

Arithmetic with R | R - Google Chrome

Arithmetic with

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-1-intro-to-basics-1?ex=2

Apps Gmail WhatsApp i1ERP | Login



Exercise

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.

With this knowledge, follow the instructions below to complete the exercise.

Instructions

100 XP

R Console

>

script.R

```
1 # An addition
2 5 + 5
3
4 # A subtraction
5 5 - 5
6
7 # A multiplication
8 3 * 5
9
10 # A division
11 (5 + 5) / 2
12
13 # Exponentiation
14
15
16 # Modulo
17
```

Variable assignment | R - Google Chrome

Variable assignn x

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-1-intro-to-basics-1?ex=3

Apps Gmail WhatsApp i1ERP | Login

DataCamp

Exercise

script.R

```
1 # Assign the value 42 to the variable x
2 x <- 42
3
4 # Print out the value of x
5 x
```

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. Click 'Submit Answer'. Notice that when you ask R to print `x`, the value 42 appears.

Take Hint (-30 XP)

R Console

```
X script.R  
1 # Assign  
2 x <- 42  
3  
4 # Print  
5 x|
```

• +100 XP

Good job! Have you noticed that R does not print the value of a variable to the console when you did the assignment? `x <- 42` did not generate any output, because R assumes that you will be needing this variable in the future. Otherwise you wouldn't have stored the value in a variable in the first place, right? Proceed to the next exercise!

PRESS ENTER TO

Continue

R Console

```
> # Assign the value  
> x <- 42  
>  
> # Print out the value  
> x
```

Basic data types in R | R - Google Chrome

Basic data types

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-1-intro-to-basics/1/ex-7

Apps Gmail WhatsApp i1ERP | Login

DataCamp

Exercise

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**.
- Natural 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 on the right indicate that "some text" is a character.

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

What's that data type? | R - Google Chrome

What's that date x https://campus.datacamp.com/courses/free-introduction-to-r/chapter-1-intro-to-basics?t=ex=8
Apps Gmail WhatsApp i1ERP | Login

DataCamp

Exercise

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 on the right shows.

Instructions

100 XP

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

Take Hint (-30 XP)

script.R

```
1 # Declare variables of different types
2 my_numeric <- 42
3 my_character <- "universe"
4 my_logical <- FALSE
5
6 # Check class of my_numeric
7 class(my_numeric)
8
9 # Check class of my_character
10
11
12 # Check class of my_logical
13
```

R Console

Chrome
Create a vector | x
https://campus.datacamp.com/courses/free-introduction-to-r/chapter-2-vectors-2?ex=1
Apps Gmail WhatsApp i1ERP | Login

DataCamp

Exercise

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!

Take Hint (-30 XP)

script.R

```
1 # Define the
2 vegas <-
```

Exercise

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-dimensional 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).

R Console

 Take Hint (-30 XP)

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

Take Hint (-30 XP)

Correct Submission

Code contains a syntax error. Check the console output and try to fix the issue.
Did this feedback help you?

✓ Yes ✗ No

```
script.R
1 numeric_vector <- c(1, 10, 49)
2 character_vector <- c("a", "b", "c")
3
4 # Complete the code for boolean_vector
5 boolean_vector <- c(TRUE, FALSE, TRUE)
```

R Console
4: # Complete the code for boolean_vector
5: boolean_vector <- c(TRUE, FALSE and

Parsing error in script.R:5:32: unexpected symbol
4: # Complete the code for boolean_vector
5: boolean_vector <- c(TRUE, FALSE and
> numeric_vector <- c(1, 10, 49)
> character_vector <- c("a", "b", "c")
> # Complete the code for boolean_vector

DataCamp

Exercise

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:

```
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:

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

R Console

Instructions

100 XP

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

[Take Hint \(-30 XP\)](#)

← Course Outline →

script.R

```
1 # Poker winnings from Monday to Friday
2 poker_vector <- c(140, -50, 20, -120, 240)
3
4 # Roulette winnings from Monday to Friday
5 roulette_vector <- c(-24, -50, 100, -350, 10)
6
7 # Assign days as names of poker_vector
8 names(poker_vector) <- c("Monday", "Tuesday", "Wednesday", "Thursday",
9
10 # Assign days as names of roulette_vector
11
```

¶

R Console

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

- 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`.

Take Hint (-30 XP)

script.R

```
1 # Poker winnings from Monday to Friday  
2 poker_vector <- c(140, -50, 20, -120, 240)  
3 # Roulette winnings from Monday to Friday  
4 roulette_vector <- c(-24, -50, 100, -350, 10)  
5 # The variable days_vector  
6 days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")  
7 # Assign the names of the day to roulette_vector and poker_vector  
8 names(poker_vector) <-  
9 names(roulette_vector) <-  
10 names(days_vector) <-  
11 names(poker_vector) <-  
12 names(roulette_vector) <-
```

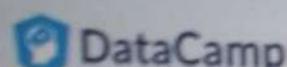
R Console

Naming a vector (2) | R - Google Chrome

Naming a vector

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-2-vectors-2?ex=5

Apps Gmail WhatsApp i1ERP | Login



Exercise

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`.

Take Hint (-30 XP)

script.R

```

1 # Poker winnings from Monday to Friday
2 poker_vector <- c(140, -50, 20, -120, 240)
3
4 # Roulette winnings from Monday to Friday
5 roulette_vector <- c(-24, -50, 100, -350, 10)
6
7 # The variable days_vector
8 days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
9
10 # Assign the names of the day to roulette_vector and poker_vector
11 names(poker_vector) <- (days_vector)
12 names(roulette_vector) <-(days_vector)

```



Run Code

R Console

```

> # 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)

```

- How can we calculate our 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?

8

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:

```
c(1, 2, 3) + c(4, 5, 6)  
c(1 + 4, 2 + 5, 3 + 6)  
c(5, 7, 9)
```

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

```
a <- c(1, 2, 3)  
b <- c(4, 5, 6)  
c <- a + b
```

R Console

>

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`.

 Take Hint (-30 XP)

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`.

 Take Hint (-30 XP)

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?

 Take Hint (-30 XP)

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`.

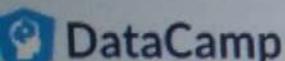
Take Hint (-30 XP)

Vector selection: the good times (2) | R - Google Chrome

Vector selection x

← → ⌂ https://campus.datacamp.com/courses/free-introduction-to-r/chapter-2-vectors-2?ex=11

Apps Gmail WhatsApp i1ERP | Login



Exercise

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`.

Take Hint (-30 XP)

script.R

```
1 # Poker and roulette winnings from Monday to Friday:  
2 poker_vector <- c(140, -50, 20, -120, 240)  
3 roulette_vector <- c(-24, -50, 100, -350, 10)  
4 days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")  
5 names(poker_vector) <- days_vector  
6 names(roulette_vector) <- days_vector  
7  
8 # Define a new variable based on a selection  
9 poker_midweek <- poker_vector[c(2,3,4)]
```

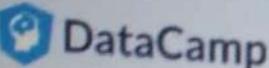
R Console

```
>  
> # Define a new variable based on a selection  
> poker_midweek <- poker_vector[2,3,4]  
Error: incorrect number of dimensions  
> # 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) | R - Google Chrome

Vector selection x

← → C https://campus.datacamp.com/courses/free-introduction-to-r/chapter-2-vectors-2?ex=12
Apps Gmail WhatsApp i1ERP | Login



Exercise

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.

Take Hint (-30 XP)



R Cons

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["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.

Take Hint (-30 XP)

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:

```
> c(4, 5, 6) > 5
[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`.

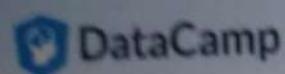
R Console

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.

Take Hint (-30 XP)



Exercise

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`

Time limit: 30 min

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.

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

R Console

```
> # Construct a matrix with 3 rows that contain the numbers 1 up to 9  
> matrix(1:9,byrow=TRUE,nrow=3)  
     [,1] [,2] [,3]  
[1,]    1    2    3  
[2,]    4    5    6  
[3,]    7    8    9  
>
```

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

R Console

- 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`.

Take Hint (-30 XP)

Course Outline

script.R

```
1 # Box office Star Wars (in millions!)
2 new_hope <- c(460.998, 314.4)
3 empire_strikes <- c(290.475, 247.900)
4 return_jedi <- c(309.306, 165.8)
5
6 # Create box_office
7 box_office <- c(new_hope, empire_strikes, return_jedi)
8
9 # Construct star_wars_matrix
10 star_wars_matrix <- matrix(box_office, nrow=3, byrow=TRUE)
```

Run all/selected code



Run Code

Stop

R Console

```
> # 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)
```

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.

