## R - Packages

R packages are a collection of R functions, complied code and sample data. They are stored under a directory called **"library"** in the R environment. By default, R installs a set of packages during installation. More packages are added later, when they are needed for some specific purpose. When we start the R console, only the default packages are available by default. Other packages which are already installed have to be loaded explicitly to be used by the R program that is going to use them.

All the packages available in R language are listed at R Packages.

Below is a list of commands to be used to check, verify and use the R packages.

Check Available R Packages

Example: Get library locations containing R packages

```
> .libPaths()
[1] "C:/Users/hp/AppData/Local/R/win-library/4.4"
[2] "C:/Program Files/R/R-4.4.0/library"
```

Example: Get the list of all the packages installed:

```
> library()
```

When we execute the above code, it produces the following result. It may vary depending on the local settings of your pc. In our case, the results will be:

```
Packages in library ‘C:/Users/hp/AppData/Local/R/win-library/4.4’:

abind                Combine Multidimensional Arrays
AnnotationDbi        Manipulation of SQLite-based annotations
                     in Bioconductor
askpass              Password Entry Utilities for R, Git, and
                     SSH
base64enc            Tools for base64 encoding
BH                   Boost C++ Header Files
Biobase              Biobase: Base functions for Bioconductor
BiocGenerics         S4 generic functions used in Bioconductor
BiocIO               Standard Input and Output for
```

### Get all packages currently loaded in the R environment

```
> search()
 [1] ".GlobalEnv"        "tools:rstudio"     "package:stats
"
 [4] "package:graphics"  "package:grDevices" "package:utils
"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

When we execute the above code, it produces the above result. It may vary depending on the local settings of your pc.

### Install a New Package

There are two ways to add new R packages. One is installing directly from the CRAN (Comprehensive R Archive Network) directory and another is downloading the package to your local system and installing it manually.

### Install directly from CRAN

The following command gets the packages directly from CRAN webpage and installs the package in the R environment. You may be prompted to choose a nearest mirror. Choose the one appropriate to your location.

**install.packages("Package Name")**

```
> # Install the package named "XML".
> install.packages("XML")

WARNING: Rtools is required to build R packages but is not curren
tly installed. Please download and install the appropriate versio
n of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into ‘C:/Users/hp/AppData/Local/R/win-library/
4.4’
(as ‘lib’ is unspecified)

  There is a binary version available but the source version
  is later:
          binary    source needs_compilation
XML 3.99-0.16.1 3.99-0.17              TRUE

  Binaries will be installed
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.4/XML_
3.99-0.16.1.zip'
Content type 'application/zip' length 3103340 bytes (3.0 MB)
```

```
downloaded 3.0 MB

package ‘XML’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Public\Documents\iSkysoft\CreatorTemp\RtmpWeH194\
downloaded_packages
```

### Install package manually

Go to the link R Packages to download the package needed. Save the package as a **.zip** file in a suitable location in the local system.

Now you can run the following command to install this package in the R environment.

install.packages(file_name_with_path, repos = NULL, type = "source")

Example:

```
> # Install the package named "XML"
> install.packages("E:/XML_3.98-1.3.zip", repos = NULL, typ
e = "source")
```

### Load Package or Library

Before a package can be used in the code, it must be loaded to the current R environment. You also need to load a package that is already installed previously but not available in the current environment. A package is loaded using the following command:

```
> # Loading library called "xlsx"
> library("xlsx")
```

### Viewing the contents of loaded library

In R, you can view the contents of a library (also known as a package) using the ls() function or by exploring the package's documentation. Here's how you can do it:

