1. How to Import SAS XPORT files into R with the foreign package?

### Ans. Read SAS Files into R

For those R users that also want to import SAS file into R, it’s very simple! For starters, install the sas7bdat package. Load it, and then invoke the read.sas7bdat() function contained within the package and you are good to go!

# Activate the `sas7bdat` library

library(sas7bdat)

# Read in the SAS data

mySASData <- read.sas7bdat("example.sas7bdat")

Does this function interest you and do you want to know more? Visit the [R documentation](http://www.rdocumentation.org/packages/sas7bdat/functions/read.sas7bdat) page.

Note that you can also use the foreign library to load in SAS data in R. In such cases, you’ll start from a SAS Permanent Dataset or a SAS XPORT Format Library with the read.ssd() and read.xport() functions, respectively.

1. How to Import SAS Files into R with the Haven package?

Ans. Source: [R/haven.R](https://github.com/tidyverse/haven/blob/master/R/haven.R)

Reading supports both sas7bdat files and the accompanying sas7bdat files that SAS uses to record value labels. Writing value labels is not currently supported.

read\_sas(data\_file, catalog\_file = NULL, encoding = NULL,

catalog\_encoding = encoding, cols\_only = NULL)

write\_sas(data, path)

## Arguments

|  |  |
| --- | --- |
| **data\_file, catalog\_file** | Path to data and catalog files. The files are processed with [readr::datasource()](http://www.rdocumentation.org/packages/readr/topics/datasource). |
| **encoding, catalog\_encoding** | The character encoding used for the data\_file and catalog\_encoding respectively. A value of NULL uses the encoding specified in the file; use this argument to override it if it is incorrect. |
| **cols\_only** | A character vector giving an experimental way to read in only specified columns. |
| **data** | Data frame to write. |
| **path** | Path to file where the data will be written. |

## Value

A tibble, data frame variant with nice defaults.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

write\_sas() returns the input data invisibly.

## Examples

path <- system.file("examples", "iris.sas7bdat", package = "haven")

read\_sas(path)

#> # A tibble: 150 x 5

#> Sepal\_Length Sepal\_Width Petal\_Length Petal\_Width Species

#> <dbl> <dbl> <dbl> <dbl> <chr>

#> 1 5.1 3.5 1.4 0.2 setosa

#> 2 4.9 3 1.4 0.2 setosa

#> 3 4.7 3.2 1.3 0.2 setosa

#> 4 4.6 3.1 1.5 0.2 setosa

#> 5 5 3.6 1.4 0.2 setosa

#> 6 5.4 3.9 1.7 0.4 setosa

#> 7 4.6 3.4 1.4 0.3 setosa

#> 8 5 3.4 1.5 0.2 setosa

#> 9 4.4 2.9 1.4 0.2 setosa

#> 10 4.9 3.1 1.5 0.1 setosa

#> # ... with 140 more rows

1. How to read Weka Attribute-Relation File Format (ARFF) files in R?

## Ans. Usage

|  |  |
| --- | --- |
| 1 | [read.arff](https://rdrr.io/cran/RWeka/man/read.arff.html)([file](https://rdrr.io/r/base/connections.html)) |

## Arguments

|  |  |
| --- | --- |
| **file** | a character string with the name of the ARFF file to read from, or a connection which will be opened if necessary, and if so closed at the end of the function call. |

## Value

A data frame containing the data from the ARFF file.

## References

Attribute-Relation File Format <https://weka.wikispaces.com/ARFF>

## See Also

[write.arff](https://rdrr.io/cran/RWeka/man/write.arff.html)

## Examples

|  |  |
| --- | --- |
| 1  2 | [read.arff](https://rdrr.io/cran/RWeka/man/read.arff.html)([system.file](https://rdrr.io/r/base/system.file.html)("arff", "contact-lenses.arff",  package = "RWeka")) |

1. How to read a heavy csv/tsv file using readr package?

## Ans. Overview

The goal of readr is to provide a fast and friendly way to read rectangular data (like csv, tsv, and fwf). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes. If you are new to readr, the best place to start is the [data import chapter](http://r4ds.had.co.nz/data-import.html) in R for data science.

