

Working with large data at the RDSC

Technical Report 2021-04

The views expressed in this technical report are personal views of the authors and do not necessarily reflect the views of the Deutsche Bundesbank or the Eurosystem.

Version: 1 (September 2021)

Deutsche Bundesbank, Research Data and Service Centre

Matthias Gomolka

Jannick Blaschke

Christian Hirsch

Abstract

This report provides recommendations on writing efficient code when working with large data in the technical settings of the Research Data and Service Centre (RDSC). We place specific emphasis on tweaks to code that are easy to implement. In benchmark tests, these tweaks lead to a decrease in computing time of roughly 50 % in both Stata and R where R generally is faster than Stata. We urge researchers to consider both computing time and programming time before making a decision on which language to use.

Keywords: large data, data manipulation, data handling, R, Stata

Version: 1

Citation: Gomolka, M., J. Blaschke and C. Hirsch (2021). Working with large data at the RDSC, Technical Report 2021-04 – Version 1. Deutsche Bundesbank, Research Data and Service Centre.

Contents

1	Introduction	4
2	Deciding on a suitable programming language: Stata or R	6
3	Recommendations on importing data into memory	7
3.1	Use suitable file formats	7
3.2	Read only relevant variables and time periods into memory	9
4	Recommendations on data wrangling	12
4.1	Use software packages built for large data	12
4.2	Use suitable column types	13
5	General recommendations	15
5.1	Start with a small subset of your data	15
5.2	Use Virtual Clients	15
6	Conclusion	16
	References	17

1 Introduction

Working with granular data in the Research Data and Service Centre (RDSC) covers (almost) the entire life cycle of a research project. After gaining data access, researchers usually start reading the raw data into memory and manipulating them in order to prepare them for analysis. After that, researchers continue with the analysis and conduct the Statistical Disclosure Control (SDC) which is finally checked again by the RDSC. This is very similar to the data science project cycle described by Wickham & Grolemund (2016) and visualized in Figure 1.

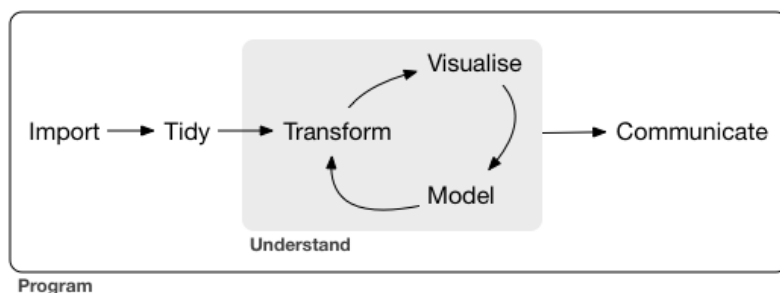


Figure 1: Data science project cycle according to Wickham & Grolemund (2016).

In general, the complexity of each of these steps increases with the size of the data.¹⁾ For example, commands run on larger data require significantly more computing time, which slows down the analysis and could require visiting researchers to travel to the RDSC more often.

Figure 2 shows a size comparison of selected datasets from the RDSC's portfolio. For instance, the "Monthly Balance Sheet Statistics (BISTA)" (Gomolka, Schäfer, & Stahl, 2021), which is one of the RDSC's most requested datasets, is only 0.5 GB while the "Markets in Financial Instruments Directive (MiFID)" data (Cagala, Gomolka, Krüger, & Sachs, 2021) is 471 times as big.

