Reading Data into R

Norman Lo

1/10/2022

Reading Different Data Source into R

We have covered the basic features of R in the first two sections. In this section, we would like to explore how to read data from different data source into R for analysis. R offers several methods to read data from different sources, the most common one is reading from CSV (comma separated values) file and the basic excel file format. R also has the capability to read from relational database, internet data, and computer science data format "JSON".

3.1 - Reading CSV File

The best way to read a CSV file to R is using read.table() function. Many people starts with read.csv() function, but it's actually a wrapper function from read.table(), which is the same as using the **sep** argument with a comma (",") in read.table() function. read.table() function returns a R data.frame object.

The way to use read.table() function is to put the **absolute path** to the CSV file in the first argument. The path could be the location to the file in the machine or it could be a file from the internet. We are going to demonstrate to read a CSV file from the URL in this exercise.

```r
# Assign the URL to a variable
theUrl <- "http://www.jaredlander.com/data/TomatoFirst.csv"

# Read the CSV file from the URL to R
tomato <- read.table(file=theUrl, header=TRUE, sep=",")

# Check the data.frame object
head(tomato)
```

```
##   Round             Tomato Price      Source Sweet Acid Color Texture Overall
## 1     1         Simpson SM  3.99 Whole Foods   2.8  2.8   3.7     3.4     3.4
## 2     1  Tuttorosso (blue)  2.99     Pioneer   3.3  2.8   3.4     3.0     2.9
## 3     1 Tuttorosso (green)  0.99     Pioneer   2.8  2.6   3.3     2.8     2.9
## 4     1     La Fede SM DOP  3.99   Shop Rite   2.6  2.8   3.0     2.3     2.8
## 5     2       Cento SM DOP  5.49  D Agostino   3.3  3.1   2.9     2.8     3.1
## 6     2      Cento Organic  4.99  D Agostino   3.2  2.9   2.9     3.1     2.9
##   Avg.of.Totals Total.of.Avg
## 1          16.1         16.1
## 2          15.3         15.3
## 3          14.3         14.3
## 4          13.4         13.4
## 5          14.4         15.2
## 6          15.5         15.1
```

In this example, we use the three arguments file, header, and sep from the read.table() function to load the CSV file to R, which returns a data.frame object. Note that the argument names (file, header, and sep) are optional and can be missing in the function, such that read.table(theUrl, TRUE, “,”), but it’s a good habit to include the argument names for readability. The second argument “header=TRUE” reads the first row of the CSV file as the column names. The third argument ‘sep=“,” ’ defines the data is separated by commas. We can also separate the data by different notations, such as “"̂ (tab),”;" (semi-colon), etc, for different data formats.

In some cases, we may want to use “stringAsFactors=FALSE” argument to avoid a string column automatically read as factor. It saves the computational time when reading a large character data set. “stringAsFactor” can also be used in data.frame, here is an example to demonstrate.

```
# Create three individual vectors
x <- 10:1
y <- -4:5
q <- c("Hockey", "Football", "Baseball", "Curling", "Rugby",
       "Lacrosse", "Basketball", "Tennis", "Cricket", "Soccer")

# Create data.frame by using the three vectors
theDF <- data.frame(First=x, Second=y, Sport=q, stringsAsFactors=FALSE)

# Check the Sport column
theDF$Sport
```

```
##  [1] "Hockey"     "Football"   "Baseball"   "Curling"    "Rugby"
##  [6] "Lacrosse"   "Basketball" "Tennis"     "Cricket"    "Soccer"
```

