4.0.4; R Core Team, 20
References
----------
  - Elin Waring, Michael Quinn, Amelia McNamara, Eduardo Arino de la Rubia, Hao Zhu and
  - H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2
  - Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2021). dplyr: A Gra
  - John Fox and Sanford Weisberg (2019). An {R} Companion to Applied Regression, Third
  - John Fox, Sanford Weisberg and Brad Price (2019). carData: Companion to Applied Regr
  - Long JA (2020). _jtools: Analysis and Presentation of Social ScientificData_. R pack
  - Makowski, D., Ben-Shachar, M.S., Patil, I. & Lüdecke, D. (2020). Automated reporting
  - Navarro, D. J. (2015) Learning statistics with R: A tutorial for psychology students
  - R Core Team (2021). R: A language and environment for statistical computing. R Found
  - Sarkar, Deepayan (2008) Lattice: Multivariate Data Visualization with R. Springer, N
  - Taiyun Wei and Viliam Simko (2017). R package "corrplot": Visualization of a Correla
  - Yihui Xie (2020). bookdown: Authoring Books and Technical Documents with R Markdown.
  - Yihui Xie (2021). knitr: A General-Purpose Package for Dynamic Report Generation in
```

Correlation and Regression

Correlation

Correlation measures the degree to which two variables are related to or associated with each other. It provides information on the strength and direction of relationships. The most widely used correlation index, also known as Pearson correlation, is

$$\text{For populations: } \rho = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

$$\text{For samples: } r = \frac{S_{xy}}{S_x S_y}$$

Here are some notes on how to interpret correlations:

- Correlation can range from -1 to +1.
- The sign (either - or +) shows the direction of the correlation
- The value of correlation shows the magnitude of the correlation.
- As correlations get closer to either -1 or +1, the strength increases.
- Correlations near zero indicate very weak to no correlation.

There are many guidelines for categorizing weak, moderate, and strong correlations. Typically, researchers refers to Cohen's guidelines[1] as shown below:

Correlation	Interpretation
0.1	Small
0.3	Moderate
0.5	Strong

[1]**Source:** http://imaging.mrc-cbu.cam.ac.uk/statswiki/FAQ/effectSize

In **R**, the `cor()` function provides correlation coefficients and matrices. For the correlation between two variables (`depression1` and `mental1`):

```
cor(medical$depression1, medical$mental1)
```

```
[1] -0.6629
```

For the correlation between multiple variables:

```
cor(medical[, c("depression1", "mental1", "physical1")])
```

```
            depression1  mental1 physical1
depression1      1.0000 -0.66289  -0.32000
mental1         -0.6629  1.00000   0.05698
physical1       -0.3200  0.05698   1.00000
```

If some of the variables include missing values, then we can add either `use = "complete.obs"` (listwise deletion of missing cases) or `use = "pairwise.complete.obs"` (pairwise deletion of missing cases) inside the `cor` function.

To test the significance of the correlation, we can use the `correlate` function from the `lsr` package:

```
# Activate the package
library("lsr")

correlate(medical[, c("depression1", "mental1", "physical1")],
          test = TRUE, corr.method="pearson")
```

```
CORRELATIONS
============
- correlation type:  pearson
- correlations shown only when both variables are numeric

            depression1    mental1    physical1
depression1           .     -0.663***    -0.320***
mental1         -0.663***        .          0.057
physical1       -0.320***    0.057              .

---
Signif. codes: . = p < .1, * = p<.05, ** = p<.01, *** = p<.001
```

```
p-VALUES
========
- total number of tests run:  3
- correction for multiple testing:  holm

            depression1 mental1 physical1
depression1           .   0.000     0.000
mental1           0.000       .     0.373
physical1         0.000   0.373         .


SAMPLE SIZES
============

            depression1 mental1 physical1
depression1         246     246       246
mental1             246     246       246
physical1           246     246       246
```

In addition to printing correlation matrices, **R** also provides nice ways to visualize correlations. In the following example, we will use the `corrplot` function from the `corrplot` package (Wei and Simko 2017) to draw a correlation matrix plot.

```
# Install and activate the package
install.packages("corrplot")
library("corrplot")
```

```
# First, we need to save the correlation matrix
cor_scores <- cor(medical[, c("depression1", "mental1", "physical1",
                              "depression2", "mental2", "physical2")])

