R Programming

Learn the fundamentals of data analysis with R.

Course Modules

- ✓ Introduction
- ✓ Elementary Programming
- ✓ **Working With Data**
- ✓ Selection Statements
- ✓ Loops
- ✓ Functions
- ✓ Debugging
- ✓ Unit Testing

Working With Data

- ✓ Data Types
- ✓ Data Structures
- ✓ Data Creation
- ✓ Data Info
- ✓ Data Subsetting
- ✓ Comparing R Objects

- ✓ **Importing Data**
- ✓ Exporting Data
- ✓ Data Transformation
- ✓ Numeric Functions
- ✓ String Functions
- ✓ Mathematical Functio

Importing Data In R

Objectives

In this module, we will learn to:

- Read data from the console
- Read data from files
- Import data from
    - Text/Excel/CSV files
    - Stata/SAS/SPSS files
- Load .Rdata files
- Source R scripts

Read Data From Console

In this section, we will learn to read data from the console interactively and store them
R objects using the following functions:

- ✓ scan
- ✓ readline

scan() (1/4)

Description:
`scan()` allows user to input data from console or from a file and stores the input in a vector or list.

Syntax:
```
x <- scan()                       # stores input as vector
x <- scan("", what = integer())  # stores input as integer
x <- scan("", what = list())     # stores input as list
```

Returns:
A vector or list of the input data.

Documentation
```
help(scan)
```

scan() (2/4)

Examples

```
> # example 1
> x <- scan()
1: 1
2: 2
3: 3
4:
Read 3 items

# to end input, do not enter anything.

> x
[1] 1 2 3

> typeof(x)
[1] "double"

# if numbers are entered, they will be stored as double. In the next example, we will learn
how to store numbers as integers.
```

scan() (3/4)

Examples

```
> # example 2
> x <- scan("", what = integer())
1: 1
2: 2
3: 3
4:
Read 3 items

# mention the data type in the what argument to store the data in the preferred mode.

> x
[1] 1 2 3

> typeof(x)
[1] "integer"
```

scan() (4/4)

Examples

```
> # example 3
> x <- scan("", what = list(name = "", age = 0))
1: Jovial 28
2: Manual 27
3: Funnel 25
4: Tunnel 29
5:
Read 4 records

# suppose we want the user to enter multiple attributes and store the input in a list. Use
list in the what argument with the names for the attributes.

> x
$name
[1] "Jovial" "Manual" "Funnel" "Tunnel"

$age
[1] 28 27 25 29
```

`readline()` (1/3)

Description:
`readline()` prompts the user for an input and stores the input as a character vector.

Syntax:
`readline(prompt = "")`

Returns:
A character vector of the input data.

Documentation
`help(readline)`

readline() (2/3)

Examples

```
> # example 1
> x <- readline(prompt = "Enter your name: ")
Enter your name: Jovial

> x
[1] "Jovial"

> class(x)
[1] "character"


# input is stored as character type. It has to be converted to other data types as necessary.
In the next example, we will input a number and then store it as an integer.
```

readline() (3/3)

Examples

```
> # example 2
> x <- readline(prompt = "Enter your age: ")
Enter your age: 28

> x
[1] "28"

> class(x)
[1] "character"

> x <- as.integer(x)

> x
[1] 28
```

Read Data From Files

In this section, we will learn to read data from files using the following functions:

- ✓ scan
- ✓ readLines

scan() (1/5)

Description:

`scan()` allows user to input data from console or from a file and stores the input in a ve
or list.

Syntax:

```
scan(file = "", what = double(), nmax = -1L, n = -1L, sep = "",
    quote = if (identical(sep, "\n")) "" else "'\"", dec = ".",
    skip = 0L, nlines = 0L, na.strings = "NA", flush = FALSE,
    fill = FALSE, strip.white = FALSE, quiet = FALSE, blank.lines.skip = TRUE,
    multi.line = TRUE, comment.char = "", allowEscapes = FALSE,
    fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

Returns:

A vector or list of the input data.

Documentation

`help(scan)`

scan() (2/5)

Arguments:

file: name of the file from which the data must be read.
what: mode in which data must be stored.
nmax: maximum number of data values or lines to be read from a file.
n: maximum number of data values to be read.
sep: delimiter
skip: number of lines to be skipped before reading reading data from a fil
nlines: maximum number of lines to be read from a file.
quiet: how many items have been read.
blank.lines.skip: if blank lines must be skipped.
multi.line: whether all lines must appear in one line or multi-line.

scan() (3/5)

Examples