When reading a large CSV file, read.table() function is relatively slow compares to other packages. For instance, we can use the read_delim() and data.table() from the **readr** package. The two functions are relatively quick and won’t automatically transfrom character data to factor data type.

```
# Loading readr package
library(readr)

# Assign URL to a variable
theUrl <- "http://www.jaredlander.com/data/TomatoFirst.csv"

# Loading the CSV to R and create a tibble object
tomato2 <- read_delim(file=theUrl, delim=", ")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Round = col_double(),
##   Tomato = col_character(),
##   Price = col_double(),
##   Source = col_character(),
##   Sweet = col_double(),
##   Acid = col_double(),
##   Color = col_double(),
##   Texture = col_double(),
##   Overall = col_double(),
##   ‘Avg of Totals‘ = col_double(),
##   ‘Total of Avg‘ = col_double()
## )
```

```
head(tomato2)
```

```
## # A tibble: 6 x 11
##   Round Tomato    Price Source Sweet  Acid Color Texture Overall `Avg of Totals`
##   <dbl> <chr>     <dbl> <chr>  <dbl> <dbl> <dbl>   <dbl>   <dbl>           <dbl>
## 1     1 Simpson ~  3.99 Whole~   2.8   2.8   3.7     3.4     3.4            16.1
## 2     1 Tuttoros~  2.99 Pione~   3.3   2.8   3.4     3       2.9            15.3
## 3     1 Tuttoros~  0.99 Pione~   2.8   2.6   3.3     2.8     2.9            14.3
## 4     1 La Fede ~  3.99 Shop ~   2.6   2.8   3       2.3     2.8            13.4
## 5     2 Cento SM~  5.49 D Ago~   3.3   3.1   2.9     2.8     3.1            14.4
## 6     2 Cento Or~  4.99 D Ago~   3.2   2.9   2.9     3.1     2.9            15.5
## # ... with 1 more variable: Total of Avg <dbl>
```

Similar to read.table(), the first argument of the read_delim() function is the path to the CSV file. The second argument col_names is default to be TRUE and the third argument delim="," reads the data separated by commas. Note that read_delim() function or other data reading functions from readr package return a tibble object, which is a modern reimagining of the data.frame. In the readr package, read_csv, read_csv2, and read_tsv functions are the special cases, which separate the data by comma (,), semi-colon (;), and tab ().

An alternative would be fread() function from data.table package. Similar to the previous data reading function, the first argument is the path or URL to the CSV file. The second argument sep=',' represents the data separated by comma. The third argument header=TRUE reads the first row as the column names. This function also has the stringsAsFactor argument, which is default to be FALSE.

```
# Loading data.table package
library(data.table)

# Assign URL to a variable
theUrl <- "http://www.jaredlander.com/data/TomatoFirst.csv"

# Loading the CSV to R and create a data.table object
tomato3 <- fread(input=theUrl, sep=',', header=TRUE)
head(tomato3)
```

```
##    Round              Tomato Price      Source Sweet Acid Color Texture Overall
## 1:     1          Simpson SM  3.99 Whole Foods   2.8  2.8   3.7     3.4     3.4
## 2:     1  Tuttorosso (blue)  2.99     Pioneer   3.3  2.8   3.4     3.0     2.9
## 3:     1 Tuttorosso (green)  0.99     Pioneer   2.8  2.6   3.3     2.8     2.9
## 4:     1      La Fede SM DOP  3.99   Shop Rite   2.6  2.8   3.0     2.3     2.8
## 5:     2        Cento SM DOP  5.49  D Agostino   3.3  3.1   2.9     2.8     3.1
## 6:     2      Cento Organic  4.99  D Agostino   3.2  2.9   2.9     3.1     2.9
##    Avg of Totals Total of Avg
## 1:          16.1         16.1
## 2:          15.3         15.3
## 3:          14.3         14.3
## 4:          13.4         13.4
## 5:          14.4         15.2
## 6:          15.5         15.1
```

Genearlly speaking, both read_delim() and fread() are efficient functions for loading CSV file into R. The choice depends on the package (dplyr or data.table) that you are more fimilar to work with.

