List and Dataframe

- 1 Objectives
- 2 About the Dataset
- 3 Lists
  - 3.1 Accessing items in a list
  - 3.2 Named lists
  - 3.3 Coding Exercise 1
  - 3.4 Adding, modifying, and removing items
  - 3.5 Coding Exercise 2:
  - 3.6 Concatenating lists
- 4 DataFrames
  - 4.1 Coding Exercise 3:
  - 4.2 Coding Exercise 4:
- 5 Scaling R with big data
- 6 Author(s)
  - 6.1 Other contributors
- 7 Change Log

Estimated time needed: **15** minutes

1 Objectives

After completing this lab you will be able to:

- Understand the list and how list correlated with DataFrame
- Understand DataFrame
- Operate on DataFrames

2 About the Dataset

You have received many movie recommendations from your friends and compiled all of the recommendations into a table, with information about each movie.

This table has one row for each movie and several columns.

- **name** - The name of the movie;
- **year** - The year the movie was released;
- **length_min** - The length of the movie in minutes;
- **genre** - The genre of the movie;
- **average_rating** - Average rating on Imdb;
- **cost_millions** - The movie's production cost in millions;
- **foreign** - Indicative of whether the movie is foreign (1) or domestic (0);
- **age_restriction** - The age restriction for the movie;

Here's what the data looks like:

name	year	length_min	genre	average_rating	cost_millions	foreign	age_restriction
Toy Story	1995	81	Animation	8.3	30.0	0	0
Akira	1998	125	Animation	8.1	10.4	1	14
The Breakfast Club	1985	97	Drama	7.9	1.0	0	14
The Artist	2011	100	Romance	8.0	15.0	1	12
Modern Times	1936	87	Comedy	8.6	1.5	0	10
Fight Club	1999	139	Drama	8.9	63.0	0	18
City of God	2002	130	Crime	8.7	3.3	1	18
The Untouchables	1987	119	Drama	7.9	25.0	0	14
Star Wars Episode IV	1977	121	Action	8.7	11.0	0	10
American Beauty	1999	122	Drama	8.4	15.0	0	14

3 Lists

First of all, we're gonna take a look at lists in R.

A list is a sequenced collection of different objects of R, like vectors, numbers, characters, other lists and so on. You can consider a list as a container of correlated information, well structured, and easy to read with a context.

A list accepts items of different types, but a vector (or a matrix, which is a multidimensional vector) doesn't. To create a list just type `list()` with your content inside the parenthesis and separated by commas. Let's try it!

```
movie <- list("Toy Story", 1995, c("Animation", "Adventure", "Comedy"))
```

In the code above, the variable movie contains a list of 3 objects, which are a string, a numeric value, and a vector of strings.

Easy, eh? Now let's print the content of the list. We just need to call its name.

```
movie
```

```
## [[1]]
## [1] "Toy Story"
##
## [[2]]
## [1] 1995
##
## [[3]]
## [1] "Animation" "Adventure" "Comedy"
```

3.1 Accessing items in a list

It is possible to retrieve only a part of a list using the **single square** bracket operator `[ ]`. This operator can be also used to get a single element in a specific position. Take a look at the next example:

The index number 2 returns the second element of a list, if that length of list is at least 2:

```
movie[2]
```

```
## [[1]]
## [1] 1995
```

Or you can select a part or interval of elements of a list. In our next example we are retrieving the 1st, 2nd, and 3rd elements:

```
movie[1:3]
```

```
## [[1]]
## [1] "Toy Story"
##
## [[2]]
## [1] 1995
##
## [[3]]
## [1] "Animation" "Adventure" "Comedy"
```

3.2 Named lists

It looks a little confusing, but lists can also store names for its elements. The following list is a named list:

```
movie <- list(
  name = "Toy Story",
  year = 1995,
  genre = c("Animation", "Adventure", "Comedy")
)
```

Let me explain that: the list **movie** has some named objects within it. **name**, for example, is an object of type **character**, **year** is an object of type **number**, and **genre** is a vector with objects of type **character**.

Now take a look at this list. This time, it's full of information and well organized. It's clear what each element means. You can see that the elements have different types, and that's ok because it's a list.

```
movie
```

```
## $name
## [1] "Toy Story"
##
## $year
## [1] 1995
##
## $genre
## [1] "Animation" "Adventure" "Comedy"
```

You can also get separated information from the list. You can use `listName$selectorName`. The *dollar-sign operator* `$` will give you the block of data that is related to selector Name.

