

How can I input data into R?

Authored by
stats writer

June 30, 2024

RECOMMENDED CITATION

stats writer (2024). *How can I input data into R?*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=161309>

Inputting data into R is a crucial step in data analysis and statistical computing. It involves the process of entering raw data from various sources such as Excel spreadsheets, text files, or databases, into the R software. There are several ways to input data into R, including using the built-in functions, importing data from external files, or manually entering data through the console. The most efficient method depends on the type and size of the data, as well as the user's familiarity with R. Proper data input is essential for accurate analysis and obtaining meaningful insights from the data.

How to Input data into R | R FAQ

Importing formatted data files using the functions in the foreign package

The foreign package contains functions that will allow you to import data files from some of the most commonly used statistical software packages such as SAS, Stata and SPSS.

To download the foreign package from the CRAN website from within R, click on "Packages" and then "Install package(s) from CRAN". You will then need to load the package, and you can use the help function.

```
library(foreign)
```

```
help(package=foreign)
```

The package contains the following functions:

data.restore Read an S3 Binary File

lookup.xport Lookup Information on a SAS XPORT Format Library

read.dbf Read a DBF File

read.dta Read Stata binary files

read.epiinfo Read Epi Info data files

read.mtp Read a Minitab Portable Worksheet

read.octave Read Octave Text Data Files

read.spss Read an SPSS data file

read.ssd Obtain a Data Frame from a SAS Permanent Dataset, via read.xport

read.systat Obtain a Data Frame from a Systat File

read.xport Read a SAS XPORT Format Library

write.dbf Write a DBF File

write.dta Write Files in Stata Binary Format

write.foreign Write text files and code to read them

write.foreign Write text files and code to read them

To download the package:

```
install.packages("foreign")
```

To view the functions in the package:

```
library(help=foreign)
```

To view the help file for a specific function, for example

the function read.dta:

?read.dta

Note that this is an outdated package and will not work for Stata 12 or higher data files. There is a package called "readstata13" available to read Stata 13 files.

Here are examples of importing a Stata data file called test.dta.

```
test.stata                                     <-  
read.dta("https://stats.idre.ucla.edu/stat/data/test.dta")  
print(test.stata)
```

make model mpg weight price

1 AMC Concord 22 2930 4099

2 AMC Pacer 17 3350 4749

3 AMC Spirit 22 2640 3799

4 Buick Century 20 3250 4816

5 Buick Electra 15 4080 7827

Importing free formatted (delimited) data files using the read.table function

The read.table function is very useful when reading in

ASCII files that contain rectangular data.

When the file contains the variable names in the first line of data the option header should be set to TRUE. The default delimiter is blank space, other delimiters must be specified by using the sep option and setting it equal to the delimiter in quotes (i.e., sep=";" for the semicolon delimited data file).

Here are some examples of data with different types of delimiters. We will start by looking at a typical bread and butter type of data file namely a space delimited ASCII file called test.txt.

```
test.txt <-  
read.table("https://stats.idre.ucla.edu/wp-content/uploads/2016/02/test.txt", header=T)  
print(test.txt)
```

```
make model mpg weight price  
1 AMC Concord 22 2930 4099  
2 AMC Pacer 17 3350 4749  
3 AMC Spirit 22 2640 3799  
4 Buick Century 20 3250 4816
```

5 Buick Electra 15 4080 7827

Another very common type of file is the comma delimited file. The file test-1.csv has been saved out of Excel as a comma delimited file.

This file can be read in by the read.table function by using the sep option, but it can also be read in by the read.csv function which was written specifically for comma delimited files.

```
test.csv <-  
read.csv("https://stats.idre.ucla.edu/wp-content/uploads  
/2016/02/test-1.csv", header=T)  
print(test.csv)
```

make model mpg weight price

1 AMC Concord 22 2930 4099

2 AMC Pacer 17 3350 4749

3 AMC Spirit 22 2640 3799

4 Buick Century 20 3250 4816

5 Buick Electra 15 4080 7827

```
test.csv1 <-
```

```
read.table("https://stats.idre.ucla.edu/wp-content/uploads/2016/02/test-1.csv", header=T, sep=",")  
print(test.csv1)
```

```
make model mpg weight price  
1 AMC Concord 22 2930 4099  
2 AMC Pacer 17 3350 4749  
3 AMC Spirit 22 2640 3799  
4 Buick Century 20 3250 4816  
5 Buick Electra 15 4080 7827
```

