Statistical Inference For Data Science

Saul Diaz-Infante

1/16/23

Table of contents

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.
- What the book is about.
- When I writing this book.
- Why I write this book.
- Where I wrote this book.

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.

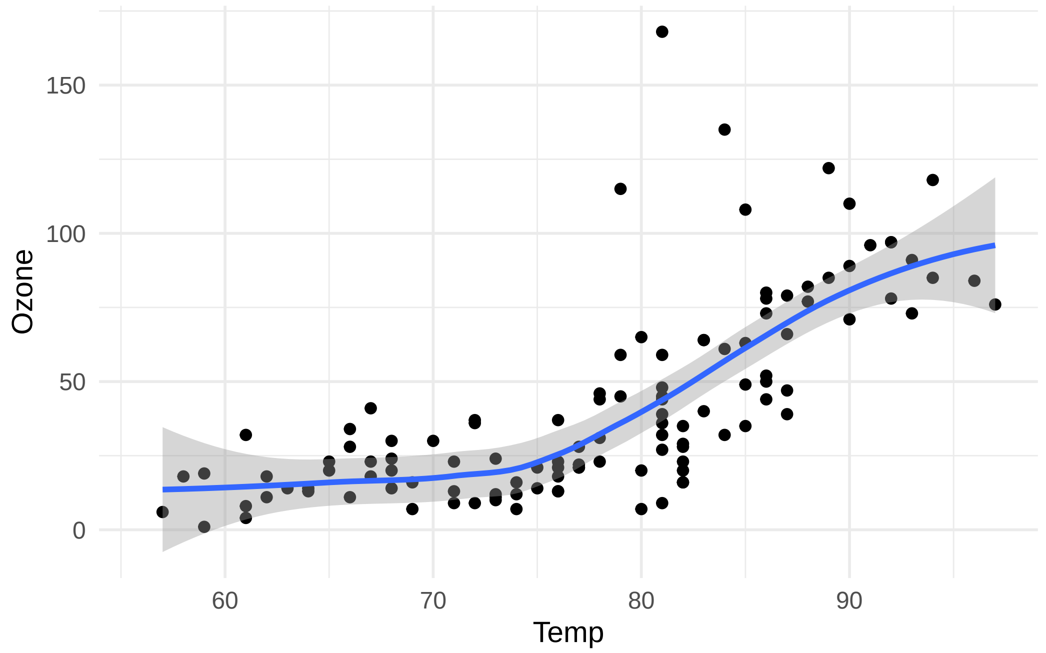

Figure 1: Temperature and ozone level.

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

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.

1 R programming fundamentals for Data Science

1.1 Nuts and bolts: Data types

1.1.1 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.

1.2 Intro Baisics

Take your first steps with R. In this chapter, you will learn how to use the console as a calculator and how to assign variables. You will also get to know the basic data types in R. Let's get started. The bellow lines has been taken from the course Introduction to R of Data Camp. You can see the first chapter for free. I recommend this course to get started on R see https://campus.datacamp.com/courses/free-introduction-to-r

1.2.1 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.

1.2.1.1 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

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

1.2.2 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.

1.2.2.1 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
```

1.2.3 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
```

1.2.3.1 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.

```
[1] 42
```

1.2.4 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.

1.2.4.1 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

```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)
```

1.2.5 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
```

1.2.5.1 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

```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
```

1.2.6 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.

1.2.6.1 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

```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
```

1.2.7 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.

1.2.8 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`.

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
```

1.2.9 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.

1.2.9.1 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)
```

1.3 Vectors

1.3.1 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!!

1.3.1.1 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!"
```

1.3.2 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.

1.3.2.1 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)
```

1.3.3 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.

1.3.3.1 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

```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)
```

1.3.4 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:

```r
#| 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"
```

1.3.4.1 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

```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")
```

1.3.5 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.

1.3.5.1 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

```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
```

1.3.6 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:

1.3.6.1 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)
```

1.3.7 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`.

1.3.7.1 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
```

1.3.8 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)
```

1.3.8.1 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)
```

1.3.9 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

```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)
```

1.3.10 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.

1.3.10.1 Instructions 100 XP

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

ex_17.R

```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]
```

1.3.11 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:

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

1.3.11.1 Instructions `100 XP`

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

ex_18.R

```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)]
```

1.3.12 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.

1.3.12.1 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

```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]
```

1.3.13 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")]
```

1.3.13.1 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)
```

1.3.14 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`.

1.3.14.1 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

```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)
```

1.3.15 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`.

1.3.15.1 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]
```

1.3.16 Advanced selection

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

1.3.16.1 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

```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]
```

1.4 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!

1.4.1 What's a matrix?

1.4.2

1.5 Best Coding Practices for R

1.5.1 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/

1.5.2 Folder Structure

1.5.3 Code Structure

1.5.4 Sections

1.5.5 Structural Composition

1.5.6 Identation

1.5.7 Styling

1.5.8 Final Comments

Tidyverse Fundamentals with R

1.6 Introduction to Tidyverse

1.7 Reshaping Data with tidyr

1.8 Project

1.9 Modeling with Data in the Tidyverse

1.10 Communication with Data in the Tidyverse

1.11 Categorical Data in the Tydiverse

Data Manipulation

1.12 Data Manipulation with dplyr

1.13 Joining Data with dplyr

1.14 Case Study: Exploratory Data Analysis in R

1.15 Data Manipulation with data.table in R

1.16 Joining Data with data.table in R

2 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.

3 Statistical Inference with resampling: Bootstrap and Jacknife.

3.1 Likelihood inference.

3.2 Variance analisys.

3.3 ROC Curves

4 Lienar Regression

4.1 Linear Regression

4.2 Multiple linear_regression and generalized linear regresion

5 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.