# Plot 1 with circles
corrplot(cor_scores, method="circle")
```

```
# Plot 2 with colors
corrplot(cor_scores, method="color")
```

```
# Plot 3 with numbers
corrplot(cor_scores, method="number")
```

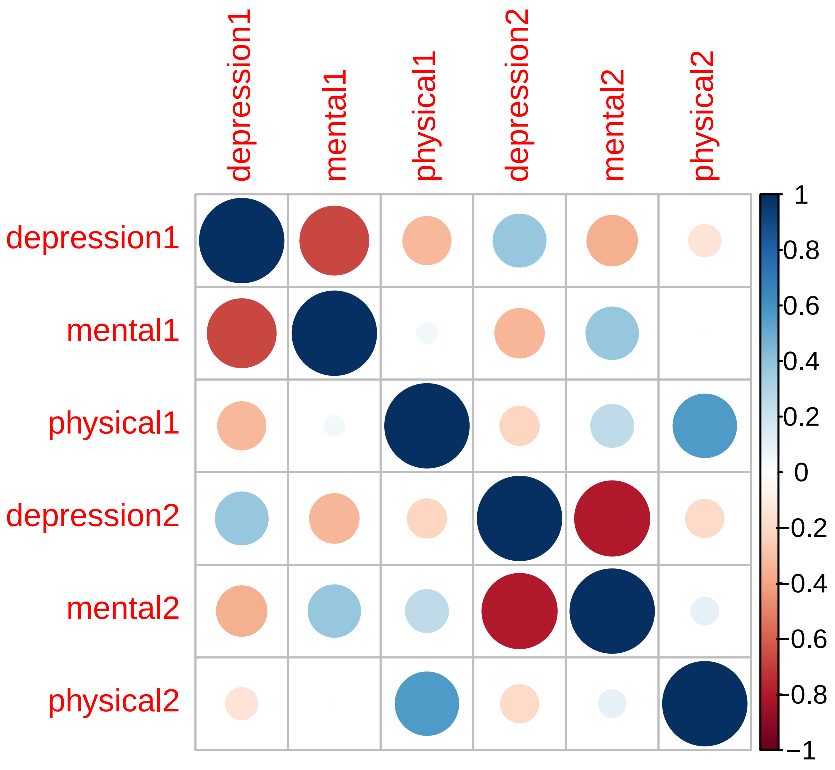


Figure 32: Correlation matrix plot with circles

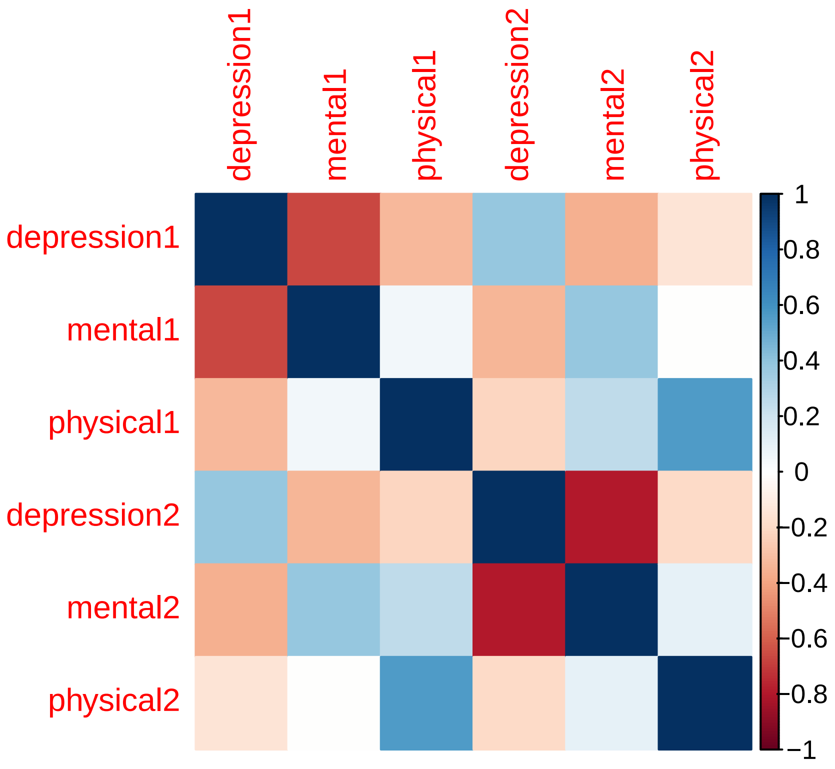


Figure 33: Correlation matrix plot with colours

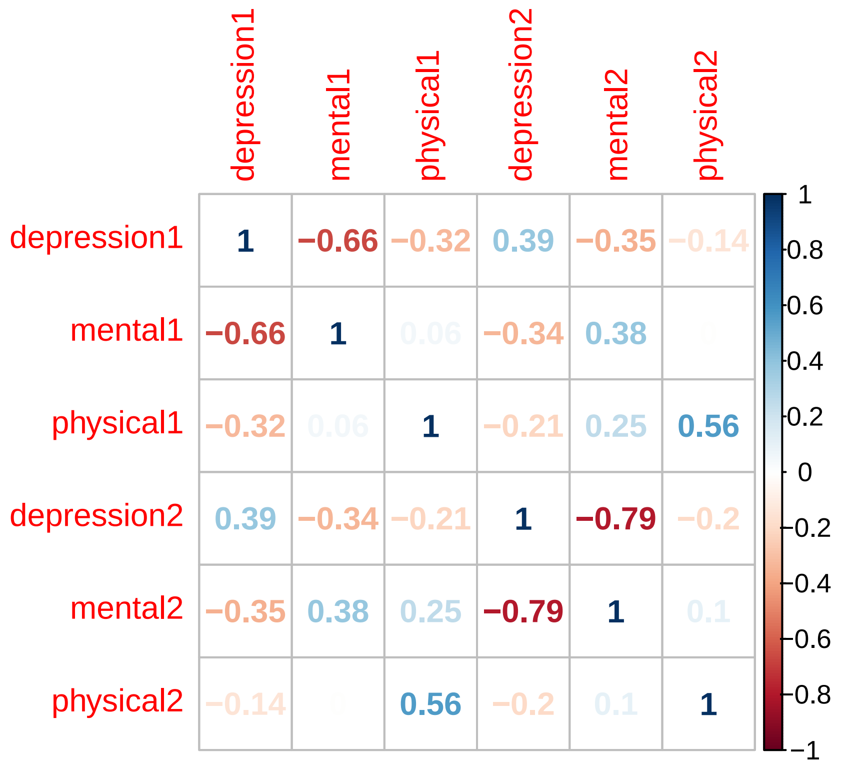


Figure 34: Correlation matrix plot with numbers

```
# Plot 4 with circles + lower triangular
corrplot(cor_scores, method="circle", type="lower")
```