```
2 new  
3 emp  
4 retu  
5  
6 # Co  
7 star  
byron  
8  
9 # Vec  
10 regio  
11 titl  
12  
13 # Name
```

R Console

① 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.

```
byrow = TRUE)
8
9 # Vectors region and titles, used for naming
10 region <- c("US", "non-US")
11 titles <- c("A New Hope", "The Empire Strikes Back",
12
13 # Name the columns with region
14 colnames(star_wars_matrix) <- region
15
16 # Name the rows with titles
17 rownames(star_wars_matrix) <- titles
18
19 # Print out star_wars_matrix
20 star_wars_matrix
```

5

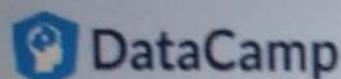
R Console

```
> colnames(star_wars_matrix) <- region
>
> # Name the rows with titles
> rownames(star_wars_matrix) <- titles
>
> # Print out star_wars_matrix
> star_wars_matrix
```

	US	non-US
A New Hope	460,998	314,4
The Empire Strikes Back	290,475	247,9
Return of the Jedi	309,386	165,8

Calculating the worl x

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-1



Exercise

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

R Co

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

💡 Take Hint (-30 XP)

pter-3-matrices-3?ex=4



Course Outline



script.R

```

1 # Construct star_wars_matrix
2 box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306, 165.8)
3 star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
4                               dimnames = list(c("A New Hope", "The Em
    Back", "Return of the Jedi"),
5
6
7 # Calculate worldwide box office figures
8 worldwide_vector <- rowSums(box_office)

```

Run all/selected code



Run Code

R Console

```

> # Construct star_wars_matrix
> box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306, 165.8)
> star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
>                               dimnames = list(c("A New Hope", "The Empire Str
    , "Return of the Jedi"),
>                               c("US", "non-US")))
>
> # Calculate worldwide box office figures
> worldwide_vector <- rowSums(box_office)
Error: 'x' must be an array of at least two dimensions

```

ecome an instant

ar Wars movies,
the non-US

the totals for each

100 XP

```
3 star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
4                                     dimnames = list(c("A New Hope", "The Empire St
5                                         Back", "Return of the Jedi"),
6                                     c("US", "non-US")))
7 # Calculate worldwide box office figures
8 worldwide_vector <- rowSums(star_wars_matrix)
```

Run all/selected code



Run Code

Submit Ans

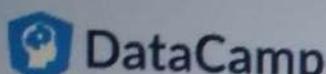
R Console

```
> # Construct star_wars_matrix
> box_office <- c(460.998, 314.4, 298.475, 247.900, 309.306, 165.8)
> star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
>                                     dimnames = list(c("A New Hope", "The Empire Strikes Back",
>                                         "Return of the Jedi"),
>                                     c("US", "non-US")))
>
> # Calculate worldwide box office figures
> worldwide_vector <- rowSums(star_wars_matrix)
>
```

Adding a column for the Worldwide box office | R - Chromium

Adding a column for

<https://campus.datacamp.com/courses/free-introduction-to-r/chapter-1>



Exercise

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.

Take Hint (-30 XP)

R Consol

matrices-3?ex=5

Course Outline →

```
R

# Construct star_wars_matrix
box_office <- c(460.998, 314.4, 298.475, 247.900, 309.306, 165.8)
star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
                           dimnames = list(c("A New Hope", "The Empire Strikes
Back", "Return of the Jedi"),
c("US", "non-US")))

# 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(worldwide_vector)
```

Run all/selected code



Run Code

Submit

sole

```
star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
                           dimnames = list(c("A New Hope", "The Empire Strikes
Back", "Return of the Jedi"),
c("US", "non-US")))
```

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(worldwide_vector)
```

Exercise

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.

Take Hint (-30 XP)

Incorrect Submission

Have you correctly used `cbind()` to add `worldwide_vector` to `star_wars_matrix`? You should pass `star_wars_matrix` and `worldwide_vector` to `cbind()`, in this order. The resulting matrix, `all_wars_matrix`, should consist of three rows and three columns.

Yes No

Did you find this feedback helpful?

script.R

```
1 # Construct star_wars_matrix
2 box_office <- c(460.998, 314.4, 290.475, 247.0)
3 star_wars_matrix <- matrix(box_office, nrow = 3,
4                               dimnames = list(c("A New Hope",
5                                "The Empire Strikes Back", "Return of the Jedi"),
6                                c("US", "UK", "Germany")))
7 # The worldwide box office figures
8 worldwide_vector <- rowSums(star_wars_matrix)
9
10 # Bind the new variable worldwide_vector as a column
11 all_wars_matrix <- cbind(worldwide_vector, star_wars_matrix)
```

R Console

```
> star_wars_matrix <- matrix(box_office, nrow = 3,
>                               dimnames = list(c("A New Hope",
>                                "The Empire Strikes Back", "Return of the Jedi"),
>                                c("US", "UK", "Germany")))
>
> # The worldwide box office figures
> worldwide_vector <- rowSums(star_wars_matrix)
>
> # Bind the new variable worldwide_vector as a column
> all_wars_matrix <- cbind(worldwide_vector, star_wars_matrix)
```

Ctrl+Shift+Enter

 *100 XP

Adding column to a matrix, the logical next step
rows. Learn how in the next exercise.

Please rate this exercise:

★ ★ ★ ★ ★

ENTER

ENTER TO

The screenshot shows a DataCamp RStudio interface. The top navigation bar includes a back arrow, forward arrow, and course outline. The URL in the address bar is `/chapter-3-matrices-3?ex=5`. The main area displays a script named `script.R` with the following content:

```
script.R
1 # Construct star-wars-matrix
2 box_office <- c(460.998, 314.4, 290.475,
3 star_wars_matrix <- matrix(box_office, nrow = 3,
4 dimnames = list(c("US", "non-US"), c("A New Hope", "The Empire Strikes Back", "Return of the Jedi")))
5 Back", "Return of the Jedi"),
6 worldwide_box_office <- rowSums(star_wars_matrix)
7 # The worldwide box office figures
8 worldwide_vector <- worldwide_box_office
9 worldwide_vector <- worldwide_vector / 3
10 # Bind the new variable worldwide_vector as a column to star-wars-matrix
11 all_wars_matrix <- cbind(star_wars_matrix, worldwide_vector)
```

A blue button labeled `Ctrl+Shift+Enter` is highlighted at the bottom right. The R Console below shows the execution of the script:

```
RConsole
> star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE, dimnames = list(c("A New Hope", "The Empire Strikes Back", "Return of the Jedi"), c("US", "non-US"))))
> worldwide_box_office <- rowSums(star_wars_matrix)
> worldwide_vector <- worldwide_box_office
> worldwide_vector <- worldwide_vector / 3
> # Bind the new variable worldwide_vector as a column to star-wars-matrix
> all_wars_matrix <- cbind(star_wars_matrix, worldwide_vector)
```

The sidebar on the left shows a progress bar with 100 XP earned and 5 stars. It also includes a section titled "Rate this exercise" with a 5-star rating.

The Workspace

The workspace is your current R working environment and includes any user-defined objects (vectors, matrices, data frames, lists, functions). At the end of an R session, the user can save an image of the current workspace that is automatically reloaded the next time R is started. Commands are entered interactively at the R user prompt. Up and down arrow keys scroll through your command history.

You will probably want to keep different projects in different physical directories. Here are some standard commands for managing your workspace.

```
getwd() # print the current working directory - cwd  
ls() # list the objects in the current workspace
```

```
setwd(mydirectory) # change to mydirectory  
setwd("c:/docs/mydir") # note / instead of \ in windows  
setwd("/usr/rob/mydir") # on linux
```

```
# view and set options for the session  
help(options) # learn about available options  
options() # view current option settings  
options(digits=3) # number of digits to print on output
```

Exercise

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 along with data on the original trilogy,
- `star_wars_matrix2`, with similar data for the prequels trilogy.

Type the name of these matrices in the console and hit Enter 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`.

