10 years of rio and readODS

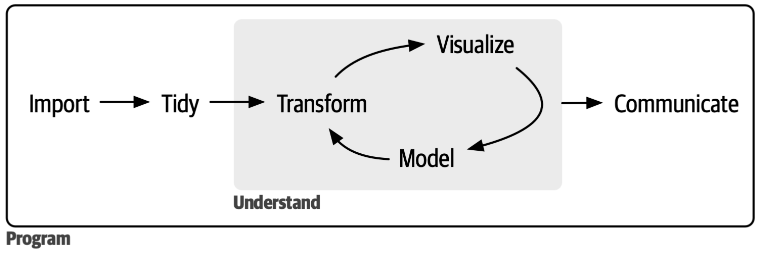
Maintaining the I/O infrastructure of R

Chung-hong Chan

GESIS

2024-07-10

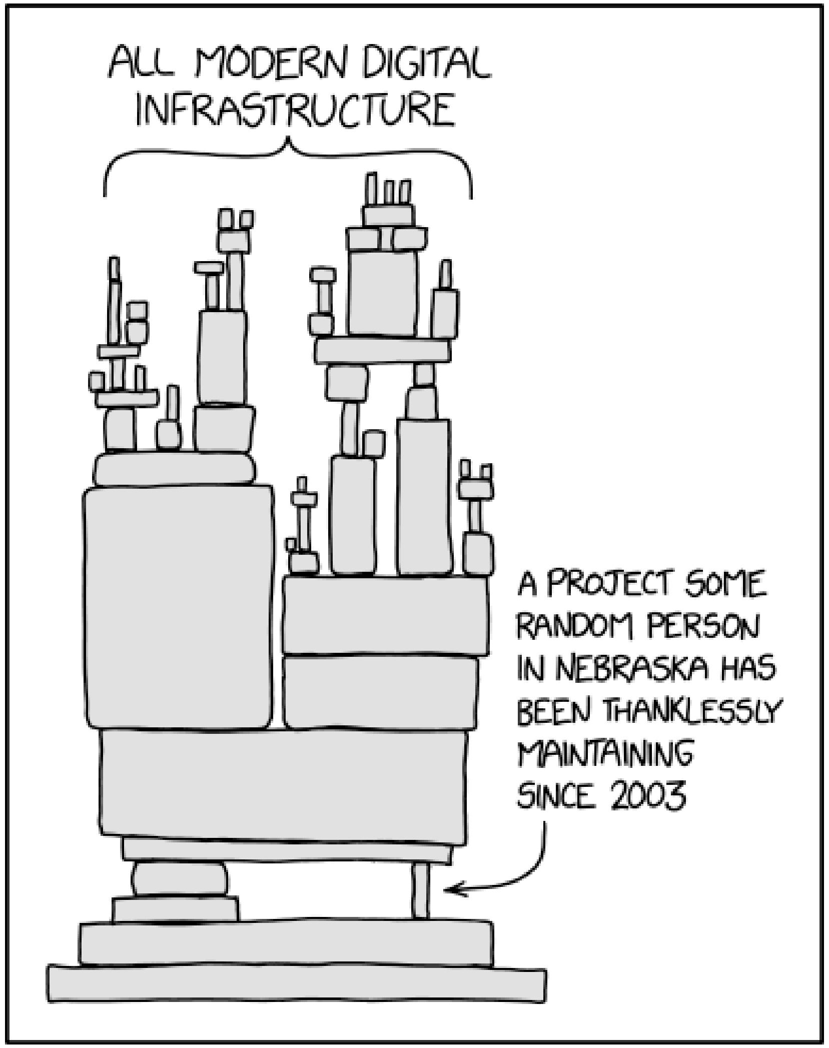
Why you should care about I/O



Source: R for Data Science

“First, you must import your data into R. This typically means that you take data stored in a file, database, or web application programming interface (API) and load it into a data frame in R. **If you can’t get your data into R, you can’t do data science on it!**”

Also I/O


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

Source: https://xkcd.com/2347/

Hello from ~~Nebraska~~ Mannheim!

Before 2013, data import and export

```r
write.csv(iris, "iris.csv")
saveRDS(iris, "iris.rds")
save(iris, "iris.Rdata")
# 2013: No way to write to spss

x <- read.csv("iris.csv")
x <- readRDS("iris.rds")
x <- read.spss("iris.sav")
load("iris.Rdata")
```

rio, since 2013

```r
library(rio)
export(iris, "iris.csv")
export(iris, "iris.rds")
export(iris, "iris.sav")

x <- import("iris.csv")
x <- import("iris.rds")
x <- import("iris.sav")
```

