Statistical Inference For Data Science

Saul Diaz-Infante

1/16/23

Table of contents

Preface **6**

Introduction **7**
The tidyverse . . . 7

I R fundamentals in programming, data managment and visualization **9**

Nuts and bolts: Data types **11**
Entering Input: the assigment operator . . . 11

Intro to basics **12**
How it works . . . 12
Instructions `100 XP` . . . 12
Arithmetic with R . . . 12
Instructions `100 XP` . . . 13
Variable assignment . . . 13
Variable assignment (2) . . . 14
Variable assignment (3) . . . 15
Instructions `100 XP` . . . 15
Apples and oranges . . . 15
Instructions `100 XP` . . . 16
Basic data types in R . . . 16
Instructions `100 XP` . . . 17
What's that data type? . . . 18
Instructions `100 XP` . . . 18

Vectors **19**
Create a vector . . . 19
Instructions `100 XP` . . . 19
Create a vector (2) . . . 19
Create a vector (3) . . . 20
Naming a vector . . . 21
Naming a vector (2) . . . 22
Calculating total winnings . . . 23
Calculating total winnings (2) . . . 24

Calculating total winnings (3) . . . 25
Comparing total winnings . . . 26
Instructions 100 XP . . . 26
Vector selection: the good times . . . 26
Instructions 100 XP . . . 27
Vector selection: the good times (2) . . . 27
Instructions 100 XP . . . 28
Vector selection: the good times (3) . . . 28
Instructions 100 XP . . . 28
Vector selection: the good times (4) . . . 29
Instructions 100 XP . . . 29
Selection by comparison - Step 1 . . . 30
Instructions 100 XP . . . 30
Selection by comparison - Step 2 . . . 31
Instructions 100 XP . . . 31
Advanced selection . . . 32
Instructions 100 XP . . . 32

Matrices **33**
What's a matrix? . . . 33
Instructions 100 XP . . . 33
Analyze matrices, you shall . . . 34
Instructions 100 XP . . . 34
Naming a matrix . . . 34
Instructions 100 XP . . . 35
Calculating the worldwide box office . . . 35
Instructions 100 XP . . . 36
Adding a column for the Worldwide box office . . . 36
Instructions 100 XP . . . 37
Adding a row . . . 37
Instructions 100 XP . . . 37
The total box office revenue for the entire saga . . . 38
Instructions 100 XP . . . 38
Selection of matrix elements . . . 38
Instructions 100 XP . . . 39
A little arithmetic with matrices . . . 40
Instructions 100 XP . . . 40
A little arithmetic with matrices (2) . . . 40
Instructions 100 XP . . . 41

Factors **42**
What's a factor and why would you use it? . . . 42
Instructions 100 XP . . . 42

What's a factor and why would you use it? (2) . . . 43
Instructions 100 XP . . . 43
What's a factor and why would you use it? (3) . . . 43
Instructions 100 XP . . . 44
Factor levels . . . 44
Instructions 100 XP . . . 45
Summarizing a factor . . . 45
Instructions 100 XP . . . 46
Battle of the sexes . . . 46
Instructions 100 XP . . . 46
Ordered factors . . . 47
Instructions 100 XP . . . 47
Ordered factors (2) . . . 48
Instructions 100 XP . . . 48
Comparing ordered factors . . . 49
Instructions 100 XP . . . 49

Data frames **50**
What's a data frame? . . . 50
Instructions 100 XP . . . 50
Quick, have a look at your dataset . . . 51
Instructions 100 XP . . . 52
Have a look at the structure . . . 52
Instructions 100 XP . . . 52
Creating a data frame . . . 53
Instructions 100 XP . . . 53
Creating a data frame (2) . . . 54
Instructions 100 XP . . . 54
Selection of data frame elements . . . 54
Instructions 100 XP . . . 54
Selection of data frame elements (2) . . . 55
Instructions 100 XP . . . 55
Only planets with rings . . . 55
Instructions 100 XP . . . 56
Only planets with rings (2) . . . 56
Instructions 100 XP . . . 57
Only planets with rings but shorter . . . 57
Instructions 100 XP . . . 57
Sorting . . . 58
Instructions 100 XP . . . 58
Sorting your data frame . . . 59
Instructions 100 XP . . . 59

Best Coding Practices for R **60**
What we mean when say "better coding practice" . . . 60
Folder Structure . . . 61
Code Structure . . . 61
Sections . . . 61
Structural Composition . . . 61
Identation . . . 61
Styling . . . 61
Final Comments . . . 61

Tidyverse Fundamentals with R **62**
Introduction to Tidyverse . . . 62
Reshaping Data with tidyr . . . 62
Project . . . 62
Modeling with Data in the Tidyverse . . . 62
Communication with Data in the Tidyverse . . . 62
Categorical Data in the Tydiverse . . . 62

Data Manipulation **63**
Data Manipulation with dplyr . . . 63
Joining Data with dplyr . . . 63
Case Study: Exploratory Data Analysis in R . . . 63
Data Manipulation with data.table in R . . . 63
Joining Data with data.table in R . . . 63

1 Data visualization with ggplot2 and friends **64**

II The whole game of statistical Inference **65**

2 Statistical Inference with resampling: Bootstrap and Jacknife. **67**
2.1 Likelihood inference. . . . 67
2.2 Variance analisys. . . . 67
2.3 ROC Curves . . . 67

3 Lienar Regression **68**
3.1 Linear Regression . . . 68
3.2 Multiple linear_regression and generalized linear regresion . . . 68

4 Summary **69**

References **70**

Preface

Who I am. I am Saul Diaz Infante Velasco. I just starting as assistant professor at the Data Science graduate program of Universidad de Sonora at Hermosillo Mexico. My Background is related with numerical analysis and stochastic models. I'm are a enthusiastic of this treading topic called Data-Science, but perhaps at the moment I only have just intuition about what really it is. However, I have been programming almost 20 years an moved from old programming langues as FORTRAN, Pascal, Basic, Cobol, C, C++ to the new well established treading development workflows like R, Python and Julia. This is my firs attempt in R.

More of this book is work in progress. We aim to provide material as well we review and improve our basic skills to face the study of the most popular methods in Machine learning. Thus the book try to cover fundamentals in R programming as data types, flow control structures and put particular importance in the well practice of coding functions. Then we moves to the management of data whit dplyr and other packages. To finish this part we discuss some package to visualize data. Then, we face the problem o estimation and explore techniques based on bootstrap–and another sampling flavors.

This book has been started on January, 2023 as part of a course to the Master on Data Science from Universidad de Sonora.

I'm writing this book to follow a path of self learning, understanding and and joy for this matter called Data Science. I'm not try to become and expert instead I just pursuit the joy of the interaction of math, computational sciences and the generosity of this virtuous learning-teaching process.

https://sauldiazinfante.github.io/statisticalInferenceinR4DS/

Introduction

The focus of this course is int the programming and basic techniques for inference that are usually applied in data science. We start by reviewing and enforcing programming skills. Then we will use the database of entomological data practice and build the required bases for more structured tools like bootstrap or Jackknife cuts.

Figure 1 further explores the impact of temperature on ozone level.