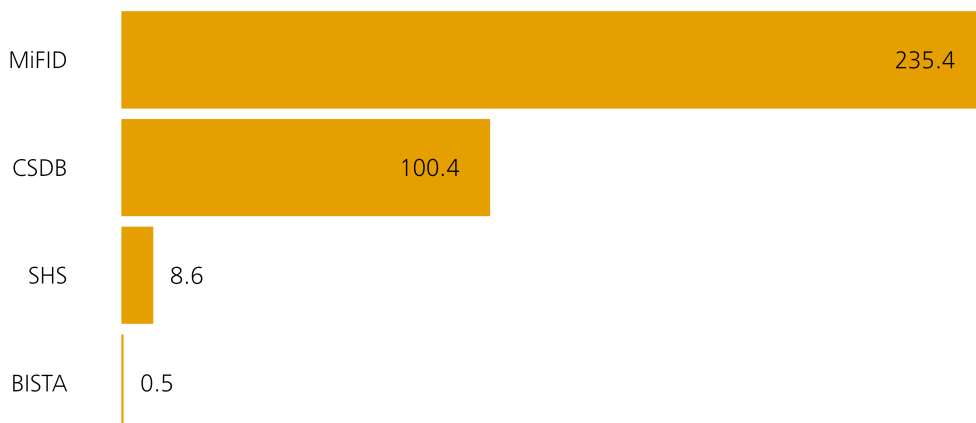


Figure 2: Size comparison of selected datasets offered at the RDSC. Sizes are reported in GB. Datasets are stored as gzip compressed dta (dta.gz).

The goal of this technical report is to provide visiting researchers at the RDSC with some basic tips and tricks on how to speed up their code and increase efficiency under the given technical settings

¹ Of course, also small datasets can be complex. However, in this report we focus on complexity due to size.

of the Deutsche Bundesbank's working environment. We write this with our largest datasets²⁾ in mind since benefits of efficient code are most pronounced in these cases. However, also users of smaller datasets might find our recommendations helpful, as it is always beneficial to reduce the computing time.

In the next sections, we highlight some programming tasks which commonly occur during a research project. These tasks are loosely oriented at the data science project cycle (Wickham & Grolemund, 2016). For each task, we compare simple default solutions to optimised and therefore recommended solutions. Additionally, we provide benchmarks to prove our claims. We provide code and benchmarks for Stata (StataCorp, 2019) and R (R Core Team, 2020), the two main programming languages used at the RDSC.³⁾

This report starts with some general considerations on choosing the right programming language for your research project (it is not about computing time only). Then, we present how to tackle some common tasks of a research project efficiently using Stata and R. This reveals that R is significantly faster in almost every task. Finally, Section 6 concludes.

²⁾ The largest datasets that the RDSC provides for visiting researchers are "Markets in Financial Instruments Directive (MiFID)" data, "Money Market Statistical Reporting (MMSR)" data, the "Centralised Securities Database (CSDB)" and the "Securities Holdings Statistics (SHS-Base plus)."

³⁾ In September 2021, the RDSC provides Stata SE 16.1, Stata MP 16.1 and R 4.0.2. All benchmarks displayed in this report ran on Stata MP 16.1 and R 4.0.2 with the following package versions: `appendgz` 0.3.1, `savegz` 0.3.0, `gtools` 1.1.2, `data.table` 1.14.0, `arrow` 4.0.1.

2 Deciding on a suitable programming language: Stata or R

By default, the RDSC provides Stata to researchers, which is the software most researchers in economics are familiar with. However, upon request, researchers can also use other analytical tools such as R, Matlab (The Mathworks, Inc., 2018) or Python (Van Rossum & Drake, 2009). Especially for the preparation of large datasets, the choice of a suitable programming language can speed things up. However, we highly recommend users to choose either Stata or R for research projects at the RDSC, as we currently only provide our SDC packages for Stata⁴ and R (Gomolka & Becker, 2021).