 Take Hint (-30 XP)

as a reaction, every `cbind()` has an `rbind()`.
Pretty bad with metaphors.)

Here all variables you defined 'live' (check out what a
variable has been initialized and contains two matrices:

matrix that we have used all along, with data on the

`matrix2`, with similar data for the prequels trilogy.

Type these matrices in the console and hit Enter if you want to
see what you want to check out the contents of the workspace,
in the console.

100 XP

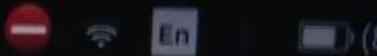
script.R

```
1 # star_wars_matrix and star_wars_matrix2 are available in your workspace
2 star_wars_matrix
3 star_wars_matrix2
4 # Combine both Star Wars trilogies in one matrix
5 all_wars_matrix <-
```

R Console

```
> star_wars_matrix
A New Hope US non-US
The Empire Strikes Back 461.8 314.4
Return of the Jedi 290.5 247.9
> ls()
[1] "star_wars_matrix" "star_wars_matrix2"
> star_wars_matrix2
The Phantom Menace US non-US
Attack of the Clones 474.5 552.6
Attack of the Clones 310.2 338.7
```

Run Code



er-3-matrices-3?ex=6

← Course Outline →

script.R

```
1 # star_wars_matrix and star_wars_matrix2 are available in your works
2 star_wars_matrix
3 star_wars_matrix2
4 # Combine both Star Wars trilogies in one matrix
5 all_wars_matrix <- rbind(star_wars_matrix,star_wars_matrix2)
```

Ctrl+Shift+Enter

R Console

```
A New Hope           461.0 314.4
The Empire Strikes Back 290.5 247.9
Return of the Jedi      309.3 165.8
> star_wars_matrix2
                           US non-US
The Phantom Menace    474.5 552.5
Attack of the Clones 310.7 338.7
Revenge of the Sith   380.3 468.5
> # 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 | R - Chromium

The total box office ×

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-3-matrices-3?ex=7

DataCamp

Exercise

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.

Take Hint (-30 XP)

script.R

```
1 # all_wars_matrix is available in your workspace
2 all_wars_matrix
3
4 # Total revenue for US and non-US
5 total_revenue_vector <- 
6
7 # Print out total_revenue_vector
```

R Console

```
> all_wars_matrix
   [,1]      [,2]
A New Hope    461.0  314.4
The Empire Strikes Back 298.5  247.9
Return of the Jedi 309.3  165.8
The Phantom Menace 474.5  552.5
Attack of the Clones 318.7  338.7
Revenge of the Sith 388.3  468.5
>
```

matrices-3/ex-1

Course Outline

script.R

```
1 # all_wars_matrix is available in your workspace
2 all_wars_matrix
3
4 # Total revenue for US and non-US
5 total_revenue_vector <- colSums(all_wars_matrix)
6
7 # Print out total_revenue_vector
8 total_revenue_vector
```

Run all

R Console

```
Attack of the Clones    310./ 338./
Revenge of the Sith / 380.3 468.5
>
> # Total revenue for US and non-US
> total_revenue_vector <- colSums(all_wars_matrix)
>
> # Print out total_revenue_vector
> total_revenue_vector
  US non-US
2226.3 2087.8
```

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.

```
1 # all_wars_matrix is
2 all_wars_matrix
3
4 # Select the non-US
5 non_us_all <-
6
7 # Average non-US revenue
8
9
10 # Select the non-US revenue
11 non_us_some <-
12
13 # Average non-US revenue
14
```

R Console

> `all_wars_matrix`

	US
A New Hope	461.0
The Empire Strikes Back	298.5
Return of the Jedi	389.3
The Phantom Menace	474.5
Attack of the Clones	310.7
Revenge of the Sith	388.3

> |

Exercise

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`.

 Take Hint (-30 XP)

script.R

```
1 # all_wars_matrix is available in your workspace
2 all_wars_matrix
3
4 # Select the non-US revenue for all movies
5 non_us_all <- all_wars_matrix[,2]
6
7 # Average non-US revenue
8 mean(non_us_all)
9
10 # Select the non-US revenue for first two movies
11 non_us_some <- all_wars_matrix[1:2,2]
12
13 # Average non-US revenue for first two movies
14 mean(non_us_some)
```

R Console

```
> # Select the non-US revenue for all movies
> non_us_all <- all_wars_matrix[,2]
>
> # Average non-US revenue
> mean(non_us_all)
[1] 347.9667
>
> # 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
```

Exercise

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`.
- Printout `visitors` so you can have a look.

 Take Hint (-30 XP)

script.R

```
1 # all_wars_matrix is available in your workspace
2 all_wars_matrix
3
4 # Estimate the visitors
5 visitors <- 
6
7 # Print the estimate to the console
8
```

R Console

```
> all_wars_matrix
   US non-US
A New Hope      461.0  314.4
The Empire Strikes Back 290.5  247.9
Return of the Jedi    309.3  165.8
The Phantom Menace  424.5  552.5
Attack of the Clones 318.3  330.7
Revenge of the Sith  359.3  468.9
```

```
1 # all_wars_matrix is available in your workspace
2 all_wars_matrix
3
4 # Estimate the visitors
5 visitors <- (all_wars_matrix)/5
6
7 # Print the estimate to the console
8 visitors
```

100 XP

Run Code

Submit An

R Console

```
"Estimating the visitors"
> visitors <- (all_wars_matrix)/5
>
> # Print the estimate to the console
> visitors
   US non-US
A New Hope      92.20  62.88
The Empire Strikes Back 58.10  49.58
Return of the Jedi    61.86  33.16
The Phantom Menace  94.98 118.58
Attack of the Clones 62.14  67.74
```

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-3-matrices-3?ex=10

DataCamp

Exercise

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.

Take Hint (-30 XP)

script.R

```
1 # all_wars_matrix and ticket_prices_matrix are available
2 all_wars_matrix
3 ticket_prices_matrix
4
5 # Estimated number of visitors
6 visitors <- 
7
8 # US visitors
9 us_visitors <-
10
11 # Average number of US visitors
12
```

R Console

	US	non-US
Attack of the Clones	310.7	338.7
Revenge of the Sith	380.3	468.5
> ticket_prices_matrix		
A New Hope	5.8	5.8
The Empire Strikes Back	6.8	6.8
Return of the Jedi	7.0	7.0
The Phantom Menace	4.8	4.8
Attack of the Clones	4.5	4.5
Revenge of the Sith	4.9	4.9



script.R

```

1 # all_wars_matrix and ticket_prices_matrix are available in you
2 all_wars_matrix
3 ticket_prices_matrix
4
5 # Estimated number of visitors
6 visitors <- all_wars_matrix/ticket_prices_matrix
7
8 # US visitors
9 us_visitors <- visitors[,1]
10
11 # Average number of US visitors
12 mean(us_visitors)
13 # neeraj:printing result
14 us_visitors

```



Run Code

R Console

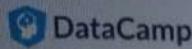
```

> us_visitors <- visitors[,1]
>
> # Average number of US visitors
> mean(us_visitors)
[1] 75.01401
> # neeraj:printing result
> us_visitors

```

	A New Hope	The Empire Strikes Back
	92.20000	48.41667
The Phantom Menace	Attack of the Clones	
	118.62500	69.04444

Return of the Jedi	44.18571
Revenge of the Sith	77.61224



Exercise

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 for categorical variables".

[Take Hint \(-30 XP\)](#)

Course Outline

script.R

```
1 # Assign to the variable theory what this chapter is about
2 theory<- "factors for categorical variables"
```

Run all

R Console

```
> # Assign to the variable theory what this chapter is about
> theory<- "factors for categorical variables"
>
```

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.

Take Hint (-30 XP)

script.R

```
1 # Sex vector
2 sex_vector <- c("Male", "Female", "Female", "Male",
3
4 # Convert sex_vector to a factor
5 factor_sex_vector <- factor(sex_vector)
6
7 # Print out factor_sex_vector
8 factor_sex_vector
```

R Console

```
> # 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
> factor_sex_vector
[1] Male   Female Female Male   Male
Levels: Female Male
```

4-factors-4?ex=2

Course Outline →

RDocumentation

Search for packages, functions, etc.

R Enterprise Training

factor

From base v3.5.2
by R-core R-core@R-project.org

16th Percentile

Factors

The function `factor` is used to encode a vector as a factor (the terms 'category' and 'enumerated type' are also used for factors). If argument `ordered` is `TRUE`, the factor levels are assumed to be ordered. For compatibility with S there is also a function `ordered`. `is.factor`, `is.ordered`, `as.factor` and `as.ordered` are the membership and coercion functions for these classes.

Keywords NA, category

Created by DataCamp.com

R documentation

R Console