```
# Plot 5 with circles + lower triangular + ordered correlations
corrplot(cor_scores, method="circle", type="lower", order="hclust")
```

Simple Linear Regression

Linear regression is a standard tool that researchers often rely on when analyzing the relationship between some predictors (i.e., independent variables) and an outcome (i.e., dependent variable). In a simple linear regression model, we aim to fit the best linear line to the data based on a single predictor so that the sum of residuals is the smallest. This method is known as “ordinaly least squares” (OLS). Once the regression equation is computed, we can use it to make predictions for a new sample of observations.

If Y is the dependent variable (DV) and X is the independent variable (IV), then the formula that describes our simple regression model becomes:

$$Y_i = b_1 X_i + b_0 + \epsilon_i$$

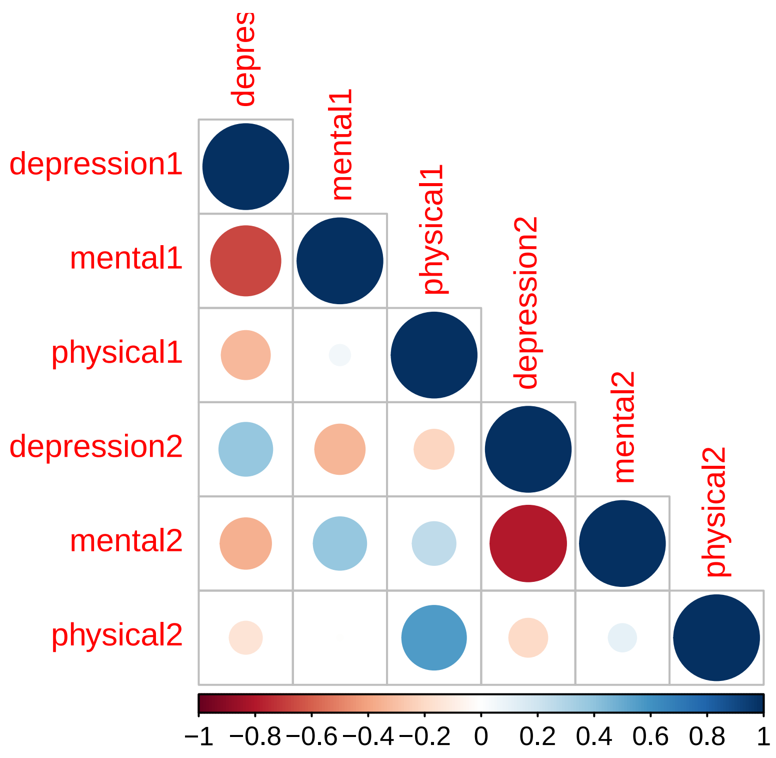


Figure 35: Correlation matrix plot with circle and lower triangle

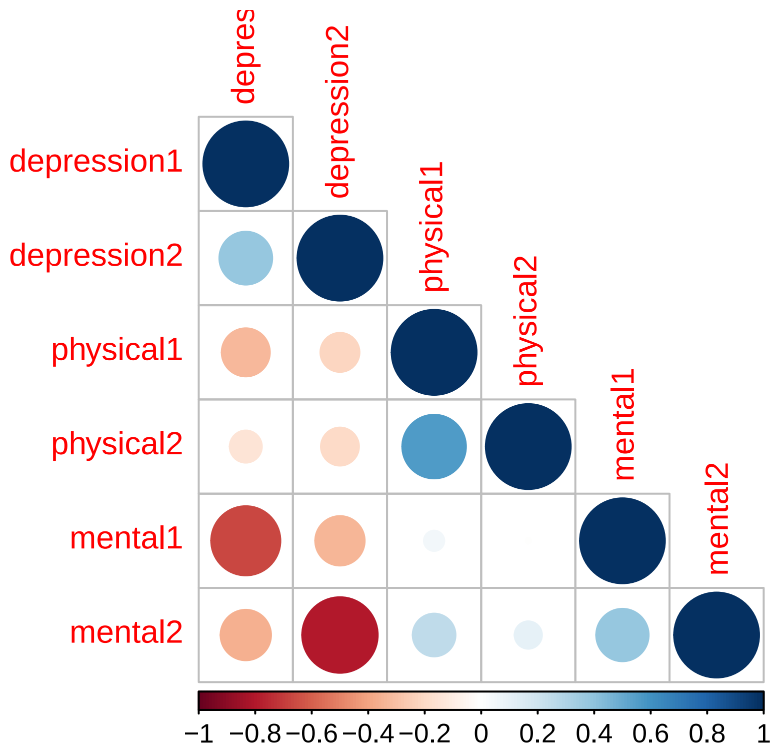


Figure 36: Correlation matrix plot with ordered correlations

where b_1 is the slope (the increase in Y for every one unit increase in X), b_0 is the intercept (the value of Y when $X = 0$), and ϵ_i is the residual (the difference between the predicted values based on the regression model and the actual values of the dependent variable).

In **R**, there are many ways to run regression analyses. However, the simplest way to run a regression model in **R** is to use the `lm()` function that fits a linear model to a given dataset (see `?lm` in the console for the help page). Here are the typical elements of `lm()`:

- `formula`: A formula that specifies the regression model. This formula is of the form DV ~ IV.
- `data`: The data frame containing the variables.

Example

Now let's see how regression works in **R**. We want to predict patients' mental scores at the baseline (i.e., `mental1`) using their depression scores at the baseline (i.e., `depression1`). To see how this relationship looks like, we can first take a look at the scatterplot as well as the correlation between the two variables.

```
cor(medical$depression1, medical$mental1)
```

```
[1] -0.6629
```

$r = -.66$ is indicating that there is a negative and moderate relationship between the two variables. Although we know that correlation does **NOT** mean causation, our knowledge or theory from the literature may suggest that these two variables are indeed associated with each other. So, we can move to the next step and plot these two variables in a scatterplot using the '`ggplot2` package (see http://sape.inf.usi.ch/quick-reference/ggplot2/colour for many colour options in `ggplot2`).