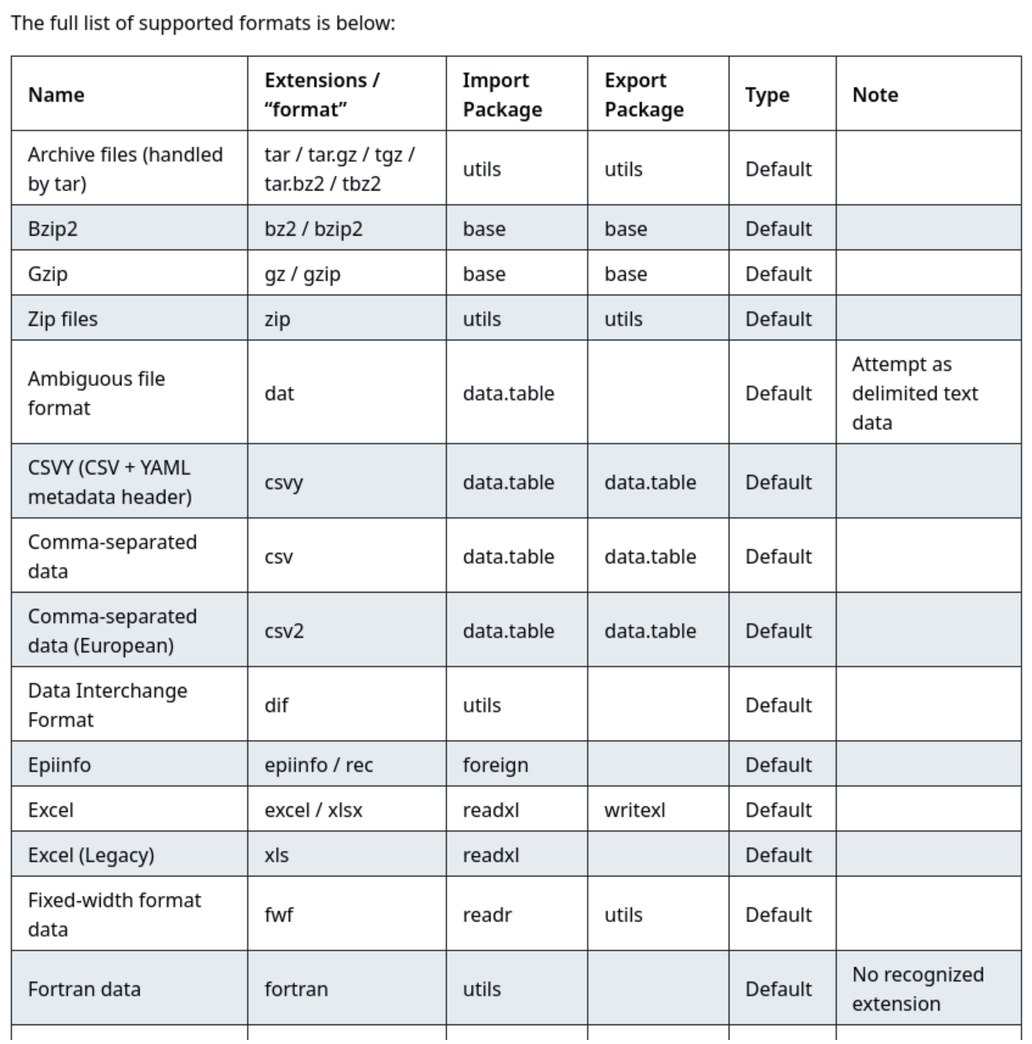
`rio` version 0.1.1 2013-08-26 14:02 CEST

```r
import <- function(file="", format=NULL, header=TRUE, ... ) {
  format <- .guess(file, format)
  x <- switch(format,
              txt=read.table(file=file, sep="\t", header=header, ..
              rds=readRDS(file=file, ...),
              csv=read.csv(file=file, ...),
              dta=read.dta(file=file, ...),
              sav=read.spss(file=file,to.data.frame=TRUE, ...),
              mtp=read.mtp(file=file, ...),
              rec=read.epiinfo(file=file, ...),
              stop("Unknown file format")
              )
  return(x)
}
```

`rio` development

- 2015 Feb: Transfer maintainership to Thomas Leeper
- 2015 Mar: Add support for ODS (OpenDocument Spreadsheet)
- 2016 Jan: Use S3 class (no longer `switch()`) by Jason Becker
- 2016 - 2023: Add many supported formats

Supported formats (partial list)



The full list of supported formats is below:

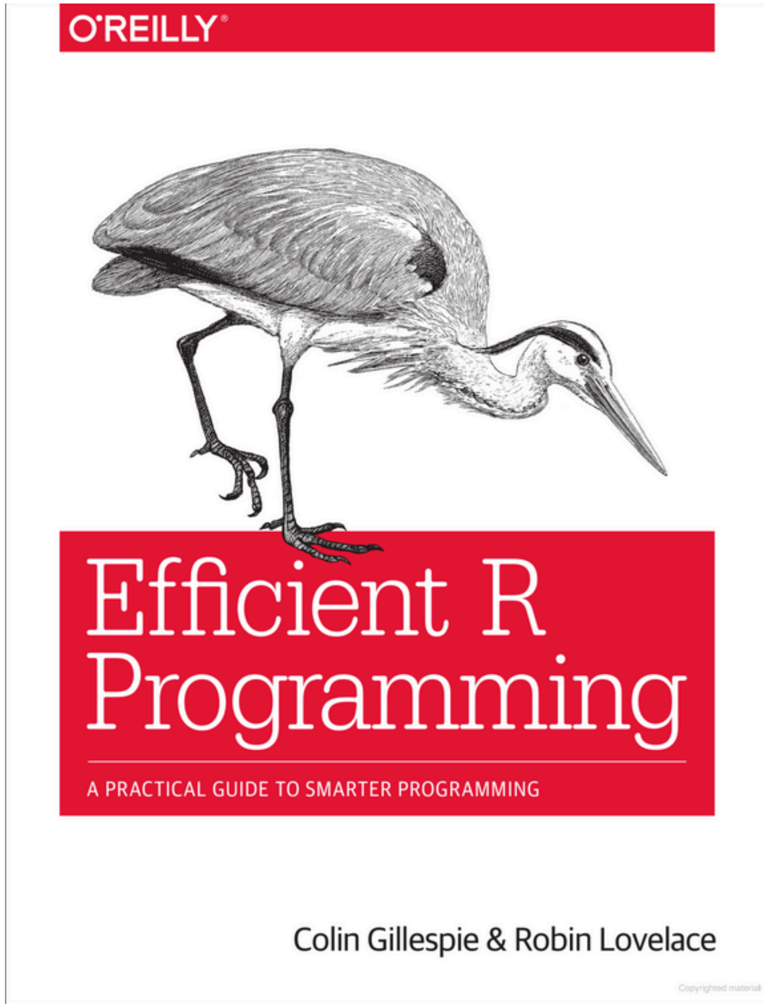
Name	Extensions / "format"	Import Package	Export Package	Type	Note
Archive files (handled by tar)	tar / tar.gz / tgz / tar.bz2 / tbz2	utils	utils	Default	
Bzip2	bz2 / bzip2	base	base	Default	
Gzip	gz / gzip	base	base	Default	
Zip files	zip	utils	utils	Default	
Ambiguous file format	dat	data.table		Default	Attempt as delimited text data
CSVY (CSV + YAML metadata header)	csvy	data.table	data.table	Default	
Comma-separated data	csv	data.table	data.table	Default	
Comma-separated data (European)	csv2	data.table	data.table	Default	
Data Interchange Format	dif	utils		Default	
Epiinfo	epiinfo / rec	foreign		Default	
Excel	excel / xlsx	readxl	writexl	Default	
Excel (Legacy)	xls	readxl		Default	
Fixed-width format data	fwf	readr	utils	Default	
Fortran data	fortran	utils		Default	No recognized extension

