Informatics

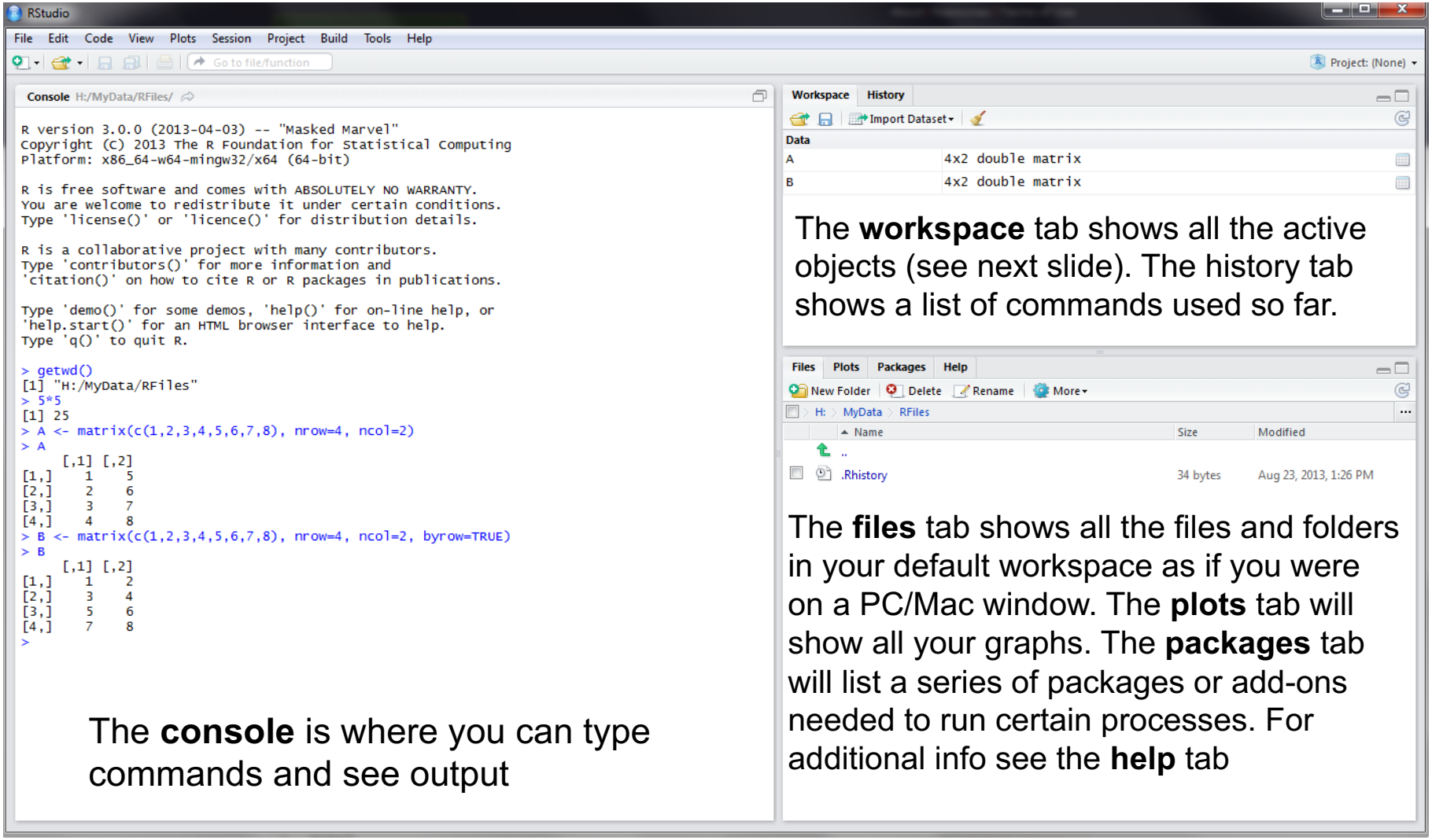
Programming and using R

Claudio Sartori
Department of Computer Science and Engineering
claudio.sartori@unibo.it
https://www.unibo.it/sitoweb/claudio.sartori/

Introduction

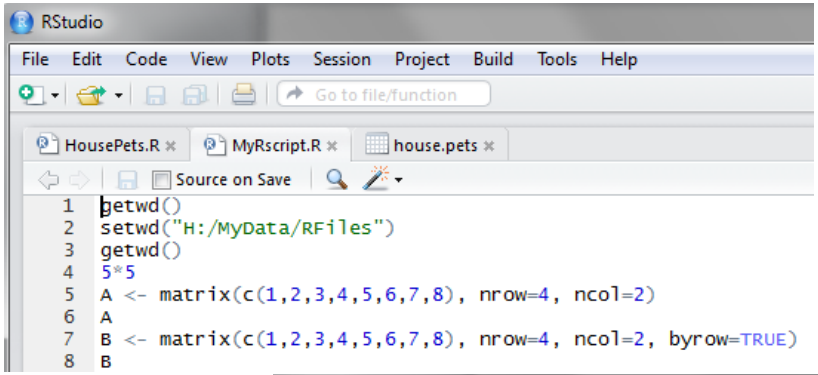
- RStudio allows the user to run R in a more user-friendly environment. It is open-source (i.e. free) and available at http://www.rstudio.com/
- For R related tutorials and/or resources see the following links:
- http://dss.princeton.edu/training/
- http://libguides.princeton.edu/dss