```
> ls("package:xlsx")
```

```
 [1] "addAutoFilter"          "addDataFrame"
 [3] "addHyperlink"           "addMergedRegion"
 [5] "addPicture"             "Alignment"
 [7] "autoSizeColumn"         "Border"
 [9] "BORDER_STYLES_"         "CB.setBorder"
[11] "CB.setColData"          "CB.setFill"
[13] "CB.setFont"             "CB.setMatrixData"
[15] "CB.setRowData"          "CELL_STYLES_"
[17] "CellBlock"              "CellProtection"
[19] "CellStyle"              "createCell"
[21] "createCellComment"      "createFreezePane"
[23] "createRange"            "createRow"
[25] "createSheet"            "createSplitPane"
[27] "createWorkbook"         "DataFormat"
[29] "Fill"                   "FILL_STYLES_"
[31] "Font"                   "forceFormulaRefresh"
[33] "forcePivotTableRefresh" "get_java_tmp_dir"
[35] "getCellComment"         "getCells"
[37] "getCellStyle"           "getCellValue"
[39] "getRanges"              "getRows"
[41] "getSheets"              "HALIGN_STYLES_"
[43] "INDEXED_COLORS_"        "is.Alignment"
[45] "is.Border"              "is.CellBlock"
[47] "is.CellProtection"      "is.CellStyle"
[49] "is.DataFormat"          "is.Fill"
[51] "is.Font"                "loadWorkbook"
[53] "printSetup"             "read.xlsx"
[55] "read.xlsx2"             "readColumns"
[57] "readRange"              "readRows"
[59] "removeCellComment"      "removeMergedRegion"
[61] "removeRow"              "removeSheet"
[63] "saveWorkbook"           "set_java_tmp_dir"
[65] "setCellStyle"           "setCellValue"
[67] "setColumnWidth"         "setPrintArea"
[69] "setRowHeight"           "setZoom"
[71] "VALIGN_STYLES_"         "write.xlsx"
[73] "write.xlsx2"
```

You can also explore the documentation of a package to see its contents by Using **help()** or **?** to Explore Documentation in the help tab (down right window of RStudio).

```
> # Explore the documentation of package in help tab
> help(package = "xlsx")
>
> # Using ? to get help (this is used to get help of everyt
hing)
> ? "xlsx"
```

### Detaching a Package

you can **detach** a package from the search path, which effectively removes its functions and datasets from your current R session. Use the **detach()** function to remove a package from the search path:

```
> # Viewing current loaded packages
>
> search()
 [1] ".GlobalEnv"        "package:xlsx"      "tools:rstudio
"
 [4] "package:stats"     "package:graphics"  "package:grDev
ices"
 [7] "package:utils"     "package:datasets"  "package:metho
ds"
[10] "Autoloads"         "package:base"
>
> # Detach “xlsx” package
> detach("package:xlsx", unload = TRUE)
>
> # Viewing loaded packages after detaching “xlsx” package
>
> search()
 [1] ".GlobalEnv"        "tools:rstudio"     "package:stats
"
 [4] "package:graphics"  "package:grDevices" "package:utils
"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

- "package:dplyr": Specifies the package to detach.
- unload = TRUE: Ensures the package is unloaded from memory (optional but recommended).

### R Data Interferences

### CSV files

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

In this section we will learn to read data from a csv file and then write data into a csv file. **The file should be present in current working directory so**

**that R can read it, elsewhere you have to specify path** . Of course we can also set our own directory and read files from there.

### Getting and Setting the Working Directory

You can check which directory the R workspace is pointing to using the **getwd()** function. You can also set a new working directory using **setwd()**function.

Example:

```
> # Get and print current working directory.
> print(getwd())

"C:/Users/hp/R"


> # Set current working directory.
> setwd("C:/Users/hp/OneDrive/Documents")
>
> # Get and print current working directory.
> print(getwd())
"C:/Users/hp/R"
```

This result depends on your OS and your current directory where you are working.

### Input as CSV File

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named **input.csv**.

You can create this file using windows notepad by copying and pasting this data. Save the file as **input.csv** using the save As All files(*.*) option in notepad.

```
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Michelle,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
5,Gary,843.25,2015-03-27,Finance
```

```
6,Nina,578,2013-05-21,IT
7,Simon,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance
```

### Reading a CSV File

Following is a simple example of **read.csv()** function to read a CSV file available in your current working directory:

```
> data <- read.csv("input.csv")
> print(data)

  id     name salary start_date       dept
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5  5     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
```

### Analyzing the CSV File

By default the **read.csv()** function gives the output as a data frame. This can be easily checked as follows. Also we can check the number of columns and rows.

Example:

```
> data <- read.csv("input.csv")
>
> print(is.data.frame(data))
[1] TRUE
>
> print(ncol(data))
[1] 5
>
> print(nrow(data))
[1] 8
```

Once we read data in a data frame, we can apply all the functions applicable to data frames as explained in subsequent section.