3.2 - Reading Excel Files

Excel is the most popular data analytic tools, which is simple, powerful, and easy to learn. In many documentation from the R community, it was suggested to transform excel file to CSV file and read it into R by read.csv() function. In the recent years, reading excel file into R is getting simplier, which we can use the **readxl** package developed by Hadley Wickham. The function we are going to use to read either .xls or .xlsx files is read_excel(). Unlike read.table(), read_delim(), and fread(), read_excel() cannot read file from internet source (URL). In this example, we use download.file() to download the excel file from the internet source, then use read_excel() reading the data into R.

```
# Downloading an excel file from the internet
download.file(url='http://www.jaredlander.com/data/ExcelExample.xlsx',
              destfile='data/ExcelExample.xlsx', mode='wb')

# Loading readxl package
library(readxl)

# Loading the first sheet in the xlsx file from the data folder
excel_sheets('data/ExcelExample.xlsx')
```

```
## [1] "Tomato" "Wine"   "ACS"
```

```
# Create a tibble object and assign to a variable
tomatoXL <- read_excel('data/ExcelExample.xlsx')
tomatoXL
```

```
## # A tibble: 16 x 11
##    Round Tomato   Price Source Sweet  Acid Color Texture Overall `Avg of Totals`
##    <dbl> <chr>    <dbl> <chr>  <dbl> <dbl> <dbl>   <dbl>   <dbl>           <dbl>
##  1     1 Simpson~  3.99 Whole~   2.8   2.8   3.7     3.4     3.4            16.1
##  2     1 Tuttoro~  2.99 Pione~   3.3   2.8   3.4     3       2.9            15.3
##  3     1 Tuttoro~  0.99 Pione~   2.8   2.6   3.3     2.8     2.9            14.3
##  4     1 La Fede~  3.99 Shop ~   2.6   2.8   3       2.3     2.8            13.4
##  5     2 Cento S~  5.49 D Ago~   3.3   3.1   2.9     2.8     3.1            14.4
##  6     2 Cento O~  4.99 D Ago~   3.2   2.9   2.9     3.1     2.9            15.5
##  7     2 La Vall~  3.99 Shop ~   2.6   2.8   3.6     3.4     2.6            14.7
##  8     2 La Vall~  3.99 Faicos   2.1   2.7   3.1     2.4     2.2            12.6
##  9     3 Stanisl~  4.53 Resta~   3.4   3.3   4.1     3.2     3.7            17.8
## 10     3 Ciao     NA    Other    2.6   2.9   3.4     3.3     2.9            15.3
## 11     3 Scotts ~  0    Home ~   1.6   2.9   3.1     2.4     1.9            11.9
## 12     3 Di Casa~ 12.8  Eataly   1.7   3.6   3.8     2.3     1.4            12.7
## 13     4 Trader ~  1.49 Trade~   3.4   3.3   4       3.6     3.9            17.8
## 14     4 365 Who~  1.49 Whole~   2.8   2.7   3.4     3.1     3.1            14.8
## 15     4 Muir Gl~  3.19 Whole~   2.9   2.8   2.7     3.2     3.1            14.8
## 16     4 Bionatu~  3.39 Whole~   2.4   3.3   3.4     3.2     2.8            15.1
## # ... with 1 more variable: Total of Avg <dbl>
```

Note: read_excel() is default to read the first sheet in the excel file, which is the "tomato" sheet in our example. The return object is a tibble, not data.frame object.

If we want to read the second sheet from the excel file into R, we can put in the argument "sheet=2" or "sheet='Wine' " to read a specific sheet from the file. Here are two examples:

```
# Loading readxl package
library(readxl)

# Loading the second sheet in the xlse file from the data folder
wineXL1 <- read_excel('data/ExcelExample.xlsx', sheet=2)
head(wineXL1)
```

```
## # A tibble: 6 x 14
##   Cultivar Alcohol `Malic acid`   Ash `Alcalinity of ~ Magnesium `Total phenols`
##      <dbl>   <dbl>        <dbl> <dbl>            <dbl>     <dbl>           <dbl>
## 1        1    14.2         1.71  2.43             15.6       127            2.8
## 2        1    13.2         1.78  2.14             11.2       100            2.65
## 3        1    13.2         2.36  2.67             18.6       101            2.8
## 4        1    14.4         1.95  2.5              16.8       113            3.85
## 5        1    13.2         2.59  2.87             21         118            2.8
## 6        1    14.2         1.76  2.45             15.2       112            3.27
## # ... with 7 more variables: Flavanoids <dbl>, Nonflavanoid phenols <dbl>,
## #   Proanthocyanins <dbl>, Color intensity <dbl>, Hue <dbl>,
## #   OD280/OD315 of diluted wines <dbl>, Proline <dbl>
```

```
# Loading the second sheet in the xlse file by sheet name
wineXL2 <- read_excel('data/ExcelExample.xlsx', sheet='Wine')
head(wineXL2)
```

```
## # A tibble: 6 x 14
##   Cultivar Alcohol `Malic acid`   Ash `Alcalinity of ~ Magnesium `Total phenols`
##      <dbl>   <dbl>        <dbl> <dbl>            <dbl>     <dbl>           <dbl>
## 1        1    14.2         1.71  2.43             15.6       127            2.8
## 2        1    13.2         1.78  2.14             11.2       100            2.65
## 3        1    13.2         2.36  2.67             18.6       101            2.8
## 4        1    14.4         1.95  2.5              16.8       113            3.85
## 5        1    13.2         2.59  2.87             21         118            2.8
## 6        1    14.2         1.76  2.45             15.2       112            3.27
## # ... with 7 more variables: Flavanoids <dbl>, Nonflavanoid phenols <dbl>,
## #   Proanthocyanins <dbl>, Color intensity <dbl>, Hue <dbl>,
## #   OD280/OD315 of diluted wines <dbl>, Proline <dbl>
```