```
# Activate the ggplot2 first
library("ggplot2")

ggplot(data = medical,
       mapping = aes(x = depression1, y = mental1)) +
  geom_point(size = 2, color = "grey25") +
  geom_smooth(method = lm, color = "blue", se = TRUE) +
  labs(x = "Depression scores",
       y = "Mental scores",
       title = "Depression and mental scores at the baseline") +
  theme_bw()
```

The scatterplot also confirms that there is a negative relationship between the two variables. Now we can fit a regression model to the data to quantify this relationship.

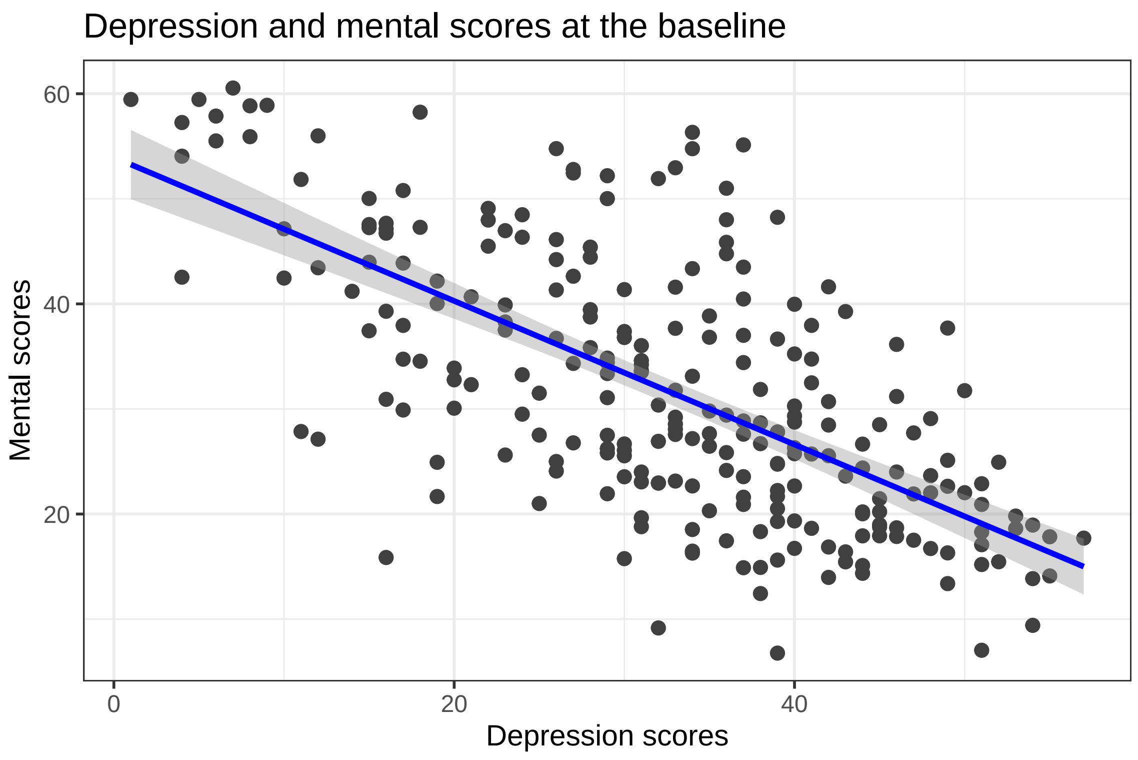


Figure 37: Scatterplot of depression and mental scores at the baseline

```r
# Set up the model and save it as model1
model1 <- lm(mental1 ~ depression1, data = medical)

# Print basic model output
print(model1)
```

```

Call:
lm(formula = mental1 ~ depression1, data = medical)

Coefficients:
(Intercept)  depression1
     53.948       -0.683
```

```r
# Print detailed summary of the model
summary(model1)
```

```

Call:
lm(formula = mental1 ~ depression1, data = medical)

Residuals:
    Min      1Q  Median      3Q     Max
-27.152  -6.993   0.039   5.853  26.464

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  53.9477     1.7173    31.4   <2e-16 ***
depression1  -0.6834     0.0494   -13.8   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.37 on 244 degrees of freedom
Multiple R-squared:  0.439, Adjusted R-squared:  0.437
F-statistic:  191 on 1 and 244 DF,  p-value: <2e-16
```

The output shows that our regression equation is:

$$\hat{mental1} = 53.9477 - 0.6834(depression1)$$

suggesting that one unit increase in the depression scores corresponds to -0.6834 points increase in the mental scores at the baseline.

`t value` in the output indicates the individual t tests for testing whether intercept and slope are significantly different from zero; i.e., $H_0 = b_0 = 0$ and $H_0 = b_1 = 0$. The test for the intercept is not really interesting as we rarely care about whether or not the intercept is zero. However, for the slope, we want this test to be significant in order to conclude that the predictor is indeed useful for predicting the dependent variable. In our example, $t = -13.8$ for the slope and its p-value is less than .001. This indicates that the slope was significantly different from zero (i.e., an important predictor in our model). In the output `***` shows the level of significance.

Another important information is R^2, which indicates the proportion of the variance explained by the predictor (`depression1`) in the dependent variable (`mental1`). In our model, $R^2 = 0.439$ – which means 43.9 of the variance in the mental scores can be explained by the depression scores.

There are many guidelines for categorizing for interpreting R-squared values. Researchers often refers to Cohen's guidelines as shown below:

R-squared	Interpretation
0.1	Small
0.09	Moderate
0.25	Large

Using these guidelines, we can say that our model indicates a large effect between the dependent and independent variables.

Finally, the output has additional information about the overall significance of the regression model: $F(1, 244) = 191, p < .001$, suggesting that the model is statistically significant. This information may not be useful when we have a simple regression model because we already know that our predictor is significantly predicts the dependent variable. However, this information will be more useful when we look at multiple regression where only some variables might be significant, not all of them.

Multiple Regression

The simple linear regression model that we have discussed up to this point assumes that there is a single predictor variable that we are interested in. However, in many (perhaps most) research projects, we actually have multiple predictors that we want to examine. Therefore, we can extend the linear regression framework to be able to include multiple predictors – which is called **multiple regression**.

Multiple regression is conceptually very simple. All we do is to add more predictors into our regression equation. Let's suppose that we are interested in predicting the mental scores at the baseline (`mental1`) using both the depression scores at the baseline (`depression1`), the physical scores at the baseline (`physical1`), and sex (`sex`). Our new regression equation should look like this:

$$\hat{mental1} = b_0 + b_1(depression1) + b_2(physical1) + b_3(sex)$$

We would hope that the additional variables we include in the model will make our regression model more precise. In other words, we will be able to better predict the mental scores with the help of these predictors. The caveat is that these two variables should be correlated with the dependent variable (i.e., mental scores) but they should not be highly correlated with each other or the other predictor (i.e., depression scores). If this is the case, adding new variables would not bring additional value to the regression model. Rather, the model would have some redundant variables.

Example

Now let's see how this will look like in **R**.

