R - Lists

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

Creating a List

Following is an example to create a list containing strings, numbers, vectors and a logical values.

# Create a list containing strings, numbers, vectors and a logical

# values.

list\_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)

print(list\_data)

When we execute the above code, it produces the following result −

[[1]]

[1] "Red"

[[2]]

[1] "Green"

[[3]]

[1] 21 32 11

[[4]]

[1] TRUE

[[5]]

[1] 51.23

[[6]]

[1] 119.1

AD

Naming List Elements

The list elements can be given names and they can be accessed using these names.

# Create a list containing a vector, a matrix and a list.

list\_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),

list("green",12.3))

# Give names to the elements in the list.

names(list\_data) <- c("1st Quarter", "A\_Matrix", "A Inner list")

# Show the list.

print(list\_data)

When we execute the above code, it produces the following result −

$`1st\_Quarter`

[1] "Jan" "Feb" "Mar"

$A\_Matrix

[,1] [,2] [,3]

[1,] 3 5 -2

[2,] 9 1 8

$A\_Inner\_list

$A\_Inner\_list[[1]]

[1] "green"

$A\_Inner\_list[[2]]

[1] 12.3

Accessing List Elements

Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We continue to use the list in the above example −

# Create a list containing a vector, a matrix and a list.

list\_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),

list("green",12.3))

# Give names to the elements in the list.

names(list\_data) <- c("1st Quarter", "A\_Matrix", "A Inner list")

# Access the first element of the list.

print(list\_data[1])

# Access the thrid element. As it is also a list, all its elements will be printed.

print(list\_data[3])

# Access the list element using the name of the element.

print(list\_data$A\_Matrix)

When we execute the above code, it produces the following result −

$`1st\_Quarter`

[1] "Jan" "Feb" "Mar"

$A\_Inner\_list

$A\_Inner\_list[[1]]

[1] "green"

$A\_Inner\_list[[2]]

[1] 12.3

[,1] [,2] [,3]

[1,] 3 5 -2

[2,] 9 1 8

AD

Manipulating List Elements

We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

# Create a list containing a vector, a matrix and a list.

list\_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),

list("green",12.3))

# Give names to the elements in the list.

names(list\_data) <- c("1st Quarter", "A\_Matrix", "A Inner list")

# Add element at the end of the list.

list\_data[4] <- "New element"

print(list\_data[4])

# Remove the last element.

list\_data[4] <- NULL

# Print the 4th Element.

print(list\_data[4])

# Update the 3rd Element.

list\_data[3] <- "updated element"

print(list\_data[3])

When we execute the above code, it produces the following result −

[[1]]

[1] "New element"

$<NA>

NULL

$`A Inner list`

[1] "updated element"

Merging Lists

You can merge many lists into one list by placing all the lists inside one list() function.

# Create two lists.

list1 <- list(1,2,3)

list2 <- list("Sun","Mon","Tue")

# Merge the two lists.

merged.list <- c(list1,list2)

# Print the merged list.

print(merged.list)

When we execute the above code, it produces the following result −

[[1]]

[1] 1

[[2]]

[1] 2

[[3]]

[1] 3

[[4]]

[1] "Sun"

[[5]]

[1] "Mon"

[[6]]

[1] "Tue"

Converting List to Vector

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the **unlist()** function. It takes the list as input and produces a vector.

# Create lists.

list1 <- list(1:5)

print(list1)

list2 <-list(10:14)

print(list2)

# Convert the lists to vectors.

v1 <- unlist(list1)

v2 <- unlist(list2)

print(v1)

print(v2)

# Now add the vectors

result <- v1+v2

print(result)

When we execute the above code, it produces the following result −

[[1]]

[1] 1 2 3 4 5

[[1]]

[1] 10 11 12 13 14

[1] 1 2 3 4 5

[1] 10 11 12 13 14

[1] 11 13 15 17 19

# R - Data Frames

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

* The column names should be non-empty.
* The row names should be unique.
* The data stored in a data frame can be of numeric, factor or character type.
* Each column should contain same number of data items.