3.3 - Connecting to Database

In practice, most of the company data is stored into the database. Most of these database, such as PostgreSQL, MySQL, Microsoft SQL server, or Microsoft Access, can be connected using different program. A common way to connect to a database is using ODBC connection. Two of the most popular open source packages for database connection are RPostgreSQL and RMySQL. Building a connection to different database is not an easy task, which DBI package aims to construct a standardized connection structure to achieve the task.

In this section, we only demonstrate building a connect to a simple database SQLite. The example illustrates the steps that also apply to most of the database connection cases. First, we download a database file from the internet source. Then, we import the RSQLite package for building the connection to the SQLite database file.

```
# Downloading database file
# download.file("http://www.jaredlander.com/data/diamonds.db",
#               destfile="data/diamonds.db", mode="wb")

# Loading RSQLite package
# library(RSQLite)
```

To connect to the database, we need to first create a database driver and define the database type for activating the driver program.

```
# Creating a database driver
# drv <- dbDriver('SQLite')
# class(drv)
```

Next, we connect to the database using dbConnect() function. The first argument is always the database driver and the second argument is usually DSN connection or direction path to the database file. Additional argument may be needed for other database structures, such as database user name, password, host, and port.

```
# Code Block Setting: drv=drv
# Build connection to database
# con <- dbConnect(drv, 'data/diamonds.db')
# class(con)
```

Once we build the connection to the database, we can use functions from the DBI package to extract information from the database, such as table names, field names, etc.

```
# Code Block Setting: con=con
# Extracting table names from the database
# dbListTables(con)

# Extracting field (column) names from diamond table
# dbListFields(con, name='diamonds')

# Extracting field (column) names from DiamondColors table
# dbListFields(con, name='DiamondColors')
```

Now we can use dbGetQuery() function to perform query from the database. dbGetQuery() returns a data.frame object from the query. Since dbGetQuery() has the argument “stringAsFactors”, it’s suggested to set it to FALSE, so the character data can be extract as its original type.

```
# Code Block Setting: con=con
# Use SELECT * to query diamonds table from database
# diamondsTable <- dbGetQuery(con,
#                             "SELECT * FROM diamonds",
#                             stringsAsFactors=FALSE)
# head(diamondsTable)

# Use SELECT * to query DiamondColors table from database
# colorTable <- dbGetQuery(con,
#                             "SELECT * FROM DiamondColors",
#                             stringsAsFactors=FALSE)
# head(colorTable)

# Join the two tables
# longQuery <- "SELECT * FROM diamonds, diamondColors WHERE diamonds.color = DiamondColors.Color"
# diamondsJoin <- dbGetQuery(con, longQuery,
#                            stringsAsFactors=FALSE)
# head(diamondsJoin)

# Disconnect the connection to database
# dbDisconnect(con)
```

Even though ODBC connection will disconnect once R is closed or connecting to a different database using dbConnect(), it is recommended to build a good habit to manually close the ODBC connect with dbDisconnect() to avoid any complication or error.

3.4 - Reading Data from other Software Formats

Technically speaking, we don't need to worry much about data saved in other statistical package format once we adopted to R. In some cases, we may need to load data sources in different format, for instance, a project that you had worked on before or a project that was completed by other researchers or scentist in different software package. The **foreign** package in R offers several functions similar to read.table for reading data source saved with different software package format. Here is a list of these functions.