```
> 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
> factor_sex_vector
[1] Male   Female Female Male   Male
Levels: Female Male
> ?factor
> |
```

factor and why would you use it? (2) | R - Chromium

It's a factor and x

https://campus.datacamp.com/courses/free-introduction-to-r/chapt

caCamp



Factor and why would you use it? (2)

+100 XP

Great! If you want to find out more about the `factor()` function, do not hesitate to type `?factor` in the console. This will open up a help page. Continue to the next exercise.

Please rate this exercise:



PRESS ENTER TO

Continue

Become a power user!

Exercise

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

Click 'Submit 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.

Take Hint (-30 XP)

ied order. This
an the other'.
ector with the
"Horse".
y the other. (Note

consider for
with the
s obvious that
above

100 XP

```
2 animals_vector <- c("Elephant", "Giraffe", "Donkey", "Horse")
3 factor_animals_vector <- factor(animals_vector)
4 factor_animals_vector
5
6 # Temperature
7 temperature_vector <- c("High", "Low", "High", "Low", "Medium")
8 factor_temperature_vector <- factor(temperature_vector, order = TRUE, levels =
  ("Low", "Medium", "High"))
9 factor_temperature_vector
```



Run Code

Submit Answer

R Console

```
> factor_animals_vector
[1] Elephant Giraffe Donkey Horse
Levels: Donkey Elephant Giraffe Horse
>
> # 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
[1] High Low High Low Medium
Levels: Low < Medium < High
```

Exercise

Factor levels

When you first get a data set, 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:

 Instructions

100 XP

Exercise

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.

 Take Hint (-30 XP)



script.R

```
1 # Code to build factor_survey_vector
2 survey_vector <- c("M", "F", "F", "M", "M")
3 factor_survey_vector <- factor(survey_vector)
4
5 # Specify the levels of factor_survey_vector
6 levels(factor_survey_vector) <- c("Female", "Male")
7
8 factor_survey_vector
```



Run

R Console

```
> # 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("Female", "Male")
>
> factor_survey_vector
[1] Male   Female Female Male   Male
Levels: Female Male
```

Summarizing a factor | R - Chromium

Summarizing a factor x

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-4-factors-4?ex=5

DataCamp

Exercise

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?

Take Hint (-30 XP)

script.R

```
1 # Build factor_survey_vector with clean levels
2 survey_vector <- c("M", "F", "F", "M", "M")
3 factor_survey_vector <- factor(survey_vector)
4 levels(factor_survey_vector) <- c("Female", "Male")
5 factor_survey_vector
6
7 # Generate summary for survey_vector
8 summary(survey_vector)
9
10 # Generate summary for factor_survey_vector
11 summary(factor_survey_vector)
```

R Console

```
>
> # Generate summary for survey_vector
> summary(survey_vector)
  Length   Class    Mode
      5 character character
>
> # Generate summary for factor_survey_vector
> summary(factor_survey_vector)
Female   Male
      2       3
>
```

script.R

Course Outline

```
1 # Build factor_survey_vector with clean levels
2 survey_vector <- c("M", "F", "F", "M", "4")
3 factor_survey_vector <- factor(survey_vector)
4 levels(factor_survey_vector) <- c("Female", "Male", "Other")
5 factor_survey_vector
6
7 # Generate summary for survey_vector
8 summary(survey_vector)
9
10 # Generate summary for factor_survey_vector
11 summary(factor_survey_vector)
```

Console

```
survey_vector <- c("M", "F", "F", "M", "4")
factor_survey_vector <- factor(survey_vector)
levels(factor_survey_vector) <- c("Female", "Male")
Warning: number of levels differs
factor_survey_vector
[1] M F F M 4
levels: 4 F M

Generate summary for survey_vector
summary(survey_vector)
Length: 5
Classes: factor Mode: factor
```



Course Outline



script.R

```
1 # Build factor_survey_vector with clean levels
2 survey_vector <- c("M", "F", "F", "M", "4")
3 factor_survey_vector <- factor(survey_vector)
4 levels(factor_survey_vector) <- c("Female", "Male", "44")
5 factor_survey_vector
6
7 # Generate summary for survey_vector
8 summary(survey_vector)
9
10 # Generate summary for factor_survey_vector
11 summary(factor_survey_vector)
```



Run Code

R Console

```
> # Build factor_survey_vector with clean levels
> survey_vector <- c("M", "F", "F", "M", "4")
> factor_survey_vector <- factor(survey_vector)
> levels(factor_survey_vector) <- c("Female", "Male", "44")
> factor_survey_vector
[1] 44     Male   Male   44     Female
levels: Female Male 44
```

```
> # Generate summary for survey_vector
> summary(survey_vector)
Length: 5 Class : Factor
```

script.R

```
1 # Build factor_survey_vector with clean levels
2 survey_vector <- c(43,4)
3 factor_survey_vector <- factor(survey_vector)
4 levels(factor_survey_vector) <- c(2,4)
5 factor_survey_vector
6
7 # Generate summary for survey_vector
8 summary(survey_vector)
9
10 # Generate summary for factor_survey_vector
11 summary(factor_survey_vector)
```

R Console

```
>
> # Generate summary for survey_vector
> summary(survey_vector)
  Min. 1st Qu. Median      Mean 3rd Qu.      Max.
  4.00   13.75  23.50    23.50  33.25    43.00
>
> # Generate summary for factor_survey_vector
> summary(factor_survey_vector)
2 4
3 1
>
```

5

Run



script.R

```
1 # Build factor_survey_vector with clean levels
2 survey_vector <- c("M", "F", "F", "M", "M")
3 factor_survey_vector <- factor(survey_vector)
4 levels(factor_survey_vector) <- c("Female", "Male")
5
6 # Male
7 male <- factor_survey_vector[1]
8
9 #neeraj:male
10 male
11
12 # Female
13 female <- factor_survey_vector[2]
14
```



Run Code

R Console

```
> male
[1] Male
Levels: Female Male
>
> # Female
> female <- factor_survey_vector[2]
>
> # Battle of the sexes: Male 'larger' than female?
> male > female
Warning message: '>' not meaningful for factors
[1] NA
```

dered factors | R - Chromium
Ordered factors | R

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-4-factors-4?ex=7

DataCamp

Exercise

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...

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,

script.R

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

R Console

```
1: # Create speed_vector
2: speed_vector <- ("medium",
   ^

Parsing error in script.R:2:25: unexpected ','
```

```
1: # Create speed_vector
2: speed_vector <- ("medium",
   ^
   > # Create speed_vector
   > speed_vector <- c("medium", "slow", "slow", "medium", "fast")
```

Exercise

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") .
```

Take Hint (-30 XP)

script.R

```
1 # Create speed_vector  
2 speed_vector <- c("medium", "slow", "slow", "medium"  
3  
4 # Convert speed_vector to ordered factor vector  
5 factor_speed_vector <- factor(speed_vector,  
6  
7  
8  
9 # Print factor_speed_vector  
10 factor_speed_vector  
11 summary(factor_speed_vector)
```

R Console

```
ordered=TRUE,  
levels=c("slow", "medium", "fast"))  
>  
> # Print factor_speed_vector  
> factor_speed_vector  
[1] medium slow slow medium fast  
Levels: slow < medium < fast  
> summary(factor_speed_vector)  
slow medium fast  
      2      2      1
```

```
1 # Create speed_vector
2 speed_vector <- c("medium", "slow", "slow", "medium", "fast")
3
4 # Convert speed_vector to ordered factor vector
5 factor_speed_vector <- factor(
6
7
8
9 # Print factor_speed_vector
10 factor_speed_vector
11 summary(factor_speed_vector)
```



Run Code

Submit Answer

R Console

100 XP

```
ordered=TRUE,
levels=c("slow", "medium", "fast"))
```

```
>
> # Print factor_speed_vector
> factor_speed_vector
ordered(0)
Levels: slow < medium < fast
> summary(factor_speed_vector)
  slow medium   fast
    0     0     0
```

```
    "slow", "medium", "fast")
4 # Convert speed_vector to ordered factor vector
5 factor_speed_vector <- factor(speed_vector,
6                               ordered=FALSE,
7                               levels=c("slow", "medium", "fast"))
8
9 # Print factor_speed_vector
10 factor_speed_vector
11 summary(factor_speed_vector)
```

II

R Console



Run Code

Submit Answer

```
ordered=FALSE,
levels=c("slow", "medium", "fast"))
>
> # Print factor_speed_vector
> factor_speed_vector
[1] medium slow  slow  medium fast
Levels: slow medium fast
> summary(factor_speed_vector)
 slow medium  fast
 2      2      1
>
```

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.