## Create Data Frame

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

salary = c(623.3,515.2,611.0,729.0,843.25),

start\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",

"2015-03-27")),

stringsAsFactors = FALSE

)

# Print the data frame.

print(emp.data)

When we execute the above code, it produces the following result −

emp\_id emp\_name salary start\_date

1 1 Rick 623.30 2012-01-01

2 2 Dan 515.20 2013-09-23

3 3 Michelle 611.00 2014-11-15

4 4 Ryan 729.00 2014-05-11

5 5 Gary 843.25 2015-03-27

AD

## Get the Structure of the Data Frame

The structure of the data frame can be seen by using **str()** function.

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

salary = c(623.3,515.2,611.0,729.0,843.25),

start\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",

"2015-03-27")),

stringsAsFactors = FALSE

)

# Get the structure of the data frame.

str(emp.data)

When we execute the above code, it produces the following result −

'data.frame': 5 obs. of 4 variables:

$ emp\_id : int 1 2 3 4 5

$ emp\_name : chr "Rick" "Dan" "Michelle" "Ryan" ...

$ salary : num 623 515 611 729 843

$ start\_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...

## Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying **summary()** function.

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

salary = c(623.3,515.2,611.0,729.0,843.25),

start\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",

"2015-03-27")),

stringsAsFactors = FALSE

)

# Print the summary.

print(summary(emp.data))

When we execute the above code, it produces the following result −

emp\_id emp\_name salary start\_date

Min. :1 Length:5 Min. :515.2 Min. :2012-01-01

1st Qu.:2 Class :character 1st Qu.:611.0 1st Qu.:2013-09-23

Median :3 Mode :character Median :623.3 Median :2014-05-11

Mean :3 Mean :664.4 Mean :2014-01-14

3rd Qu.:4 3rd Qu.:729.0 3rd Qu.:2014-11-15

Max. :5 Max. :843.2 Max. :2015-03-27

AD

## Extract Data from Data Frame

Extract specific column from a data frame using column name.

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

salary = c(623.3,515.2,611.0,729.0,843.25),

start\_date = as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11",

"2015-03-27")),

stringsAsFactors = FALSE

)

# Extract Specific columns.

result <- data.frame(emp.data$emp\_name,emp.data$salary)

print(result)

When we execute the above code, it produces the following result −

emp.data.emp\_name emp.data.salary

1 Rick 623.30

2 Dan 515.20

3 Michelle 611.00

4 Ryan 729.00

5 Gary 843.25

Extract the first two rows and then all columns

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

salary = c(623.3,515.2,611.0,729.0,843.25),

start\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",

"2015-03-27")),

stringsAsFactors = FALSE

)

# Extract first two rows.

result <- emp.data[1:2,]

print(result)

When we execute the above code, it produces the following result −

emp\_id emp\_name salary start\_date

1 1 Rick 623.3 2012-01-01

2 2 Dan 515.2 2013-09-23

Extract 3rd and 5th row with 2nd and 4th column

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

salary = c(623.3,515.2,611.0,729.0,843.25),

start\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",

"2015-03-27")),

stringsAsFactors = FALSE

)

# Extract 3rd and 5th row with 2nd and 4th column.

result <- emp.data[c(3,5),c(2,4)]

print(result)

When we execute the above code, it produces the following result −

emp\_name start\_date

3 Michelle 2014-11-15

5 Gary 2015-03-27

## Expand Data Frame

A data frame can be expanded by adding columns and rows.

### Add Column

Just add the column vector using a new column name.

# Create the data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

salary = c(623.3,515.2,611.0,729.0,843.25),

start\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",

"2015-03-27")),

stringsAsFactors = FALSE

)

# Add the "dept" coulmn.

emp.data$dept <- c("IT","Operations","IT","HR","Finance")

v <- emp.data

print(v)

When we execute the above code, it produces the following result −

emp\_id emp\_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

### Add Row

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the **rbind()** function.