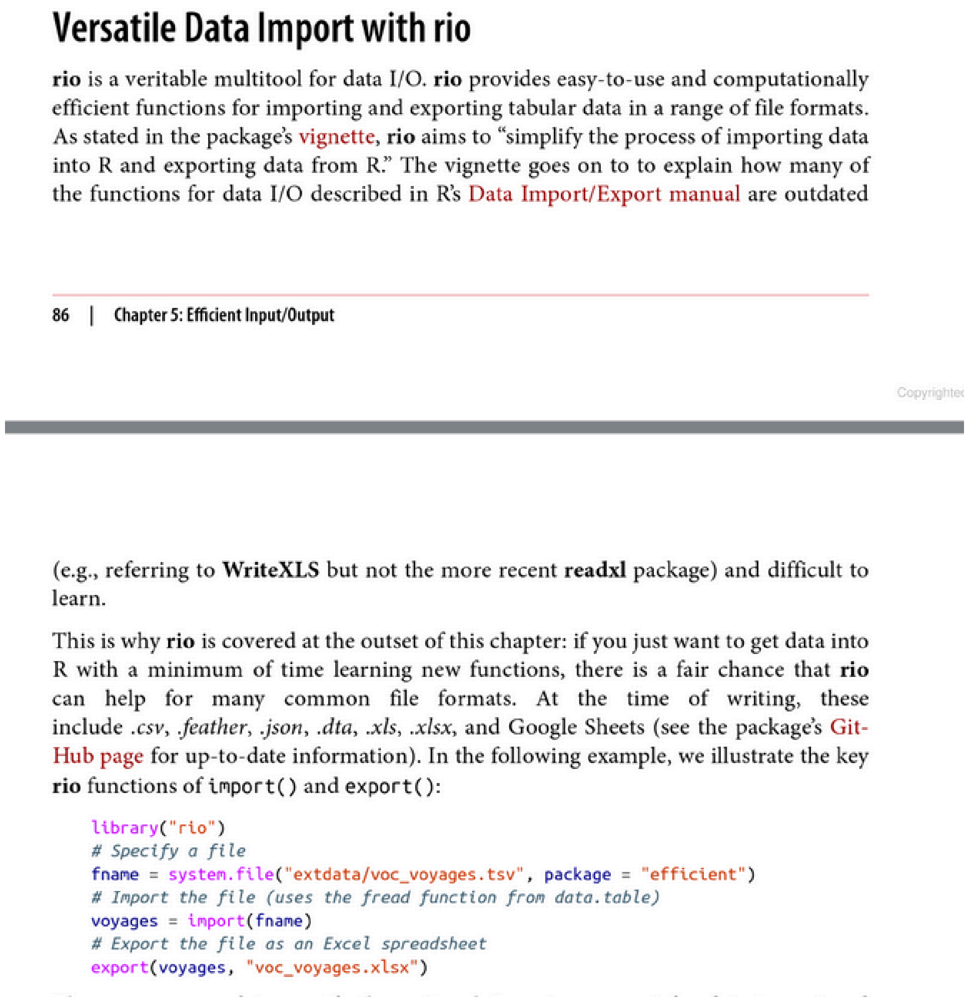
Full list

`rio` development

- 2023 Aug: 10 years
- 2023 Aug: Maintain collectively by GESIS Transparent Social Analytics Team
- 2023 Sep: `rio` 1.0.0

rio in real world 1


O'REILLY
Efficient R Programming
A PRACTICAL GUIDE TO SMARTER PROGRAMMING
Colin Gillespie & Robin Lovelace



Versatile Data Import with rio

rio is a veritable multitool for data I/O. **rio** provides easy-to-use and computationally efficient functions for importing and exporting tabular data in a range of file formats. As stated in the package's vignette, **rio** aims to "simplify the process of importing data into R and exporting data from R." The vignette goes on to to explain how many of the functions for data I/O described in R's Data Import/Export manual are outdated