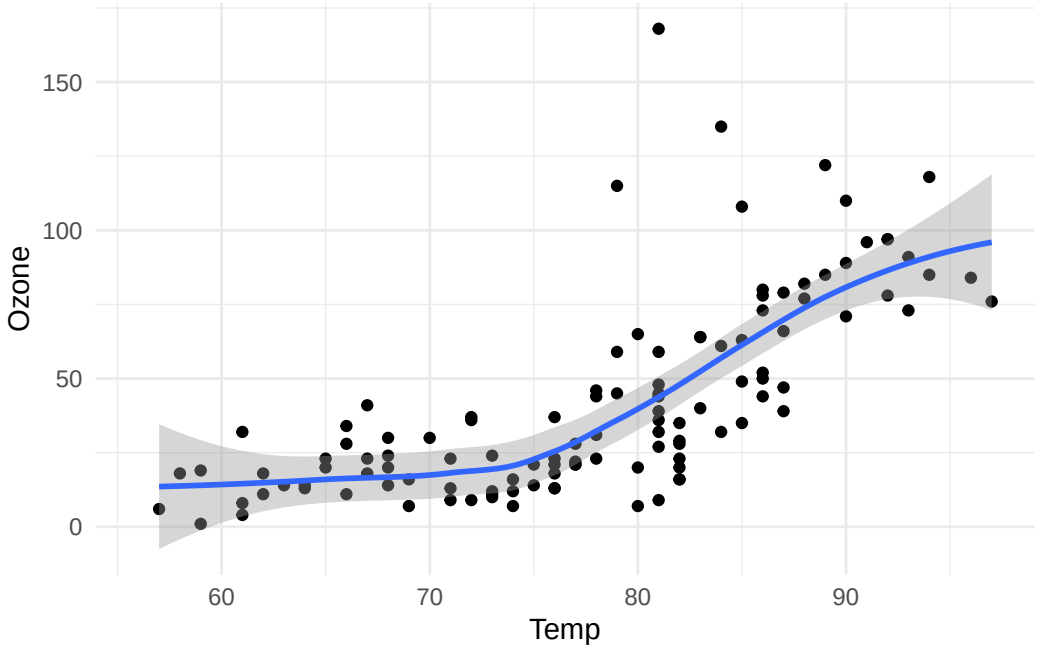

Figure 1: Temperature and ozone level.

The tidyverse

We need to install a R package. The majority of the packages that we will use are part of the so-called tidyverse package. The packages in the tidyverse share a common philosophy of data and R programming, and are designed to work together naturally.

You can install the complete tidyverse with the line of code:

then we can use it by loading in the preamble section with

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.0      v stringr 1.5.0
v readr   2.1.2      v forcats 0.5.2
v purrr   0.3.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

see https://www.tidyverse.org/ documentation.

Part I

R fundamentals in programming, data managment and visualization

We dedicate this part to overview the basics to program in R. The aim of this part is building the basis for Machine learning, namely **data visualization**, **data manipulation** and the **good coding practices** to type script of industrial production quality.

Nuts and bolts: Data types

Entering Input: the assigment operator

The thing that we type on the R console prompt are expressions. The firs expression we discuses here is the assignment operator, please watch the following video https://www.youtube.com/watch?v=vGY5i_J2c-c&t=283s

At the R console, any executable typed text that we put a side of the prompt are called expressions. We start by the `<-` symbol is the assignment operator.

```
[1] 0
```

```
[1] 0
```

```
[1] "what's up"
```

The `[1]` shown in the output indicates that `x` is a `vector` and 0 is the element at position with `index` 1.

Intro to basics

How it works

In the text editor you should type R code to solve the exercises. When you hit **ctrl + enter**, every line of code is interpreted and executed by R and you get a message whether or not your code was correct.

R makes use of the # sign to add comments, so that you and others can understand what the R code is about. Comments are not run as R code, so they will not influence your result. For example, Calculate 3 + 4 in the editor on the right is a comment.

You can also execute R commands straight in the console. This is a good way to experiment with R code.

Instructions `100 XP`

```
- In the text editor on the right there is already some sample code.
- Can you see which lines are actual R code and which are comments?
- Add a line of code that calculates the sum of 6 and 12,
    and hit the enter button
```

ex_01.R

```
# Calculate 3 + 4
3 + 4
# Calculate 6 + 12
6 + 12
```

Arithmetic with R

In its most basic form, R can be used as a simple calculator. Consider the following arithmetic operators:

- Addition: `+`

- Subtraction: `-`
- Multiplication: `*`
- Division: `/`
- Exponentiation: `^`
- Modulo: `%%`

The last two might need some explaining:

- The `^` operator raises the number to its left to the power of the number to its right: for example `3^2` is 9.
- The modulo returns the remainder of the division of the number to the left by the number on its right, for example 5 modulo 3 or `5 %% 3` is 2.

Instructions `100 XP`

```
- Type 2^5 in the editor to calculate 2 to the power 5.
- Type 28 %% 6 to calculate 28 modulo 6.
- Run the answer in the console and have a look at the R output .
- Note how the # symbol is used to add comments on the R code.
```

ex_02.R

```
# An addition
5 + 5

# A subtraction
5 - 5

# A multiplication
3 * 5
    # A division
(5 + 5) / 2
# Exponentiation
2 ^ 5
# Modulo
28 %% 6
```

Variable assignment

A basic concept in (statistical) programming is called a variable.

A variable allows you to store a value (e.g. 4) or an object (e.g. a function description) in R. You can then later use this variable's name to easily access the value or the object that is stored within this variable.

> You can assign a value 4 to a variable my_var with the command
>
> my_var <- 4

Instructions 100 XP

Over to you: complete the code in the editor such that it assigns the value 42 to the variable x in the editor. Submit the answer. Notice that when you ask R to print x, the value 42 appears.

ex_03.R

```
# Assign the value 42 to x
x <- 42
# Print out the value of the variable x
print (x)
```

Variable assignment (2)

Suppose you have a fruit basket with five apples. As a data analyst in training, you want to store the number of apples in a variable with the name my_apples.

Instructions 100 XP

- Type the following code in the editor: my_apples <- 5. This will assign the value 5 to my_apples.
- Type: my_apples below the second comment. This will print out the value of my_apples.
- Run your answer, and look at the output: you see that the number 5 is printed. So R now links the variable my_apples to the value 5.

ex_04.R

```
# Assign the value 5 to the variable my_apples
my_apples <- 5
# Print out the value of the variable my_apples
```

```
print(my_apples)
```

Variable assignment (3)

Every tasty fruit basket needs oranges, so you decide to add six oranges. As a data analyst, your reflex is to immediately create the variable my_oranges and assign the value 6 to it. Next, you want to calculate how many pieces of fruit you have in total. Since you have given meaningful names to these values,

> you can now code this in a clear way:
>
> ```
> my_apples + my_oranges
> ```

Instructions 100 XP

- Assign to my_oranges the value 6.
- Add the variables my_apples and my_oranges and have R simply print the result.
- Assign the result of adding my_apples and my_oranges to a new variable my_fruit.

ex_05.R

```
# Assign a value to the variables my_apples and my_oranges
my_apples <- 5
my_oranges <- 6

# Add these two variables together
my_apples + my_oranges

# Create the variable my_fruit
my_fruit <- my_apples + my_oranges
```

Apples and oranges

Common knowledge tells you not to add apples and oranges. But hey, that is what you just did, no :-)? The my_apples and my_oranges variables both contained a number in the previous exercise. The + operator works with numeric variables in R. If you really tried to add "apples" and "oranges", and assigned a text value to the variable my_oranges (see the

editor), you would be trying to assign the addition of a numeric and a character variable to the variable my_fruit. This is not possible.

Instructions `100 XP`

- Run the answer and read the error message. Make sure to understand why this did not work.
- Adjust the code so that R knows you have 6 oranges and thus a fruit basket with 11 pieces of fruit.

ex_06.R

```
# Assign a value to the variable my_apples
my_apples <- 5
# Fix the assignment of my_oranges
my_oranges <- "six"
# Create the variable my_fruit and print it out
my_fruit <- my_apples + my_oranges
my_fruit
```

Response

ex_06.R

```
# Assign a value to the variable my_apples
my_apples <- 5
# Fix the assignment of my_oranges
my_oranges <- 6
# Create the variable my_fruit and print it out
my_fruit <- my_apples + my_oranges
my_fruit
```

Basic data types in R

R works with numerous data types. Some of the most basic types to get started are:

- Decimal values like 4.5 are called numerics.
- Whole numbers like 4 are called integers. Integers are also numerics.
- Boolean values (TRUE or FALSE) are called logical.
- Text (or string) values are called characters.

Note how the quotation marks in the editor indicate that “some text” is a string.

Instructions `100 XP`

Change the value of the:

- `my_numeric` variable to 42.
- `my_character` variable to `"universe"`. Note that the quotation marks indicate that “universe” is a character.
- `my_logical` variable to `FALSE`.