```
> # example 1
> scan("words.txt", what = character(), skip = 2, nlines = 2,
+ quiet = TRUE)
 [1] "Morbi"       "consequat"   "commodo"     "orci"        "ut"          "volutpat."
 [7] "Sed"         "accumsan"    "eleifend"    "egestas."    "Nullam"      "ac"
[13] "posuere"     "eros."       "Donec"       "rutrum"      "gravida"     "felis,"
[19] "quis"        "fermentum"   "orci."       "Pellentesque" "purus"       "lacus,"
[25] "tincidunt"   "eget"        "enim"        "ut,"         "facilisis"   "rutrum"
[31] "odio."


# read two lines from the file “words.txt” as type character after skipping the first two
lines and do not print the number of lines read on the console.
```

scan() (4/5)

Examples

```
> # example 2
> scan("words.txt", what = list("", ""), skip = 2, nlines = 2, sep = " ",
+      quiet = TRUE)
[[1]]
 [1] "Morbi"     "commodo"   "ut"         "Sed"        "eleifend"  "Nullam"    "posuere"
 [8] "Donec"     "gravida"   "quis"       "orci."      "purus"     "tincidunt" "enim"
[15] "facilisis" "odio."

[[2]]
 [1] "consequat"    "orci"          "volutpat."    "accumsan"     "egestas."     "ac"
 [7] "eros."        "rutrum"        "felis,"       "fermentum"    "Pellentesque" "lacus,"
[13] "eget"         "ut,"           "rutrum"       ""


# read two lines from the file "words.txt" as a list, after skipping the first two lines and
do not print the number of lines read on the console.
```

scan() (5/5)

Examples

```
> # example 3
> scan("words.txt", what = list("", "", ""), skip = 2, nlines = 3, sep = " ",
+       quiet = TRUE)
[[1]]
 [1] "Morbi"     "orci"      "Sed"       "egestas."  "posuere"   "Donec"     "felis,"
 [8] "orci."     "lacus,"    "enim"      "rutrum"    "Donec"     "tincidunt" "eu,"
[15] "tortor."   "turpis"    "bibendum"

[[2]]
 [1] "consequat"    "ut"           "accumsan"     "Nullam"       "eros."        "rutrum"
 [7] "quis"         "Pellentesque" "tincidunt"    "ut,"          "odio."        "mi"
[13] "a"            "euismod"      "In"           "vel"          "posuere."

[[3]]
 [1] "commodo"      "volutpat."    "eleifend"     "ac"           ""             "gravida"
 [7] "fermentum"    "purus"        "eget"         "facilisis"    ""             "urna,"
[13] "sollicitudin" "non"          "dignissim"    "lorem"        ""

# read three lines from the file “words.txt” as a list, after skipping the first two lines
and do not print the number of lines read on the console.
```

`readLines() (1/3)`

Description:
`readLines()` allows user to input data from console or from a file and stores the input
vector or list.

Syntax:
`readLines(file_name)`

Returns:
A vector of the input data.

Documentation
`help(readLines)`

readLines() (2/3)

Examples

```
> # example 1
> readLines("words.txt")
 [1] "Lorem ipsum dolor sit amet, consectetur adipiscing elit. In sodales nulla quis interdum
dictum. "
 [2] "Maecenas molestie suscipit libero lobortis ornare. Nam quam magna, tincidunt id
vulputate nec, elementum ac lorem. "
 [3] "Morbi consequat commodo orci ut volutpat. Sed accumsan eleifend egestas. Nullam ac
posuere eros. "
.    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .
.    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .
.    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .
[15] "Vivamus pulvinar consectetur tellus, quis mollis libero lobortis at. "
[16] "Quisque tincidunt purus fermentum augue auctor ultricies."
[17] ""


# reads all the lines from the file
```

readLines() (3/3)

Examples

```
> # example 2
> readLines("words.txt", n = 5)
[1] "Lorem ipsum dolor sit amet, consectetur adipiscing elit. In sodales nulla quis interdum
dictum. "
[2] "Maecenas molestie suscipit libero lobortis ornare. Nam quam magna, tincidunt id
vulputate nec, elementum ac lorem. "
[3] "Morbi consequat commodo orci ut volutpat. Sed accumsan eleifend egestas. Nullam ac
posuere eros. "
[4] "Donec rutrum gravida felis, quis fermentum orci. Pellentesque purus lacus, tincidunt
eget enim ut, facilisis rutrum odio. "
[5] "Donec mi urna, tincidunt a sollicitudin eu, euismod non tortor. In dignissim turpis vel
lorem bibendum posuere. "