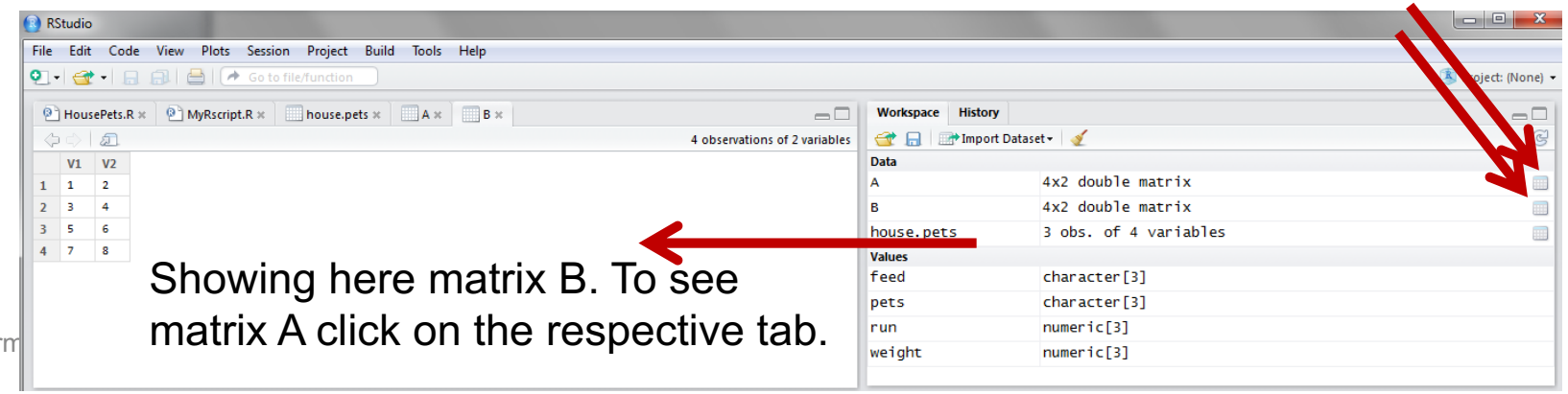
RStudio screen



Workspace tab

- The workspace tab stores any object, value, function or anything you create during your R session. In the example below, if you click on the dotted squares you can see the data on a screen to the left

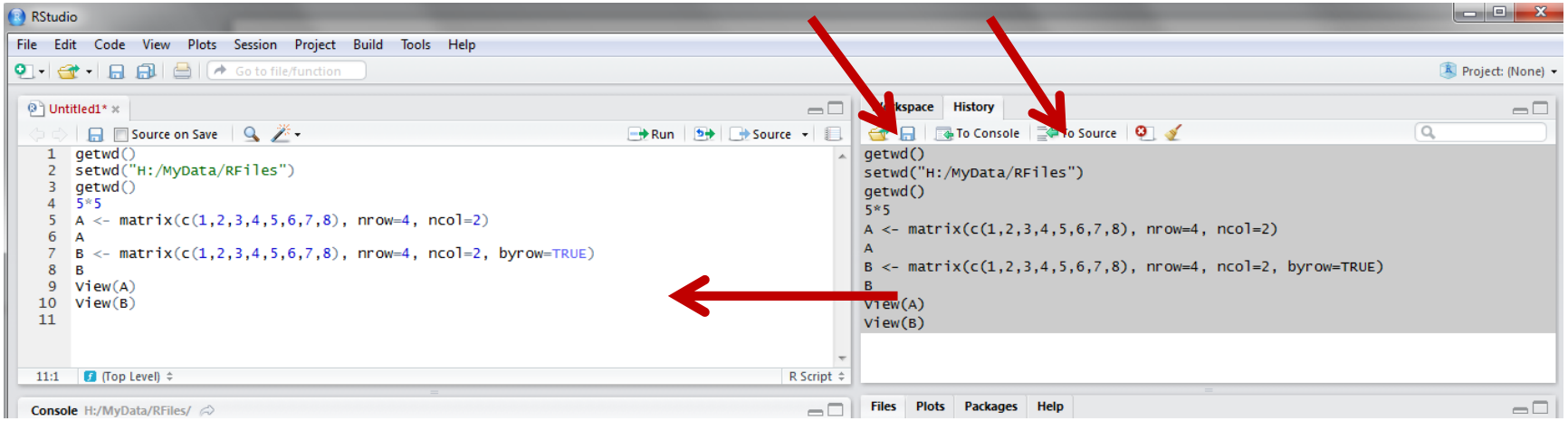




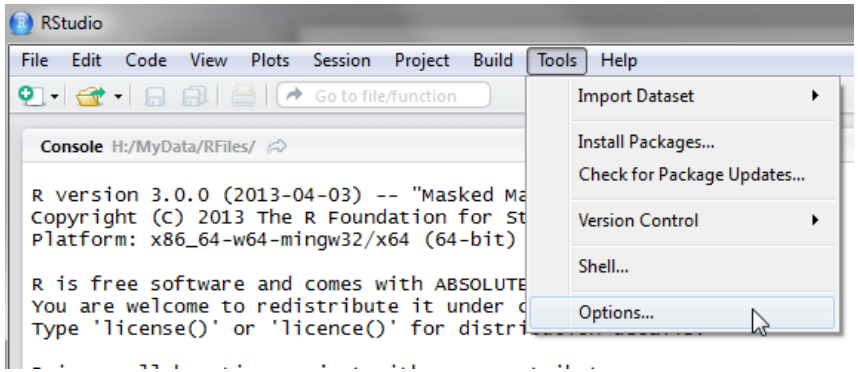
Showing here matrix B. To see matrix A click on the respective tab.