Function	Data Type
read.spss	SPSS
read.dta	Stata
read.ssd	SAS
read.octave	Octava
read.mtp	Minitab
read.systat	Systa

These functions return a data.frame object, which is similar to the R built-in function read.table().

Hadley Wickham developed a package, **haven**, which is similar to the **foreing** package, but significantly increase the computational time and efficiency. The return object is tibble. Here is a list for the functions in the **haven** package.

Function	Data Type
read_spss	SPSS
read_sas	SAS
read_stata	STATA

3.5 - Reading RData File

The best way to store and share objects from R is with **RData** files are specific to R and can store as many objects as you'd like within a single file. It can also be opened in different operation systems, Windows, Mac, and Linux.

In this example, we first create a RData file storing the R objects. Next, we remove the objects from the global environment. Then we load the objects from the RData file to retrieve the objects.

```
# Save tomato data.frame to the local drive
save(tomato, file="data/tomato.rdata")

# Remove tomato from the global environment
rm(tomato)

# Check if tomato exists
head(tomato)
```

```
## Error in head(tomato): object 'tomato' not found
```

```
# Read the tomato data.frame from the local drive
load("data/tomato.rdata")

# Check if tomato exists
head(tomato)
```

```
##   Round             Tomato Price      Source Sweet Acid Color Texture Overall
## 1     1         Simpson SM  3.99 Whole Foods   2.8  2.8   3.7     3.4     3.4
## 2     1  Tuttorosso (blue)  2.99     Pioneer   3.3  2.8   3.4     3.0     2.9
## 3     1 Tuttorosso (green)  0.99     Pioneer   2.8  2.6   3.3     2.8     2.9
## 4     1     La Fede SM DOP  3.99   Shop Rite   2.6  2.8   3.0     2.3     2.8
## 5     2       Cento SM DOP  5.49  D Agostino   3.3  3.1   2.9     2.8     3.1
## 6     2      Cento Organic  4.99  D Agostino   3.2  2.9   2.9     3.1     2.9
##   Avg.of.Totals Total.of.Avg
## 1          16.1         16.1
## 2          15.3         15.3
## 3          14.3         14.3
## 4          13.4         13.4
## 5          14.4         15.2
## 6          15.5         15.1
```

```
# Create two objects and store into a data.frame
n <- 20
r <- 1:10
w <- data.frame(n, r)

# Check these objects
n
```

```
## [1] 20
```

```
r
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
w
```

```
##     n  r
## 1  20  1
## 2  20  2
## 3  20  3
## 4  20  4
## 5  20  5
## 6  20  6
## 7  20  7
## 8  20  8
## 9  20  9
## 10 20 10
```

```
# Save the objects to the local drive
save(n, r, w, file="data/multiple.rdata")

# Check if the objects are in memory
n
```

```
## [1] 20
```

```
r
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
w
```

```
##     n  r
## 1  20  1
## 2  20  2
## 3  20  3
## 4  20  4
## 5  20  5
## 6  20  6
## 7  20  7
## 8  20  8
## 9  20  9
## 10 20 10
```

```
# Load the data from the local drive
load("data/multiple.rdata")

# Check the objects again
n
```

```
## [1] 20
```

```
r
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
w
```

```
##     n  r
## 1  20  1
## 2   20  2
## 3   20  3
## 4   20  4
## 5   20  5
## 6   20  6
## 7   20  7
## 8   20  8
## 9   20  9
## 10  20 10
```

Note: Loading the RData file into R does not need to assign to a variable like loading the Excel or CSV file because the RData file load all the saved objects to the working environment.