In the example below we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

# Create the first data frame.

emp.data <- data.frame(

emp\_id = c (1:5),

emp\_name = c("Rick","Dan","Michelle","Ryan","Gary"),

salary = c(623.3,515.2,611.0,729.0,843.25),

start\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",

"2015-03-27")),

dept = c("IT","Operations","IT","HR","Finance"),

stringsAsFactors = FALSE

)

# Create the second data frame

emp.newdata <- data.frame(

emp\_id = c (6:8),

emp\_name = c("Rasmi","Pranab","Tusar"),

salary = c(578.0,722.5,632.8),

start\_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),

dept = c("IT","Operations","Fianance"),

stringsAsFactors = FALSE

)

# Bind the two data frames.

emp.finaldata <- rbind(emp.data,emp.newdata)

print(emp.finaldata)

When we execute the above code, it produces the following result −

emp\_id emp\_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 Rasmi 578.00 2013-05-21 IT

7 7 Pranab 722.50 2013-07-30 Operations

8 8 Tusar 632.80 2014-06-17 Fianance

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.](https://cran.r-project.org/web/packages/available_packages_by_name.html)

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

## Check Available R Packages

Get library locations containing R packages

[Live Demo](http://tpcg.io/MfTDys)

.libPaths()

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

[2] "C:/Program Files/R/R-3.2.2/library"

AD

## Get the list of all the packages installed

[Live Demo](http://tpcg.io/DEnHgl)

library()

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

Packages in library ‘C:/Program Files/R/R-3.2.2/library’:

base The R Base Package

boot Bootstrap Functions (Originally by Angelo Canty

for S)

class Functions for Classification

cluster "Finding Groups in Data": Cluster Analysis

Extended Rousseeuw et al.

codetools Code Analysis Tools for R

compiler The R Compiler Package

datasets The R Datasets Package

foreign Read Data Stored by 'Minitab', 'S', 'SAS',

'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...

graphics The R Graphics Package

grDevices The R Graphics Devices and Support for Colours

and Fonts

grid The Grid Graphics Package

KernSmooth Functions for Kernel Smoothing Supporting Wand

& Jones (1995)

lattice Trellis Graphics for R

MASS Support Functions and Datasets for Venables and

Ripley's MASS

Matrix Sparse and Dense Matrix Classes and Methods

methods Formal Methods and Classes

mgcv Mixed GAM Computation Vehicle with GCV/AIC/REML

Smoothness Estimation

nlme Linear and Nonlinear Mixed Effects Models

nnet Feed-Forward Neural Networks and Multinomial

Log-Linear Models

parallel Support for Parallel computation in R

rpart Recursive Partitioning and Regression Trees

spatial Functions for Kriging and Point Pattern

Analysis

splines Regression Spline Functions and Classes

stats The R Stats Package

stats4 Statistical Functions using S4 Classes

survival Survival Analysis

tcltk Tcl/Tk Interface

tools Tools for Package Development

utils The R Utils Package

Get all packages currently loaded in the R environment

[Live Demo](http://tpcg.io/VPFDZb)

search()

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

[1] ".GlobalEnv" "package:stats" "package:graphics"

[4] "package:grDevices" "package:utils" "package:datasets"

[7] "package:methods" "Autoloads" "package:base"

## Install a New Package

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

AD

## 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")

## Install package manually

Go to the link [R Packages](https://cran.r-project.org/web/packages/available_packages_by_name.html) 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")

# Install the package named "XML"

install.packages("E:/XML\_3.98-1.3.zip", repos = NULL, type = "source")

## Load Package to 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 −

library("package Name", lib.loc = "path to library")

# Load the package named "XML"

install.packages("E:/XML\_3.98-1.3.zip", repos = NULL, type = "sou

# R - CSV Files

[**Previous Page**](https://www.tutorialspoint.com/r/r_data_reshaping.htm)

[**Next Page**](https://www.tutorialspoint.com/r/r_excel_files.htm)

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 chapter 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. 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.