[Take Hint \(-30 XP\)](#)

script.R

```
1 # Create factor_speed_vector
2 speed_vector <- c("medium", "slow", "slow", "medium", "fast")
3 factor_speed_vector <- factor(speed_vector, ordered = TRUE)
4
5 # Factor value for second data analyst
6 da2 <- factor_speed_vector[2]
7 #neeraj:da2
8 da2
9
10 # Factor value for fifth data analyst
11 da5 <- factor_speed_vector[5]
12
13 # Is data analyst 2 faster than data analyst 5?
```

R Console

```
> # Factor value for second data analyst
> da2 <- factor_speed_vector[2]
> #neeraj:da2
> da2
[1] slow
Levels: slow < medium < fast
>
> # Factor value for fifth data analyst
> da5 <- factor_speed_vector[5]
>
> # Is data analyst 2 faster than data analyst 5?
```

Comparing ordered

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-4-factors-4?ex=9

DataCamp

Exercise

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.

Take Hint (-30 XP)

script.R

```
2 speed_vector <- c("medium", "slow", "slow", "medium", "fast")
3 factor_speed_vector <- factor(speed_vector, ordered = TRUE, levels
  "medium", "fast"))
4
5 # Factor value for second data analyst
6 da2 <- factor_speed_vector[2]
7 #neeraj:da2
8 da2
9
10 # Factor value for fifth data analyst
11 da5 <- factor_speed_vector[5]
12
13 # Is data analyst 2 faster than data analyst 5?
14 da2>da5
```

R Console

```
> da2
[1] slow
Levels: slow < medium < fast
>
> # Factor value for fifth data analyst
> da5 <- factor_speed_vector[5]
>
> # Is data analyst 2 faster than data analyst 5?
> da2>da5
[1] FALSE
```

Run Code

What's a data frame

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-5-data-frames?ex=1

DataCamp

Exercise

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 data set 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 data set of different data types. You will often find yourself working with data sets that contain different data types instead of only one.

A data frame has the variables of a data set 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

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

script.R

```
1 # Print out built-in R data frame
2 mtcars
```

R Console

```
> # Print out built-in R data frame
> mtcars
```

	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	18.67	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	3.45	3.440	17.02	0	0	2	1
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Cougar 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Fiat 128	24.4	4	140.0	62	3.69	2.900	18.00	1	0	2	1

Quick, have a look at your data set

Wow, that is a lot of cars!

Working with large data sets is not uncommon in data analysis. When you work with (extremely) large data sets 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 data set.

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 data set.

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

Instructions

100 XP

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

Take Hint (-30 XP)

script.R

```
1 # Call head() on mtcars
2 head(mtcars)
3 tail(mtcars
4 )
```

R Console

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4		
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4		
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4		
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3		
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3		
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3		
> tail(mtcars)												
Porsche 914-2	26.0	4	120	3	91.4	4.43	2.140	16.7	0	1	5	2

+100 XP

! So, what do we have in this data set? For example, presents the car's horsepower; the Datsun has theorse power of the 6 cars that are displayed. For a fullw of the variables' meaning, type `?mtcars` in the and read the help page. Continue to the next exercise!

PRESS ENTER TO

Continue

Become a power user!

SUBMIT ANSWER: **CTRL + SHIFT + ENTER**

See all keyboard shortcuts

script.R RDocumentation

MOTOR TREND CAR ROAD TESTS

The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel of automobile design and performance for 32 automobiles (1973–74 models).

Keywords datasets

Usage

mtcars

Format

A data frame with 32 observations on 11 (numeric) variables.

[1]	mpg	Miles/(US) gallon
[2]	cyl	Number of cylinders
[3]	disp	Displacement (cu.in.)
Ferrari Dino	19.7	6 145.0 175 3.62 2.770 15.5 0 1 5 6
Maserati Bora	15.0	8 301.0 335 3.54 3.570 14.6 0 1 5 8
Volvo 142E	21.4	4 121.0 109 4.11 2.780 18.6 1 1 4 2
> ?mtcars		

R Console

Created by DataCamp.com

Exercise

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 data set. 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 data set or data frame. It is a great way to get more insight in your data set 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.

Take Hint (-30 XP)

script.R

```
1 # Investigate the structure of mtcars  
2 str(mtcars)
```

R Console

```
> # Investigate the structure of mtcars  
> str(mtcars)  
'data.frame': 32 obs. of 11 variables:  
 $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...  
 $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...  
 $ disp: num 160 160 108 258 360 ...  
 $ hp : num 110 110 93 110 175 105 245 62 95 123 ...  
 $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.90 ...  
 $ wt : num 2.62 2.88 2.32 3.21 3.44 ...  
 $ qsec: num 16.5 17 18.6 19.4 17 ...  
 $ vs : num 0 0 1 1 0 1 0 1 1 1 ...  
 $ am : num 1 1 1 0 0 0 0 0 0 0 ...  
 $ gear: num 4 4 4 3 3 3 3 4 4 4 ...  
 $ carb: num 4 4 1 1 2 1 4 2 2 4 ...  
>
```

Exercise

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 on the right. 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` .

script.R

```

1 # Definition of vectors
2 name <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn",
  "Uranus", "Neptune")
3 type <- c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
  "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant")
4
5 diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
6 rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
7 rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)
8
9 # Create a data frame from the vectors
10 planets_df <- data.frame(name, type, diameter, rotation, rings)
11

```

Submit

R Console

```

> 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")
> 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)
>
> # Create a data frame from the vectors
> planets_df <- data.frame(name, type, diameter, rotation, rings)
>

```

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-5-data-frames?ex=6

DataCamp

Exercise

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.

Take Hint (-30 XP)

script.R

```
1 # The planets_df data frame from the previous exercise
2
3 # Print out diameter of Mercury (row 1, column 3)
4 planets_df[1,3]
5
6 # Print out data for Mars (entire fourth row)
7 planets_df[4,]
```

R Console

```
> # The planets_df data frame from the previous exercise is pre-loaded
>
> # Print out diameter of Mercury (row 1, column 3)
> planets_df[1,3]
[1] 0.382
>
> # Print out data for Mars (entire fourth row)
> planets_df[4,]
  name           type diameter rotation rings
4 Mars Terrestrial planet     0.532      1.03 FALSE
```

Exercise

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`.

 Take Hint (-30 XP)

Incorrect Submission

Have you correctly selected the first five values from the diameter column and printed them out? You can use `[1:5, "diameter"]` here.

Did you find this feedback helpful?

Yes No

script.R

```
1 # The planets_df data frame from the previous exercise is pre-loaded
2
3 # Select first 5 values of diameter column
4 planets_df[1:5,"diameter"]
```

R Console

```
> # The planets_df data frame from the previous exercise is pre-loaded
>
> # Select first 5 values of diameter column
> planets_df[1:5,"diameter"]
[1] 11.209
>
```



Course Outline



script.R

```
1 # The planets_df data frame from the previous exercise is pre-loaded
2
3 # Select first 5 values of diameter column
4 planets_df[1:5,"diameter"]
```



Run Code

Su

R Console

```
>
> # Select first 5 values of diameter column
> planets_df[5,"diameter"]
[1] 11.209
>
> # The planets_df data frame from the previous exercise is pre-loaded
>
> # Select first 5 values of diameter column
> planets_df[1:5,"diameter"]
[1] 0.382 0.949 1.000 0.532 11.209
>
```

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.

 Take Hint (-30 XP)

script.R

```
1 # planets_df is pre-loaded in your workspace  
2  
3 # Select the rings variable from planets_df  
4 rings_vector <- planets_df$rings  
5  
6 # Print out rings_vector  
7 rings_vector
```

R Console

```
> # planets_df is pre-loaded in your workspace  
>  
> # Select the rings variable from planets_df  
> rings_vector <- planets_df$rings  
>  
> # Print out rings_vector  
> rings_vector  
[1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE  
>
```

Exercise

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 on the right 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.