**Get the maximum salary:**

```
> # Create a data frame.
> data <- read.csv("input1.csv")
>
> # Get the max salary from data frame.
> sal <- max(data$salary)
> print(sal)
```

```
[1] 843.25
```

### Writing into a CSV File

R can create csv file form existing data frame. The **write.csv()** function is used to create the csv file. This file gets created in the working directory.

```
> # Create a data frame.
> data <- read.csv("input.csv")
> retval <- subset(data, start_date >= "2014-01-01")
>
> # Write filtered data into a new file.
> write.csv(retval,"output.csv")
> newdata <- read.csv("output.csv")
> print(newdata)
```

```
  X id     name salary start_date    dept
1 3  3 Michelle 611.00 2014-11-15      IT
2 4  4     Ryan 729.00 2014-05-11      HR
3 5  5     Gary 843.25 2015-03-27 Finance
4 8  8     Guru 722.50 2014-06-17 Finance
```

## R Data Interferences

### Excel files

Microsoft Excel is the most widely used spreadsheet program which stores data in the .xls or .xlsx format. R can read directly from these files using some excel specific packages. The popular R packages for reading and writing Excel files are:

### Reading Excel Files:

- **readxl:** This is a popular and user-friendly choice for reading data from both .xls and .xlsx Excel files into R data frames. It's known for its simplicity and lack of external dependencies, making it work seamlessly across different operating systems.

**Example:** reading excel file using **read_excel()** function of readxl package

```
> # Loading the required library “readxl”
> library(readxl)
>
> input <- read_excel("input.xlsx", sheet = “sheet1”)
> View(input) # this command optional, view the table in ed
iting window (upper left part of R Studio)
> input
# A tibble: 8 × 5
     id name      salary start_date          dept
  <dbl> <chr>      <dbl> <dttm>              <chr>
1     1 Rick      623.   2002-01-01 00:00:00 012          IT
2     2 Dan       515.   2013-09-23 00:00:00 Operations
3     3 Michelle  611    2014-11-15 00:00:00 IT
4     4 Ryan      729    2014-05-11 00:00:00 HR
5     5 Gary       43.2  2015-03-27 00:00:00 Finance
6     6 Nina      578    2013-05-21 00:00:00 IT
7     7 Simon     633.   2013-07-30 00:00:00 Operations
8     8 Guru      722.   2014-06-17 00:00:00 Finance
>
> # exploring specific field “name” of read table
> name <- input$name
> name
[1] "Rick"     "Dan"       "Michelle" "Ryan"     "Gary"
[6] "Nina"     "Simon"     "Guru"
> salary <- input$salary
> salary
[1] 623.30 515.20 611.00 729.00  43.25 578.00 632.80 722.50
>
> # If we want to get the sum of salaries
> Sum <- sum(input$salary)
> Sum
[1] 4455.05
> # We can convert the loaded data to a data frame
> df <-data.frame(input)
> df
  id     name salary start_date          dept
1  1     Rick 623.30 2002-01-01 012          IT
2  2      Dan 515.20 2013-09-23    Operations
3  3 Michelle 611.00 2014-11-15            IT
4  4     Ryan 729.00 2014-05-11            HR
5  5     Gary  43.25 2015-03-27       Finance
6  6     Nina 578.00 2013-05-21            IT
7  7    Simon 632.80 2013-07-30    Operations
8  8     Guru 722.50 2014-06-17       Finance
>
```

```
> #exploring selective fields
> df$name
[1] "Rick"     "Dan"      "Michelle" "Ryan"     "Gary"
[6] "Nina"     "Simon"    "Guru"
>
> # Summarize data
> summary(df)
       id           name                  salary
 Min.   :1.00   Length:8           Min.   : 43.25
 1st Qu.:2.75   Class :character   1st Qu.:562.30
 Median :4.50   Mode  :character   Median :617.15
 Mean   :4.50                      Mean   :556.88
 3rd Qu.:6.25                      3rd Qu.:655.23
 Max.   :8.00                      Max.   :729.00
   start_date                      dept
 Min.   :2002-01-01 00:00:00   Length:8
 1st Qu.:2013-07-12 12:00:00   Class :character
 Median :2014-01-16 00:00:00   Mode  :character
 Mean   :2012-09-13 06:00:00
 3rd Qu.:2014-07-24 18:00:00
 Max.   :2015-03-27 00:00:00
```

We can also, import excel file

**Writing Excel Files:**

- **writexl:** This package complements readxl and excels (pun intended) at writing R data frames to new .xlsx Excel files. It offers a straightforward approach without requiring additional dependencies.

**Example:** using **write_exlsx()** function of “writexl” package to generate excel file in working directory (if not you have to specify the path)

```
> # Load the “writexl” package
> library(writexl)
>
> # Create a sample data frame
> df <- data.frame(
+   Name = c("Alice", "Bob", "Charlie"),
+   Age = c(25, 30, 35),
+   Salary = c(50000, 60000, 70000)
+ )
>
> # Write the data frame to an Excel file
> write_xlsx(df, path = "example.xlsx")
```