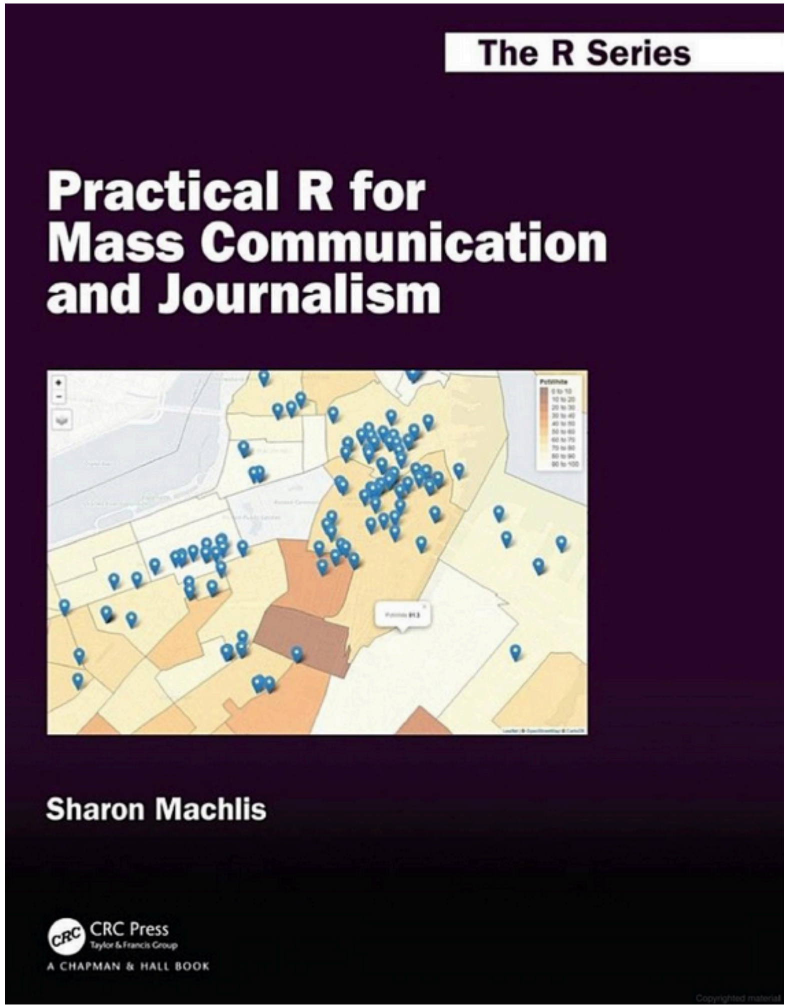
86 | Chapter 5: Efficient Input/Output

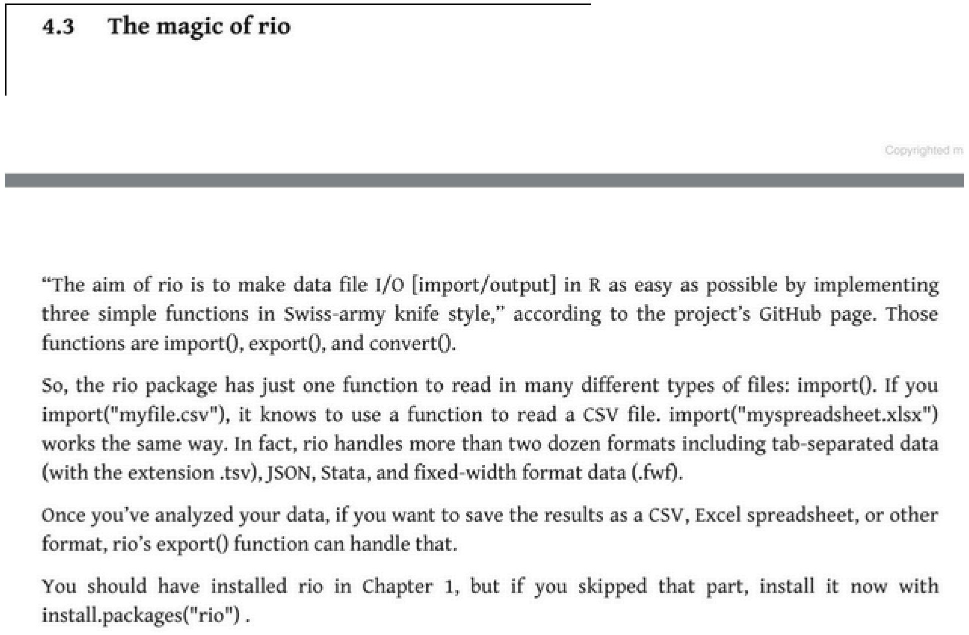
(e.g., referring to **WriteXLS** but not the more recent **readxl** package) and difficult to learn.

This is why **rio** is covered at the outset of this chapter: if you just want to get data into R with a minimum of time learning new functions, there is a fair chance that **rio** can help for many common file formats. At the time of writing, these include *.csv*, *.feather*, *.json*, *.dta*, *.xls*, *.xlsx*, and Google Sheets (see the package's Git-Hub page for up-to-date information). In the following example, we illustrate the key **rio** functions of `import()` and `export()`:

```
library("rio")
# Specify a file
fname = system.file("extdata/voc_voyages.tsv", package = "efficient")
# Import the file (uses the fread function from data.table)
voyages = import(fname)
# Export the file as an Excel spreadsheet
export(voyages, "voc_voyages.xlsx")
```

rio in real world 2





4.3 The magic of rio

“The aim of rio is to make data file I/O [import/output] in R as easy as possible by implementing three simple functions in Swiss-army knife style,” according to the project’s GitHub page. Those functions are import(), export(), and convert().

So, the rio package has just one function to read in many different types of files: import(). If you import("myfile.csv"), it knows to use a function to read a CSV file. import("myspreadsheet.xlsx") works the same way. In fact, rio handles more than two dozen formats including tab-separated data (with the extension .tsv), JSON, Stata, and fixed-width format data (.fwf).

Once you’ve analyzed your data, if you want to save the results as a CSV, Excel spreadsheet, or other format, rio’s export() function can handle that.

You should have installed rio in Chapter 1, but if you skipped that part, install it now with install.packages("rio") .

`rio` development

- 2015 Feb: Transfer maintainership to Thomas Leeper
- 2016 Jan: Use S3 class (no longer `switch()`) by Jason Becker
- 2016 - 2023: Add many supported formats

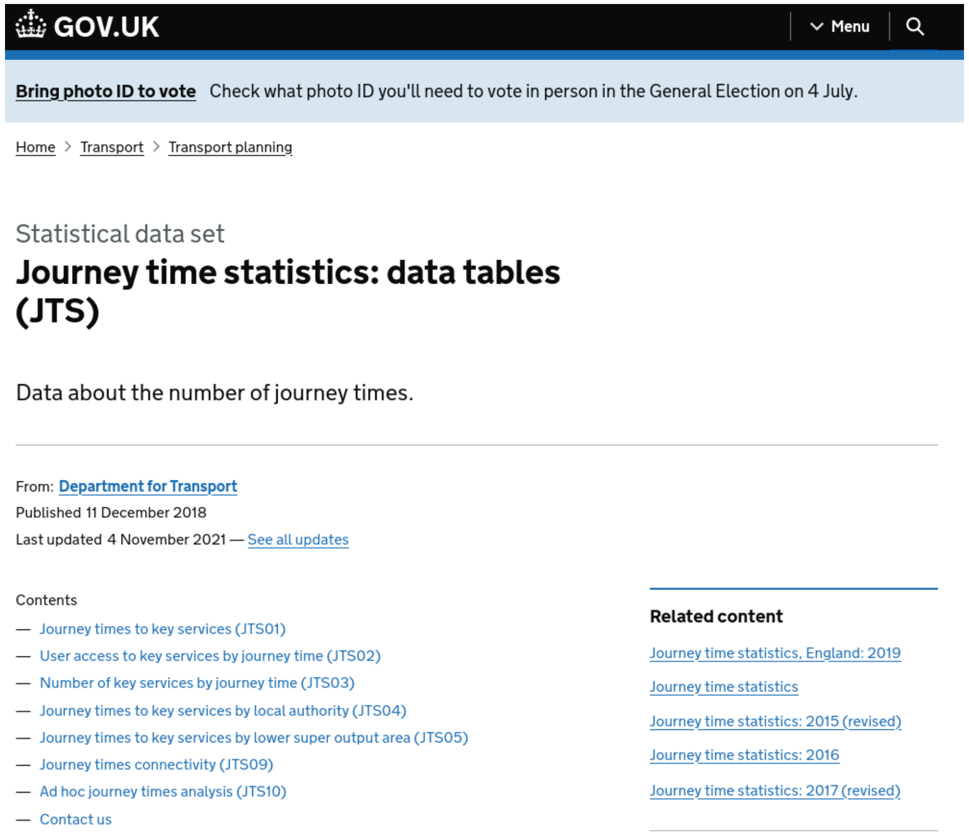
`rio` development

- 2015 Feb: Transfer maintainership to Thomas Leeper
- 2015 Mar: Add support for ODS (OpenDocument Spreadsheet)
- 2016 Jan: Use S3 class (no longer `switch()`) by Jason Becker
- 2016 - 2023: Add many supported formats

OpenDocument Spreadsheet

- Truly open format (ISO/IEC 26300)
- LibreOffice, Google Sheets, Microsoft Office etc.
- Technology: Zipped XML file (like xlsx)
- Adoption: NATO, EU, and many governments

Example: UK Gov



GOV.UK | Menu

Bring photo ID to vote Check what photo ID you'll need to vote in person in the General Election on 4 July.

Home > Transport > Transport planning

Statistical data set

Journey time statistics: data tables (JTS)

Data about the number of journey times.