```
# Add physical1 into the previous model
model2 <- lm(mental1 ~ depression1 + physical1, data = medical)

# Print detailed summary of the model
summary(model2)
```

```
Call:
```

```
lm(formula = mental1 ~ depression1 + physical1, data = medical)

Residuals:
   Min     1Q Median     3Q    Max 
-23.51  -6.30  -0.45   6.11  26.79 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  64.9234     3.5619   18.23  < 2e-16 ***
depression1  -0.7404     0.0510  -14.52  < 2e-16 ***
physical1    -0.1920     0.0549   -3.49  0.00057 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.16 on 243 degrees of freedom
Multiple R-squared:  0.466, Adjusted R-squared:  0.462 
F-statistic:  106 on 2 and 243 DF,  p-value: <2e-16
```

The output shows that our new regression equation is:

$$\hat{mental1} = 64.92 - 0.74(depression1) - 0.19(physical1)$$

Both predictors negatively predict the mental scores and they are statistically significant in the model. Our R-squared value increased to $R^2 = 0.466$, suggesting that 46.6 of the variance can be explained by the two predictors that we have in the model.

By looking at the slopes for each predictor, we **cannot** tell which predictor plays a more important role in the prediction. Therefore, we need to see the standardized slopes – which are directly comparable based on their magnitudes. We can use the `jtools` package (Long 2021) to visualize the standardized slopes. To use the `jtools` package, we have to install both `jtools` and `ggstance`.

```
# Install and activate the packages
install.packages(c("jtools", "ggstance"))
library("jtools")
```

We will use the `plot_summ` function from the `jtools` package to visualize the slopes:

```
plot_summs(model2, scale = TRUE)
```

In the plot, the circles in the middle show the location of the standardized slopes (i.e., standardized regression coefficients) for both predictors and the line around the circles represent the confidence interval. We can see from Figure 38 that the standardized slope for `depression1` is much larger (around -8.5) whereas the standardized slope for `physical1` is

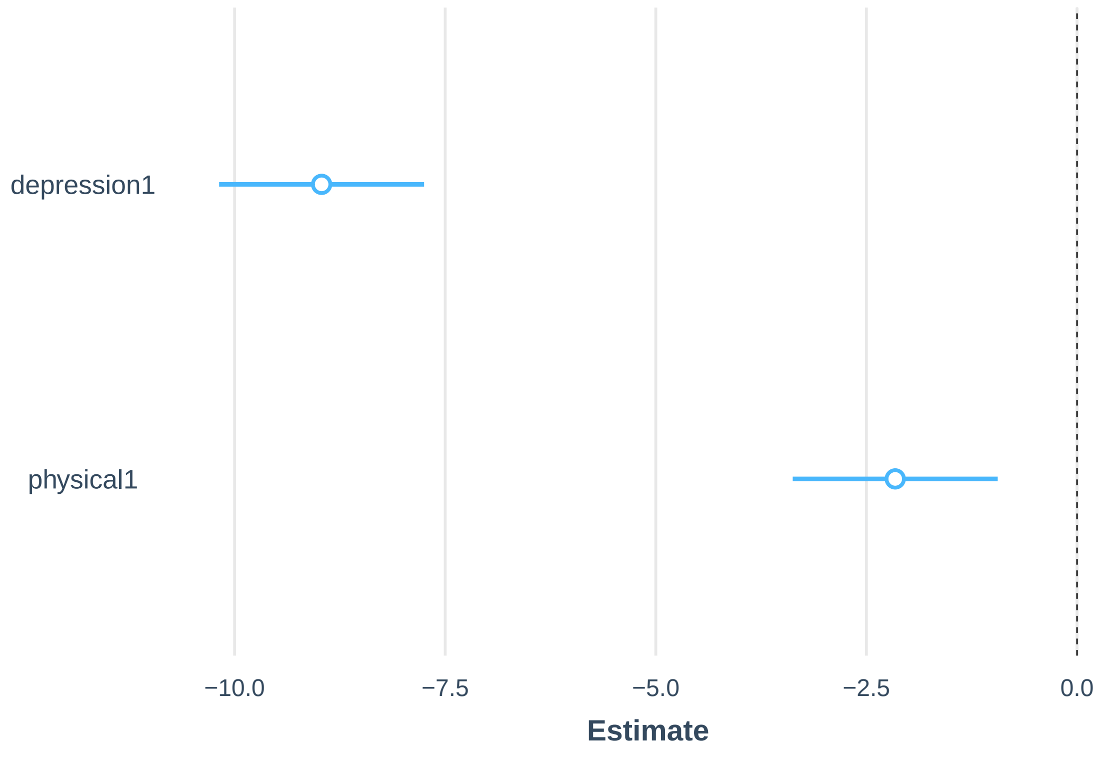


Figure 38: Standardized regression coefficients in Model 2

smaller (around -2). So, we could say that the `depression1` is a stronger predictor than `physical1` in predicting `mental1`.

Let's expand our model by adding `sex`. Because `sex` is a categorical variable, **R** will choose one level of this variable as the reference category and give the results for the other category. The default reference category is selected alphabetically. In the context of `sex`, the values are male and female. Because **f** alphabetically comes first, female will be selected as the reference category and we will see the results for male.