Take Hint (-30 XP)

script.R

```
1 # planets_df and rings_vector are pre-loaded in your workspace
2
3 # Adapt the code to select all columns for planets with rings
4 planets_df[rings_vector, "name"]
```

Run all/selected



Run C

R Console

```
> # 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"]
[1] Jupiter Saturn Uranus Neptune
Levels: Earth Jupiter Mars Mercury Neptune Saturn Uranus Venus
```

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 on the right selects the `name` column of all planets that have rings. Adapt the code so that instead of only the `name` column, *all columns* or planets that have rings are selected.

Take Hint (-30 XP)

Incorrect Submission

Have you correctly adapted the code to select *all columns* for the planets that have rings? You can use `planets_df[rings_vector,]`. Make sure to include the comma here, it's crucial!

Did you find this feedback helpful?

Yes No

script.R

```
1 # planets_df and rings_vector are pre-loaded in your workspace
2
3 # Adapt the code to select all columns for planets with rings
4 planets_df[rings_vector, "name"]
```

R Console

```
> # Adapt the code to select all columns for planets with rings
> planets_df[rings_vector, "name"]
[1] Jupiter Saturn Uranus Neptune
Levels: Earth Jupiter Mars Mercury Neptune Saturn Uranus Venus
> # 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"]
[1] Jupiter Saturn Uranus Neptune
Levels: Earth Jupiter Mars Mercury Neptune Saturn Uranus Venus
```

Exercise

nes. Could K help you out?

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 on the right 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.

Take Hint (-30 XP)

Incorrect Submission

Have you correctly adapted the code to select *all* columns for the planets that have rings? You can use `planets_df[rings_vector,]`. Make sure to include the comma here, it's crucial!

Did you find this feedback helpful?

Yes No

script.R

```
1 # planets_df and rings_vector are pre-loaded in your workspace
2
3 # Adapt the code to select all columns for planets with rings
4 planets_df[rings_vector, ]
```

R Console

Levels: Earth Jupiter Mars Mercury Neptune Saturn Uranus Venus
> # 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 type diameter rotation rings
5 Jupiter Gas giant 11.209 0.41 TRUE
6 Saturn Gas giant 9.449 0.43 TRUE
7 Uranus Gas giant 4.887 -0.72 TRUE
8 Neptune Gas giant 3.883 0.67 TRUE

Exercise

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 data set 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

Use `subset()` on `planets_df` to select planets that have a diameter

script.R

```
1 # planets_df is pre-loaded in your workspace
2
3 # Select planets with diameter < 1
4 subset(planets_df, diameter<1)
```

R Console

```
> # planets_df is pre-loaded in your workspace
>
> # Select planets with diameter < 1
> subset(planets_df, diameter<1)
   name          type diameter rotation rings
1 Mercury Terrestrial planet    0.382    58.64 FALSE
2 Venus Terrestrial planet    0.949   -243.02 FALSE
4 Mars Terrestrial planet    0.532     1.03 FALSE
>
```

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 data set. 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)
[1] 2 1 3
```

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)]
[1] 10 100 1000
```

Instructions

100 XP

script.R

```
1 # Play around with the order function in t
2 d<-c(2,32,4,26,9)
3 d
4 order(d)
5 e<-d[order(d)]
6 e
7 d[order(d)]
```

I

R Console

```
> d<-c(2,32,4,26,9)
> d
[1] 2 32 4 26 9
> order(d)
[1] 1 3 5 4 2
> e<-d[order(d)]
> e
[1] 2 4 9 26 32
> d[order(d)]
[1] 2 4 9 26 32
```

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 results as `positions`.
- Now reshuffle `planets_df` with the `positions` vector as row indexes inside square brackets. Keep all columns. Simply print out the result.

Take Hint (-30 XP)

Incorrect Submission

Use `planets_df[positions,]` to sort `planets_df`; the comma inside the square brackets is crucial!

Did you find this feedback helpful?

Yes No

script.R

```
1 # planets_df is pre-loaded in your workspace
2
3 # Use order() to create positions
4 positions <- order(planets_df$diameter)
5
6 # Use positions to sort planets_df(keeping all columns)
7 planets_df[positions, ]
8
```

R Console

```
> planets_df[positions, ]
   name          type diameter rotation rings
1 Mercury Terrestrial planet    0.382    58.64 FALSE
4 Mars Terrestrial planet    0.532     1.03 FALSE
2 Venus Terrestrial planet    0.949   -243.02 FALSE
3 Earth Terrestrial planet    1.000     1.00 FALSE
8 Neptune Gas giant      3.883    61.67 TRUE
7 Uranus  Gas giant      4.007    -8.72 TRUE
6 Saturn   Gas giant     9.449     0.43 TRUE
5 Jupiter  Gas giant    11.289     0.41 TRUE
```

Exercise

Lists, why would you need them?

Congratulations! At this point in the course you are already familiar with:

- **Vectors** (one dimensional array): can hold numeric, character or logical values. The elements in a vector all have the same data type.
- **Matrices** (two dimensional array): can hold numeric, character or logical values. The elements in a matrix all have the same data type.
- **Data frames** (two-dimensional objects): can hold numeric, character or logical values. Within a column all elements have the same data type, but different columns can be of different data type.

Pretty sweet for an R newbie, right? :-)

⊕ Instructions

100 XP



 DataCamp

 Exercise

Lists, why would you need them? (2)

A list in R is similar to your to-do list at work or school: the different items on that list most likely differ in length, characteristic, and type of activity that has to be done.

A list in R allows you to gather a variety of objects under one name (that is, the name of the list) in an ordered way. These objects can be matrices, vectors, data frames, even other lists, etc. It is not even required that these objects are related to each other in any way.

You could say that a list is some kind super data type: you can store practically any piece of information in it!

 Instructions 100 XP

Click 'Submit Answer' to start the first exercise on lists.

 Take Hint (-30 XP)

Exercise

Creating a list

Let us create our first list! To construct a list you use the function `list()` :

```
my_list <- list(comp1, comp2 ...)
```

The arguments to the list function are the list components. Remember, these components can be matrices, vectors, other lists, ...

Instructions

100 XP

Construct a list, named `my_list`, that contains the variables `my_vector`, `my_matrix` and `my_df` as list components.

Take Hint (-30 XP)

script.R

```
1 # Vector with numerics from 1 up to 10
2 my_vector <- 1:10
3
4 # Matrix with numerics from 1 up to 9
5 my_matrix <- matrix(1:9, ncol = 3)
6 my_matrix
7
8 # First 10 elements of the built-in data frame mtcars
9 my_df <- mtcars[1:10,]
10
11 # Construct list with these different elements:
12 my_list <- list(my_vector, my_matrix, my_df)
13 my_list
```

R Console

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

[[2]]
 [,1] [,2] [,3]
 [1,]    1    4    7
 [2,]    2    5    8
 [3,]    3    6    9

[[3]]
#> Mazda RX4      mpg cyl  disp  hp drat   wt  qsec vs am gear carb
#> 21.0   6 160.0 110 3.98 2.620 16.46 0  1   4
```

DataCamp

Exercise

names to them:

```
my_list <- list(name1 = your_comp1,  
                 name2 = your_comp2)
```

This creates a list with components that are named `name1`, `name2`, and so on. If you want to name your lists after you've created them, you can use the `names()` function as you did with vectors. The following commands are fully equivalent to the assignment above:

```
my_list <- list(your_comp1, your_comp2)  
names(my_list) <- c("name1", "name2")
```

Instructions

100 XP

- Change the code of the previous exercise (see editor) by adding names to the components. Use for `my_vector` the name `vec`, for `my_matrix` the name `mat` and for `my_df` the name `df`.
- Print out `my_list` so you can inspect the output.

Take Hint (-30 XP)

script.R

```
3  
4 # Matrix with numerics from 1 up to 9  
5 my_matrix <- matrix(1:9, ncol = 3)  
6  
7 # First 10 elements of the built-in data frame  
8 my_df <- mtcars[1:10,]  
9  
10 # Adapt list() call to give the components names  
11 my_list <- list(my_vector, my_matrix, my_df)  
12 names(my_list)<-c("vec", "mat", "df")  
13  
14 # Print out my_list  
15 my_list
```

R Console