# Get and print current working directory.

print(getwd())

# Set current working directory.

setwd("/web/com")

# Get and print current working directory.

print(getwd())

When we execute the above code, it produces the following result −

[1] "/web/com/1441086124\_2016"

[1] "/web/com"

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

AD

## 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)

When we execute the above code, it produces the following result −

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 NA 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

AD

## 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.

data <- read.csv("input.csv")

print(is.data.frame(data))

print(ncol(data))

print(nrow(data))

When we execute the above code, it produces the following result −

[1] TRUE

[1] 5

[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("input.csv")

# Get the max salary from data frame.

sal <- max(data$salary)

print(sal)

When we execute the above code, it produces the following result −

[1] 843.25

### Get the details of the person with max salary

We can fetch rows meeting specific filter criteria similar to a SQL where clause.

# Create a data frame.

data <- read.csv("input.csv")

# Get the max salary from data frame.

sal <- max(data$salary)

# Get the person detail having max salary.

retval <- subset(data, salary == max(salary))

print(retval)

When we execute the above code, it produces the following result −

id name salary start\_date dept

5 NA Gary 843.25 2015-03-27 Finance

### Get all the people working in IT department

# Create a data frame.

data <- read.csv("input.csv")

retval <- subset( data, dept == "IT")

print(retval)

When we execute the above code, it produces the following result −

id name salary start\_date dept

1 1 Rick 623.3 2012-01-01 IT

3 3 Michelle 611.0 2014-11-15 IT

6 6 Nina 578.0 2013-05-21 IT

### Get the persons in IT department whose salary is greater than 600

# Create a data frame.

data <- read.csv("input.csv")

info <- subset(data, salary > 600 & dept == "IT")

print(info)

When we execute the above code, it produces the following result −

id name salary start\_date dept

1 1 Rick 623.3 2012-01-01 IT

3 3 Michelle 611.0 2014-11-15 IT

### Get the people who joined on or after 2014

# Create a data frame.

data <- read.csv("input.csv")

retval <- subset(data, as.Date(start\_date) > as.Date("2014-01-01"))

print(retval)

When we execute the above code, it produces the following result −

id name salary start\_date dept

3 3 Michelle 611.00 2014-11-15 IT

4 4 Ryan 729.00 2014-05-11 HR

5 NA Gary 843.25 2015-03-27 Finance

8 8 Guru 722.50 2014-06-17 Finance

## 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, as.Date(start\_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.

write.csv(retval,"output.csv")

newdata <- read.csv("output.csv")

print(newdata)

When we execute the above code, it produces the following result −

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 NA Gary 843.25 2015-03-27 Finance

4 8 8 Guru 722.50 2014-06-17 Finance

Here the column X comes from the data set newper. This can be dropped using additional parameters while writing the file.

# Create a data frame.

data <- read.csv("input.csv")

retval <- subset(data, as.Date(start\_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.

write.csv(retval,"output.csv", row.names = FALSE)

newdata <- read.csv("output.csv")

print(newdata)

When we execute the above code, it produces the following result −

id name salary start\_date dept

1 3 Michelle 611.00 2014-11-15 IT

2 4 Ryan 729.00 2014-05-11 HR

3 NA Gary 843.25 2015-03-27 Finance

4 8 Guru 722.50 2014-06-17 Finance

# R - Excel File

[**Previous Page**](https://www.tutorialspoint.com/r/r_csv_files.htm)

[**Next Page**](https://www.tutorialspoint.com/r/r_binary_files.htm)

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. Few such packages are - XLConnect, xlsx, gdata etc. We will be using xlsx package. R can also write into excel file using this package.

