Strings and Regular Expressions

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

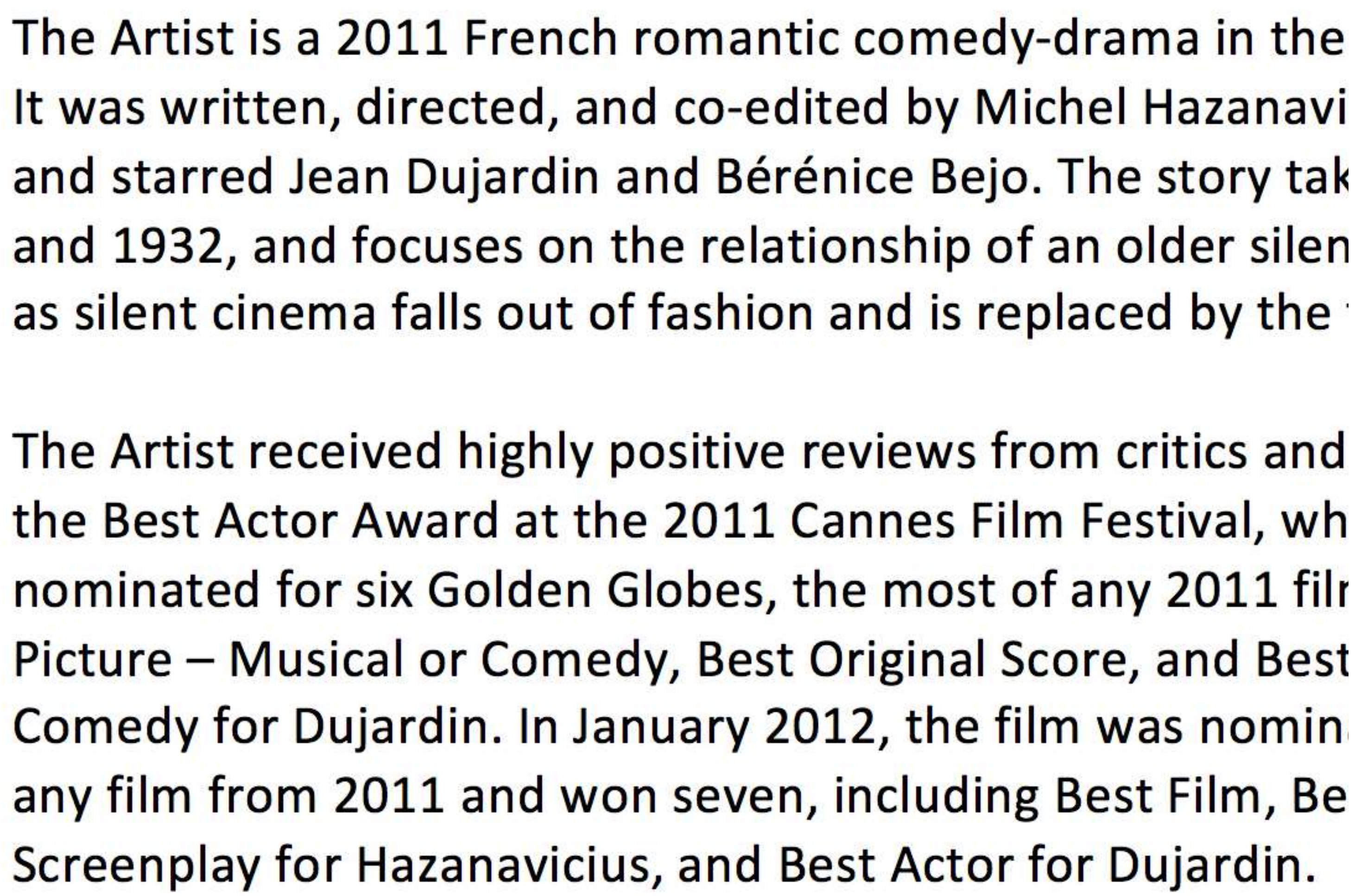
- Understand and perform string operations in R
- Understand and use regular expressions to find patterns in strings

You will be working on this lab in cloud-based RStudio hosted by IBM Skills Network Labs.

Note that your lab will be reset after about 1 hour inactive window. It's recommended to backup the files you created.

About the Dataset

In this lab, we are going to use `The_Artist.txt` file.
This file contains text data which is basically summary of
the `The Artist` movie and we are going to perform various operations on this data.



The Artist is a 2011 French romantic comedy-drama in the
It was written, directed, and co-edited by Michel Hazanavi
and starred Jean Dujardin and Bérénice Bejo. The story tak
and 1932, and focuses on the relationship of an older silen
as silent cinema falls out of fashion and is replaced by the

The Artist received highly positive reviews from critics and
the Best Actor Award at the 2011 Cannes Film Festival, wh
nominated for six Golden Globes, the most of any 2011 fil
Picture – Musical or Comedy, Best Original Score, and Best
Comedy for Dujardin. In January 2012, the film was nomin
any film from 2011 and won seven, including Best Film, Be
Screenplay for Hazanavicius, and Best Actor for Dujardin.