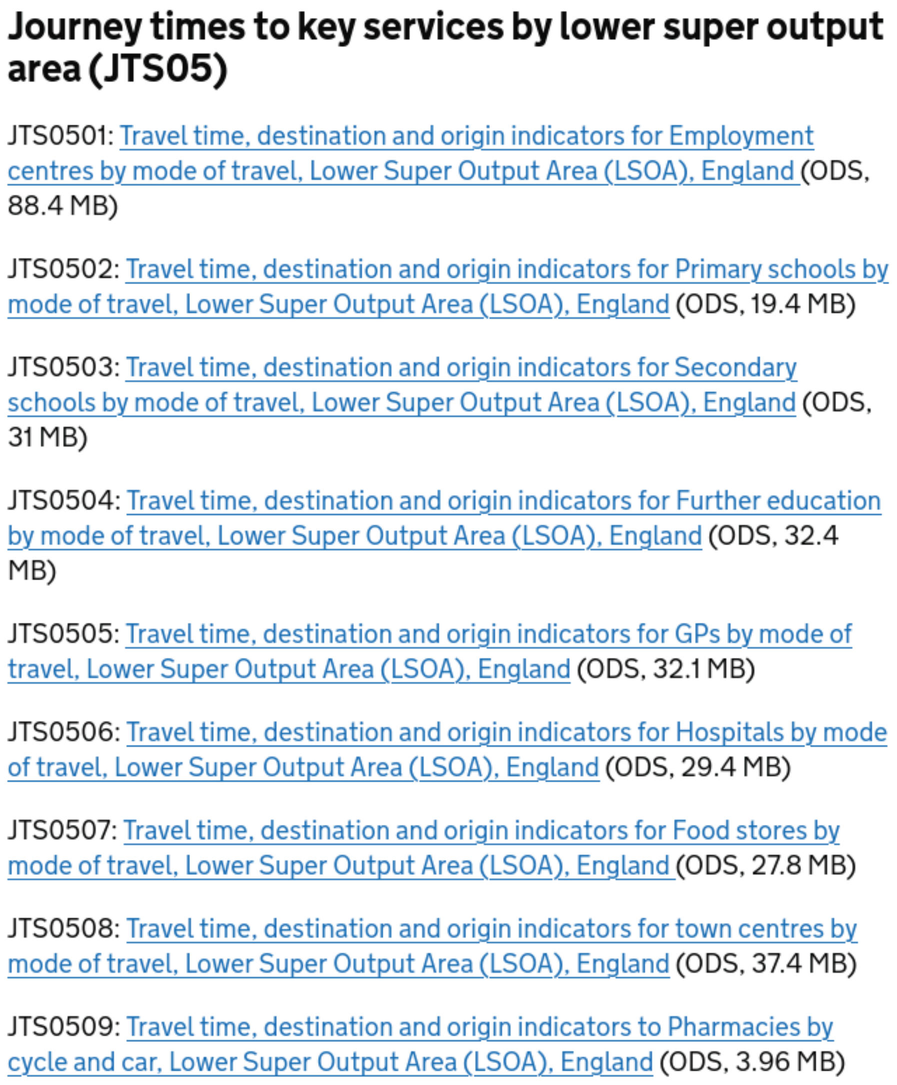
From: Department for Transport
Published 11 December 2018
Last updated 4 November 2021 — See all updates

Contents

- Journey times to key services (JTS01)
- User access to key services by journey time (JTS02)
- Number of key services by journey time (JTS03)
- Journey times to key services by local authority (JTS04)
- Journey times to key services by lower super output area (JTS05)
- Journey times connectivity (JTS09)
- Ad hoc journey times analysis (JTS10)
- Contact us

Related content

- Journey time statistics, England: 2019
- Journey time statistics
- Journey time statistics: 2015 (revised)
- Journey time statistics: 2016
- Journey time statistics: 2017 (revised)



Journey times to key services by lower super output area (JTS05)

JTS0501: Travel time, destination and origin indicators for Employment centres by mode of travel, Lower Super Output Area (LSOA), England (ODS, 88.4 MB)

JTS0502: Travel time, destination and origin indicators for Primary schools by mode of travel, Lower Super Output Area (LSOA), England (ODS, 19.4 MB)

JTS0503: Travel time, destination and origin indicators for Secondary schools by mode of travel, Lower Super Output Area (LSOA), England (ODS, 31 MB)

JTS0504: Travel time, destination and origin indicators for Further education by mode of travel, Lower Super Output Area (LSOA), England (ODS, 32.4 MB)

JTS0505: Travel time, destination and origin indicators for GPs by mode of travel, Lower Super Output Area (LSOA), England (ODS, 32.1 MB)

JTS0506: Travel time, destination and origin indicators for Hospitals by mode of travel, Lower Super Output Area (LSOA), England (ODS, 29.4 MB)

JTS0507: Travel time, destination and origin indicators for Food stores by mode of travel, Lower Super Output Area (LSOA), England (ODS, 27.8 MB)

JTS0508: Travel time, destination and origin indicators for town centres by mode of travel, Lower Super Output Area (LSOA), England (ODS, 37.4 MB)

JTS0509: Travel time, destination and origin indicators to Pharmacies by cycle and car, Lower Super Output Area (LSOA), England (ODS, 3.96 MB)