## Install xlsx Package

You can use the following command in the R console to install the "xlsx" package. It may ask to install some additional packages on which this package is dependent. Follow the same command with required package name to install the additional packages.

install.packages("xlsx")

AD

## Verify and Load the "xlsx" Package

Use the following command to verify and load the "xlsx" package.

# Verify the package is installed.

any(grepl("xlsx",installed.packages()))

# Load the library into R workspace.

library("xlsx")

When the script is run we get the following output.

[1] TRUE

Loading required package: rJava

Loading required package: methods

Loading required package: xlsxjars

## Input as xlsx File

Open Microsoft excel. Copy and paste the following data in the work sheet named as sheet1.

id name salary start\_date dept

1 Rick 623.3 1/1/2012 IT

2 Dan 515.2 9/23/2013 Operations

3 Michelle 611 11/15/2014 IT

4 Ryan 729 5/11/2014 HR

5 Gary 43.25 3/27/2015 Finance

6 Nina 578 5/21/2013 IT

7 Simon 632.8 7/30/2013 Operations

8 Guru 722.5 6/17/2014 Finance

Also copy and paste the following data to another worksheet and rename this worksheet to "city".

name city

Rick Seattle

Dan Tampa

Michelle Chicago

Ryan Seattle

Gary Houston

Nina Boston

Simon Mumbai

Guru Dallas

Save the Excel file as "input.xlsx". You should save it in the current working directory of the R workspace.

AD

## Reading the Excel File

The input.xlsx is read by using the **read.xlsx()** function as shown below. The result is stored as a data frame in the R environment.

# Read the first worksheet in the file input.xlsx.

data <- read.xlsx("input.xlsx", sheetIndex = 1)

print(data)

When we execute the above code, it produces the following result −

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 NA 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

## Data Set

A data set is a collection of data, often presented in a table.

There is a popular built-in data set in R called "**mtcars**" (Motor Trend Car Road Tests), which is retrieved from the 1974 Motor Trend US Magazine.

In the examples below (and for the next chapters), we will use the mtcars data set, for statistical purposes:

### Example

# Print the mtcars data set  
mtcars

Result:

mpg cyl disp hp drat wt qsec vs am gear carb

Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4

Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4

Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1

Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1

Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2

Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1

Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4

Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2

Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2

Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4

Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4

Merc 450SE 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3

Merc 450SL 17.3 8 275.8 180 3.07 3.730 17.60 0 0 3 3

Merc 450SLC 15.2 8 275.8 180 3.07 3.780 18.00 0 0 3 3

Cadillac Fleetwood 10.4 8 472.0 205 2.93 5.250 17.98 0 0 3 4

Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82 0 0 3 4

Chrysler Imperial 14.7 8 440.0 230 3.23 5.345 17.42 0 0 3 4

Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1

Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2

Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1

Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1

Dodge Challenger 15.5 8 318.0 150 2.76 3.520 16.87 0 0 3 2

AMC Javelin 15.2 8 304.0 150 3.15 3.435 17.30 0 0 3 2

Camaro Z28 13.3 8 350.0 245 3.73 3.840 15.41 0 0 3 4

Pontiac Firebird 19.2 8 400.0 175 3.08 3.845 17.05 0 0 3 2

Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1

Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2

Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2

Ford Pantera L 15.8 8 351.0 264 4.22 3.170 14.50 0 1 5 4

Ferrari Dino 19.7 6 145.0 175 3.62 2.770 15.50 0 1 5 6

Maserati Bora 15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8

Volvo 142E 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2

## Information About the Data Set

You can use the question mark (?) to get information about the mtcars data set:

### Example

# Use the question mark to get information about the data set  
  
?mtcars

Result:

|  |  |
| --- | --- |
| mtcars {datasets} | R Documentation |

## Motor Trend Car Road Tests

### Description

The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973-74 models).