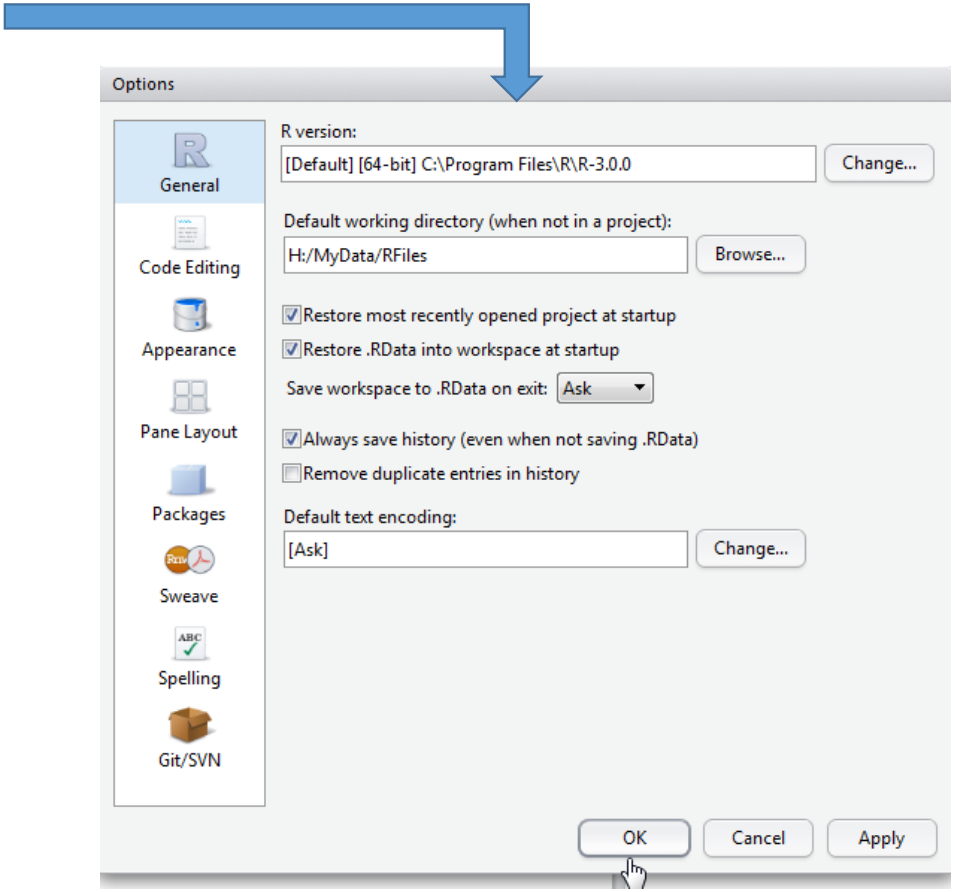
History tab

- The history tab keeps a record of all previous commands. It helps when testing and running processes. Here you can either **save** the whole list or you can **select** the commands you want and send them to an R script to keep track of your work.
- In this example, we select all and click on the “To Source” icon, a window on the left will open with the list of commands. Make sure to save the ‘untitled1’ file as an *.R script.



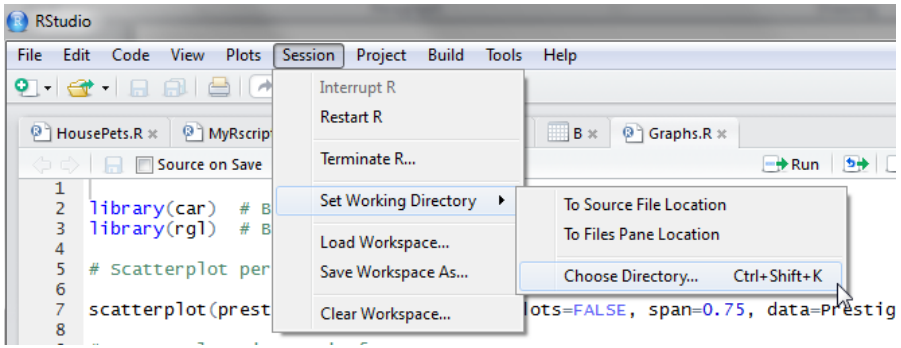
Setting a default working directory

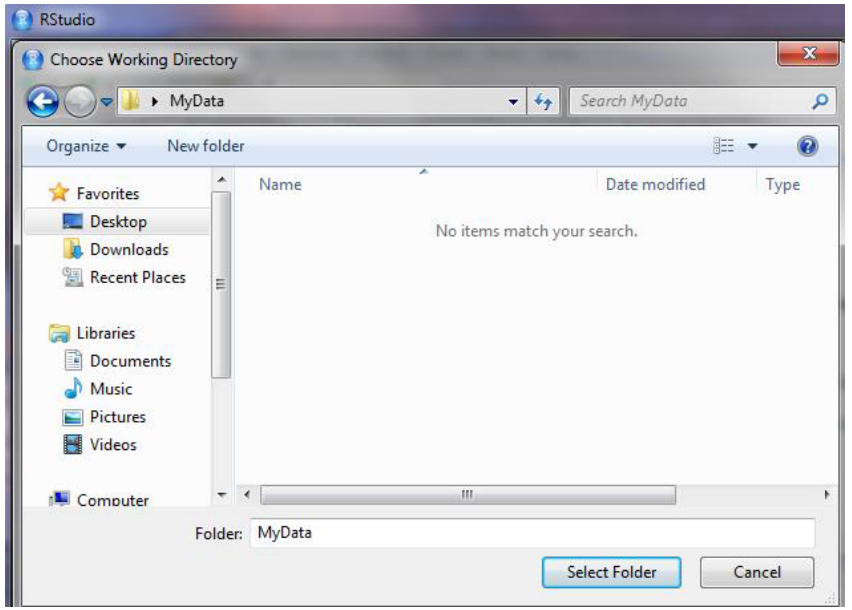




- Every time you open RStudio, it goes to a default directory. You can change the default to a folder where you have your datafiles so you do not have to do it every time. In the menu go to `Tools -> Options`