R has another data format for saving a single R object from the working environment, which is **RDS**. We use the saveRDS() function to save the object to the RDS file. Note that RDS file does not save the object name, so when we reading the file into R using readDRS() function, we need to assign it to a variable. Here is an example.

```r
# Create a vector object
smallVector <- c(1, 5, 4)
smallVector
```

```
## [1] 1 5 4
```

```r
# Save it to the local drive in RDS format
saveRDS(smallVector, file="data/thisObject.rds")

# Read the RDS file and assign to a new variable
thatVect <- readRDS("data/thisObject.rds")
thatVect
```

```
## [1] 1 5 4
```

```r
# Check if the two variables are identical
identical(smallVector, thatVect)
```

```
## [1] TRUE
```

3.6 - Reading R Built-In Data

Similar to other statistical software, R offers a list of datasets from different packages, such as ggplot2, MASS, and ISLR. We demonstrate loading the diamonds dataset from ggplot2 package in this example.

```r
# Reading the built-in data set diamonds
data(diamonds, package='ggplot2')

# Check the data set
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

Note: An alternative way to load the built-in data sets in R is importing the package before using the data set.

3.7 - Reading Data from the Internet

Since the invention of the internet, many data resources have become available for public to use. If you are fortunate to find a well-managed data source from the internet, it could be efficiently scraped from the HTML format. In a more complex case, we may need to break down the HTML structure and extract the data that we need.

In this example, we are going to demonstrate how to extract data from a well-managed data source from a HTML sheet using readHTMLTable() function from the XML package. The target web page in this example contains a table with 15 columns, which can be easily extracted:

```r
# Importing XML package
library(XML)
```

```
## Error in library(XML): there is no package called 'XML'
```

```r
library(RCurl)
```

```
## Error in library(RCurl): there is no package called 'RCurl'
```

```r
# Assign the URL link
theURL <- "https://normanlo4319.github.io/Norman_Lo_Web/data.html"
html <- getURL(theURL)
```

```
## Error in getURL(theURL): could not find function "getURL"
```

```r
# Reading the data from the URL
table <- readHTMLTable(html, which=1, header=TRUE,
                          stringsAsFactors=FALSE)
```

```
## Error in readHTMLTable(html, which = 1, header = TRUE, stringsAsFactors = FALSE): could not find func
```

Here, readHTMLTable() function takes the URL path from the first argument to locate the page and read the HTML sheet. "which=1" definds the first table tag item in the HTML. "header=TRUE" means treating the first row as headers of the columns. "stringsAsFactors=FALSE" keeps the character format in the data source.

However, in most cases, data are not stored in a HTML table, but in different tag elements, such as divs, span, h, etc. We demonstrates the use of read_html() function in rvest package developed by Hadley Wickham to extract the data from different tag element and create a xml_document object to store these HTML elements.

```r
# Step 1:
# Importing rvest package
library(rvest)
```

```
## Warning: package 'rvest' was built under R version 4.1.2
```

```
##
## Attaching package: 'rvest'
```

```
## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```r
# Reading the data from the URL
ribalta <- read_html('http://www.jaredlander.com/data/ribalta.html')

# Check the class of the return object
class(ribalta)
```

```
## [1] "xml_document" "xml_node"
```

```r
# Print the return object
ribalta
```

```
## {html_document}
## <html xmlns="http://www.w3.org/1999/xhtml">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body>\r\n<ul>\n<li class="address">\r\n    <span class="street">48 E 12t ...
```

```r
# Step 2:
# Filtering the data from the HTML
ribalta %>% html_nodes('ul') %>% html_nodes('span')
```

```
## {xml_nodeset (6)}
## [1] <span class="street">48 E 12th St</span>
## [2] <span class="city">New York</span>
## [3] <span class="zip">10003</span>
## [4] <span>\r\n    \t<span id="latitude" value="40.733384"></span>\r\n    \t<s ...
## [5] <span id="latitude" value="40.733384"></span>
## [6] <span id="longitude" value="-73.9915618"></span>
```

```r
# Step 3:
# Find the class with element "street"
ribalta %>% html_nodes('.street')
```

```
## {xml_nodeset (1)}
## [1] <span class="street">48 E 12th St</span>
```

```r
# Step 4:
# Extract the characters data within the span
ribalta %>% html_nodes('.street') %>% html_text()
```

```
## [1] "48 E 12th St"
```

```r
# Step 5:
# Find the attribute using its ID
ribalta %>% html_nodes('#longitude') %>% html_attr('value')
```

```
## [1] "-73.9915618"
```

```r
# Step 6:
# Extract food items from the table
ribalta %>% html_nodes('table.food-items') %>%
  magrittr:: extract2(5) %>% html_table()
```