# reads the first 5 lines from the file.
```

Import Data Files

In this section, we will learn to import the following data files:

- ✓ Text file
- ✓ Excel/CSV file
- ✓ Stata file
- ✓ SAS file
- ✓ SPSS file

Importing Text File

Description:
`read.table()` reads a file in table format and creates a data frame from it.

Syntax:
```
read.table(file_name, header, sep)
```

Returns:
A data frame.

Documentation
```
help(read.table)
```

read.table()

Examples

```
> # example 1
> # read data from a semicolon delimited file and retain the column names
> text_data <- read.table("data.txt", header = TRUE, sep = ";")

> # example 2
> # read data from a comma delimited file and retain the column names
> text_data1 <- read.table("data1.txt", header = TRUE, sep = ",")

> # example 3
> # read data from a tab delimited file and retain the column names
> text_data2 <- read.table("data2.txt", header = TRUE, sep = "\t")
```

`read.csv()`

Description:

`read.csv()` reads a CSV file in table format and creates a data frame from it.

Syntax:

```
read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".",
    fill = TRUE, comment.char = "", ...)
```

Returns:

A data frame.

Documentation

```
help(read.csv)
```

read.csv()

Examples

```
> # example 1
> # read data from a CSV file and retain the column names
> data_csv <- read.csv("data.csv", header = TRUE)

> # example 2
> # read data from a CSV file without the column names
> data_csv <- read.csv("data.csv", header = FALSE)

> # example 3
> # read data from a CSV file and retain the column names and add blank fields
> # when rows are of unequal length
> data_csv <- read.csv("data.csv", header = TRUE, fill = TRUE)
```

`read.xls()`

Description:
`read.xls()` reads an excel file in table format and creates a data frame from it. You ne
to install the gdata package in order to use the `read.xls()` function.

Syntax:
```
read.xls(file, sheet)
```

Returns:
A data frame.

Documentation:
```
library(gdata)
help(read.xls)
```

read.xls()

Examples

```
> # example 1
> # read data from a excel file
> data_xls <- read.xls("data.csv")

> # example 2
> # read data from a particular sheet in a excel file
> data_xls <- read.xls("data.csv", sheet = 1)
```

Stata File

Description

`read.dta()` reads a Stata binary file into a data frame.

Package

Install the `foreign` package to import stata files.

Syntax

```
read.csv(file, convert.dates = TRUE, convert.factors = TRUE, missing.t
= FALSE, convert.underscore = FALSE, warn.missing.labels = TRUE)
```

Returns

A data frame.

Documentation

```
help(read.dta)
```

read.dta()

Examples

```
> # example 1
> install.packages("foreign")
> library(foreign)
> data_stata <- read.dta("auto.dta")
```

SPSS File

Description

`read.spss()` reads a SPSS file into a data frame.

Package

Install the `foreign` package to import stata files.

Syntax

```
read.spss(file, use.value.labels = TRUE, to.data.frame = FALSE, max.value.label
Inf, trim.factor.names = FALSE, trim_values = TRUE, reencode = NA, use.missings
to.data.frame)
```

Returns

A data frame.

Documentation

`help(read.spss)`

read.spss()

Examples

```
> # example 1
> install.packages("foreign")
> library(foreign)
> data_spss <- read.spss("binary.sav")
```

SAS File

Description

`read.sas7bdat()` reads SAS files in the sas7bdat data format into a dataframe.

Package

Install the `sas7bdat` package to import stata files.

Syntax:

```
read.sas7bdat(file, debug=FALSE)
```

Returns:

A data frame.

Documentation

```
help(read.sas7bdat)
```

read.sas7bdat()

Examples

```
> # example 1
> install.packages("sas7bdat")
> library(sas7bdat)
> data_sas <- read.sas7bdat("crime.sas7bdat")
```

`load()`

Description
`load()` reloads saved datasets and workspaces. Datasets and workspaces have the extension `.RData`

Syntax:
`load(file)`

Returns:
R object or workspace.

Documentation
`help(load)`

Example
```
> load("x.RData")
```

`source()`

Description
`source()` reads R codes from a file and makes those codes available in the current session. R scripts have the extension `.R`

Syntax:
`source(file_name, file_path)`

Returns
Codes from a R file.

Documentation
`help(source)`

Example

```
> source("functions.R")
```