> **i** Note that R is case sensitive!
>
> Thus despite the varibales called `var`, `Var`, `vAr`, has the same fonetic characters, R understand each of these as different memory addresses.

ex_07.R

```
# Change my_numeric to be 42
my_numeric <- 42.5

# Change my_character to be "universe"
my_character <- "some text"

# Change my_logical to be FALSE
my_logical <- TRUE
```

Response

ex_07.R

```
# Change my_numeric to be 42
my_numeric <- 42

# Change my_character to be "universe"
my_character <- "universe"

# Change my_logical to be FALSE
my_logical <- FALSE
```

What's that data type?

Do you remember that when you added 5 + "six", you got an error due to a mismatch in data types? You can avoid such embarrassing situations by checking the data type of a variable beforehand. You can do this with the class() function, as the code in the editor shows.

Instructions 100 XP

Complete the code in the editor and also print out the classes of my_character and my_logical.

ex_08.R

```
# Declare variables of different types

my_numeric <- 42
my_character <- "universe"
my_logical <- FALSE
# Check class of my_numeric
class(my_numeric)

# Check class of my_character
class(my_character)

# Check class of my_logical
class(my_logical)
```

Vectors

Create a vector

Feeling lucky? You better, because this chapter takes you on a trip to the City of Sins, also known as Statisticians Paradise!

Thanks to R and your new data-analytical skills, you will learn how to uplift your performance at the tables and fire off your career as a professional gambler. This chapter will show how you can easily keep track of your betting progress and how you can do some simple analyses on past actions. Next stop, Vegas Baby… VEGAS!!

Instructions `100 XP`

- Do you still remember what you have learned in the first chapter? Assign the value `"Go!"` to the variable vegas. Remember: R is case sensitive!

ex_08.R

```
# Define the variable vegas
vegas <- "Go!"
```

Create a vector (2)

Let us focus first!

On your way from rags to riches, you will make extensive use of vectors. Vectors are one-dimension arrays that can hold numeric data, character data, or logical data. In other words, a vector is a simple tool to store data. For example, you can store your daily gains and losses in the casinos.

In R, you create a vector with the combine function `c()`. You place the vector elements separated by a comma between the parentheses.

For example:

```
numeric_vector <- c(1, 2, 3)
character_vector <- c("a", "b", "c")
```

Once you have created these vectors in R, you can use them to do calculations.

Instructions 100 XP

Complete the code such that boolean_vector contains the three elements: `TRUE`,FALSE`and`TRUE‘ (in that order).

ex_09.R

```
numeric_vector <- c(1, 10, 49)
character_vector <- c("a", "b", "c")

# Complete the code for boolean_vector
boolean_vector <-c(TRUE, FALSE, TRUE)
```

Create a vector (3)

After one week in Las Vegas and still zero Ferraris in your garage, you decide that it is time to start using your data analytical superpowers.

Before doing a first analysis, you decide to first collect all the winnings and losses for the last week:

For poker_vector:

- On Monday you won `$140`
- Tuesday you lost `$50`
- Wednesday you won `$20`
- Thursday you lost `$120`
- Friday you won `$240`

For roulette_vector:

- On Monday you lost `$24`
- Tuesday you lost `$50`
- Wednesday you won `$100`
- Thursday you lost `$350`

- Friday you won $10

You only played poker and roulette, since there was a delegation of mediums that occupied the craps tables. To be able to use this data in R, you decide to create the variables poker_vector and roulette_vector.

Instructions 100 XP

Assign the winnings/losses for roulette to the variable roulette_vector. You lost $24, then lost $50 , won $100, lost $350, and won $10.

ex_10.R

```
# Poker winnings from Monday to Friday
poker_vector <- c(140, -50, 20, -120, 240)

# Roulette winnings from Monday to Friday
roulette_vector <- c(-24, -50, 100, -350, 10)
```

Naming a vector

As a data analyst, it is important to have a clear view on the data that you are using. Understanding what each element refers to is therefore essential.

In the previous exercise, we created a vector with your winnings over the week. Each vector element refers to a day of the week but it is hard to tell which element belongs to which day. It would be nice if you could show that in the vector itself.

You can give a name to the elements of a vector with the `names()` function. Have a look at this example:

```
#| code-line-numbers: false
#| code-fold: false
#| code-summary: "Show the code"

some_vector <- c("John Doe", "poker player")
names(some_vector) <- c("Name", "Profession")
```

This code first creates a vector some_vector and then gives the two elements a name. The first element is assigned the name Name, while the second element is labeled Profession. Printing the contents to the console yields following output:

Output

```
      Name     Profession
"John Doe" "poker player"
```

Instructions 100 XP

The code in the editor names the elements in `poker_vector` with the days of the week. Add code to do the same thing for `roulette_vector`.

ex_11.R

```
# Poker winnings from Monday to Friday
poker_vector <- c(140, -50, 20, -120, 240)

# Roulette winnings from Monday to Friday
roulette_vector <- c(-24, -50, 100, -350, 10)

# Assign days as names of poker_vector
names(poker_vector) <-
  c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")

# Assign days as names of roulette_vector

names(roulette_vector) <-
  c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
```

Naming a vector (2)

If you want to become a good statistician, you have to become lazy. (If you are already lazy, chances are high you are one of those exceptional, natural-born statistical talents.)

In the previous exercises you probably experienced that it is boring and frustrating to type and retype information such as the days of the week. However, when you look at it from a higher perspective, there is a more efficient way to do this, namely, to assign the days of the week vector to a **variable**!

Just like you did with your poker and roulette returns, you can also create a variable that contains the days of the week. This way you can use and re-use it.

Instructions `100 XP`

- A variable days_vector that contains the days of the week has already been created for you.
- Use `days_vector` to set the names of `poker_vector` and `roulette_vector`.

ex_12.R

```
# Poker winnings from Monday to Friday
poker_vector <- c(140, -50, 20, -120, 240)

# Roulette winnings from Monday to Friday
roulette_vector <- c(-24, -50, 100, -350, 10)

# The variable days_vector
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")

# Assign the names of the day to roulette_vector and poker_vector
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector
```

Calculating total winnings

Now that you have the poker and roulette winnings nicely as named vectors, you can start doing some data analytical magic.

You want to find out the following type of information:

- How much has been your overall profit or loss per day of the week?
- Have you lost money over the week in total?
- Are you winning/losing money on poker or on roulette? To get the answers, you have to do arithmetic calculations on vectors.

It is important to know that if you sum two vectors in R, it takes the element-wise sum. For example, the following three statements are completely equivalent:

You can also do the calculations with variables that represent vectors:

Instructions 100 XP

- Take the sum of the variables `A_vector` and `B_vector` and assign it to total_vector.
- Inspect the result by printing out `total_vector`.

ex_13.R

```
A_vector <- c(1, 2, 3)
B_vector <- c(4, 5, 6)

# Take the sum of A_vector and B_vector
total_vector <- A_vector + B_vector

# Print out total_vector
print(total_vector)
```

Calculating total winnings (2)

Now you understand how R does arithmetic with vectors, it is time to get those Ferraris in your garage! First, you need to understand what the overall profit or loss per day of the week was. The total daily profit is the sum of the `profit / loss` you realized on poker per day, and the `profit / loss` you realized on roulette per day.

In R, this is just the sum of `roulette_vector` and `poker_vector`.

Instructions 100 XP

Assign to the variable `total_daily` how much you won or lost on each day in total (poker and roulette combined).

ex_14.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Assign to total_daily how much you won/lost on each day
total_daily <- roulette_vector + poker_vector
```

Calculating total winnings (3)

Based on the previous analysis, it looks like you had a mix of good and bad days. This is not what your ego expected, and you wonder if there may be a very tiny chance you have lost money over the week in total?

A function that helps you to answer this question is `sum()`. It calculates the sum of all elements of a vector. For example, to calculate the total amount of money you have lost/won with poker you do:

```
total_poker <- sum(poker_vector)
```

Instructions 100 XP

- Calculate the total amount of money that you have won/lost with roulette and assign to the variable `total_roulette`.
- Now that you have the totals for roulette and poker, you can easily calculate `total_week` (which is the sum of all gains and losses of the week).
- Print out `total_week`.

ex_15.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Total winnings with poker
total_poker <- sum(poker_vector)

# Total winnings with roulette
total_roulette <-  sum(roulette_vector)

# Total winnings overall
total_week <- total_poker + total_roulette

# Print out total_week
print(total_week)
```

Comparing total winnings

Oops, it seems like you are losing money. Time to rethink and adapt your strategy! This will require some deeper analysis…

After a short brainstorm in your hotel's jacuzzi, you realize that a possible explanation might be that your skills in roulette are not as well developed as your skills in poker. So maybe your total gains in poker are higher (or `>`) than in roulette.