Changing the working directory



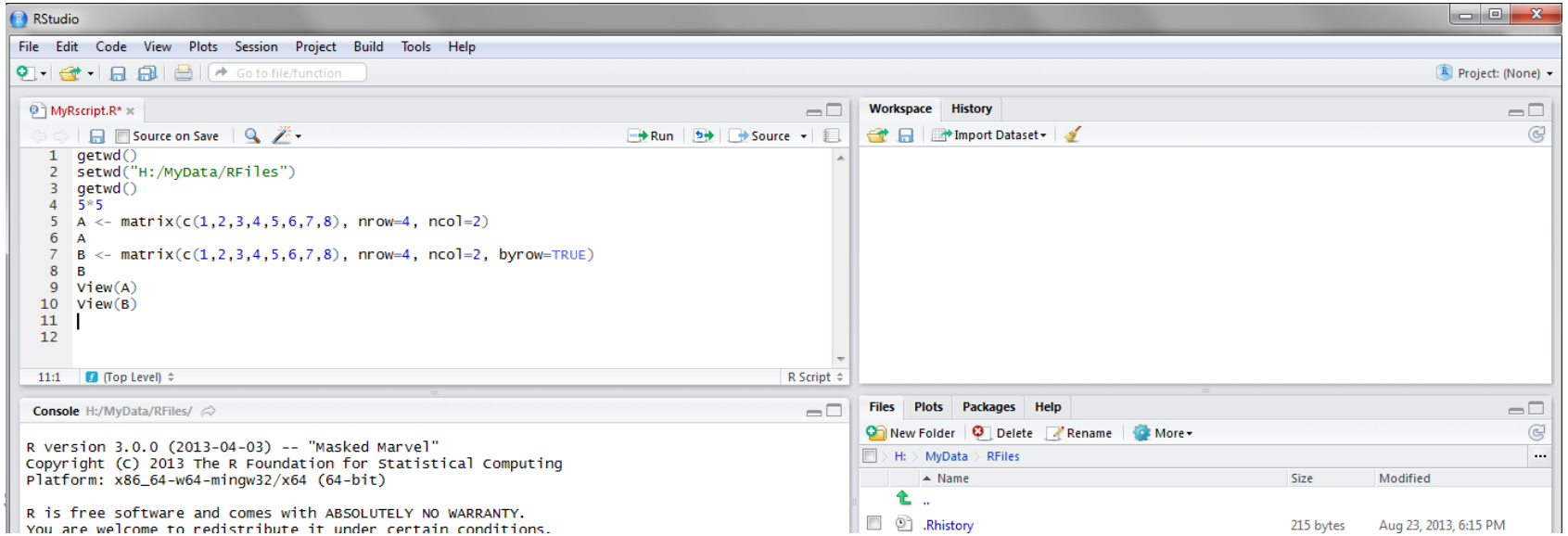


- If you have different projects you can change the working directory for that session, see above. Or you can type:
- `# Shows the working directory (wd)`
- `getwd()`
- `# Changes the wd`
- `setwd("C:/myfolder/data")`

R Script (I)

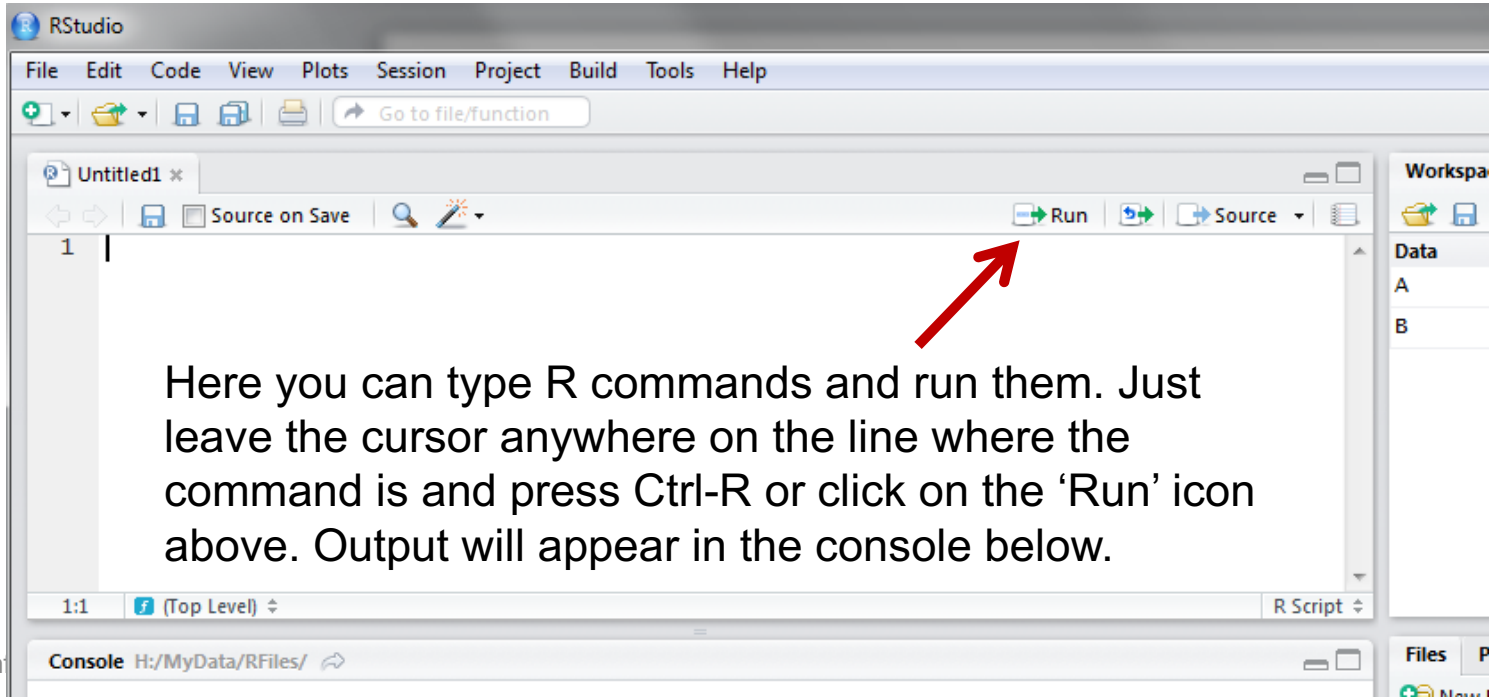
- The usual Rstudio screen has four windows:

1. Console.
2. Workspace and history.
3. Files, plots, packages and help.
4. The R script(s) and data view. The R script is where you keep a record of your work.