### Usage

mtcars

### Format

A data frame with 32 observations on 11 (numeric) variables.

|  |  |  |
| --- | --- | --- |
| [, 1] | mpg | Miles/(US) gallon |
| [, 2] | cyl | Number of cylinders |
| [, 3] | disp | Displacement (cu.in.) |
| [, 4] | hp | Gross horsepower |
| [, 5] | drat | Rear axle ratio |
| [, 6] | wt | Weight (1000 lbs) |
| [, 7] | qsec | 1/4 mile time |
| [, 8] | vs | Engine (0 = V-shaped, 1 = straight) |
| [, 9] | am | Transmission (0 = automatic, 1 = manual) |
| [,10] | gear | Number of forward gears |
| [,11] | carb | Number of carburetors |

### Note

Henderson and Velleman (1981) comment in a footnote to Table 1: 'Hocking [original transcriber]'s noncrucial coding of the Mazda's rotary engine as a straight six-cylinder engine and the Porsche's flat engine as a V engine, as well as the inclusion of the diesel Mercedes 240D, have been retained to enable direct comparisons to be made with previous analyses.'

### Source

Henderson and Velleman (1981), Building multiple regression models interactively. Biometrics, **37**, 391-411.

### Examples

require(graphics)

pairs(mtcars, main = "mtcars data", gap = 1/4)

coplot(mpg ~ disp | as.factor(cyl), data = mtcars,

panel = panel.smooth, rows = 1)

## possibly more meaningful, e.g., for summary() or bivariate plots:

mtcars2 <- within(mtcars, {

vs <- factor(vs, labels = c("V", "S"))

am <- factor(am, labels = c("automatic", "manual"))

cyl <- ordered(cyl)

gear <- ordered(gear)

carb <- ordered(carb)

})

summary(mtcars2)

## Get Information

Use the dim() function to find the dimensions of the data set, and the names() function to view the names of the variables:

### Example

Data\_Cars <- mtcars # create a variable of the mtcars data set for better organization  
  
# Use dim() to find the dimension of the data set  
dim(Data\_Cars)  
  
# Use names() to find the names of the variables from the data set  
names(Data\_Cars)

Result:

[1] 32 11

[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"

[11] "carb"

Use the rownames() function to get the name of each row in the first column, which is the name of each car:

### Example

Data\_Cars <- mtcars  
  
rownames(Data\_Cars)

Result:

[1] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710"

[4] "Hornet 4 Drive" "Hornet Sportabout" "Valiant"

[7] "Duster 360" "Merc 240D" "Merc 230"

[10] "Merc 280" "Merc 280C" "Merc 450SE"

[13] "Merc 450SL" "Merc 450SLC" "Cadillac Fleetwood"

[16] "Lincoln Continental" "Chrysler Imperial" "Fiat 128"

[19] "Honda Civic" "Toyota Corolla" "Toyota Corona"

[22] "Dodge Challenger" "AMC Javelin" "Camaro Z28"

[25] "Pontiac Firebird" "Fiat X1-9" "Porsche 914-2"

[28] "Lotus Europa" "Ford Pantera L" "Ferrari Dino"

[31] "Maserati Bora" "Volvo 142E"

From the examples above, we have found out that the data set has **32** observations (Mazda RX4, Mazda RX4 Wag, Datsun 710, etc) and **11** variables (mpg, cyl, disp, etc).

A variable is defined as something that can be measured or counted.

Here is a brief explanation of the variables from the mtcars data set:

|  |  |
| --- | --- |
| **Variable Name** | **Description** |
| mpg | Miles/(US) Gallon |
| cyl | Number of cylinders |
| disp | Displacement |
| hp | Gross horsepower |
| drat | Rear axle ratio |
| wt | Weight (1000 lbs) |
| qsec | 1/4 mile time |
| vs | Engine (0 = V-shaped, 1 = straight) |
| am | Transmission (0 = automatic, 1 = manual) |
| gear | Number of forward gears |
| carb | Number of carburetors |