First, let's identify your current working directory

- In the RStudio Console, run the following code snippet:

```
getwd()
```

Copied!

then you should see the following result in console:

```
[1] "/resources/rstudio"
```

Copied!

In the Files panel on the right, if you are not in `/resources/rstudio`,
click `resources` folder and you should find a `rstudio` folder. This will be your
current working directory in RStudio.

- Now let's download the text file by copying the following code in the console, press the `Enter` key to run the code:

```
::page{title="code to download the dataset"}

download.file("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0101EN-Coursera/v2/dataset/The_Artist.t
```

Copied!

Reading Text Files

To read text files in R, we can use the built-in R function **readLines()**.
This function takes **file path** as the argument and read the whole file.

Let's read the **The_Artist.txt** file and see how it looks like.

- Copy and run following code snippet in the console

```
my_data <- readLines("The_Artist.txt")
my_data
```

Copied!

and you should see the text content shown in the console:

```
[1] "The Artist is a 2011 French romantic comedy-drama in the style of a black-and-white silent film. It was written, directed, and co-edited by Mich
[2] ""
[3] "The Artist received highly positive reviews from critics and won many accolades. Dujardin won the Best Actor Award at the 2011 Cannes Film Festi
[4] ""
[5] ""
```

Copied!

If you got an error message here, make sure that you run the code cell above first to download the dataset

So, we get a character vector which has several elements and these elements can be accessed as we access array.

Let's check the length of **my_data** variable:

```
length(my_data)
```

Copied!

Length of **my_data** variable is **5** which means it contains 5 elements.

Similarly, we can check the size of the file by using the **file.size()** method of R
and it takes **file path** as argument and returns the number of bytes.

- By executing code snippet below, we will get **1065** at the output, which is the size of the file **in bytes**.

```
file.size("The_Artist.txt")
```

Copied!

There is another method **scan()** which can be used to read **.txt** files.

The Difference in **readLines()** and **scan()** method is that, **readLines()**
is used to read text files line by line whereas **scan()** method read the text files word by word.

scan() method takes two arguments. First is the **file path** and second argument is
the string expression according to which we want to separate the words.

Like in example below, we pass an empty string as the separator argument.

```
my_data1 <- scan("The_Artist.txt", "")
my_data1
```

Copied!

If we will check length of **my_data1** variable then we will get total number of elements at the output.

```
length(my_data1)
```

Copied!

Coding Exercise: Get the frequency for each word in “The_Artist.txt” file. (HINT: convert my_data1 to factor and use summary() function)

▶ Click here to see solution

String Operations

There are many string operation methods in R which can be used to manipulate the data.
We are going to use some basic string operations on the data that we read before.

nchar()

The first function is **nchar()** which will return the total number of characters in the given string.
Let’s find out how many characters are there in the first element of **my_data** variable.

```
nchar(my_data[1])
```

Copied!

toupper()

Now, sometimes we need the whole string to be in Upper Case. To do so, there is a function called `toupper()` in R which takes a string as input and provides the whole string in upper case at output.

```
toupper(my_data[3])
```

Copied!

In above code block, we convert the third element of the character vector in upper case.

tolower()

Similarly, `tolower()` method can be used to convert whole string into lower case.
Let’s convert the same string that we convert in upper case, into lower case.

```
tolower(my_data[3])
```

Copied!

We can clearly see the difference between the outputs of last two methods.

chartr()

what if we want to replace any characters in given string?

This operation can also be performed in R using `chartr()` method which takes three arguments.
The first argument is the characters which we want to replace in string,
second argument is the new characters and the last argument is the string on which operation will be performed.

Let’s replace white spaces in the string with the hyphen (-) sign in the first element of the `my_data` variable.

```
chartr(" ", "-", my_data[1])
```

Copied!

strsplit()

Previously, we learned that we can read file word by word using `scan()` function.
But what if we want to split the given string word by word?

This can be done using `strsplit()` method. Let's split the string according to the white spaces.

```
character_list <- strsplit(my_data[1], " ")
word_list <- unlist(character_list)
word_list
```

Copied!

In above code block, we separate the string word by word, but `strsplit()` method provides a list at the output which contains all the separated words as single element which is more complex to read.
Thus, to make it more easy to read each word as single element, we used `unlist()` method which converts the list into character vector and now we can easily access each word as a single element.

Coding Exercise: Replace the space characters in the 3rd element of my_data vector with underscore _, then split the string by _

▶ Click here to see solution

sort()

Sorting is also possible in R.
Let's use `sort()` method to sort elements of the `word_list` character vector in ascending order.

```
sorted_list <- sort(word_list)
sorted_list
```

Copied!

paste()

Now, we sort all the elements of `word_list` character vector. Let's use `paste()` function here, which is used to concatenate strings. This method takes two arguments, the strings we want to concatenate and collapse argument which defines the separator in the words.