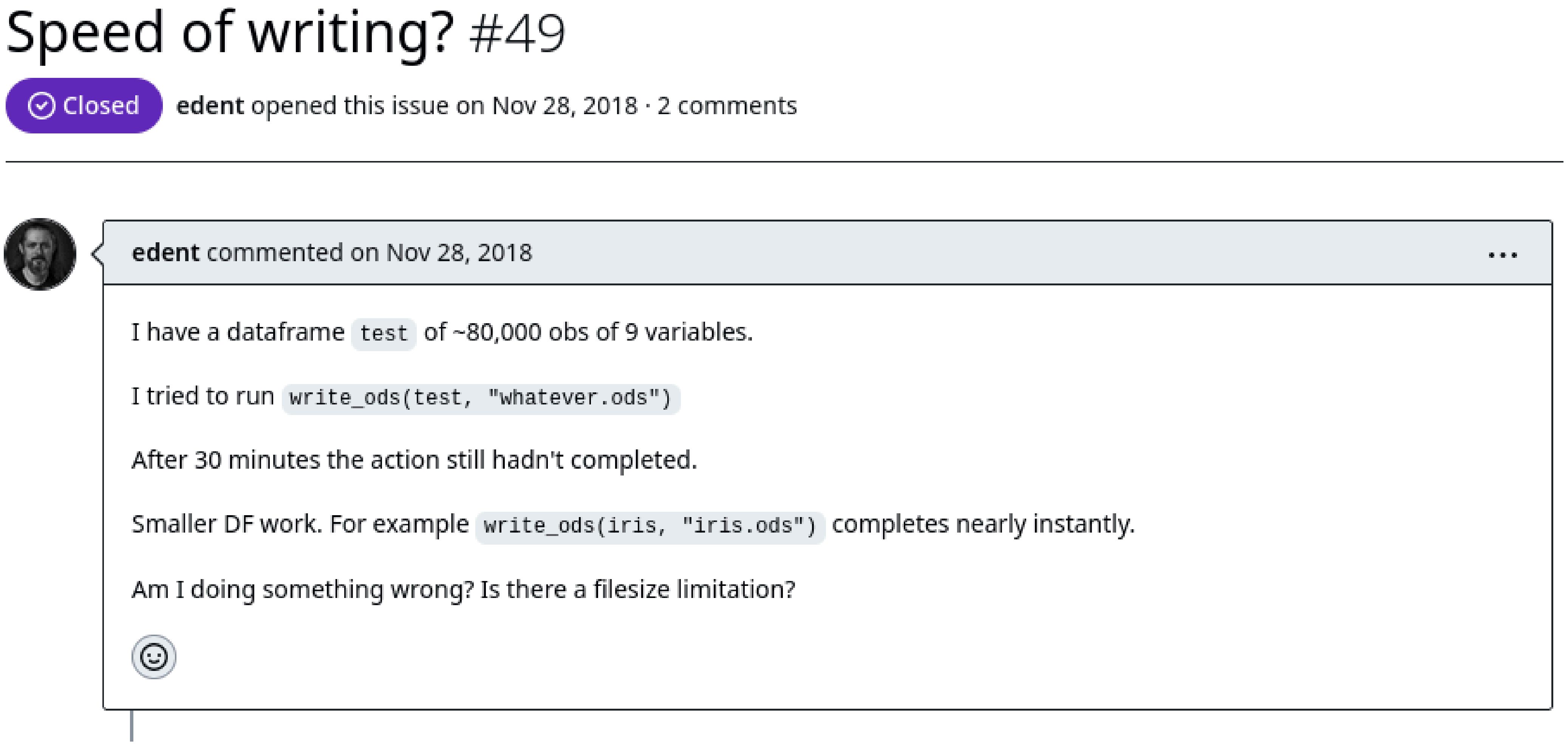
`readODS`

- Created by Gerrit-Jan Schutten in 2014
- Maintained by me since 2016
- Peer reviewed and accepted by rOpenSci since 2022-06-24
- Emulate `readxl::read_excel()` and `writexl::write_xlsx()`

`readODS` prior 2.0.0

- Based on `XML`, and then `xml2`, in pure R
- Was in a bad state due to performance

readODS issue 49



Speed of writing? #49

Closed **edent** opened this issue on Nov 28, 2018 · 2 comments

edent commented on Nov 28, 2018

I have a dataframe `test` of ~80,000 obs of 9 variables.

I tried to run `write_ods(test, "whatever.ods")`

After 30 minutes the action still hadn't completed.

Smaller DF work. For example `write_ods(iris, "iris.ods")` completes nearly instantly.

Am I doing something wrong? Is there a filesize limitation?

`readODS` issue 71

`readODS` issue 71

Not Working also

- Google Sheets, Python `odfpy` (Also Julia `OdsIO.jl`), JS `SheetJS`

Working but

- LibreOffice ~ 15s
- Export as … first by LibreOffice and then
  - CSV via `data.table::fread()` - 1s
  - XLSX via `readxl::read_excel()` - 2s

Projekt 71

> "I'll devote my 2023 to the project I tentatively called"Projekt 71". The idea is simple: I want to have a way that can read the aforementioned "jts0501.ods" directly as an R data frame without memory issues; but yet pass at least 80% of the current unit tests of `readODS`. So, I am embarking on solving just one Github issue of `readODS`. I will put other of my R packages into maintenance mode and focus only on this."

My Projekt 71 manifesto

Peter Brohan is our hero

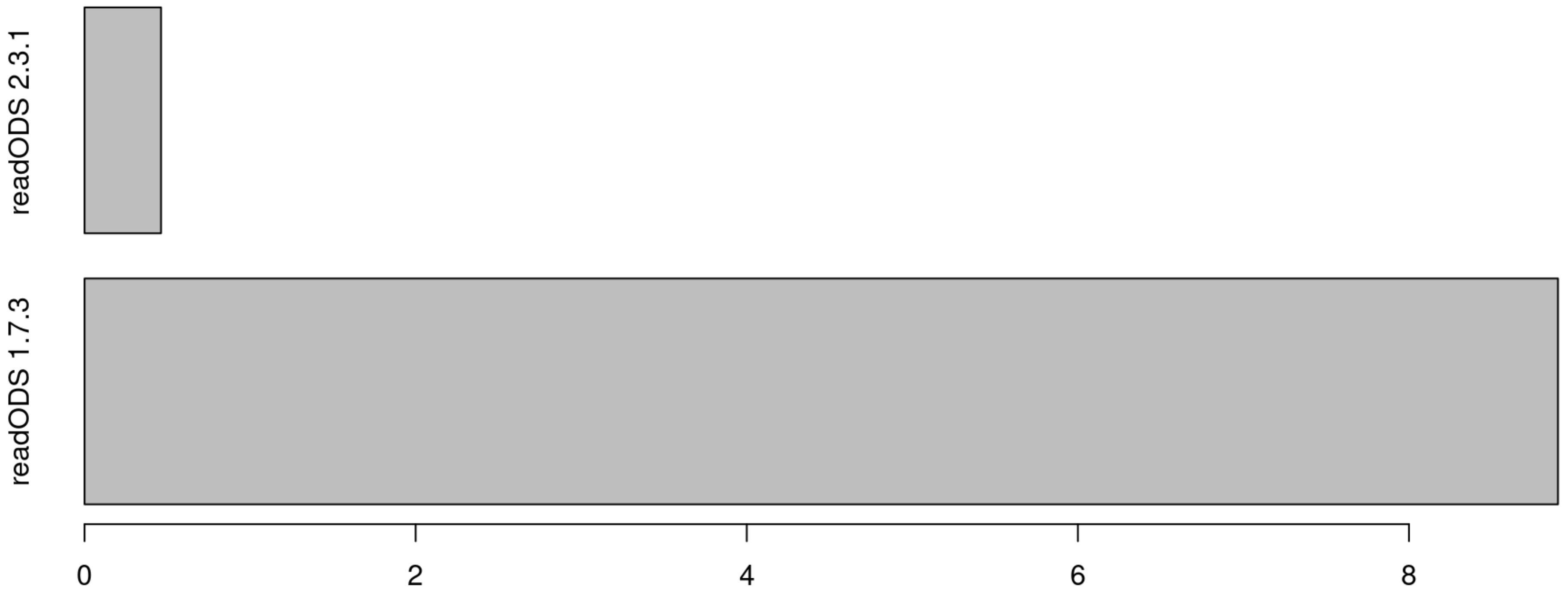
- In July 2023, Peter rewrote `readODS::read_ods()` in C++ (RapidXML) - super speed improvement
- I switched to tackle issue 49 (Writing speed)