Let's get the genre part of our movies list, for example.

```
movie$genre
```

```
## [1] "Animation" "Adventure" "Comedy"
```

Here is another way of selecting the genre column:

```
movie["genre"]
```

```
## $genre
## [1] "Animation" "Adventure" "Comedy"
```

3.3 Coding Exercise 1

In the code cell below, get the second element in the `genre` column

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

You can also use numerical selectors like an array. Here we are selecting elements with indexes ranged from 2 to 3.

```
movie[2:3]
```

```
## $year
## [1] 1995
##
## $genre
## [1] "Animation" "Adventure" "Comedy"
```

The function `class()` returns the type of a object. You can use that function to retrieve the type of specific elements of a list:

```
class(movie$name)
```

```
## [1] "character"
```

3.4 Adding, modifying, and removing items

Adding a new element is also very easy. The code below adds a new field named **age** and puts the numerical value 5 into it.

In this case we use the double square brackets operator, because we are directly referencing a list member (and we want to change its content).

```
movie[["age"]] <- 5
movie
```

```
## $name
## [1] "Toy Story"
##
## $year
## [1] 1995
##
## $genre
## [1] "Animation" "Adventure" "Comedy"
##
## $age
## [1] 5
```

In order to modify, you just need to refer a list member that already exists, then change its content.

```
movie[["age"]] <- 6
# Now it's 6, not 5
movie
```

```
## $name
## [1] "Toy Story"
##
## $year
## [1] 1995
##
## $genre
## [1] "Animation" "Adventure" "Comedy"
##
## $age
## [1] 6
```

And removing is also easy! You just put `NULL`, which means empty object, into it.

```
movie[["age"]] <- NULL
movie
```

```
## $name
## [1] "Toy Story"
##
## $year
## [1] 1995
##
## $genre
## [1] "Animation" "Adventure" "Comedy"
```

3.5 Coding Exercise 2:

In the code cell below, add a logical column with a name `WillWatch` with value `TRUE` or `T`.

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

3.6 Concatenating lists

Concatenation is the process of putting things together, in sequence. And yes, you can do it with lists. Just call the function `c()`. Take a look at the next example:

```
# We split our previous list in two sublists
movie_part1 <- list(name = "Toy Story")
movie_part2 <- list(year = 1995, genre = c("Animation", "Adventure", "Comedy"))

# Now we call the function c() to put everything together again
movie_concatenated <- c(movie_part1, movie_part2)

# Check it out
movie_concatenated
```

```
## $name
## [1] "Toy Story"
##
## $year
## [1] 1995
##
## $genre
## [1] "Animation" "Adventure" "Comedy"
```

Lists are really handy for organizing different types of elements in R, and also easy to use.

Additionally, lists are also important since this type of data structure is essential to create data frames, our next covered topic.

4 DataFrames

A DataFrame is a structure that is used for storing data tables. Underneath it all, a data frame is a list of vectors of identical length, exactly like a table (each vector is a column).

We can use the function `data.frame()` to create a data frame and pass vectors, which are our columns, as arguments. It is required to name the columns that will compose the data frame.

```
movies <-
  data.frame(
    name = c(
      "Toy Story",
      "Akira",
      "The Breakfast Club",
      "The Artist",
      "Modern Times",
      "Fight Club",
      "City of God",
      "The Untouchables"
    ),
    year = c(1995, 1998, 1985, 2011, 1936, 1999, 2002, 1987),
    stringsAsFactors = F
  )
```

Let's print its content of our recently created data frame:

```
movies
```

```
##                 name year
## 1          Toy Story 1995
## 2              Akira 1998
## 3 The Breakfast Club 1985
## 4         The Artist 2011
## 5       Modern Times 1936
## 6         Fight Club 1999
## 7        City of God 2002
## 8   The Untouchables 1987
```

It's very easy! You can note how it looks like a table.

We can also use the `$` selector to get some type of information. This operator returns the content of a specific column of a data frame (that's why we have to choose a name for each column).

```
movies$name
```

```
## [1] "Toy Story"          "Akira"              "The Breakfast Club"
## [4] "The Artist"         "Modern Times"       "Fight Club"
## [7] "City of God"        "The Untouchables"
```

You retrieve data using numeric indexing, like in lists:

```
# This returns the first (1st) column
movies[1]
```

```
##                 name
## 1          Toy Story
## 2              Akira
## 3 The Breakfast Club
## 4         The Artist
## 5       Modern Times
## 6         Fight Club
## 7        City of God
## 8   The Untouchables
```

4.1 Coding Exercise 3:

In the code cell below, select the first row of movies data frame:

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

4.2 Coding Exercise 4:

In the code cell below, select the first and second rows but only with first column selected from the movies data frame.

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

The function called `str()` is one of most useful functions in R. With this function you can obtain textual information about an object. In this case, it delivers information about the objects within a data frame. Let's see what it returns:

```
str(movies)
```

```
## 'data.frame':    8 obs. of  2 variables:
##  $ name: chr  "Toy Story" "Akira" "The Breakfast Club" "The Artist" ...
##  $ year: num  1995 1998 1985 2011 1936 ...
```

It shows this data frame has 8 observations, for 2 columns, so called `name` and `year`. The "name" column is a factor with 8 levels and "year" is a numerical column.