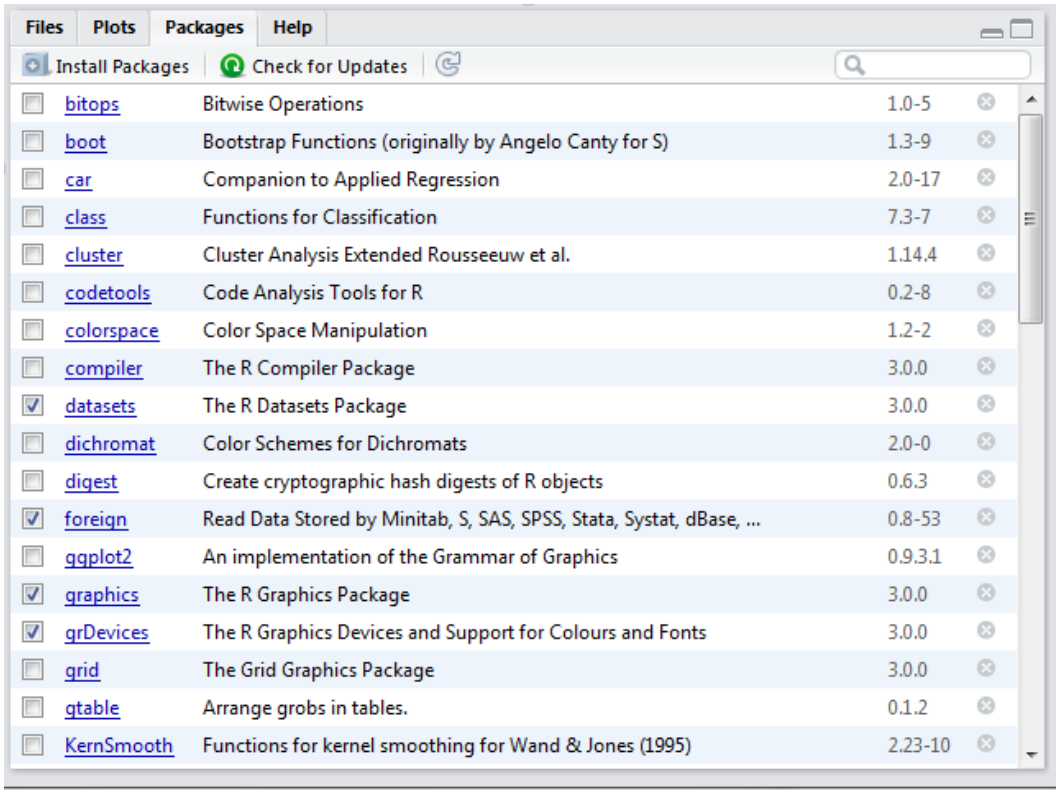
R Script (II)

- To create a new R script you can either go to `File -> New -> R Script`, or click on the icon with the "+" sign and select "R Script", or simply press Ctrl+Shift+N. Make sure to save the script.



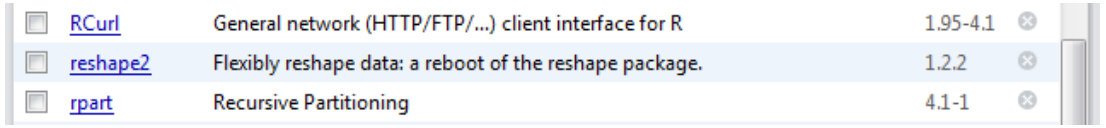
Packages tab

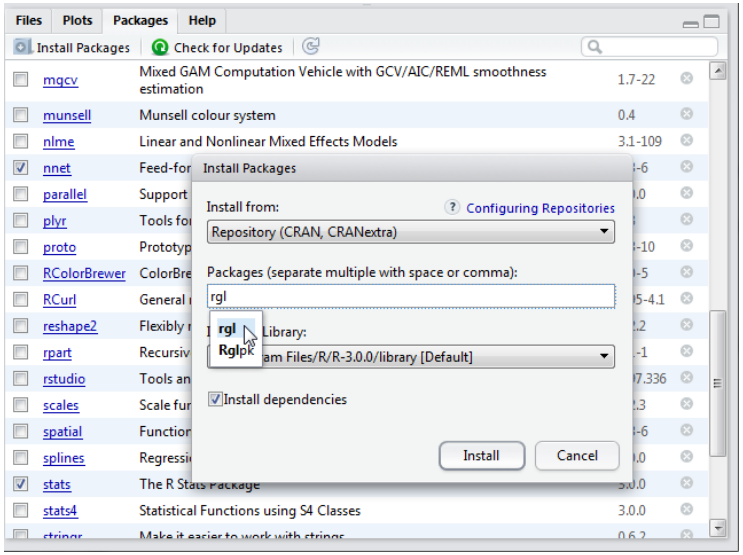
- The **package tab** shows the list of add-ons included in the installation of RStudio. If checked, the package is loaded into R, if not, any command related to that package won't work, you will need select it. You can also install other add-ons by clicking on the 'Install Packages' icon.
- Another way to activate a package is by typing, for example, `library(foreign)`. This will automatically check the `--foreign` package (it helps bring data from proprietary formats like Stata, SAS or SPSS).



Installing a package

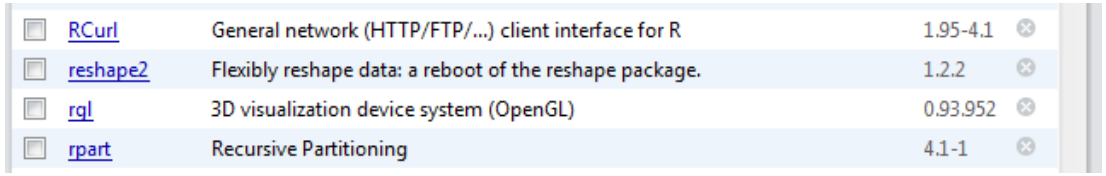
Before





- We are going to install the package `rgl` (useful to plot 3D images). It does not come with the original R install.
- Click on "`Install Packages`", write the name in the pop-up window and click on "`Install`".