```
# Add sex into the previous model
model3 <- lm(mental1 ~ depression1 + physical1 + sex, data = medical)

# Print detailed summary of the model
summary(model3)
```

```
Call:
lm(formula = mental1 ~ depression1 + physical1 + sex, data = medical)

Residuals:
    Min      1Q  Median      3Q     Max
-23.538  -6.409  -0.314   6.040  26.712
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  64.6403     3.7293   17.33  < 2e-16 ***
depression1  -0.7386     0.0515  -14.33  < 2e-16 ***
physical1    -0.1932     0.0552   -3.50  0.00056 ***
sexmale       0.3689     1.4105    0.26  0.79390
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.18 on 242 degrees of freedom
Multiple R-squared:  0.466, Adjusted R-squared:  0.46
F-statistic: 70.5 on 3 and 242 DF,  p-value: <2e-16
```

The output shows that the estimated slope for `sexmale` is 0.37; but this slope is not statistically significant as its p-value is 0.79. We could probably conclude that `sex` does not explain any further variance in the model and therefore we can remove it from the model to keep our regression model simple.

So far we tested three models: Model 1, Model 2, and Model 3. Because the models are nested within each other (i.e., we incrementally added new variables), we can make a model comparison to see if adding those additional predictors brought significant added-value to the models with more predictors. We will use the `anova` function in base **R** to accomplish this. This function does not necessarily run the same ANOVA that we discussed earlier. It will compare the models based on their R-squared values.

```
anova(model1, model2, model3)
```

```
Analysis of Variance Table

Model 1: mental1 ~ depression1
Model 2: mental1 ~ depression1 + physical1
Model 3: mental1 ~ depression1 + physical1 + sex
  Res.Df   RSS Df Sum of Sq     F  Pr(>F)
1    244 21411
2    243 20387  1      1024 12.16 0.00058 ***
3    242 20381  1         6  0.07 0.79390
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the output, there are two comparisons:

- Model 1 vs. Model 2
- Model 2 vs. Model 3

The comparison of Model 1 and Model 2 shows that $F(1, 1024) = 12.16, p < .001$, suggesting that the larger model (Model 2) explains significantly more variance than the smaller model (Model 1).

The comparison of Model 2 and Model 3 shows that $F(1, 6) = 0.07, p = .793$, suggesting that the larger model (Model 3) does **NOT** explain significantly more variance than the smaller model (Model 2), which confirms our earlier finding that `sex` did not bring additional value to the model.

We can also compare the three models visually.

```
plot_summs(model1, model2, model3, scale = TRUE)
```

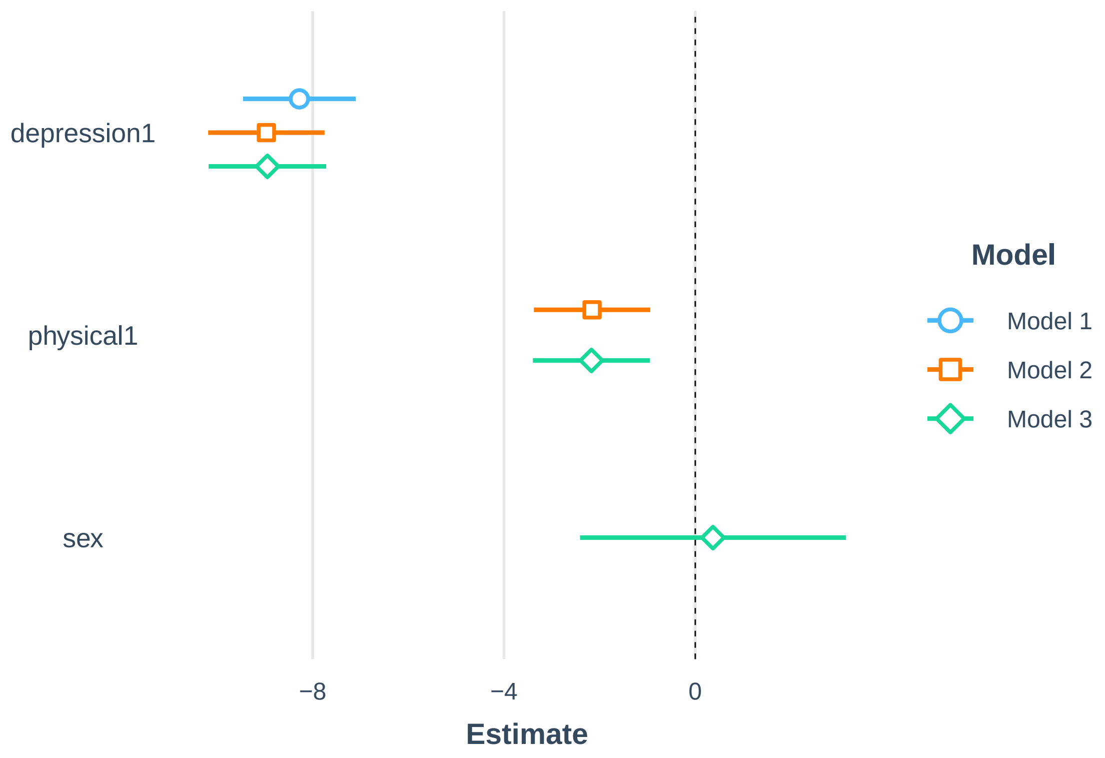


Figure 39: Comparison of three models

Exercise 10

Run a multiple regression model where your dependent variable will be `depression2` (i.e., depression scores after 6 months) and your predictors will be:

- `depression1` (i.e., depression scores at the baseline)
- `treat1` (i.e., whether the patient received treatment)

- `sex` (i.e., patients' sex)

The questions we can answer are:

1. Which variables are significant?
2. Do we need all three predictors?
3. How is the R-squared value for the model?

Additional Packages to Consider

There are also additional **R** packages that might be useful when conducting regression analysis. In the last chapter, we used **report** (https://github.com/easystats/report) to produce reports of *t*-tests. We can use the same package to create reports of regression models.

```
# Activate the package
library("report")

model <- lm(mental1 ~ depression1 + physical1 + sex, data = medical)

# Print model parameters and additional information
report(model)
```

```
We fitted a linear model (estimated using OLS) to predict mental1 with depression1, phys

  - The effect of depression1 is significantly negative (beta = -0.74, 95% CI [-0.84, -0
  - The effect of physical1 is significantly negative (beta = -0.19, 95% CI [-0.30, -0.0
  - The effect of sex [male] is non-significantly positive (beta = 0.37, 95% CI [-2.41,

Standardized parameters were obtained by fitting the model on a standardized version of
```

```
# Or, print out specific parts of the model output
report_model(model)
```

```
linear model (estimated using OLS) to predict mental1 with depression1, physical1 and se
```

```
report_performance(model)
```

```
The model explains a significant and substantial proportion of variance (R2 = 0.47, F(3,
report_statistics(model)
```

```
beta = 64.64, 95% CI [57.29, 71.99], t(242) = 17.33, p < .001; Std. beta = -0.02, 95% CI
beta = -0.74, 95% CI [-0.84, -0.64], t(242) = -14.33, p < .001; Std. beta = -0.72, 95% C
beta = -0.19, 95% CI [-0.30, -0.08], t(242) = -3.50, p < .001; Std. beta = -0.17, 95% CI
beta = 0.37, 95% CI [-2.41, 3.15], t(242) = 0.26, p = 0.794; Std. beta = 0.03, 95% CI [-
```

Another useful package for reporting regression results is **parameters** (https://github.com/easystats/parameters). The goal of this package is to facilitate and streamline the process of reporting results of statistical models, which includes the easy and intuitive calculation of standardized estimates or robust standard errors and p-values.

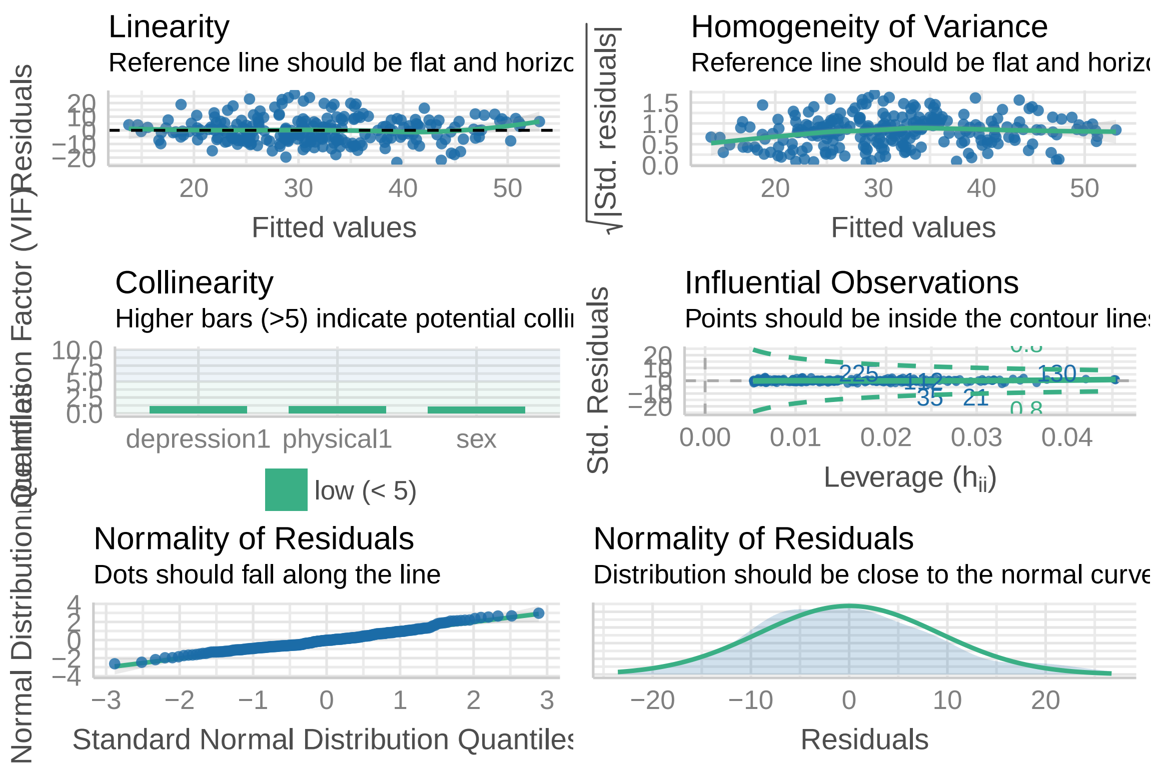
```
# Install the activate the package
install.packages("parameters")
library("parameters")

# To view regular model parameters
model_parameters(model)

# To view standardized model parameters
model_parameters(model, standardize = "refit")
```

```
Parameter   | Coefficient |   SE |         95% CI | t(242) |      p
-------------------------------------------------------------------
(Intercept) |       64.64 | 3.73 | [57.29, 71.99] |  17.33 | < .001
depression1 |       -0.74 | 0.05 | [-0.84, -0.64] | -14.33 | < .001
physical1   |       -0.19 | 0.06 | [-0.30, -0.08] |  -3.50 | < .001
sex [male]  |        0.37 | 1.41 | [-2.41,  3.15] |   0.26 | 0.794

Parameter   | Coefficient |   SE |         95% CI | t(242) |      p
-------------------------------------------------------------------
(Intercept) |       -0.02 | 0.10 | [-0.22,  0.17] |  -0.23 | 0.818
depression1 |       -0.72 | 0.05 | [-0.81, -0.62] | -14.33 | < .001
physical1   |       -0.17 | 0.05 | [-0.27, -0.08] |  -3.50 | < .001
sex [male]  |        0.03 | 0.11 | [-0.19,  0.25] |   0.26 | 0.794
```