Detlef Steuer is also our hero

- Detlef proposed a speedy solution to write XML using base R without `xml2`
- I rewrote his solution in C++

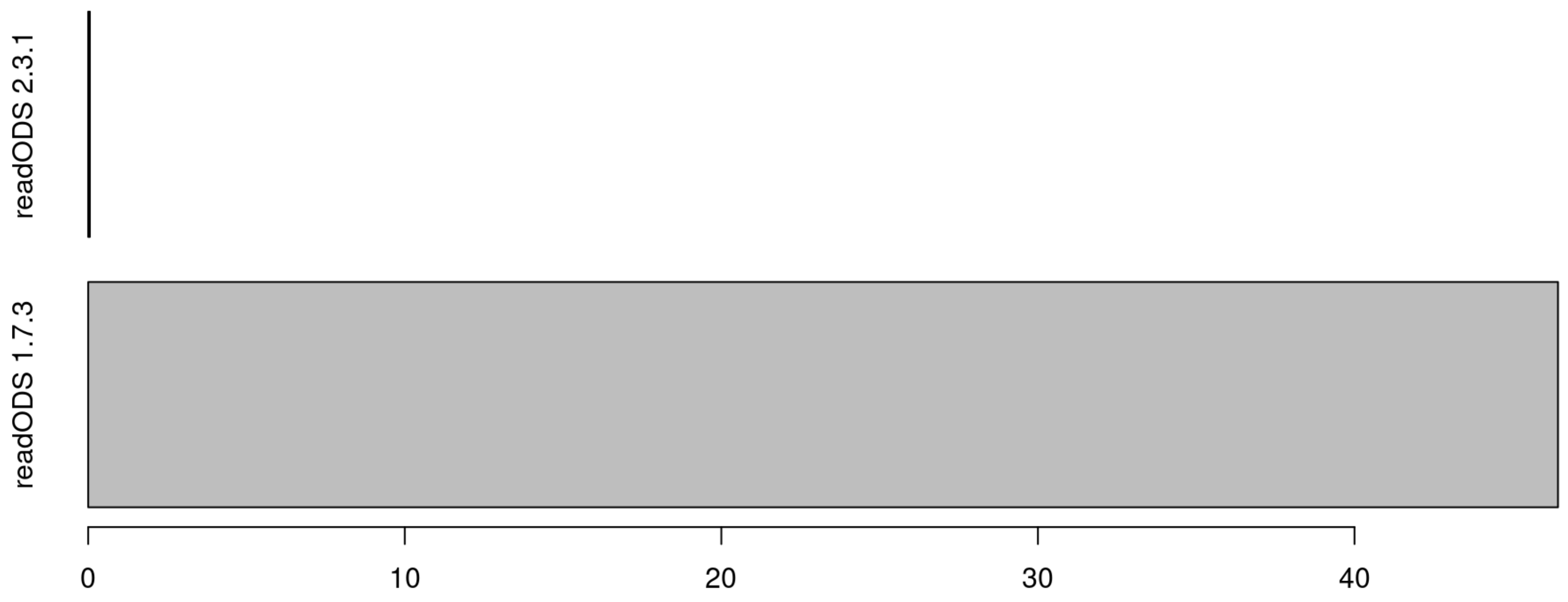
Benchmark I

Reading speed: 5539 x 11



Benchmark II

Writing speed: 3000 x 8



In summary

- Issue 71: “jts0501.ods” is readable with a modest computer in 120s (was impossible)
- Issue 49: “80000 x 9” can be written in < 2s (was > 30 mins)

So, are we done yet?

> “Software, unlike papers or grants, is never done.”

Free as in mummies

Issues

- Type guessing, not 100% the same as `readxl` - working on `minty`
- Speed vs `readxl` (and other formats)

`rio`-based comparison

`Lahman::Batting` (11,2164 x 22)

Rank	Format	Export	Import	Size	Accuracy
4	csv	1	1	1	2

Benchmark of R file formats using rio and friends

rio-based comparison

Rank	Format	Export	Import	Size	Accuracy
1	feather	0.9	0.3	0.5	0
2	parquet	3.6	0.4	0.3	0
3	qs	1.7	0.7	0.2	2
4	csv	1	1	1	2

Benchmark of R file formats using rio and friends

`rio`-based comparison

Rank	Format	Export	Import	Size	Accuracy
1	feather	0.9	0.3	0.5	0
2	parquet	3.6	0.4	0.3	0
3	qs	1.7	0.7	0.2	2
4	csv	1	1	1	2
…					
16	xlsx	141.4	36.3	1.3	21

Benchmark of R file formats using rio and friends

rio-based comparison

Rank	Format	Export	Import	Size	Accuracy
1	feather	0.9	0.3	0.5	0
2	parquet	3.6	0.4	0.3	0
3	qs	1.7	0.7	0.2	2
4	csv	1	1	1	2
16	xlsx	141.4	36.3	1.3	21
23	fods	77.3	119.7	42.2	21
25	ods	258.2	253.7	0.8	21

Benchmark of R file formats using rio and friends

10 years of `rio` and `readODS`

10 years of `rio` and `readODS`

> “A good beginning requires enthusiasm, a good ending requires discipline.”

The Motto of the German Football Association for Word Cup 2014

10 years of `rio` and `readODS`

> “Ein guter Anfang braucht Begeisterung, ein gutes Ende Disziplin.”

das Motto des DFB für die WM 2014

10 years of `rio` and `readODS`

> “Ein guter Anfang braucht Begeisterung, ein gutes Ende Disziplin.”