Instructions `100 XP`

- Calculate `total_poker` and `total_roulette` as in the previous exercise. Use the `sum()` function twice.
- Check if your total gains in poker are higher than for roulette by using a comparison. Simply print out the result of this comparison. What do you conclude, should you focus on roulette or on poker?

ex_16.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Calculate total gains for poker and roulette
total_poker <- sum(poker_vector)
total_roulette <- sum(roulette_vector)

# Check if you realized higher total gains in poker than in roulette

print(total_poker > total_roulette)
```

Vector selection: the good times

Your hunch seemed to be right. It appears that the poker game is more your cup of tea than roulette.

Another possible route for investigation is your performance at the beginning of the working week compared to the end of it. You did have a couple of Margarita cocktails at the end of the week…

To answer that question, you only want to focus on a selection of the `total_vector`. In other words, our goal is to select specific elements of the vector. To select elements of a vector (and later matrices, data frames, …), you can use square brackets. Between the square brackets, you indicate what elements to select. For example, to select the first element of the vector, you type `poker_vector[1]`. To select the second element of the vector, you type `poker_vector[2]`, etc. Notice that the first element in a vector has index 1, not 0 as in many other programming languages.

Instructions 100 XP

Assign the poker results of Wednesday to the variable `poker_wednesday`.

ex_17.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Define a new variable based on a selection
poker_wednesday <- poker_vector[3]
```

Vector selection: the good times (2)

How about analyzing your midweek results?

To select multiple elements from a vector, you can add square brackets at the end of it. You can indicate between the brackets what elements should be selected. For example: suppose you want to select the first and the fifth day of the week: use the vector `c(1, 5)` between the square brackets. For example, the code below selects the first and fifth element of poker_vector:

```
poker_vector[c(1, 5)]
```

Instructions 100 XP

Assign the poker results of Tuesday, Wednesday and Thursday to the variable `poker_midweek`.

ex_18.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Define a new variable based on a selection
poker_midweek <- poker_vector[c(2, 3, 4)]
```

Vector selection: the good times (3)

Selecting multiple elements of poker_vector with `c(2, 3, 4)` is not very convenient. Many statisticians are lazy people by nature, so they created an easier way to do this: `c(2, 3, 4)` can be abbreviated to `2:4`, which generates a vector with all natural numbers from 2 up to 4.

So, another way to find the mid-week results is `poker_vector[2:4]`. Notice how the vector `2:4` is placed between the square brackets to select element 2 up to 4.

Instructions 100 XP

Assign to `roulette_selection_vector` the roulette results from Tuesday up to Friday; make use of : if it makes things easier for you.

ex_19.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Define a new variable based on a selection
```

```
roulette_selection_vector <- roulette_vector[2:5]
```

Vector selection: the good times (4)

Another way to tackle the previous exercise is by using the names of the vector elements (Monday, Tuesday, ...) instead of their numeric positions. For example,

```
poker_vector[c("Monday"]
```

will select the first element of poker_vector since "Monday" is the name of that first element.

Just like you did in the previous exercise with numerics, you can also use the element names to select multiple elements, for example:

```
poker_vector[c("Monday","Tuesday")]
```

Instructions 100 XP

- Select the first three elements in `poker_vector` by using their names: `"Monday"`, `"Tuesday"` and `"Wednesday"`. Assign the result of the selection to `poker_start`.
- Calculate the average of the values in `poker_start` with the `mean()` function. Simply print out the result so you can inspect it.

ex_20.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Select poker results for Monday, Tuesday and Wednesday
poker_start <- poker_vector[c("Monday", "Tuesday", "Wednesday")]

# Calculate the average of the elements in poker_start
mean(poker_start)
```

Selection by comparison - Step 1

By making use of comparison operators, we can approach the previous question in a more proactive way.

The (logical) comparison operators known to R are:

- `<` for less than
- `>` for greater than
- `<=` for less than or equal to
- `>=` for greater than or equal to
- `==` for equal to each other
- `!=` not equal to each other

As seen in the previous chapter, stating $6 > 5$ returns TRUE. The nice thing about R is that you can use these comparison operators also on vectors. For example:

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

This command tests for every element of the vector if the condition stated by the comparison operator is `TRUE` or `FALSE`.

Instructions `100 XP`

- Check which elements in `poker_vector` are positive (i.e. `> 0`) and assign this to selection_vector.
- Print out selection_vector so you can inspect it. The printout tells you whether you won `(TRUE)` or lost `(FALSE)` any money for each day.

ex_21.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Which days did you make money on poker?
selection_vector <-
    poker_vector > 0
```

```
# Print out selection_vector
print(selection_vector)
```

Selection by comparison - Step 2

Working with comparisons will make your data analytical life easier. Instead of selecting a subset of days to investigate yourself (like before), you can simply ask R to return only those days where you realized a positive return for poker.

In the previous exercises you used `selection_vector <- poker_vector > 0` to find the days on which you had a positive poker return. Now, you would like to know not only the days on which you won, but also how much you won on those days.

You can select the desired elements, by putting `selection_vector` between the square brackets that follow poker_vector:

```
poker_vector[selection_vector]
```

R knows what to do when you pass a logical vector in square brackets: it will only select the elements that correspond to `TRUE` in `selection_vector`.

Instructions `100 XP`

Use `selection_vector` in square brackets to assign the amounts that you won on the profitable days to the variable `poker_winning_days`.

ex_22.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Which days did you make money on poker?
selection_vector <- poker_vector > 0

# Select from poker_vector these days
poker_winning_days <- poker_vector[selection_vector]
```

Advanced selection

Just like you did for poker, you also want to know those days where you realized a positive return for roulette.

Instructions `100 XP`

- Create the variable selection_vector, this time to see if you made profit with roulette for different days.
- Assign the amounts that you made on the days that you ended positively for roulette to the variable roulette_winning_days. This vector thus contains the positive winnings of roulette_vector.

ex_23.R

```
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Which days did you make money on roulette?
selection_vector <- roulette_vector > 0

# Select from roulette_vector these days
roulette_winning_days <- roulette_vector[selection_vector]
```

Matrices

In this chapter, you will learn how to work with matrices in R. By the end of the chapter, you will be able to create matrices and understand how to do basic computations with them. You will analyze the box office numbers of the Star Wars movies and learn how to use matrices in R. May the force be with you!

What's a matrix?

In R, a matrix is a collection of elements of the same data type (numeric, character, or logical) arranged into a fixed number of rows and columns. Since you are only working with rows and columns, a matrix is called two-dimensional.

You can construct a matrix in R with the `matrix()` function. Consider the following example:

```
matrix(1:9, byrow = TRUE, nrow = 3)
```

In the `matrix()` function:

- The first argument is the collection of elements that R will arrange into the rows and columns of the matrix. Here, we use 1:9 which is a shortcut for `c(1, 2, 3, 4, 5, 6, 7, 8, 9)`.
- The argument byrow indicates that the matrix is filled by the rows. If we want the matrix to be filled by the columns, we just place `byrow = FALSE`.
- The third argument nrow indicates that the matrix should have three rows.

Instructions `100 XP`

Construct a matrix with 3 rows containing the numbers 1 up to 9, filled row-wise.

ex_24.R

```
# Construct a matrix with 3 rows that contain the numbers 1 up to 9
matrix(1:9, byrow = TRUE, nrow = 3)
```

Analyze matrices, you shall

It is now time to get your hands dirty. In the following exercises you will analyze the box office numbers of the Star Wars franchise. May the force be with you!

In the editor, three vectors are defined. Each one represents the box office numbers from the first three Star Wars movies. The first element of each vector indicates the US box office revenue, the second element refers to the Non-US box office (source: Wikipedia).

In this exercise, you'll combine all these figures into a single vector. Next, you'll build a matrix from this vector.

Instructions `100 XP`

- Use `c(new_hope, empire_strikes, return_jedi)` to combine the three vectors into one vector. Call this vector `box_office`.
- Construct a matrix with 3 rows, where each row represents a movie. Use the `matrix()` function to do this. The first argument is the vector `box_office`, containing all box office figures. Next, you'll have to specify `nrow = 3` and `byrow = TRUE`. Name the resulting matrix `star_wars_matrix`.

ex_25.R