Lastly, **performance** (https://github.com/easystats/performance) is another useful package for evaluating the quality of model fit for regression models. The package provides several fit indices to evaluate model fit. Also, it has data visualization tools that facilitate model diagnostics.

```
# Install the activate the package
install.packages("performance")
library("performance")

# Model performance summaries
model_performance(model)

# Check for heteroskedasticity
check_heteroscedasticity(model)

# Comprehensive visualization of model checks
check_model(model)
```

```
# Indices of model performance

AIC      |      BIC |    R2 | R2 (adj.) |  RMSE | Sigma
-------------------------------------------------------
1794.707 | 1812.234 | 0.466 |     0.460 | 9.102 | 9.177

OK: Error variance appears to be homoscedastic (p = 0.346).
```



Additional R Features

As we conclude this training, I also want to mention a few additional features that you might find useful when analyzing and visualizing data using **R**.

Removing Objects from the Workspace

When you define an object (e.g., a dataset) within an **R** session, it stays in the workspace until you close **R**. However, if you decide to delete a particular object while still using **R**, one of the following methods can be used:

```r
x <- 1:100

# Use rm() to remove x entirely
rm(x)

# Or, turn x into NULL (deletes the content of x)
x <- NULL
```

Saving Output in R

There are several ways to save the output of your analysis in R (i.e., everything printed on the console section). The following are *some* of the options to save output in R:

Using the `sink` Function

Using the `sink` Function, it is possible to create an empty text document (called my_output.txt in the following example) and capture the output printed in the console. All the output, including warning messages, in the console is saved into the document once the final `sink()` function is executed.

```r
# Start saving the output
sink("my_output.txt")

fit <- lm(mpg ~ wt, data = mtcars)
summary(fit)

# Stop saving the output
sink()
```

Saving Objects from an R Session

If you want to save on particular object that you made (e.g., a dataset) for use in **R** later, you can use the `save` function.

```r
mydata <- data.frame(person = 1:10,
                     age = sample.int(30, 10))

# Save the data
save(mydata, file="myfile.Rdata")

# Load the data when needed again
load("myfile.Rdata")
```

Alternatively, you can save the entire **R** session using `save.image()`. This saves all of the objects to a file ".RData" in your working directory.

```r
mydata1 <- data.frame(person = 1:10,
                      age = sample.int(30, 10))

mydata2 <- data.frame(person = 1:10,
                      weight = rnorm(10, 75, 4))

x <- 1:100

# It saves all objects in one place
save.image(file = "myfile.Rdata")

# Load the data when needed again
load("myfile.Rdata")
```

Saving the History of an R Session

If you want to save all the commands you used in a session, then you can use the `savehistory` function. This is helpful because it allows you to browse the history (i.e., all the commands or functions you have entered) when you load the same history into R in the future. You simply press the up-arrow and down-arrow keys to view the commands you typed earlier.

```
# Save the history
savehistory(file = "my_analysis.Rhistory")

# Load the history when needed again
loadhistory("my_analysis.Rhistory")
```

Using R Markdown

R Markdown (https://rmarkdown.rstudio.com/) is an excellent way to save all the output from R (e.g., codes, output, data visualizations) in a nice, readable format. R Markdown allows users to turn their analysis into high quality documents, which then can be exported as an HTML file, a PDF, or a Word document. To see how R Markdown works, check out this nice tutorial: https://rmarkdown.rstudio.com/lesson-1.html

Unloading a Package

Sometimes you may want to deactivate or unload some packages that you have do not need for your analysis anymore. The `detach` function allows users to remove a particular package from an **R** session. For example, let's assume that we activated the **dplyr** package but realized that we do not need this particular package anymore. The following line will remove the package from our session (i.e., the package will be deactivated):

```
detach("package:dplyr", unload=TRUE)
```

Updating R and R Packages

Every now and then the authors of packages release updated versions of their packages. The updated versions add new functions, fix bugs, and so on. Therefore, it might be a good idea to update your packages periodically. To update a particular package, we can use the `update.packages` function.

```
# Update the dplyr package
update.packages("dplyr")
```

About every six months or so, a new version of R is released. It is not possible to update **R** from within RStudio. To get the new version, you can go to the CRAN website (https://cran.r-project.org/) and download the most recent version of R. Alternatively, you can update R using the R GUI (not RStudio). The following commands will install the **installr** package, download and install the latest version of **R**, and finally carry all of the packages in your library to the new version of **R**.

```r
# Install and activate the package
install.packages("installr")
library("installr")

# Update R with default options
updateR(TRUE)

# Or, installr asks what you want at each step
updateR()
```

References

Fox, John, Sanford Weisberg, and Brad Price. 2020. *Car: Companion to Applied Regression.* https://CRAN.R-project.org/package=car.

Long, Jacob A. 2021. *Jtools: Analysis and Presentation of Social Scientific Data.* https://jtools.jacob-long.com.

Navarro, Daniel. 2015. *Lsr: Companion to "Learning Statistics with r".* http://health.adelaide.edu.au/psychology/ccs/teaching/lsr/.

R Core Team. 2021. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Sarkar, Deepayan. 2020. *Lattice: Trellis Graphics for r.* http://lattice.r-forge.r-project.org/.

Waring, Elin, Michael Quinn, Amelia McNamara, Eduardo Arino de la Rubia, Hao Zhu, and Shannon Ellis. 2020. *Skimr: Compact and Flexible Summaries of Data.* https://CRAN.R-project.org/package=skimr.

Wei, Taiyun, and Viliam Simko. 2017. *Corrplot: Visualization of a Correlation Matrix.* https://github.com/taiyun/corrplot.

Wheelan, Charles. 2013. *Naked Statistics: Stripping the Dread from Data.* New York, NY: Norton; Company.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2020. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics.* https://CRAN.R-project.org/package=ggplot2.