After



R Basics

How it works

100xp

In the editor on the right you should type R code to solve the exercises. When you hit the 'Submit Answer' button, every line of code is interpreted and executed by R and you get a message whether or not your code was correct. The output of your R code is shown in the console in the lower right corner.

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

You can also execute R commands straight in the console. This is a good way to experiment with R code, as your submission is not checked for correctness.

Instructions

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

script.R

```
1  # Calculate 3 + 4
2
3
4  # Calculate 6 + 12
5
```

Little arithmetics with R

100xp

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

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

script.R

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

Variable assignment

100xp

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_variable` with the command

```
my_variable <- 4
```

Instructions

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.

script.R

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

Variable assignment (2)

100xp

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

Instructions

- Type the following code in the editor: `my_apples <- 5` . This will assign the value 5 to `my_apples` .
- Type: `my_apples` below the second comment. This will print out the value of `my_apples` .
- Click 'Submit Answer', and look at the console: you see that the number 5 is printed. So R now links the variable `my_apples` to the value 5.

script.R

```
1  # Assign the value 5 to the variable called 'my_apples'
2
3
4  # Print out the value of the variable 'my_apples'
5
```

Variable assignment (3)

100xp

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

```
my_apples + my_oranges
```

Instructions

- Assign to `my_oranges` the value 6.
- Add the variables `my_apples` and `my_oranges` and have R simply print the result.
- Combine the variables `my_apples` and `my_oranges` into a new variable `my_fruit`, which is the total amount of fruits in your fruit basket.

script.R

```
# Assign a value to the variables called 'my_apples' and 'my_oranges'


# Add these two variables together and print the result

# Create the variable 'my_fruit'

```

Apples and oranges

100xp

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

Instructions

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

script.R

```
1  # Assign a value to the variable called 'my_apples'
2
3
4  # Print out the value of 'my_apples'
5
6
7  # Assign a value to the variable 'my_oranges' and print it out
8
9
10
11 # New variable that contains the total amount of fruit
12
13
```

Basic data types in R

100xp

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

- Decimals 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** (`TRUE` can be abbreviated to `T` and `FALSE` to `F`).
- Text (or string) values are called **characters**.

Note how the quotation marks on the right indicate that "some text" is a character.

Instructions

Change the value of the:

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

Note that R is case sensitive!

script.R

```
# What is the answer to the universe?


# The quotation marks indicate that the variable is of type character


```

What's that data type?

100xp

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 as follows:

```
class(some_variable_name)
```

Instructions

Complete the code in the editor and print the classes of `my_numeric`, `my_character` and `my_logical` to the console.

script.R

```
# Declare variables of different types



# Check which type these variables have:


```

Vectors

Create a vector

100xp

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

- Do you still remember what you have learned in the first chapter? Assign the value `"Here we go!"` to the variable `Vegas` !

script.R

```
1 # Define the variable 'Vegas'
2
3
```

Create a vector (2)

100xp

Let us focus first!

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

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

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

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).

script.R

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

Create a vector (3)

100xp

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

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

For `poker_vector` :

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

For `roulette_vector` :

- On Monday you lost 24$
- Tuesday you lost 50$
- Wednesday you won 100$
- Thursday you lost 350$
- Friday you won 10$

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

Instructions

Assign the winnings/losses for roulette to the variable `roulette_vector` .

script.R

```
# Poker winnings from Monday to Friday


# Roulette winnings from Monday to Friday

```

R Console

Naming a vector

100xp

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("Johnny", "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:

```
some_vector
```

Instructions

Go ahead and assign the days of the week as names to `poker_vector` and `roulette_vector`. In case you are not sure, the days of the week are: Monday, Tuesday, Wednesday, Thursday and Friday.

script.R

```
1  # Poker winnings from Monday to Friday
2
3
4  # Roulette winnings from Monday to Friday
5
6
7  # Add your code here
8
9
```

Calculating total winnings

100xp

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

You want to find out the following type of information:

- How much has been your overall profit or loss per day of the week?
- Have you lost money over the week in total?
- Are you winning/losing money on poker or on roulette?

To get the answers, you have to do arithmetic calculations on vectors.

It is important to know is 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)
```

Let us try this first!

Instructions

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

script.R

```
1  A_vector <- c(1, 2, 3)
2  B_vector <- c(4, 5, 6)
3
4  # Take the sum of 'A_vector' and 'B_vector'
5
6
7  # Print 'total_vector' to the console
8
```

Naming a vector (2)

100xp

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

- Create a variable `days_vector` that contains the days of the week, from Monday to Friday.
- Use that variable `days_vector` to set the names of `poker_vector` and `roulette_vector`.

script.R

```
# Poker winnings from Monday to Friday


# Roulette winnings from Monday to Friday


# Create the variable 'days_vector'


#Assign the names of the day to 'roulette_vector' and 'poker_vector'


```

Calculating total winnings

100xp

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

You want to find out the following type of information:

- How much has been your overall profit or loss per day of the week?
- Have you lost money over the week in total?
- Are you winning/losing money on poker or on roulette?

To get the answers, you have to do arithmetic calculations on vectors.

It is important to know is 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)
```

Let us try this first!

Instructions

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

script.R

```
1  A_vector <- c(1, 2, 3)
2  B_vector <- c(4, 5, 6)
3
4  # Take the sum of 'A_vector' and 'B_vector'
5
6
7  # Print 'total_vector' to the console
8
```

Calculating total winnings (2)

100xp

Do you understand how R does arithmetic calculations with vectors?