```
> my_list  
$vec  
[1] 1 2 3 4 5 6 7 8 9 10  
  
$mat  
[,1] [,2] [,3]  
[1,] 1 4 7  
[2,] 2 5 8  
[3,] 3 6 9  
  
$df  
mpg cyl disp hp drat wt qsec vs
```

```
6
7 # First 10 elements of the built-in data frame mtcars
8 my_df <- mtcars[1:10,]
9
10 # Adapt list() call to give the components names
11 my_list <- list("vec"=my_vector, "mat"= my_matrix, "df"=my_df)
12
13
14 # Print out my_list
15 my_list
```

100 XP

R Console

```
> # Print out my_list
> my_list
$vec
[1] 1 2 3 4 5 6 7 8 9 10

$mat
 [,1] [,2] [,3]
 [1,]    1    4    7
 [2,]    2    5    8
 [3,]    3    6    9
```



Run Code

Creating a named list (2)

Being a huge movie fan (remember your job at LucasFilms), you decide to start storing information on good movies with the help of lists.

Start by creating a list for the movie "The Shining". We have already created the variables `mov`, `act` and `rev` in your R workspace. Feel free to check them out in the console.

Instructions

100 XP

Complete the code on the right to create `shining_list`; it contains three elements:

- `movname`: a character string with the movie title (stored in `mov`)
- `actors`: a vector with the main actors' names (stored in `act`)
- `reviews`: a data frame that contains some reviews (stored in `rev`)

Do not forget to name the list components accordingly (names are `movname`, `actors` and `reviews`).

Take Hint (-30 XP)

script.R

```
1 # The variables mov, act and rev are available
2
3 # Finish the code to build shining_list
4 shining_list <- list(movname = mov)
```

R Console

```
> mov
[1] "The Shining"
> act
[1] "Jack Nicholson"   "Shelley Duvall"   "Danny Lloyd"
[5] "Barry Nelson"
> rev
  scores sources
1     4.5    IMDb1
2     4.0    IMDb2 A truly brilliant and scary film from Stanley
3     5.0    IMDb3   A masterpiece of psychological
```

ng a named list (2)

uge movie fan (remember your job at LucasFilms), you decide to
ing information on good movies with the help of lists.

creating a list for the movie "The Shining". We have already created
bles `mov`, `act` and `rev` in your R workspace. Feel free to
em out in the console.

tions

100 XP

the code on the right to create `shining_list` : it contains three

`name`: a character string with the movie title (stored in `mov`)

`act`: a vector with the main actors' names (stored in `act`)

`rev`: a data frame that contains some reviews (stored in `rev`)

forget to name the list components accordingly (names are
ame, `actors` and `reviews`).

e Hint (-30 XP)

script.R

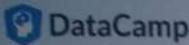
```
1 # The variables mov, act and rev are available
2
3 # Finish the code to build shining_list
4 shining_list <- list("moviename" = mov, "actors"=act, "reviews"=rev)
5 shining_list
```

R Console

```
> # FINISH THE CODE TO BUILD SHINING_LIST
> shining_list <- list("moviename" = mov, "actors"=act, "reviews"=rev)
> shining_list
$moviename
[1] "The Shining"

$actors
[1] "Jack Nicholson"   "Shelley Duvall"   "Danny Lloyd"      "Scatman Crothers"
[5] "Barry Nelson"

$reviews
  scores sources
  1     4.5    IMDb1                               comments
                                         Best Horror Film I Have Ever Seen
```



Exercise

Your list will often be built out of numerous elements and components. Therefore, getting a single element, multiple elements, or a component out of it is not always straightforward.

One way to select a component is using the numbered position of that component. For example, to "grab" the first component of `shining_list` you type

```
shining_list[[1]]
```

A quick way to check this out is typing it in the console. Important to remember: to select elements from vectors, you use single square brackets:

[]. Don't mix them up!

You can also refer to the names of the components, with `[[]]` or with the `$` sign. Both will select the data frame representing the reviews:

```
shining_list[["reviews"]]  
shining_list$reviews
```

Besides selecting components, you often need to select specific elements out of these components. For example, with `shining_list[[2]][1]` you select from the second component, `actors` (`shining_list[[2]]`), the first element (`[1]`). When you type this in the console, you will see the answer is Jack Nicholson.

Instructions

100 XP

script.R

```
1 # shining_list is already pre-loaded in the workspace  
2  
3 # Print out the vector representing the actors  
4 shining_list[[2]]  
5  
6 # Print the second element of the vector representing the actors  
7
```

R Console

```
> # shining_list is already pre-loaded in the workspace  
>  
> # Print out the vector representing the actors  
> shining_list[[2]]  
[1] "Jack Nicholson" "Shelley Duvall" "Danny Li"  
[5] "Barry Nelson"  
>  
> # Print the second element of the vector representing the actors  
> actors(shining_list[[2]])  
Error: could not find function "actors"
```

Exercise

```
shining_list[["reviews"]]
shining_list$reviews
```

Besides selecting components, you often need to select specific elements out of these components. For example, with `shining_list[[2]][1]` you select from the second component, `actors` (`shining_list[[2]]`), the first element (`[1]`). When you type this in the console, you will see the answer is Jack Nicholson.

Instructions

100 XP

- Select from `shining_list` the vector representing the actors. Simply print out this vector.
- Select from `shining_list` the second element in the vector representing the actors. Do a printout like before.

 Take Hint (-30 XP)

Incorrect Submission

To select the second actor from the vector representing actors, you should chain your selections: `shining_list$actors` represents the actors, so you can add a `[2]` to select the second element.

Did you find this feedback helpful?

Yes No

Course Outline

script.R

```
1 # shining_list is already pre-loaded in the workspace
2
3 # Print out the vector representing the actors
4 shining_list[[2]]
5 shining_list$actors
6
7 # Print the second element of the vector representing the actors
8
9 shining_list$actors[2]
```

R Console

```
>
> # Print out the vector representing the actors
> shining_list[[2]]
[1] "Jack Nicholson"   "Shelley Duvall"   "Danny Lloyd"      "Scatman Crothe
[5] "Barry Nelson"
> shining_list$actors
[1] "Jack Nicholson"   "Shelley Duvall"   "Danny Lloyd"      "Scatman Crothe
[5] "Barry Nelson"
>
> # Print the second element of the vector representing the actors
>
> shining_list$actors[2]
[1] "Shelley Duvall"
> shining_list[[2]][2]
[1] "Shelley Duvall"
```

Adding more movie information to the list

Being proud of your first list, you shared it with the members of your movie hobby club. However, one of the senior members, a guy named M. McDowell, noted that you forgot to add the release year. Given your ambitions to become next year's president of the club, you decide to add this information to the list.

To conveniently add elements to lists you can use the `c()` function, that you also used to build vectors:

```
ext_list <- c(my_list, my_val)
```

This will simply extend the original list, `my_list`, with the component `my_val`. This component gets appended to the end of the list. If you want to give the new list item a name, you just add the name as you did before:

```
ext_list <- c(my_list, my_name = my_val)
```

Instructions

100 XP

- Complete the code below such that an item named `year` is added to the `shining_list` with the value 1980. Assign the result to `shining_list_full`.

script.R

```
1 # shining_list, the list containing movie name, actors
  in the workspace
2
3 # We forgot something; add the year to shining_list
4 shining_list_full <- c(shining_list, "year"=1980)
5
6 # Have a look at shining_list_full
7 shining_list_full
8 str(shining_list_full)
```

Ctrl+Shift

R Console

```
> ## Have a look at shining_list_full
> shining_list_full
```

\$moviename

[1] "The Shining"

\$actors

[1] "Jack Nicholson" "Shelley Duvall" "Danny Lloyd" "Stanley

[5] "Barry Nelson"

\$reviews

 scores sources

1 4.5 IMDb1

2 4.8 IMDb2 A truly brilliant and scary film from Stanley Ku

3 5.0 IMDb3

Best Horror Film I Have Ever Seen
A masterpiece of psychological horror

\$year

[1] 1980

```
ext_list <- c(my_list, my_val)
```

This will simply extend the original list, `my_list`, with the component `my_val`. This component gets appended to the end of the list. If you want to give the new list item a name, you just add the name as you did before:

```
ext_list <- c(my_list, my_name = my_val)
```

Instructions

- Complete the code below such that an item named `year` is added to the `shining_list` with the value 1980. Assign the result to `shining_list_full`.
- Finally, have a look at the structure of `shining_list_full` with the `str()` function.

Take Hint (-30 XP)

100 XP

script.R

```
1 # shining_list, the list containing movie name, actors and reviews  
in the workspace  
2  
3 # We forgot something; add the year to shining_list  
4 shining_list_full <- c(shining_list, "year"=1980)  
5  
6 # Have a look at shining_list_full  
7 shining_list_full  
8 str(shining_list_full)
```

R Console

```
1 ...  
2 4.0 IMDb2 A truly brilliant and scary film from Stanley Kubrick  
3 5.0 IMDb3 A masterpiece of psychological horror  
$ year  
[1] 1980  
> str(shining_list_full)  
List of 4  
$ moviename: chr "The Shining"  
$ actors : chr [1:5] "Jack Nicholson" "Shelley Duvall" "Danny Lloyd" "Scatman  
Crothers" ...  
$ reviews : 'data.frame': 3 obs. of 3 variables:  
..$ scores : num [1:3] 4.5 4 5  
..$ sources : Factor w/ 3 levels "IMDb1","IMDb2"...: 1 2 3  
..$ comments: Factor w/ 3 levels "A masterpiece of psychological horror",...: 3 2  
$ year : num 1980
```

Run Code

https://campus.datacamp.com/courses/free-introduction-to-r/chapter-6-

 DataCamp

X script.
1
2
3
4
5
6
7
8

+100 XP

Great! This was the last exercise on R lists! You now have a solid basis in the R programming language, but there's so much more to learn. Check out all the other DataCamp courses and become a true data science expert!

Please rate this exercise:

★★★★★

PRESS ENTER TO

Continue

Take Hint (-20 XP)

See all keyboard shortcuts

R Console

2 4.0
3 5.0

\$year
[1] 1980
> str(shi
List of 4
\$movien
\$actors
\$Crot
\$review
..\$scor
..\$sourc
..\$comm
\$year

Scanned with CamScanner

Set Sail

When the Titanic sank, 1502 of the 2224 passengers and crew got killed. One of the main reasons for this high level of casualties was the lack of lifeboats on this supposedly unsinkable ship.

Those that have seen the movie know that some individuals were more likely to survive the sinking (lucky Rose) than others (poor Jack). In this course you will apply machine learning techniques to predict a passenger's chance of surviving using R.

Let's start with loading in the training and testing set into your R environment. You will use the training set to build your model, and the test set to validate it. The data is stored on the web as CSV files; their URLs are already available as character strings in the sample code. You can load this data with the `read.csv()` function.

Instructions

100 XP

- Use `read.csv()` on `train_url` to create `train`. This is the train data.
- Use `read.csv()` on `test_url` to create `test`. This is the test data.
- Print out `train` and `test` to the console to have a look.

 Take Hint (-30 XP)

script.R

```
1 # Import the training set: train
2 train_url <- "http://s3.amazonaws.com/assets.datacamp.com/titanic/train.csv"
3 train <- read.csv(train_url)
4
5 # Import the testing set: test
6 test_url <- "http://s3.amazonaws.com/assets.datacamp.com/titanic/test.csv"
7 test <- read.csv(test_url)
8
9 # Print train and test to the console
10 train
11 test
```

R Console

```
> test <- read.csv(test_url)
>
> # Print train and test to the console
> train
   PassengerId Survived Pclass
1            1       0      3
2            2       1      1
3            3       1      3
4            4       1      1
5            5       0      3
6            6       0      3
```

Exercise

Rose vs Jack, or Female vs Male

How many people in your training set survived the disaster with the Titanic? To see this, you can use the `table()` command in combination with the `$`-operator to select a single column of a data frame:

```
# absolute numbers  
table(train$Survived)  
  
# proportions  
prop.table(table(train$Survived))
```

If you run these commands in the console, you'll see that 549 individuals died (62%) and 342 survived (38%). A simple prediction heuristic could thus be "majority wins": you predict every unseen observation to not survive.

In general, the `table()` command can help you to explore which variables have predictive value. For example, maybe gender could play a role as well? For a two-way comparison, also including gender, you can use

```
table(train$Sex, train$Survived)
```

To get proportions, you can again wrap `prop.table()` around `table()`, but you'll have to specify whether you want row-wise or column-wise proportions: This is done by setting the second argument of

Instructions

100 XP

script.R

```
1 # Your train and test set are still  
2 str(train)  
3 str(test)  
4  
5 # Survival rates in absolute numbers  
6  
7  
8 # Survival rates in proportions  
9  
10  
11 # Two-way comparison: Sex and Survival  
12  
13  
14 # Two-way comparison: row-wise proportions
```

R Console

```
> table(train$Survived)
```

```
0 1  
549 342
```

```
> prop.table(table(train$Survived))
```

```
[1] 0.000000000 0.002923977 0.002923977 0.  
[7] 0.000000000 0.000000000 0.002923977 0.  
[13] 0.000000000 0.000000000 0.000000000 0.  
[19] 0.000000000 0.002923977 0.000000000 0.  
[25] 0.000000000 0.002923977 0.000000000 0.  
[31] 0.000000000 0.002923977 0.002923977 0.
```

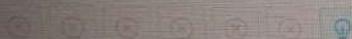
Exercise

With proportions, you can again wrap `prop.table()` around `table()`, but you'll have to specify whether you want row-wise or column-wise proportions: This is done by setting the second argument of `prop.table()`, called `margin`, to 1 or 2, respectively.

Instructions

0 XP

- Call `table()` on `train$Survived` to calculate the survival rates in absolute numbers.
- Calculate the survival rates as proportions by wrapping `prop.table()` around the previous `table()` call.
- Do a two-way comparison on the number of males and females that survived, in absolute numbers. Again, use the `train` data frame.
- Convert the numbers to row-wise proportions.



Hint

- The code for the first, second and third instruction is already given in the assignment!
- For the fourth instruction, wrap `prop.table()` around the `table()` call for the third instruction. Make sure to set the second argument of `prop.table()` correctly!

Did you find this hint helpful?

Course Outline

script.R solution.R

```
2 str(train)
3 str(test)
4
5 # Survival rates in absolute numbers
6 table(train$Survived)
7
8 # As proportions
9 prop.table(table(train$Survived))
10
11 # Two-way comparison: Sex and Survived
12 table(train$Sex, train$Survived)
13
14 # Two-way comparison: row-wise proportions
15 prop.table(table(train$Sex, train$Survived), 1)
```

R Console

```
B   T
female 81 233
male   468 109
>
> # Two-way comparison: row-wise proportions
> prop.table(table(train$Sex, train$Survived), 1)

B   T
female 0.2529618 0.7426382
male   0.8110919 0.1889081
```

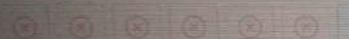
Exercise

For proportions, you can again wrap `prop.table()` around `table()`, but you'll have to specify whether you want row-wise or column-wise proportions: This is done by setting the second argument of `prop.table()`, called `margin`, to 1 or 2, respectively.

Instructions

0 XP

- Call `table()` on `train$Survived` to calculate the survival rates in absolute numbers.
- Calculate the survival rates as proportions by wrapping `prop.table()` around the previous `table()` call.
- Do a two-way comparison on the number of males and females that survived, in absolute numbers. Again, use the `train` data frame.
- Convert the numbers to row-wise proportions.



Hint

- The code for the first, second and third instruction is already given in the assignment!
- For the fourth instruction, wrap `prop.table()` around the `table()` call for the third instruction. Make sure to set the second argument of `prop.table()` correctly!

Did you find this hint helpful?

Yes No

script.R solution.R

```
2 str(train)
3 str(test)
4
5 # Survival rates in absolute numbers
6 table(train$Survived)
7
8 # As proportions
9 prop.table(table(train$Survived))
10
11 # Two-way comparison: Sex and Survived
12 table(train$Sex, train$Survived)
13
14 # Two-way comparison: row-wise proportions
15 prop.table(table(train$Sex, train$Survived), 1)
```

R Console

```
0   1
female 81 233
male   468 109
>
> # Two-way comparison: row-wise proportions
> prop.table(table(train$Sex, train$Survived), 1)

0      1
female 0.2579618 0.7420382
male   0.8110919 0.1889081
```