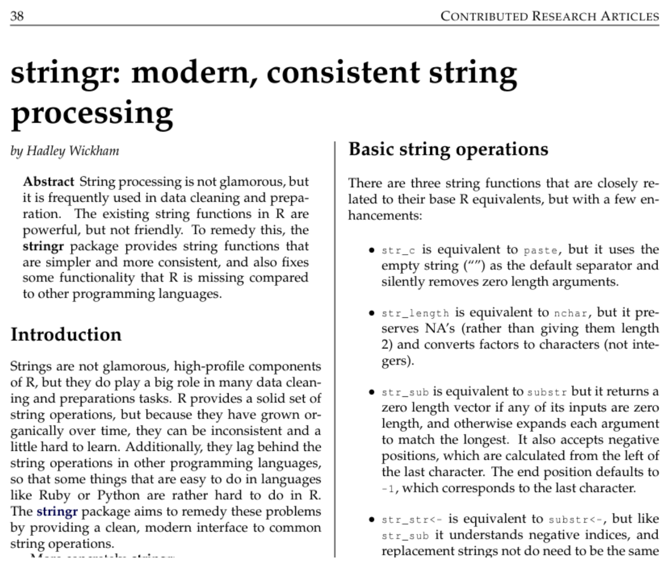
das Motto des DFB für die WM 2014

Thank you:

- Thomas Leeper (for maintaining `rio` from 2015 to 2023)
- Gerrit-Jan Schutten (for creating `readODS`)
- Peter Brohan (for the C++ implementation of `read_ods`)
- Detlef Steuer (for the quick XML generation algorithm)
- Jason Becker (for the sustained contribution to `rio`)
- David Schoch (for the contribution to `rio` and leading the GESIS TSA Team)
- Hadley Wickham, Jennifer Bryan (for allowing me to fork some code from `readr`)
- rOpenSci (for managing the infrastructure to support the development of `readODS`)
- All developers / maintainers of the I/O infrastructure
- All `rio` and `readODS` contributors and users.

More about me: https://www.chainsawriot.com/

Wickham (2010)



38 CONTRIBUTED RESEARCH ARTICLES

stringr: modern, consistent string processing

by Hadley Wickham

Abstract String processing is not glamorous, but it is frequently used in data cleaning and preparation. The existing string functions in R are powerful, but not friendly. To remedy this, the **stringr** package provides string functions that are simpler and more consistent, and also fixes some functionality that R is missing compared to other programming languages.

Introduction

Strings are not glamorous, high-profile components of R, but they do play a big role in many data cleaning and preparations tasks. R provides a solid set of string operations, but because they have grown organically over time, they can be inconsistent and a little hard to learn. Additionally, they lag behind the string operations in other programming languages, so that some things that are easy to do in languages like Ruby or Python are rather hard to do in R. The **stringr** package aims to remedy these problems by providing a clean, modern interface to common string operations.

Basic string operations

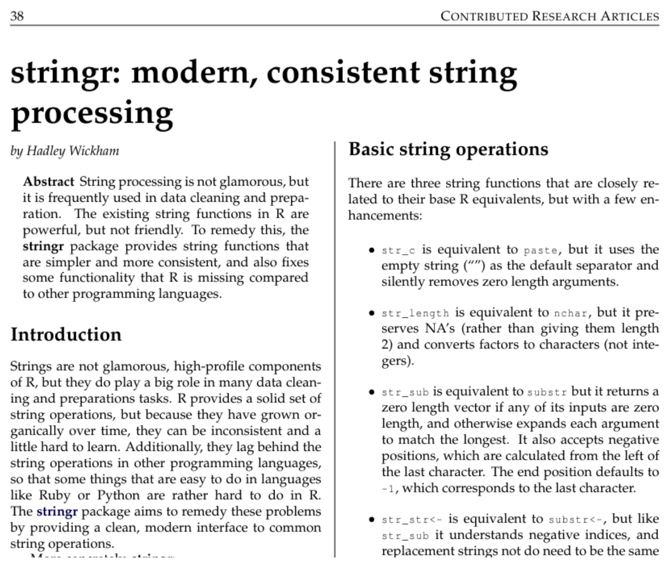
There are three string functions that are closely related to their base R equivalents, but with a few enhancements:

- `str_c` is equivalent to `paste`, but it uses the empty string ("") as the default separator and silently removes zero length arguments.
- `str_length` is equivalent to `nchar`, but it preserves NA's (rather than giving them length 2) and converts factors to characters (not integers).
- `str_sub` is equivalent to `substr` but it returns a zero length vector if any of its inputs are zero length, and otherwise expands each argument to match the longest. It also accepts negative positions, which are calculated from the left of the last character. The end position defaults to `-1`, which corresponds to the last character.
- `str_str<-` is equivalent to `substr<-`, but like `str_sub` it understands negative indices, and replacement strings not do need to be the same

```r
## From this
text <- "she dances on the sand"
grepl("sand$", text)
strsplit(text, "[dD]ances?")
```

Wickham (2010)

Wickham (2010)



38 CONTRIBUTED RESEARCH ARTICLES

stringr: modern, consistent string processing

by Hadley Wickham

Abstract String processing is not glamorous, but it is frequently used in data cleaning and preparation. The existing string functions in R are powerful, but not friendly. To remedy this, the **stringr** package provides string functions that are simpler and more consistent, and also fixes some functionality that R is missing compared to other programming languages.

Introduction

Strings are not glamorous, high-profile components of R, but they do play a big role in many data cleaning and preparations tasks. R provides a solid set of string operations, but because they have grown organically over time, they can be inconsistent and a little hard to learn. Additionally, they lag behind the string operations in other programming languages, so that some things that are easy to do in languages like Ruby or Python are rather hard to do in R. The **stringr** package aims to remedy these problems by providing a clean, modern interface to common string operations.

Basic string operations

There are three string functions that are closely related to their base R equivalents, but with a few enhancements:

- `str_c` is equivalent to `paste`, but it uses the empty string ("") as the default separator and silently removes zero length arguments.
- `str_length` is equivalent to `nchar`, but it preserves NA's (rather than giving them length 2) and converts factors to characters (not integers).
- `str_sub` is equivalent to `substr` but it returns a zero length vector if any of its inputs are zero length, and otherwise expands each argument to match the longest. It also accepts negative positions, which are calculated from the left of the last character. The end position defaults to `-1`, which corresponds to the last character.
- `str_str<-` is equivalent to `substr<-`, but like `str_sub` it understands negative indices, and replacement strings not do need to be the same

```r
## From this
text <- "she dances on the sand"
grepl("sand$", text)
strsplit(text, "[dD]ances?")
## To this
library(stringr)
str_detect(text, "sand$")
str_split(text, "[dD]ances?")
```

Wickham (2010)

rio in real world 3

- Used by multiple organizations: WHO, Médecins Sans Frontières etc.
- In data science education