## Installation

# The easiest way to get readr is to install the whole tidyverse:

install.packages("tidyverse")

# Alternatively, install just readr:

install.packages("readr")

# Or the the development version from GitHub:

# install.packages("devtools")

devtools::install\_github("tidyverse/readr")

## Usage

readr is part of the core tidyverse, so load it with:

library(tidyverse)

#> ── Attaching packages ────────────────────────────────── tidyverse 1.2.1 ──

#> ✔ ggplot2 2.2.1 ✔ purrr 0.2.4

#> ✔ tibble 1.3.4 ✔ dplyr 0.7.4

#> ✔ tidyr 0.7.2 ✔ stringr 1.2.0

#> ✔ readr 1.1.1.9000 ✔ forcats 0.2.0

#> ── Conflicts ───────────────────────────────────── tidyverse\_conflicts() ──

#> ✖ dplyr::filter() masks stats::filter()

#> ✖ dplyr::lag() masks stats::lag()

To accurately read a rectangular dataset with readr you combine two pieces: a function that parses the overall file, and a column specification. The column specification describes how each column should be converted from a character vector to the most appropriate data type, and in most cases it's not necessary because readr will guess it for you automatically.

readr supports seven file formats with seven read\_ functions:

* read\_csv(): comma separated (CSV) files
* read\_tsv(): tab separated files
* read\_delim(): general delimited files
* read\_fwf(): fixed width files
* read\_table(): tabular files where colums are separated by white-space.
* read\_log(): web log files

In many cases, these functions will just work: you supply the path to a file and you get a tibble back. The following example loads a sample file bundled with readr:

mtcars <- read\_csv(readr\_example("mtcars.csv"))

#> Parsed with column specification:

#> cols(

#> mpg = col\_double(),

#> cyl = col\_double(),

#> disp = col\_double(),

#> hp = col\_double(),

#> drat = col\_double(),

#> wt = col\_double(),

#> qsec = col\_double(),

#> vs = col\_double(),

#> am = col\_double(),

#> gear = col\_double(),

#> carb = col\_double()

#> )

Note that readr prints the column specification. This is useful because it allows you to check that the columns have been read in as you expect, and if they haven't, you can easily copy and paste into a new call:

mtcars <- read\_csv(readr\_example("mtcars.csv"), col\_types =

cols(

mpg = col\_double(),

cyl = col\_integer(),

disp = col\_double(),

hp = col\_integer(),

drat = col\_double(),

vs = col\_integer(),

wt = col\_double(),

qsec = col\_double(),

am = col\_integer(),

gear = col\_integer(),

carb = col\_integer()

)

)

vignette("readr") gives more detail on how readr guess the column types, how you can override the defaults, and provides some useful tools for debugging parsing problems.

## Alternatives

There are two main alternatives to readr: base R and data.table's fread(). The most important differences are discussed below.

### Base R

Compared to the corresponding base functions, readr functions:

* Use a consistent naming scheme for the parameters (e.g. col\_names and col\_types not header and colClasses).
* Are much faster (up to 10x).
* Leave strings as is by default, and automatically parse common date/time formats.
* Have a helpful progress bar if loading is going to take a while.
* All functions work exactly the same way regardless of the current locale. To override the US-centric defaults, use locale().

### data.table and fread()

[data.table](https://github.com/Rdatatable/data.table) has a function similar to read\_csv() called fread. Compared to fread, readr functions:

* Are slower (currently ~1.2-2x slower. If you want absolutely the best performance, use data.table::fread().
* Use a slightly more sophisticated parser, recognising both doubled ("""") and backslash escapes ("\""), and can produce factors and date/times directly.
* Forces you to supply all parameters, where fread() saves you work by automatically guessing the delimiter, whether or not the file has a header, and how many lines to skip.
* Are built on a different underlying infrastructure. Readr functions are designed to be quite general, which makes it easier to add support for new rectangular data formats. fread() is designed to be as fast as possible.