```
# Box office Star Wars (in millions!)
new_hope <- c(460.998, 314.4)
empire_strikes <- c(290.475, 247.900)
return_jedi <- c(309.306, 165.8)

# Create box_office
box_office <- c(new_hope, empire_strikes, return_jedi)

# Construct star_wars_matrix
star_wars_matrix <- matrix (box_office, nrow = 3, byrow = TRUE)
```

Naming a matrix

To help you remember what is stored in star_wars_matrix, you would like to add the names of the movies for the rows. Not only does this help you to read the data, but it is also useful to select certain elements from the matrix.

Similar to vectors, you can add names for the rows and the columns of a matrix

```
rownames(my_matrix) <- row_names_vector
colnames(my_matrix) <- col_names_vector
```

We went ahead and prepared two vectors for you: `region`, and `titles`. You will need these vectors to name the columns and rows of star_wars_matrix, respectively.

Instructions 100 XP

- Use colnames() to name the columns of star_wars_matrix with the region vector.
- Use rownames() to name the rows of star_wars_matrix with the titles vector.
- Print out star_wars_matrix to see the result of your work.

ex_26.R

```
# Box office Star Wars (in millions!)
new_hope <- c(460.998, 314.4)
empire_strikes <- c(290.475, 247.900)
return_jedi <- c(309.306, 165.8)

# Construct matrix
star_wars_matrix <- matrix(c(new_hope, empire_strikes, return_jedi), nrow = 3, byrow = TRU

# Vectors region and titles, used for naming
region <- c("US", "non-US")
titles <- c("A New Hope", "The Empire Strikes Back", "Return of the Jedi")

# Name the columns with region
colnames(star_wars_matrix) <- region

# Name the rows with titles
rownames(star_wars_matrix) <- titles

# Print out star_wars_matrix
print(star_wars_matrix)
```

Calculating the worldwide box office

The single most important thing for a movie in order to become an instant legend in Tinseltown is its worldwide box office figures.

To calculate the total box office revenue for the three Star Wars movies, you have to take the sum of the US revenue column and the non-US revenue column.

In R, the function `rowSums()` conveniently calculates the totals for each row of a matrix. This function creates a new vector:

```
rowSums(my_matrix)
```

Instructions 100 XP

Calculate the worldwide box office figures for the three movies and put these in the vector named `worldwide_vector`.

ex_26.R

```
# Construct star_wars_matrix
box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306, 165.8)
region <- c("US", "non-US")
titles <- c("A New Hope",
                "The Empire Strikes Back",
                "Return of the Jedi")

star_wars_matrix <- matrix(box_office,
                     nrow = 3, byrow = TRUE,
                     dimnames = list(titles, region))

# Calculate worldwide box office figures
worldwide_vector <- rowSums(star_wars_matrix)
```

Adding a column for the Worldwide box office

In the previous exercise you calculated the vector that contained the worldwide box office receipt for each of the three Star Wars movies. However, this vector is not yet part of star_wars_matrix.

You can add a column or multiple columns to a matrix with the cbind() function, which merges matrices and/or vectors together by column. For example:

```
  big_matrix <- cbind(matrix1, matrix2, vector1 ...)
```

Instructions 100 XP

Add `worldwide_vector` as a new column to the `star_wars_matrix` and assign the result to `all_wars_matrix`. Use the `cbind()` function.

ex_27.R

```
# Construct star_wars_matrix
box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306, 165.8)
region <- c("US", "non-US")
titles <- c("A New Hope",
            "The Empire Strikes Back",
            "Return of the Jedi")

star_wars_matrix <- matrix(box_office,
                       nrow = 3, byrow = TRUE,
                       dimnames = list(titles, region))

# The worldwide box office figures
worldwide_vector <- rowSums(star_wars_matrix)

# Bind the new variable worldwide_vector as a column to star_wars_matrix
all_wars_matrix <- cbind(star_wars_matrix, worldwide_vector)
```

Adding a row

Just like every action has a reaction, every `cbind()` has an `rbind()`. (We admit, we are pretty bad with metaphors.)

Your R workspace, where all variables you defined 'live' (check out what a workspace is), has already been initialized and contains two matrices:

- star_wars_matrix that we have used all along, with data on the original trilogy,
- star_wars_matrix2, with similar data for the prequels trilogy.

Explore these matrices in the console if you want to have a closer look. If you want to check out the contents of the workspace, you can type `ls()` in the console.

Instructions 100 XP

Use `rbind()` to paste together `star_wars_matrix` and `star_wars_matrix2`, in this order. Assign the resulting matrix to `all_wars_matrix`.

ex_28.R

```
# star_wars_matrix and star_wars_matrix2 are available in your workspace
star_wars_matrix
star_wars_matrix2

# Combine both Star Wars trilogies in one matrix
all_wars_matrix <- rbind(star_wars_matrix, star_wars_matrix2)
```

The total box office revenue for the entire saga

Just like `cbind()` has `rbind()`, `colSums()` has `rowSums()`. Your R workspace already contains the `all_wars_matrix` that you constructed in the previous exercise; type `all_wars_matrix` to have another look. Let's now calculate the total box office revenue for the entire saga.

Instructions `100 XP`

- Calculate the total revenue for the US and the non-US region and assign total_revenue_vector. You can use the colSums() function.
- Print out total_revenue_vector to have a look at the results.

ex_29.R

```
# all_wars_matrix is available in your workspace
all_wars_matrix

# Total revenue for US and non-US
total_revenue_vector <- colSums(all_wars_matrix)

# Print out total_revenue_vector
print(total_revenue_vector)
```

Selection of matrix elements

Similar to vectors, you can use the square brackets `[ ]` to select one or multiple elements from a matrix. Whereas vectors have one dimension, matrices have two dimensions. You should therefore use a comma to separate the rows you want to select from the columns. For example:

- `my_matrix[1,2]` selects the element at the first row and second column.
- `my_matrix[1:3,2:4]` results in a matrix with the data on the rows 1, 2, 3 and columns 2, 3, 4.

If you want to select all elements of a row or a column, no number is needed before or after the comma, respectively:

- `my_matrix[,1]` selects all elements of the first column.
- `my_matrix[1,]` selects all elements of the first row.

Back to Star Wars with this newly acquired knowledge! As in the previous exercise, all_wars_matrix is already available in your workspace.

Instructions `100 XP`

- Select the non-US revenue for all movies (the entire second column of `all_wars_matrix`), store the result as `non_us_all`.
- Use `mean()` on `non_us_all` to calculate the average non-US revenue for all movies. Simply print out the result.
- This time, select the non-US revenue for the first two movies in `all_wars_matrix`. Store the result as `non_us_some`.
- Use `mean()` again to print out the average of the values in `non_us_some`.

ex_30.R

```
# all_wars_matrix is available in your workspace
all_wars_matrix

# Select the non-US revenue for all movies
non_us_all <- all_wars_matrix[,2]

# Average non-US revenue
print(mean(non_us_all))

# Select the non-US revenue for first two movies
non_us_some <- all_wars_matrix[1:2, 2]

# Average non-US revenue for first two movies
print(mean(non_us_some))
```

A little arithmetic with matrices

Similar to what you have learned with vectors, the standard operators like `+`, `-`,`/`, `*`, etc. work in an element-wise way on matrices in R.

For example, `2 * my_matrix` multiplies each element of `my_matrix` by two.

As a newly-hired data analyst for Lucasfilm, it is your job to find out how many visitors went to each movie for each geographical area. You already have the total revenue figures in `all_wars_matrix`. Assume that the price of a ticket was 5 dollars. Simply dividing the box office numbers by this ticket price gives you the number of visitors.

Instructions `100 XP`

- Divide `all_wars_matrix` by 5, giving you the number of visitors in millions.
- Assign the resulting matrix to visitors.
- Print out visitors so you can have a look.

ex_31.R