It is, of course, also possible to use the `read.table` function for reading in files with other delimiters. In the data called `testsemicolon.txt` has semicolon delimiters and the dataset test called `testz.txt` uses the letter `z` as a delimiter, both of which are acceptable delimiters in R.

```
test.semi <-  
read.table("https://stats.idre.ucla.edu/wp-content/uploads/2016/02/testsemicolon.txt", header=T, sep=";")  
print(test.semi)
```

```
make model mpg weight price  
1 AMC Concord 22 2930 4099
```

```
2 AMC Pacer 17 3350 4749
3 AMC Spirit 22 2640 3799
4 Buick Century 20 3250 4816
5 Buick Electra 15 4080 7827
```

```
test.z <-
read.table("https://stats.idre.ucla.edu/wp-content/uploads/2016/02/testz.txt", header=T, sep="z")
print(test.z)
```

```
make model mpg weight price
1 AMC Concord 22 2930 4099
2 AMC Pacer 17 3350 4749
3 AMC Spirit 22 2640 3799
4 Buick Century 20 3250 4816
5 Buick Electra 15 4080 7827
```

Importing data files using the scan function

The scan function is an extremely flexible tool for importing data. It can be used to read in almost any type of data, numeric, character or complex and it can be used for fixed or free formatted files. Moreover, by using the scan function it is

possible to input data directly from the console. The scan function reads the fields of data in the file as specified by the what option with the default being numeric. If the what option is specified to be what=character() or what=" " then all the fields will be read as strings. If the data is a mix of numeric, string or complex data then a list can be used in the what option. The default separator for the scan function is any white space (single space, tab, or new line). However, unlike the read.table function which returns a data frame, the scan function returns a list or a vector. This makes the scan function less useful for inputting "rectangular" data such as the car data set that was seen in the previous examples.

In the following examples we input first numeric data and then string data directly from the console; then we input the text file, scan.txt, where the first variable is a string variable and the

second variable is numeric.

#inputting data directly from the console

```
x <- scan()
```

```
1: 3 5 6 9
```

```
5: 2 5 6
```

```
8:
```

```
Read 7 items
```

```
x
```

```
3 5 6 9 2 5 6
```

inputting string data directly from the console

```
name.x <- scan(, what="")
```

```
1: bobby
```

```
2: kate dave
```

```
4: mia
```

```
5:
```

```
Read 4 items
```

```
name.x
```

```
"bobby" "kate" "dave" "mia"
```

inputting a text file and outputting a list

```
x
```

```
<-
```

```
scan("https://stats.idre.ucla.edu/wp-content/uploads/2016/02/scan.txt", what=list(age=0, name=""))
```

Read 4 records

```
x
```

```
$age
```

```
12 24 35 20
```

```
$name
```

```
"bobby" "kate" "david" "michael"
```

```
# using the same text file and saving only the names as  
a vector
```

```
x
```

```
<-
```

```
scan("https://stats.idre.ucla.edu/wp-content/uploads/20  
16/02/scan.txt", what=list(NULL, name=character()))
```

Read 4 records

```
x <- xx
```

```
$name
```

```
"bobby" "kate" "david" "michael"
```

```
is.vector(x)
```

```
TRUE
```

Importing Fixed Format Files Using the read.fwf Function

For fixed format files the variables names are often in a

separate file from the data.

In this example the variable names are in a file called names.txt and the

data are in a file called testfixed.txt. This is especially convenient when the fixed format file is very large and has many variables; then it becomes

rather impractical to type in all the variable names. In this

situation the width option is used to specify the width of each

variable and the col.name option specifies the file containing the

variable names. So, first we read in the file for the names using the

scan function. We specify that file contains character values by

setting the what option to equal character(). By using the

col.names option in the read.fwf function names will supply

the variables names.

```
names
```

```
<-
```

```
scan("https://stats.idre.ucla.edu/wp-content/uploads/20
```

```
16/02/names.txt", what=character() )
```

```
Read 5 items
```

```
print(names)
```

```
"model" "make" "mph" "weight" "price"
```

```
test.fixed
```

```
<-
```

```
read.fwf("https://stats.idre.ucla.edu/wp-content/uploads/2016/02/testfixed.txt", col.names=names, width = c(5, 7, 2, 4, 4))
```

```
print(test.fixed)
```

```
model make mph weight price
```

```
1 AMC Concord 22 2930 4099
```

```
2 AMC Pacer 17 3350 4749
```

```
3 AMC Spirit 22 2640 3799
```

```
4 Buick Century 20 3250 4816
```

```
5 Buick Electra 15 4080 7827
```

Note that if there is a warning about an incomplete final line, it means you should add a carriage return (press Enter) after the last line so that the function will know when to stop reading the file.

Exporting files using the write.table function

The write.table function outputs data files. The first argument specifies which data frame in R is to be exported. The next argument specifies the file to be created.

The default separator is a blank space but any separator can be specified in the sep option.

The default value for both the row.names and col.names options is TRUE. In the example we specify that we do not wish to include row names. The default setting for the quote option is to include quotes around all the character values, i.e. around values in string variables and around the column names. As we have shown in the example it is very common not to want the quotes when creating a text file.

```
# using the test.csv data frame to write a text file with  
no row names
```

```
# and without quotes around the character values (both  
column names and string variables)
```

```
write.table(test.csv, "c:/temp/test1.txt", row.names=F,  
quote=F)
```