## Print Variable Values

If you want to print all values that belong to a variable, access the data frame by using the $ sign, and the name of the variable (for example cyl (cylinders)):

### Example

Data\_Cars <- mtcars  
  
Data\_Cars$cyl

Result:

[1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4

## Sort Variable Values

To sort the values, use the sort() function:

### Example

Data\_Cars <- mtcars  
  
sort(Data\_Cars$cyl)

Result:

[1] 4 4 4 4 4 4 4 4 4 4 4 6 6 6 6 6 6 6 8 8 8 8 8 8 8 8 8 8 8 8 8 8

From the examples above, we see that most cars have 4 and 8 cylinders.

## Analyzing the Data

Now that we have some information about the data set, we can start to analyze it with some statistical numbers.

For example, we can use the summary() function to get a statistical summary of the data:

### Example

Data\_Cars <- mtcars  
  
summary(Data\_Cars)

Do not worry if you do not understand the output numbers. You will master them shortly.

The summary() function returns six statistical numbers for each variable:

* Min
* First quantile (percentile)
* Median
* Mean
* Third quantile (percentile)
* Max

We will cover all of them, along with other statistical numbers in the next chapters.

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.

Factors are created using the **factor ()** function by taking a vector as input.

## Example

[Live Demo](http://tpcg.io/Q04XEe)

# Create a vector as input.

data <- c("East","West","East","North","North","East","West","West","West","East","North")

print(data)

print(is.factor(data))

# Apply the factor function.

factor\_data <- factor(data)

print(factor\_data)

print(is.factor(factor\_data))

When we execute the above code, it produces the following result −

[1] "East" "West" "East" "North" "North" "East" "West" "West" "West" "East" "North"

[1] FALSE

[1] East West East North North East West West West East North

Levels: East North West

[1] TRUE

AD

## Factors in Data Frame

On creating any data frame with a column of text data, R treats the text column as categorical data and creates factors on it.

[Live Demo](http://tpcg.io/gK9Gr0)

# Create the vectors for data frame.

height <- c(132,151,162,139,166,147,122)

weight <- c(48,49,66,53,67,52,40)

gender <- c("male","male","female","female","male","female","male")

# Create the data frame.

input\_data <- data.frame(height,weight,gender)

print(input\_data)

# Test if the gender column is a factor.

print(is.factor(input\_data$gender))

# Print the gender column so see the levels.

print(input\_data$gender)

When we execute the above code, it produces the following result −

height weight gender

1 132 48 male

2 151 49 male

3 162 66 female

4 139 53 female

5 166 67 male

6 147 52 female

7 122 40 male

[1] TRUE

[1] male male female female male female male

Levels: female male

## Changing the Order of Levels

The order of the levels in a factor can be changed by applying the factor function again with new order of the levels.

[Live Demo](http://tpcg.io/vLA7bk)

data <- c("East","West","East","North","North","East","West",

"West","West","East","North")

# Create the factors

factor\_data <- factor(data)

print(factor\_data)

# Apply the factor function with required order of the level.

new\_order\_data <- factor(factor\_data,levels = c("East","West","North"))

print(new\_order\_data)

When we execute the above code, it produces the following result −

[1] East West East North North East West West West East North

Levels: East North West

[1] East West East North North East West West West East North

Levels: East West North

AD

## Generating Factor Levels

We can generate factor levels by using the **gl()** function. It takes two integers as input which indicates how many levels and how many times each level.

### Syntax

gl(n, k, labels)

Following is the description of the parameters used −

* **n** is a integer giving the number of levels.
* **k** is a integer giving the number of replications.
* **labels** is a vector of labels for the resulting factor levels.

### Example

[Live Demo](http://tpcg.io/IjruEj)

v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))

print(v)

When we execute the above code, it produces the following result −

Tampa Tampa Tampa Tampa Seattle Seattle Seattle Seattle Boston

[10] Boston Boston Boston

Levels: Tampa Seattle Boston