```
# all_wars_matrix is available in your workspace
all_wars_matrix

# Estimate the visitors
visitors <- all_wars_matrix / 5

# Print the estimate to the console
print(visitors)
```

A little arithmetic with matrices (2)

Just like `2 * my_matrix` multiplied every element of my_matrix by two, `my_matrix1 * my_matrix2` creates a matrix where each element is the product of the corresponding elements in my_matrix1 and my_matrix2.

After looking at the result of the previous exercise, big boss Lucas points out that the ticket prices went up over time. He asks to redo the analysis based on the prices you can find in `ticket_prices_matrix` (source: imagination).

Those who are familiar with matrices should note that this is not the standard matrix multiplication for which you should use %*% in R.

Instructions 100 XP

- Divide all_wars_matrix by ticket_prices_matrix to get the estimated number of US and non-US visitors for the six movies. Assign the result to visitors.
- From the visitors matrix, select the entire first column, representing the number of visitors in the US. Store this selection as `us_visitors`.
- Calculate the average number of US visitors; print out the result.

ex_32.R

```
# all_wars_matrix and ticket_prices_matrix are available in your workspace
all_wars_matrix
ticket_prices_matrix

# Estimated number of visitors
visitors <- all_wars_matrix / ticket_prices_matrix

# US visitors
us_visitors <- visitors[, 1]

# Average number of US visitors
print(mean(us_visitors))
```

Factors

Data often falls into a limited number of categories. For example, human hair color can be categorized as black, brown, blond, red, grey, or white—and perhaps a few more options for people who color their hair. In R, categorical data is stored in factors. Factors are very important in data analysis, so start learning how to create, subset, and compare them now.

What's a factor and why would you use it?

In this chapter you dive into the wonderful world of factors.

The term factor refers to a statistical data type used to store categorical variables. The difference between a categorical variable and a continuous variable is that a categorical variable can belong to a limited number of categories. A continuous variable, on the other hand, can correspond to an infinite number of values.

It is important that R knows whether it is dealing with a continuous or a categorical variable, as the statistical models you will develop in the future treat both types differently. (You will see later why this is the case.)

A good example of a categorical variable is sex. In many circumstances you can limit the sex categories to "Male" or "Female". (Sometimes you may need different categories. For example, you may need to consider chromosomal variation, hermaphroditic animals, or different cultural norms, but you will always have a finite number of categories.)

Instructions `100 XP`

Assign to variable theory the value "factors".

ex_33.R

```
# Assign to the variable theory what this chapter is about!
theory <- "factors"
```

What's a factor and why would you use it? (2)

To create factors in R, you make use of the function `factor()`. First thing that you have to do is create a vector that contains all the observations that belong to a limited number of categories. For example, `sex_vector` contains the sex of 5 different individuals:

```
sex_vector <- c("Male","Female","Female","Male","Male")
```

It is clear that there are two categories, or in R-terms 'factor levels', at work here: "Male" and "Female".

The function `factor()` will encode the vector as a factor:

```
factor_sex_vector <- factor(sex_vector)
```

Instructions `100 XP`

- Convert the character vector `sex_vector` to a factor with `factor()` and assign the result to `factor_sex_vector`
- Print out factor_sex_vector and assert that R prints out the factor levels below the actual values.

ex_34.R

```
# Sex vector
sex_vector <- c("Male", "Female", "Female", "Male", "Male")

# Convert sex_vector to a factor
factor_sex_vector <- factor(sex_vector)

# Print out factor_sex_vector
print(factor_sex_vector)
```

What's a factor and why would you use it? (3)

There are two types of categorical variables: a nominal categorical variable and an ordinal categorical variable.

A nominal variable is a categorical variable without an implied order. This means that it is impossible to say that 'one is worth more than the other'. For example, think of the categorical variable animals_vector with the categories "Elephant", "Giraffe", "Donkey" and

"Horse". Here, it is impossible to say that one stands above or below the other. (Note that some of you might disagree ;-)).

In contrast, ordinal variables do have a natural ordering. Consider for example the categorical variable temperature_vector with the categories: "Low", "Medium" and "High". Here it is obvious that "Medium" stands above "Low", and "High" stands above "Medium".

Instructions `100 XP`

Submit the answer to check how R constructs and prints nominal and ordinal variables. Do not worry if you do not understand all the code just yet, we will get to that.

ex_35.R

```
# Animals
animals_vector <- c("Elephant", "Giraffe", "Donkey", "Horse")
factor_animals_vector <- factor(animals_vector)
factor_animals_vector

# Temperature
temperature_vector <- c("High", "Low", "High","Low", "Medium")
factor_temperature_vector <-
    factor(
        temperature_vector,
        order = TRUE,
        levels = c("Low", "Medium", "High")
    )
factor_temperature_vector
```

Factor levels

When you first get a dataset, you will often notice that it contains factors with specific factor levels. However, sometimes you will want to change the names of these levels for clarity or other reasons. R allows you to do this with the function levels():

```
levels(factor_vector) <- c("name1", "name2",...)
```

A good illustration is the raw data that is provided to you by a survey. A common question for every questionnaire is the sex of the respondent. Here, for simplicity, just two categories were recorded, "M" and "F". (You usually need more categories for survey data; either way, you use a factor to store the categorical data.)

```
survey_vector <- c("M", "F", "F", "M", "M")
```

Recording the sex with the abbreviations `"M"` and `"F"` can be convenient if you are collecting data with pen and paper, but it can introduce confusion when analyzing the data. At that point, you will often want to change the factor levels to `"Male"` and `"Female"` instead of `"M"` and `"F"` for clarity.

Watch out: the order with which you assign the levels is important. If you type `levels(factor_survey_vector)`, you'll see that it outputs `[1] "F" "M"`. If you don't specify the levels of the factor when creating the vector, `R` will automatically assign them alphabetically. To correctly map `"F"` to `"Female"` and `"M"` to `"Male"`, the levels should be set to `c("Female", "Male")`, in this order.

Instructions `100 XP`

- Check out the code that builds a factor vector from `survey_vector`. You should use `factor_survey_vector` in the next instruction.
- Change the factor levels of `factor_survey_vector` to `c("Female", "Male")`. Mind the order of the vector elements here.

ex_36.R

```
# Code to build factor_survey_vector
survey_vector <- c("M", "F", "F", "M", "M")
factor_survey_vector <- factor(survey_vector)

# Specify the levels of factor_survey_vector
levels(factor_survey_vector) <- c("F", "M")

levels(factor_survey_vector) <-  c("Female", "Male")
```

Summarizing a factor

After finishing this course, one of your favorite functions in R will be summary (). This will give you a quick overview of the contents of a variable:

```
summary(my_var)
```

Going back to our survey, you would like to know how many `"Male"` responses you have in your study, and how many `"Female"` responses. The `summary()` function gives you the answer to this question.

Instructions `100 XP`

Ask a `summary()` of the survey_vector and factor_survey_vector. Interpret the results of both vectors. Are they both equally useful in this case?

ex_37.R

```
# Build factor_survey_vector with clean levels
survey_vector <- c("M", "F", "F", "M", "M")
factor_survey_vector <- factor(survey_vector)
levels(factor_survey_vector) <- c("Female", "Male")
factor_survey_vector

# Generate summary for survey_vector
summary(survey_vector)

# Generate summary for factor_survey_vector
summary(factor_survey_vector)
```

Battle of the sexes

You might wonder what happens when you try to compare elements of a factor. In `factor_survey_vector` you have a factor with two levels: `"Male"` and `"Female"`. But how does R value these relative to each other?

Instructions `100 XP`

Read the code in the editor and submit the answer to test if `male` is greater than `(>)` `female`.

ex_38.R

```
# Build factor_survey_vector with clean levels
survey_vector <- c("M", "F", "F", "M", "M")
factor_survey_vector <- factor(survey_vector)
levels(factor_survey_vector) <- c("Female", "Male")
```

```r
# Male
male <- factor_survey_vector[1]