The `class()` function works for data frames as well. You can use it to determine the type of a column of a data frame.

```
class(movies$year)
```

```
## [1] "numeric"
```

You can use numerical selectors to reach information inside the table.

```
movies[1, 2] #1 - Toy Story, 2 - 1995
```

```
## [1] 1995
```

The `head()` function is very useful when you have a large table and you need to take a peek at the first elements. This function returns the first 6 values of a data frame (or event a list).

```
head(movies)
```

```
##                 name year
## 1          Toy Story 1995
## 2              Akira 1998
## 3 The Breakfast Club 1985
## 4         The Artist 2011
## 5       Modern Times 1936
## 6         Fight Club 1999
```

Similar to the previous function, `tail()` returns the last 6 values of a data frame or list.

```
tail(movies)
```

```
##                 name year
## 3 The Breakfast Club 1985
## 4         The Artist 2011
## 5       Modern Times 1936
## 6         Fight Club 1999
## 7        City of God 2002
## 8   The Untouchables 1987
```

Now, let's try to get the row with name "Toy Story".

```
# Find the rows with name "Toy Story"
selected <- movies["name"] == "Toy Story"
# Get the selected row(s)
toy_story <- movies[selected, ]
toy_story
```

```
##        name year
## 1 Toy Story 1995
```

Now let's try to add a new column to our data frame with the length of each movie in minutes.

```
movies['length'] <- c(81, 125, 97, 100, 87, 139, 130, 119)
movies
```

```
##                 name year length
## 1          Toy Story 1995     81
## 2              Akira 1998    125
## 3 The Breakfast Club 1985     97
## 4         The Artist 2011    100
## 5       Modern Times 1936     87
## 6         Fight Club 1999    139
## 7        City of God 2002    130
## 8   The Untouchables 1987    119
```

A new column was included into our data frame with just one line of code. We just needed to add a vector to data frame, then it will be our new column.

Now let's try to add a new movie to our data set.

```
movies <- rbind(movies, c(name="Dr. Strangelove", year=1964, length=94))
movies
```

```
##                 name year length
## 1          Toy Story 1995     81
## 2              Akira 1998    125
## 3 The Breakfast Club 1985     97
## 4         The Artist 2011    100
## 5       Modern Times 1936     87
## 6         Fight Club 1999    139
## 7        City of God 2002    130
## 8   The Untouchables 1987    119
## 9    Dr. Strangelove 1964     94
```

Remember, you can't add a list with more variables than the data frame, and vice-versa.

We don't need this movie anymore, so let's delete it. Here we are deleting row 12 by assigning to itself the movies dataframe without the 12th row.

```
movies <- movies[-12,]
movies
```

```
##                 name year length
## 1          Toy Story 1995     81
## 2              Akira 1998    125
## 3 The Breakfast Club 1985     97
## 4         The Artist 2011    100
## 5       Modern Times 1936     87
## 6         Fight Club 1999    139
## 7        City of God 2002    130
## 8   The Untouchables 1987    119
## 9    Dr. Strangelove 1964     94
```

To delete a column you can just set it as `NULL` .

```
movies[["length"]] <- NULL
movies
```

```
##                 name year
## 1          Toy Story 1995
## 2              Akira 1998
## 3 The Breakfast Club 1985
## 4         The Artist 2011
## 5       Modern Times 1936
## 6         Fight Club 1999
## 7        City of God 2002
## 8   The Untouchables 1987
## 9    Dr. Strangelove 1964
```

That is it! You learned a lot about data frames and how easy it is to work with them.

5 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 (http://cocl.us/dsx_rp0101en), which allows you to run analyses in R with two Spark executors for free.

6 Author(s)

Weiqing Wang (https://www.linkedin.com/in/weiqing-wang-641640133/).

Weiqing is a Data Scientist intern at IBM Canada Ltd. Weiqing holds an Honours Bachelor of Science from the University of Toronto with two specialist degrees, respectively in computer science and statistical sciences. He is presently working towards a graduate degree in computer science at the University of Toronto.

Weiqing avail himself of this opportunity to acknowledge that this notebook was written based heavily on past offerings of this course, in particular Thiago Felipe Correa Borges (https://www.linkedin.com/in/thiago-felipe-corr%C3%AAa-borges-a932bb114?trk=nav_responsive_tab_profile)'s, former Web Developer Intern at IBM.

6.1 Other contributors

Yan Luo, PhD (https://www.linkedin.com/in/yan-luo-96288783/).

7 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-07-10	1.0	Weiqing Wang	Initial Version Created.