```
## # A tibble: 6 x 3
##   X1                    X2                                                     X3
##   <chr>                 <chr>                                               <dbl>
## 1 Marinara Pizza Rosse basil, garlic and oregano.                              9
## 2 Doc Pizza Rosse      buffalo mozzarella and basil.                          15
## 3 Vegetariana Pizza R~ mozzarella cheese, basil and baked vegetables.         15
## 4 Brigante Pizza Rosse mozzarella cheese, salami and spicy oil.               15
## 5 Calzone Pizza Rosse  ricotta, mozzarella cheese, prosciutto cotto and b~    16
## 6 Americana Pizza Ros~ mozzarella cheese, wurstel and fries.                  16
```

Note: Web-scraping is a subject that involves extensive knowledge in web development and computer programming, therefore, we are not going into the details of different web-scraping tools in R. It's important to note that R has the capability to extract data from the internet for analytic purpose.

3.8 - Reading JSON Data

JSON (Javascript Object Notation) is a common data format used in API and documentation database. JSON saves data into plain text format, which is suitable for nested data. There are two common packages in R for reading the JSON data, **rjson** and **jsonlite**.

Below is an example of a JSON data that lists out the famous Pizza restaurants in New York. The first element in an object is the name of the restaurant then follow with Details, which is a list of address, city, State, and Zip code.

```json
[
  {
    "Name": "Di Fara Pizza",
    "Detail": [
      {
        "Address": "1424 Avenue J",
        "City": "Brooklyn",
        "State": "NY",
        "Zip": "11230"
      }
    ]
  },
  {
    "Name": "Fiore's Pizza",
    "Detail": [
      {
        "Address": "165 Bleecker St",
        "City": "New York",
        "State": "NY",
        "Zip": "10012"
      }
    ]
  },
  {
    "Name": "Juliana's",
    "Detail": [
      {
        "Address": "19 Old Fulton St",
        "City": "Brooklyn",
        "State": "NY",
        "Zip": "11201"
      }
    ]
  },
]
```

```
## Error: <text>:1:1: unexpected '['
## 1: [
##     ^
```

When we reading JSON data into R, for instance using fromJSON() function from jsonlite package, by default, it returns a data.frame object to fit all the data.

```
# Importing jsonlite package
library(jsonlite)

# Assign the JSON data from the internet to a variable
pizza <- fromJSON('http://www.jaredlander.com/data/PizzaFavorites.json')
pizza
```

```
##                     Name                                  Details
## 1          Di Fara Pizza     1424 Avenue J, Brooklyn, NY, 11230
## 2          Fiore's Pizza   165 Bleecker St, New York, NY, 10012
## 3              Juliana's  19 Old Fulton St, Brooklyn, NY, 11201
## 4     Keste Pizza & Vino   271 Bleecker St, New York, NY, 10014
## 5  L & B Spumoni Gardens      2725 86th St, Brooklyn, NY, 11223
## 6 New York Pizza Suprema       413 8th Ave, New York, NY, 10001
## 7           Paulie Gee's 60 Greenpoint Ave, Brooklyn, NY, 11222
## 8                Ribalta      48 E 12th St, New York, NY, 10003
## 9              Totonno's  1524 Neptune Ave, Brooklyn, NY, 11224
```

```r
# Check the class of the return object
class(pizza)
```

```
## [1] "data.frame"
```

```r
# Check the data type in a specific column
class(pizza$Name)
```

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

```r
class(pizza$Details)
```

```
## [1] "list"
```

```r
class(pizza$Details[[1]])
```

```
## [1] "data.frame"
```

Note: Since JSON data can be nested, the Detail column in this example is actually a column of data.frame objects. For this kind data.frame, we could use tools like dplyr, tidyr, and purrr to breakdown the nested data.frame and separate the data into columns.

Summary:

Reading data is always the first step in data analytic. Nothing can be done without the data in R. For most academic research, the most common methods are using read.table() and read_excel() functions to read the CSV or Excel file into R for analysis. Rdata also provides the convenience to save the assigned variables and objects for continuous project. Therefore, understanding the data format is an important step to become a good data analyst or scientist.