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

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

Instructions

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

script.R

```
# Poker winnings from Monday to Friday:


# Roulette winnings from Monday to Friday:


# Give names to both 'poker_vector' and 'roulette_vector'


# Up to you now:
```

Calculating total winnings (3)

100xp

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 very 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

- 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).

script.R

```
1  # Poker winnings from Monday to Friday
2
3
4  # Roulette winnings from Monday to Friday
5
6
7  # Give names to both 'poker_vector' and 'roulette_vector'
8
9
10
11
12 # Total winnings with poker
13
14
15
16
17
18
```

Comparing total winnings

100xp

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

- Calculate `total_poker` and `total_roulette` as in the previous exercise.
- Check if your total gains in poker are higher than for roulette by using a comparison. Assign the result of this comparison to the variable `answer`. What do you conclude, should you focus on roulette or on poker?

script.R

```
1  # Poker winnings from Monday to Friday
2
3
4  # Roulette winnings from Monday to Friday
5
6
7  # Give names to both 'poker_vector' and 'roulette_vector'
8
9
10
11
12 # Calculate total gains for poker and roulette
13
14
15
16 # Check if you realized higher total gains in poker than in roulette
17
```

Vector selection: the good times

100xp

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.

Instructions

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

script.R

```
1  # Poker winnings from Monday to Friday
2
3
4  # Roulette winnings from Monday to Friday
5
6
7  # Give names to both 'poker_vector' and 'roulette_vector'
8
9
10
11
12 # Define a new variable based on a selection
13
```

Vector selection: the good times (2)

100xp

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

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

script.R

```
1  # Poker winnings from Monday to Friday
2
3
4  # Roulette winnings from Monday to Friday
5
6
7  # Give names to both 'poker_vector' and 'roulette_vector'
8
9
10
11
12 # Define a new variable based on a selection
13
```

Vector selection: the good times (3)

100xp

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

Assign the results to `roulette_selection_vector` the results from Tuesday up to Friday by making use of `:`.

script.R

```
1  # Poker winnings from Monday to Friday
2
3
4  # Roulette winnings from Monday to Friday
5
6
7  # Give names to both 'poker_vector' and 'roulette_vector'
8
9
10
11
12 # Define a new variable based on a selection
13
```

Vector selection: the good times (4)

100xp

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")]
```

script.R

```
1  # Poker winnings from Monday to Friday
2
3
4  # Roulette winnings from Monday to Friday
5
6
7  # Give names to both 'poker_vector' and 'roulette_vector'
8
9
10
11
12                                                                )
```

Selection by comparison - Step 1

100xp

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, the statement `c(4,5,6) > 5` returns: `FALSE FALSE TRUE`. In other words, you test for every element of the vector if the condition stated by the comparison operator is `TRUE` or `FALSE`. Do not take our word for it! Try it in the console ;-).

Behind the scenes, R *recycles* the value `5` when you execute `c(4,5,6) > 5`. R wants to do an element-wise comparison of each element in `c(4,5,6)` with each corresponding element in `5`. However, `5` is not a vector of length three. To solve this, R automatically replicates the value 5 to generate a vector of three elements, `c(5, 5, 5)` and then carries out the element-wise comparison.

script.R

```
1  # Poker winnings from Monday to Friday
2
3
4  # Roulette winnings from Monday to Friday
5
6
7  # Give names to both 'poker_vector' and 'roulette_vector'
8
9
10
11
12 # What days of the week did you make money on poker?
13
```

Selection by comparison - Step 2

100xp

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`. This works, because R only selects those elements where `selection_vector` is `TRUE` by default. For `selection_vector` this means where `poker_vector > 0`.

Instructions

Assign the amounts that you won on the profitable days to the variable `poker_winning_days`.

script.R

```
# Poker winnings from Monday to Friday


# Roulette winnings from Monday to Friday


# Give names to both 'poker_vector' and 'roulette_vector'




# What days of the week did you make money on poker?


# Select from poker_vector these days

```

Advanced selection

100xp

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

Instructions

- Assign the amounts that you made on the days that you ended positively for roulette to the variable `roulette_winning_days` . This vector thus contains the positive winnings of `roulette_vector` .

script.R

```
1  # Poker winnings from Monday to Friday
2
3
4  # Roulette winnings from Monday to Friday
5
6
7  # Give names to both 'poker_vector' and 'roulette_vector'
8
9
10
11
12 # What days of the week did you make money on roulette?
13
14
15 # Select from roulette_vector these days
16
```

Matrices

What's a matrix?

100xp

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 constructs the vector `c(1, 2, 3, 4, 5, 6, 7, 8, 9)` .
- The argument `byrow` indicates that the matrix is filled by the rows. If we want the matrix to be filled by the columns, we just place `byrow = FALSE` .
- The third argument `nrow` indicates that the matrix should have three rows.

Instructions

Construct a matrix with 3 rows containing the numbers 1 up to 9. Click the 'Submit Answer' button and look at the output in the console.

script.R

```
1 # Construction of a matrix with 3 rows that contain the numbers 1 up to 9
2
3
```

Analyzing matrices, you shall

100xp

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).

Instructions

Construct a matrix with one row for each movie (thus 3 rows). The first column is for the US box office revenue, and the second column for the non-US box office revenue. Name the matrix `star_wars_matrix` .

script.R

```
# Box office Star Wars: In Millions!
# The first element: US, the second element: Non-US
new_hope <- c(460.998, 314.4)
empire_strikes <- c(290.475, 247.900)
return_jedi <- c(309.306, 165.8)

# Add your code below to Construct matrix

```

Naming a matrix

100xp

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 later.

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
```

Instructions

- Give the columns of `star_wars_matrix` the names `"US"` and `"non-US"`, respectively.
- Give the rows of the matrix `star_wars_matrix` the names of the three movies. In case you are not a fan ;-), the movie names are: "A New Hope", "The Empire Strikes Back" and "Return of the Jedi".