# Female
female <- factor_survey_vector[2]

# Battle of the sexes: Male 'larger' than female?
male > female
```

Ordered factors

Ordered factors Since `"Male"` and `"Female"` are unordered (or nominal) factor levels, R returns a warning message, telling you that the greater than operator is not meaningful. As seen before, R attaches an equal value to the levels for such factors.

But this is not always the case! Sometimes you will also deal with factors that do have a natural ordering between its categories. If this is the case, we have to make sure that we pass this information to R...

Let us say that you are leading a research team of five data analysts and that you want to evaluate their performance. To do this, you track their speed, evaluate each analyst as `"slow"`, `"medium"` or `"fast"`, and save the results in speed_vector.

Instructions `100 XP`

As a first step, assign speed_vector a vector with 5 entries, one for each analyst. Each entry should be either `"slow"`, `"medium"`, or `"fast"`. Use the list below:

- Analyst 1 is medium,
- Analyst 2 is slow,
- Analyst 3 is slow,
- Analyst 4 is medium and
- Analyst 5 is fast.

No need to specify these are factors yet.

ex_39.R

```r
speed_vector <- c(
    "medium",
    "slow",
```

```
    "slow",
    "medium",
    "fast"
)
```

Ordered factors (2)

`speed_vector` should be converted to an ordinal factor since its categories have a natural ordering. By default, the function `factor()` transforms `speed_vector` into an unordered factor. To create an ordered factor, you have to add two additional arguments: ordered and levels.

```
factor(some_vector,
       ordered = TRUE,
       levels = c("lev1", "lev2" ...))
```

By setting the argument ordered to TRUE in the function factor(), you indicate that the factor is ordered. With the argument levels you give the values of the factor in the correct order.

Instructions `100 XP`

From `speed_vector`, create an ordered factor vector: `factor_speed_vector`. Set `ordered` to `TRUE`, and set levels to `c("slow", "medium", "fast")`.

ex_40.R

```
# Create speed_vector
speed_vector <- c("medium", "slow", "slow", "medium", "fast")

# Convert speed_vector to ordered factor vector
factor_speed_vector <-
factor(
    speed_vector,
    ordered = TRUE,
    levels = c("slow", "medium", "fast")
)


# Print factor_speed_vector
factor_speed_vector
```

```
summary(factor_speed_vector)
```

Comparing ordered factors

Having a bad day at work, 'data analyst number two' enters your office and starts complaining that 'data analyst number five' is slowing down the entire project. Since you know that 'data analyst number two' has the reputation of being a smarty-pants, you first decide to check if his statement is true.

The fact that `factor_speed_vector` is now ordered enables us to compare different elements (the data analysts in this case). You can simply do this by using the well-known operators.

Instructions `100 XP`

- Use `[2]` to select from `factor_speed_vector` the factor value for the second data analyst. Store it as `da2`.
- Use `[5]` to select the `factor_speed_vector` factor value for the fifth data analyst. Store it as `da5`.
- Check if `da2` is greater than `da5`; simply print out the result. Remember that you can use the `>` operator to check whether one element is larger than the other.

ex_41.R

```
# Create factor_speed_vector
speed_vector <- c("medium", "slow", "slow", "medium", "fast")
factor_speed_vector <-
  factor(
    speed_vector,
    ordered = TRUE,
    levels = c("slow", "medium", "fast")
  )

# Factor value for second data analyst
da2 <- factor_speed_vector[2]

# Factor value for fifth data analyst
da5 <- factor_speed_vector[5]

# Is data analyst 2 faster than data analyst 5?
 print(da2 > da5)
```

Data frames

What's a data frame?

You may remember from the chapter about matrices that all the elements that you put in a matrix should be of the same type. Back then, your dataset on Star Wars only contained numeric elements.

When doing a market research survey, however, you often have questions such as:

- 'Are you married?' or 'yes/no' questions (logical)
- 'How old are you?' (numeric)
- 'What is your opinion on this product?' or other 'open-ended' questions (character)
- … The output, namely the respondents' answers to the questions formulated above, is a dataset of different data types. You will often find yourself working with datasets that contain different data types instead of only one.

A data frame has the variables of a dataset as columns and the observations as rows. This will be a familiar concept for those coming from different statistical software packages such as SAS or SPSS.

Instructions `100 XP`

Submit the answer. The data from the built-in example data frame `mtcars` will be printed to the console.

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
```

```
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Quick, have a look at your dataset

Wow, that is a lot of cars!

Working with large datasets is not uncommon in data analysis. When you work with (extremely) large datasets and data frames, your first task as a data analyst is to develop a clear understanding of its structure and main elements. Therefore, it is often useful to show only a small part of the entire dataset.

So how to do this in R? Well, the function `head()` enables you to show the first observations of a data frame. Similarly, the function `tail()` prints out the last observations in your dataset.

Both `head()` and `tail()` print a top line called the 'header', which contains the names of the different variables in your dataset.

Instructions 100 XP

Call head() on the mtcars dataset to have a look at the header and the first observations.

ex_42.R

```
# Call head() on mtcars
head(mtcars)
```

Have a look at the structure

Another method that is often used to get a rapid overview of your data is the function `str()`. The function `str()` shows you the structure of your dataset.

For a data frame it tells you:

- The total number of observations (e.g. 32 car types)
- The total number of variables (e.g. 11 car features)
- A full list of the variables names (e.g. mpg, cyl …)
- The data type of each variable (e.g. num)
- The first observations

Applying the `str()` function will often be the first thing that you do when receiving a new dataset or data frame. It is a great way to get more insight in your dataset before diving into the real analysis.

Instructions 100 XP

Investigate the structure of mtcars. Make sure that you see the same numbers, variables and data types as mentioned above.

ex_43.R

```
str(mtcars)
```

Creating a data frame

Since using built-in datasets is not even half the fun of creating your own datasets, the rest of this chapter is based on your personally developed dataset. Put your jet pack on because it is time for some space exploration!

As a first goal, you want to construct a data frame that describes the main characteristics of eight planets in our solar system. According to your good friend Buzz, the main features of a planet are:

- The type of planet (Terrestrial or Gas Giant).
- The planet's diameter relative to the diameter of the Earth.
- The planet's rotation across the sun relative to that of the Earth.
- If the planet has rings or not (TRUE or FALSE).

After doing some high-quality research on Wikipedia, you feel confident enough to create the necessary vectors: name, type, diameter, rotation and rings; these vectors have already been coded up in the editor. The first element in each of these vectors correspond to the first observation.

You construct a data frame with the `data.frame()` function. As arguments, you pass the vectors from before: they will become the different columns of your data frame. Because every column has the same length, the vectors you pass should also have the same length. But don't forget that it is possible (and likely) that they contain different types of data.

Instructions `100 XP`

Use the function `data.frame()` to construct a data frame. Pass the vectors `name`, `type`, `diameter`, `rotation` and `rings` as arguments to `data.frame()`, in this order. Call the resulting data frame `planets_df`.

ex_44.R

```
name <- c("Mercury", "Venus", "Earth",
          "Mars", "Jupiter", "Saturn",
          "Uranus", "Neptune")
type <- c("Terrestrial planet",
          "Terrestrial planet",
          "Terrestrial planet",
          "Terrestrial planet", "Gas giant",
          "Gas giant", "Gas giant", "Gas giant")
diameter <- c(0.382, 0.949, 1, 0.532,
              11.209, 9.449, 4.007, 3.883)
```

```
rotation <- c(58.64, -243.02, 1, 1.03,
              0.41, 0.43, -0.72, 0.67)
rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)
planets_df <-
    data.frame(name, type, diameter, rotation, rings)
```

Creating a data frame (2)

The `planets_df` data frame should have 8 observations and 5 variables. It has been made available in the workspace, so you can directly use it.

Instructions `100 XP`

Use `str()` to investigate the structure of the new planets_df variable.

ex_45.R

```
str(planets_df)
```

Selection of data frame elements

Similar to vectors and matrices, you select elements from a data frame with the help of square brackets `[ ]`. By using a comma, you can indicate what to select from the rows and the columns respectively. For example:

- `my_df[1,2]` selects the value at the first row and second column in `my_df`.
- `my_df[1:3,2:4]` selects rows 1, 2, 3 and columns 2, 3, 4 in my_df.

Sometimes you want to select all elements of a row or column. For example, `my_df[1, ]` selects all elements of the first row. Let us now apply this technique on `planets_df`!

Instructions `100 XP`

- From `planets_df`, select the diameter of Mercury: this is the value at the first row and the third column. Simply print out the result.
- From `planets_df`, select all data on Mars (the fourth row). Simply print out the result.

ex_46.R

