Lab-Importing_Data

May 22, 2021

IMPORTING DATA in R

0.0.1 Welcome!

By the end of this notebook, you will have learned how to **import and read data** from different file types in R.

0.1 Table of Contents

About the Dataset

Movies dataset

Here we have a dataset that 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

Let's learn how to **import and read data** from two common types of files used to store tabular data (when data is stored in a table or a spreadsheet.)

- **CSV files** (.csv)
- **Excel files** (.xls or .xlsx)

To begin, we'll need to **download the data**!

Download the Data

We've made it easy for you to get the data, which we've hosted online. Simply run the code cell below (Shift + Enter) to download the data to your current folder.

[ ]:
```
# Download datasets

# CSV file
download.file("https://ibm.box.com/shared/static/
 ↪n5ay5qadfe7e1nnsv5s01oe1x62mq51j.csv",
              destfile="movies-db.csv")

# XLS file
download.file("https://ibm.box.com/shared/static/
 ↪nx0ohd9sq0iz3p871zg8ehc1m39ibpx6.xls",
              destfile="movies-db.xls")
```

If you ran the cell above, you have now downloaded the following files to your current folder:

movies-db.csv
movies-db.xls

Reading CSV Files

What are CSV files? Let's read data from a CSV file. CSV (Comma Separated Values) is one of the most common formats of structured data you will find. These files contain data in a table format, where in each row, columns are separated by a delimiter – traditionally, a comma (hence comma-separated values).

Usually, the first line in a CSV file contains the column names for the table itself. CSV files are popular because you do not need a particular program to open it.

Reading CSV files in R In the `movies-db.csv` file, the first line of text is the header (names of each of the columns), followed by rows of movie information.

To read CSV files into R, we use the core function `read.csv`.

`read.csv` easy to use. All you need is the filepath to the CSV file. Let's try loading the file using the filepath to the `movies-db.csv` file we downloaded earlier:

[ ]:
```
# Load the CSV table into the my_data variable.
my_data <- read.csv("movies-db.csv")
my_data
```

The data was loaded into the `my_data` variable. But instead of viewing all the data at once, we can use the `head` function to take a look at only the top six rows of our table, like so:

[ ]:
```
# Print out the first six rows of my_data
head(my_data)
```

Additionally, you may want to take a look at the **structure** of your newly created table. R provides us with a function that summarizes an entire table's properties, called `str`. Let's try it out.

[ ]:
```
# Prints out the structure of your table.
str(my_data)
```

When we loaded the file with the `read.csv` function, we had to only pass it one parameter – the **path** to our desired file.

If you're using Data Scientist Workbench, it is simple to find the path to your uploaded file. In the **Recent Data** section in the sidebar on the right, you can click the arrow to the left of the filename to see extra options – one of these commands should be **Insert Path**, which automatically copies the path to your file into Jupyter Notebooks.

Reading Excel Files

Reading XLS (Excel Spreadsheet) files is similar to reading CSV files, but there's one catch – R does not have a native function to read them. However, thankfully, R has an extremely large repository of user-created functions, called *CRAN*. From there, we can download a library package to make us able to read XLS files.

To download a package, we use the `install.packages` function. Once installed, you do not need to install that same library ever again, unless, of course, you uninstall it.

[ ]:
```
# Download and install the "readxl" library
install.packages("readxl")
```

Whenever you are going to use a library that is not native to R, you have to load it into the R environment after you install it. In other words, you need to install once only, but to use it, you must load it into R for every new session. To do so, use the `library` function, which loads up everything we can use in that library into R.

[ ]:
```
# Load the "readxl" library into the R environment.
library(readxl)
```

Now that we have our library and its functions ready, we can move on to actually reading the file. In `readxl`, there is a function called `read_excel`, which does all the work for us. You can use it like this:

[ ]:
```
# Read data from the XLS file and attribute the table to the my_excel_data␣
↪variable.
my_excel_data <- read_excel("movies-db.xls")
```

Since `my_excel_data` is now a dataframe in R, much like the one we created out of the CSV file, all of the native R functions can be applied to it, like `head` and `str`.

[ ]:
```
# Prints out the structure of your table.
# Tells you how many rows and columns there are, and the names and type of each␣
↪column.
```

```
# This should be the very same as the other table we created, as they are the␣
 ↪same dataset.
str(my_excel_data)
```

Much like the `read.csv` function, `read_excel` takes as its main parameter the **path** to the desired file.

[Tip]
A **Library** is basically a collection of different classes and functions which are used to perform some specific operations. You can install and use libraries to add more functions that are not included on the core R files. For example, the **readxl** library adds functions to read data from excel files. It's important to know that there are many other libraries too which can be used for a variety of things. There are also plenty of other libraries to read Excel files – readxl is just one of them.

Accessing Rows and Columns

Whenever we use functions to read tabular data in R, the default method of structuring this data in the R environment is using Data Frames – R's primary data structure. Data Frames are extremely versatile, and R presents us many options to manipulate them.

Suppose we want to access the "name" column of our dataset. We can directly reference the column name on our data frame to retrieve this data, like this:

[ ]:
```
# Retrieve a subset of the data frame consisting of the "name" columns
my_data['name']
```

Another way to do this is by using the `$` notation which at the output will provide a vector:

[ ]:
```
# Retrieve the data for the "name" column in the data frame.
my_data$name
```

You can also do the same thing using **double square brackets**, to get a vector of `names` column.

[ ]:
```
my_data[["name"]]
```

Similarly, any particular row of the dataset can also be accessed. For example, to get the first row of the dataset with all column values, we can use:

[ ]:
```
# Retrieve the first row of the data frame.
my_data[1,]
```

The first value before the comma represents the **row** of the dataset and the second value (which is blank in the above example) represents the **column** of the dataset to be retrieved. By setting the first number as 1 we say we want data from row 1. By leaving the column blank we say we want all the columns in that row.

We can specify more than one column or row by using `c`, the **concatenate** function. By using `c` to concatenate a list of elements, we tell R that we want these observations out of the data frame. Let's try it out.

[ ]:
```
# Retrieve the first row of the data frame, but only the "name" and␣
↪"length_min" columns.
my_data[1, c("name","length_min")]
```

Accessing Built-in Datasets in R

R provides various built-in datasets for users to utilize for different purposes. To know which datasets are available, R provides a simple function – `data` – that returns all of the present datasets' names with a small description beside them. The ones in the `datasets` package are all inbuilt.

[ ]:
```
# Displays a list of the inbuilt datasets. Opens in a new "window".
data()
```

As you can see, there are many different datasets already inbuilt in the R environment. Having to go through each of them to take a look at their structure and try to find out what they represent might be very tiring. Thankfully, R has documentation present for each inbuilt dataset. You can take a look at that by using the `help` function.

For example, if we want to know more about the `women` dataset, we can use the following function:

[ ]:
```
# Opens up the documentation for the inbuilt "women" dataset.
help(women)
```

Since the datasets listed are inbuilt, you do not need to import or load them to use them. If you reference them by their name, R already has the data frame ready.

[ ]:
```
women
```

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.

0.1.1 About the Authors:

Hi! It's Iqbal Singh and Walter Gomes, the authors of this notebook. I hope you found it easy to learn how to import data into R! Feel free to connect with us if you have any questions.