Here, we are going to concatenate all words of `sorted_list` character vector into a single string.

```
paste(sorted_list, collapse = " ")
```

Copied!

substr()

There is another function `substr()` in R which is used to get a sub section of the string.

Let's take an example to understand it more. In example below, we use the `substr()` method and provide it three arguments. First argument is the data string from which we want the sub string.
Second argument is the starting point from where function will start reading the string and the third argument is the stopping point till where we want the function to read string.

```
sub_string <- substr(my_data[1], start = 4, stop = 50)
sub_string
```

Copied!

So, from the character vector, we start reading the first element from 4th position and read the string till 50th position and at the output, we get the resulted string which we stored in `sub_string` variable.

trimws()

As the sub string that we get in code block above, have some white spaces at the initial and end points.
So, to quickly remove them, we can use `trimws()` method of R like shown below.

```
trimws(sub_string)
```

Copied!

So, at the output, we get the string which does not contain any white spaces at the both ends.

str_sub()

To read string from last, we are using `stringr` library.
This library contains `str_sub()` method, which takes same arguments as `sub_stirng` method but read string from last.

Like in the example below, we provide a data string and both starting and end points with negative values which indicates that we are reading string from last.

```
library(stringr)
str_sub(my_data[1], -8, -1)
```

Copied!

So, we read string from -1 till -8 and it gives **talkies.** with full stop mark at the output.

Regular Expressions

Regular Expressions are generic expressions that are used to match patterns in strings and text.
A way we can exemplify this is with a simple expression that can match with an email string.
But before we write it, let's look at how an email is structured:

`test@testing.com`

So, an email is composed by a string followed by an `@` symbol followed by another string. In R regular expressions, we can express this as:

`$.+@.+$`

Where:

- The `.` symbol matches with any character.
- The `+` symbol repeats the previous symbol one or more times. So, `.+` will match with any string.
- The `@` symbol only matches with the `@` character.

Now, for our problem, which is extracting the domain from an email excluding the regional url code, we need an expression that specifically matches with what we want:

`@.+\\.`

Where the `\\.` symbol specifically matches with the '.' character.

Now let's look at some R functions that work with regular expressions.

The `grep` function below takes in a regular expression and a list of strings to search through and returns the positions of where they appear in the list.

- Before we start, let's load a sample email csv file

```
::page{title="Load a email dataframe"}

email_df <- read.csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0101EN-Coursera/v2/dataset/email
email_df
```

Copied!

- Then search the email list to find which one matches an email pattern

```
::page{title="Load a email dataframe"}

grep("@.+",  c("test@testing.com" , "not an email", "test2@testing.com"))
```

Copied!

- `grep` also has an extra parameter called 'value' that changes the output to display the strings instead of the list positions.

```
::page{title="Load a email dataframe"}

grep("@.+",  c("test@testing.com", "not an email", "test2@testing.com"), value=TRUE)
```

Copied!

Coding Exercise: Use a regular expression to check if a string is an email address ending with `.com`

▶ Click here to see solution

The next function, `gsub`, is a substitution function.
It takes in a regular expression, the string you want to swap in with the matches and a list of
strings you want to perform the swap with. The code cell below updates valid emails with a new domain:

```
gsub("@.+", "@newdomain.com", c("test@testing.com", "not an email", "test2@testing.com"))
```

Copied!

The functions below, `regexpr` and `regmatches`, work in conjunction to extract the matches found by a
regular expression specified in `regexpr`.

```
matches <- regexpr("@.*", c("test@testing.com", "not an email", "test2@testing.com"))
regmatches(c("test@testing.com", "not an email", "test2@testing.com"), matches)
```

Copied!

This function is actually perfect for our problem since we simply need to extract the specific information we want.
So let's use it with the regular expression we defined above and store the extracted strings in a new column in our data frame.

```
matches <- regexpr("@.*\\.", email_df[,'Email'])
email_df[,'Domain'] = regmatches(email_df[,'Email'], matches)
```

Copied!

And this is the resulting dataframe:

```
email_df
```

Copied!

Now we can finally construct the frequency table for the domains in our dataframe!

Authors:

Hi! It's Iqbal Singh, 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.

Other Contributor(s)

Yan Luo

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-05-11	2.4	Eric Hao & Vladislav Boyko	Updated Page Frames
2023-05-10	2.3	Eric Hao & Vladislav Boyko	Updated Page Frames
2023-05-10	2.2	Eric Hao & Vladislav Boyko	Updated Page Frames
2023-05-05	2.1	Benny	Reformatted and Re-Published
2021-03-08	2.0	Yan	Added coding tasks