lab2_jupyter_r_basics

January 29, 2022

R Basics with Jupyter Notebook

Estimated time needed: **15** minutes

0.1 Objectives

After completing this lab you will be able to:

- Understand the sample dataset
- Create variables and perform basic math operations
- Perform basic strings operations

0.2 Table of Contents

About the Dataset

Which movie should you watch next?

Let's say each of your friends tells you their favorite movies. You do some research on the movies and put it all into a table. Now you can begin exploring the dataset, and asking questions about the movies. For example, you can check if movies from some certain genres tend to get better ratings. You can check how the production cost for movies changes across years, and much more.

Movies dataset

The table gathered includes one row for each movie, with several columns for each movie characteristic:

- **name** - Name of the movie
- **year** - Year the movie was released
- **length_min** - Length of the movie (minutes)
- **genre** - Genre of the movie
- **average_rating** - Average rating on IMDB
- **cost_millions** - Movie's production cost (millions in USD)
- **foreign** - Is the movie foreign (1) or domestic (0)?
- **age_restriction** - Age restriction for the movie

0.2.1 We can use R to help us explore the dataset

But to begin, we'll need to start from the basics, so let's get started!

Simple Math in R

Let's say you want to watch *Fight Club* and *Star Wars: Episode IV (1977)*, back-to-back. Do you have enough time to **watch both movies in 4 hours?** Let's try using simple math in R.

What is the **total movie length** for Fight Club and Star Wars (1977)?

- **Fight Club**: 139 min
- **Star Wars: Episode IV**: 121 min

Tip: To run the grey code cell below, click on it, and press Shift + Enter.

[ ]:
```
139 + 121
```

Great! You've determined that the total number of movie play time is **260 min**.

What is 260 min in hours?

[ ]:
```
260 / 60
```

Well, it looks like it's **over 4 hours**, which means you can't watch *Fight Club* and *Star Wars (1977)* back-to-back if you only have 4 hours available!

[Tip] Simple math in R

You can do a variety of mathematical operations in R including:

addition: $2 + 2$

subtraction: $5 - 2$

multiplication: $3 * 2$

division: $4/2$

exponentiation: $4 * *2$ or 4^2

Variables in R

We can also **store** our output in **variables**, so we can use them later on. For example:

[ ]:
```
x <- 139 + 121
```

To return the value of **x**, we can simply run the variable as a command:

[ ]:
```
x
```

You can check its variable type using `class()` function

[ ]:
```
class(x)
```

And cast the type of `x` to character

[ ]:
```
x_char <- as.character(x)
class(x_char)
```

And cast it back to numeric

[ ]:
```
x_num <- as.numeric(x_char)
class(x_num)
```

We can also perform operations on **x** and save the result to a **new variable**:

[ ]:
```
y <- x / 60
y
```

If we save something to an **existing variable**, it will **overwrite** the previous value:

[ ]:
```
x <- x / 60
x
```

It's good practice to use **meaningful variable names**, so you don't have to keep track of what variable is what:

[ ]:
```
total <- 139 + 121
total
```

[ ]:
```
total_hr <- total / 60
total_hr
```

You can put this all into a single expression, but remember to use **round brackets** to add together the movie lengths first, before dividing by 60.

[ ]:
```
total_hr <- (139 + 121) / 60
total_hr
```

[Tip] Variables in R

As you just learned, you can use variables to store values for repeated use. Here are some more characteristics of variables in R:

variables store the output of a block of code

variables are typically assigned using $<-$, but can also be assigned using $=$, as in x $<-$ 1 or $x = 1$

once created, variables can be removed from memory using `rm(my_variable)`

Coding Exercise: in the code cell below, calculate how much longer is 139 minutes comparing to 121 minutes, in seconds

[ ]:
```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
```

Click here for the solution

```
(139 - 121) * 60
```

Strings in R

R isn't just about numbers – we can also have strings too. For example:

[ ]:
```
movie <- "Toy Story"
movie
```

In R, you can identify **character strings** when they are wrapped with **matching double (“) or single (’) quotes**.

You can also check its class using `class()` function

[ ]:
```
class(movie)
```

If you try to cast it into numeric, R will give you an error because 'Toy Story' is not number

[ ]:
```
as.numeric(movie)
```

Scaling R with big data As you learn more about R, if you are interested in exploring platforms that can help you run analyses at scale, you might want to sign up for a free account on IBM Watson Studio, which allows you to run analyses in R with two Spark executors for free.

0.2.2 Excellent! You have just completed the R basics notebook!

0.3 Authors

Hi! It's Marta Aghili, the author of this notebook. I hope you found R easy to learn! There's lots more to learn about R but you're well on your way. Feel free to connect with me if you have any questions.

0.3.1 Other Contributors

Yan Luo

0.4 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-03-02	2.0	Yan	Added coding tasks

##