script.R

```
1  # Box office Star Wars: In Millions (!)
2  # First element: US, Second element: non-US
3  new_hope <- c(460.998, 314.4)
4  empire_strikes <- c(290.475, 247.900)
5  return_jedi <- c(309.306, 165.8)
6
7  # Construct matrix
8
9
10 # Add your code here such that rows and columns of star_wars_matrix have a name!
11
12
```

Calculating the worldwide box office

100xp

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:

```
sum_of_rows_vector <- rowSums(my_matrix)
```

Instructions

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

script.R

```
# Box office Star Wars: In Millions (!)
# Construct matrix
box_office_all <- c(461, 314.4, 290.5, 247.9, 309.3, 165.8)
movie_names <- c("A New Hope","The Empire Strikes Back","Return of the Jedi")
col_titles <- c("US","non-US")
star_wars_matrix <- matrix(box_office_all, nrow=3, byrow = TRUE,
                           dimnames = list(movie_names, col_titles))

# Your code here

```

Adding a column for the Worldwide box office

100xp

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

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

script.R

```
1  # Box office Star Wars: In Millions (!)
2  # Construct matrix
3  box_office_all <- c(461, 314.4, 290.5, 247.9, 309.3, 165.8)
4  movie_names <- c("A New Hope","The Empire Strikes Back","Return of the Jedi")
5  col_titles <- c("US","non-US")
6  star_wars_matrix <- matrix(box_office_all, nrow = 3, byrow = TRUE,
7                             dimnames = list(movie_names, col_titles))
8
9  # The worldwide box office figures
10
11
12 # Bind the new variable worldwide_vector as a column to star_wars_matrix
13
```

Adding a row

100xp

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

Your R workspace (check out what a workspace is) has been initialized to and contains two matrices: the `star_wars_matrix` that we have just used for the first trilogy but also the `star_wars_matrix2` for the second trilogy. Type the name of the matrices in the console and press enter in case you want to have a closer look.

Instructions

Assign to `all_wars_matrix` a new matrix with `star_wars_matrix` in the first three rows and `star_wars_matrix2` in the next three rows.

script.R

```
# Matrix that contains the first trilogy box office
star_wars_matrix

# Matrix that contains the second trilogy box office
star_wars_matrix2

# Combine both Star Wars trilogies in one matrix

```

The total box office revenue for the entire saga

100xp

Just like every `cbind()` has a `rbind()`, every `colSums()` has a `rowSums()`. Your R workspace already contains the `all_wars_matrix` that you constructed in the previous exercise (Type `all_wars_matrix` in the console if you do not recall what it contains). Let us now calculate the total box office revenue for the entire saga.

Instructions

Calculate the total revenue for the US and the non-US region and assign `total_revenue_vector`. You can use the `colSums()` function.

script.R

```
1  # Print box office Star Wars
2
3
4  # Total revenue for US and non-US
5
```

Selection of matrix elements

100xp

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 that what to select from the rows from that what you want to select from the columns. For example:

- `my_matrix[1,2]` selects from the first row the second element.
- `my_matrix[1:3,2:4]` selects 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!

Instructions

- Calculate the average Non-US revenue for all movies. Assign this average to the `non_us_all` variable. In other words, take the average of all elements of the second column.
- Same question, but now only for the first two Star Wars movies. Assign the average to `non_us_some` .

script.R

```
1  # Box office Star Wars: In Millions (!)
2  # Construct matrix
3  box_office_all <- c(461, 314.4, 290.5, 247.9, 309.3, 165.8)
4  movie_names <- c("A New Hope","The Empire Strikes Back","Return of the Jedi")
5  col_titles <- c("US","non-US")
6  star_wars_matrix <- matrix(box_office_all, nrow = 3, byrow = TRUE,
7                             dimnames = list(movie_names, col_titles))
8
9  # Print the star_wars_matrix to the console
10
11
12 # Average non-US revenue per movie
13
14
15 # Average non-US revenue of first two movies
16
```

A little arithmetic with matrices

100xp

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. Again, R recycles the value `2` behind the scenes, building a matrix containing only `2` s with the same dimensions as `my_matrix`. Then, R carries out the element-wise operation.

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

Instructions

- Assign the matrix with the estimated number of Non-US and US visitors for the three movies to `visitors`.
- Print the resulting variable to the console.

script.R

```
1  # Box office Star Wars: In Millions (!)
2  # Construct matrix
3  box_office_all <- c(461, 314.4, 290.5, 247.9, 309.3, 165.8)
4  movie_names <- c("A New Hope","The Empire Strikes Back","Return of the Jedi")
5  col_titles <- c("US","non-US")
6  star_wars_matrix <- matrix(box_office_all, nrow = 3, byrow = TRUE, dimnames = list
   (movie_names, col_titles))
7
8  # Estimation of visitors
9
10
11 # Print the estimate to the console
12
```

A little arithmetic with matrices (2)

100xp

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 with one dollar per movie. 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

- Assign to `visitors` the matrix with your estimated number of Non-US and US visitors for the three movies.
- Assign to `average_us_visitors` the average number of visitors in the US for a Star Wars movie.
- Assign to `average_non_us_visitors` the average number of visitors in non-US areas.

script.R

```
1  # Box office Star Wars: In Millions (!)
2  # Construct matrix
3  box_office_all <- c(461, 314.4, 290.5, 247.9, 309.3, 165.8)
4  movie_names <- c("A New Hope","The Empire Strikes Back","Return of the Jedi")
5  col_titles <- c("US","non-US")
6  star_wars_matrix <- matrix(box_office_all, nrow = 3, byrow = TRUE,
7                             dimnames = list(movie_names, col_titles))
8  ticket_prices_matrix <- matrix(c(5, 5, 6, 6, 7, 7), nrow = 3, byrow = TRUE,
9                                 dimnames = list(movie_names, col_titles))
10
11 # Estimated number of visitors
12
13
14 # Average number of US visitors
15
16
17 # Average number of non-US visitors
18
```