```
# The planets_df data frame from the previous exercise is pre-loaded

# Print out diameter of Mercury (row 1, column 3)
print(planets_df[1, 3])

# Print out data for Mars (entire fourth row)
print(planets_df[4, ])
```

Selection of data frame elements (2)

Instead of using numerics to select elements of a data frame, you can also use the variable names to select columns of a data frame.

Suppose you want to select the first three elements of the type column. One way to do this is

```
planets_df[1:3, 2]
```

A possible disadvantage of this approach is that you have to know (or look up) the column number of type, which gets hard if you have a lot of variables. It is often easier to just make use of the variable name:

```
planets_df[1:3,"type"]
```

Instructions 100 XP

Select and print out the first 5 values in the `"diameter"` column of `planets_df`.

ex_47.R

```
print(planets_df[1:5, "diameter"])
```

Only planets with rings

You will often want to select an entire column, namely one specific variable from a data frame. If you want to select all elements of the variable diameter, for example, both of these will do the trick:

```
planets_df[,3]
planets_df[,"diameter"]
```

However, there is a short-cut. If your columns have names, you can use the `$` sign:

```
planets_df$diameter
```

Instructions `100 XP`

- Use the `$` sign to select the `rings` variable from `planets_df`. Store the vector that results as `rings_vector`.
- Print out `rings_vector` to see if you got it right.

ex_48.R

```
# planets_df is pre-loaded in your workspace

# Select the rings variable from planets_df
rings_vector <- planets_df$rings

# Print out rings_vector
print(rings_vector)
```

Only planets with rings (2)

You probably remember from high school that some planets in our solar system have rings and others do not. Unfortunately you can not recall their names. Could R help you out?

If you type rings_vector in the console, you get:

> [1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE

This means that the first four observations (or planets) do not have a ring (`FALSE`), but the other four do (`TRUE`). However, you do not get a nice overview of the names of these planets, their diameter, etc. Let's try to use `rings_vector` to select the data for the four planets with rings.

Instructions 100 XP

The code in the editor selects the `name` column of all planets that have rings. Adapt the code so that instead of only the `name` column, all columns for planets that have rings are selected.

ex_49.R

```
# planets_df and rings_vector are pre-loaded in your workspace

# Adapt the code to select all columns for planets with rings
planets_df[rings_vector, "name"]
planets_df[rings_vector, ]
```

Only planets with rings but shorter

So what exactly did you learn in the previous exercises? You selected a subset from a data frame (`planets_df`) based on whether or not a certain condition was true (rings or no rings), and you managed to pull out all relevant data. Pretty awesome! By now, NASA is probably already flirting with your CV ;-).

Now, let us move up one level and use the function `subset()`. You should see the `subset()` function as a short-cut to do exactly the same as what you did in the previous exercises.

```
subset(my_df, subset = some_condition)
```

The first argument of subset() specifies the dataset for which you want a subset. By adding the second argument, you give R the necessary information and conditions to select the correct subset.

The code below will give the exact same result as you got in the previous exercise, but this time, you didn't need the `rings_vector`!

```
subset(planets_df, subset = rings)
```

Instructions 100 XP

Use `subset()` on `planets_df` to select planets that have a `diameter` smaller than Earth. Because the `diameter` variable is a relative measure of the planet's diameter w.r.t that of planet Earth, your condition is `diameter < 1`.

ex_50.R

```
# planets_df is pre-loaded in your workspace

# Select planets with diameter < 1
subset(planets_df, diameter < 1)
```

Sorting

Making and creating rankings is one of mankind's favorite affairs. These rankings can be useful (best universities in the world), entertaining (most influential movie stars) or pointless (best 007 look-a-like).

In data analysis you can sort your data according to a certain variable in the dataset. In R, this is done with the help of the function `order()`.

`order()` is a function that gives you the ranked position of each element when it is applied on a variable, such as a vector for example:

```
a <- c(100, 10, 1000)
order(a)
```

10, which is the second element in a, is the smallest element, so 2 comes first in the output of order(a). 100, which is the first element in a is the second smallest element, so 1 comes second in the output of `order(a)`.

This means we can use the output of order(a) to reshuffle a:

```
a[order(a)]
```

Instructions `100 XP`

Experiment with the `order()` function in the console. Submit the answer when you are ready to continue.

ex_51.R

```
x <- rnorm(10)
order(x)
```

Sorting your data frame

Alright, now that you understand the `order()` function, let us do something useful with it. You would like to rearrange your data frame such that it starts with the smallest planet and ends with the largest one. A sort on the `diameter` column.

Instructions `100 XP`

- Call `order()` on `planets_df$diameter` (the `diameter` column of `planets_df`). Store the result as `positions`.
- Now reshuffle `planets_df` with the positions vector as row indexes inside square brackets. Keep all columns. Simply print out the result.

ex_52.R

```
# planets_df is pre-loaded in your workspace

# Use order() to create positions
positions <-  order(planets_df$diameter)

# Use positions to sort planets_df
planets_df[positions, ]
```

Best Coding Practices for R

What we mean when say "better coding practice"

R programmers have a bad reputation writing bad code. Perhaps the main reason is that the people whose write much of the package are not programmers but scientific from other areas. Sometimes we overestimate crucial aspects from a programming standpoint. As R programmers we overcome to write the code for production. Mostly we write scripts and when we deploy it the same when we just wrap it in a function and perhaps a package. It is common to face poorly written code—**columns were referred by numbers, functions were dependent upon global environment variables, 50+ lines functions without arguments and with over-sized lines code 100 characters or more, not indentation, poor naming, conventions** etc,...,.

We strongly encourage to use a style. Yea I know, there is not a unique way to do it, but the philosophy is to follow a consistent style. With respect to this regard made yourself a favor and read this great book for R

https://bookdown.org/content/d1e53ac9-28ce-472f-bc2c-f499f18264a3/

Folder Structure

Code Structure

Sections

Structural Composition

Identation

Styling

Final Comments

Tidyverse Fundamentals with R

Introduction to Tidyverse

Reshaping Data with tidyr

Project

Modeling with Data in the Tidyverse

Communication with Data in the Tidyverse

Categorical Data in the Tydiverse

Data Manipulation

Data Manipulation with dplyr

Joining Data with dplyr

Case Study: Exploratory Data Analysis in R

Data Manipulation with data.table in R

Joining Data with data.table in R

1 Data visualization with ggplot2 and friends

Part II

The whole game of statistical Inference

Our goal in this part of the book is to give you a rapid overview of the main tools of data science: **importing**, **tidying**, **transforming**, and **visualizing data**, as shown in **?@fig-ds-whole-game**. We want to show you the "whole game" of data science giving you just enough of all the major pieces so that you can tackle real, if simple, data sets. The later parts of the book, will hit each of these topics in more depth, increasing the range of data science challenges that you can tackle.

2 Statistical Inference with resampling: Bootstrap and Jacknife.

2.1 Likelihood inference.

2.2 Variance analisys.

2.3 ROC Curves

3 Lienar Regression

3.1 Linear Regression

3.2 Multiple linear_regression and generalized linear regresion

4 Summary

In summary, this book has no content whatsoever.

```
[1] 2
```

References

[1] T. Hastie, R. Tibshirani, J. Friedman, The elements of statistical learning, Second, Springer, New York, 2009.

[2] W.J. Krzanowski, D.J. Hand, ROC curves for continuous data, CRC Press, Boca Raton, FL, 2009.

[3] R. Martin, A statistical inference course based on p-values, The American Statistician. 71 (2017) 128–136.

[4] P. McCullagh, J.A. Nelder, Generalized linear models, Chapman & Hall, London, 1989.

[5] B. Ratner, Statistical and machine-learning data mining:: Techniques for better predictive modeling and analysis of big data, third edition, CRC Press, 2017.

[6] D.A. Sprott, Statistical inference in science, Springer-Verlag, New York, 2000.