In the following sections, we compare the performance of different tasks like reading and collapsing data in Stata and R. As you will see, R has large performance advantages over Stata. But before jumping to the concluding to use R at all costs, we would like to highlight one very important point that you should keep in mind before making a decision which programming language to use: Computing time is not the only time you want to minimise! Instead, your optimization problem looks as follows:

$$\min(t_{\text{overall}}) \text{ where } t_{\text{overall}} = t_{\text{programming}} + t_{\text{computing}}$$

In plain English, you want to minimize the sum of the time that it takes you to write and debug the code $t_{\text{programming}}$ and the time it takes the computer to execute the code $t_{\text{computing}}$. *The figures in the following sections only show a comparison of the computing time*, since we cannot judge how proficient you are in writing code in Stata or R. Please bear this in mind when making your choice. Also consider that you will likely have to rerun your code several times as described by Wickham & Grolemund (2016).

Note further, that you will not have access to the internet when you work at the RDSC. Therefore, useful resources such as Stack Overflow will not be readily available. Consequently, if you decide to use R simply because of the performance advantages, you might end up working significantly longer on your project in case you struggle to program in R under the conditions of the secure environment at the RDSC.⁵

⁴ The mentioned Stata ado files (`nobsdes5`, `nobsreg5` and `maxrdsc`) are available on <https://www.bundesbank.de/en/bundesbank/research/rdsc/data-access>.

⁵ For a detailed description of the rules, please see Section 2 in Research Data and Service Centre (2021).

3 Recommendations on importing data into memory

Before being able to do anything else, you need to import your data into memory.⁶ If your data is large, this may take a considerable amount of time, especially if it is stored remotely on a network drive. Often, the whole dataset will be larger than memory and you will have to work on it chunkwise. Thus, this section gives a few recommendations on how to speed up reading data into memory.

Due to its size, the MiFID data is currently provided in over 3,500 daily files, each of which is up to 2.5 GB large (in memory) and contains up to 8.5 million observations. Therefore, our baseline dataset comprises of seven daily files (2017-11-20 to 2017-11-26) of MiFID data. Altogether, this week's data has 13.7 million rows and 46 columns.

3.1 Use suitable file formats

Both Stata and R can handle a range of different file formats. There are general formats (such as csv), native formats (dta and rds), formats from other statistical tools (like sas7bdat from SAS software), and formats accessible through extension packages (such as dta.gz and parquet). This section provides guidance on appropriate formats to use for each language.

Stata

By default, Stata reads and writes dta files. This format stores data without any compression which means it can be read quickly *if you store your datasets on a fast SSD drive*. However, this makes it slow to read from a network drive, which happens to be the case in the RDSC. Assuming that there exists the global `mifid_dta` which contains the paths of the seven files from the example week these would be read using `append using ${mifid_dta}`.

But Stata can also handle gzip compressed dta files. To do so, you can simply use the following commands from the `{gziptutil}`⁷ package (Gomolka, 2018) as drop-in replacements:

- `usegz` for use
- `savegz` for save
- `mergegz` for merge
- `appendgz` for append

For more information on `{gziptutil}`, the RDSC's own package for using gzip compressed Stata files, see the package documentation. Thus, the optimised solution for this task would be:

```
// create a global macro containing the paths of one week of MiFID data
// assuming that all seven files are located in the working directory
global mifid_dtagz : dir "." files "*.dta.gz", respectcase
```

⁶ We are aware that there exist analytical tools which can operate on data without reading them into memory such as SAS software (SAS Institute Inc., 2016), but the RDSC does not provide these to researchers.

⁷ To highlight software packages which extend standard software like Stata and R, we enclose their names with curly braces. Additionally, we cite these packages whenever they first occur in this report.

```
// import data  
clear  
appendgz using ${mifid_dtagz}
```

R

R's native format to read and write objects is rds. By default, it is compressed using gzip. To read the example data from rds files, use `do.call(rbind, lapply(mifid_rds, readRDS))` where `mifid_rds` is a character vector of rds file names.

Even though rds is a compressed format, there is a faster and more flexible solution: parquet, which is available to R via the `{arrow}` package (Richardson et al., 2021). Parquet is the on-disk representation of the arrow format which is actively developed by The Apache Software Foundation.⁸⁾

The following code shows how to speed things up: We import the data using `read_parquet()` from `{arrow}` and append the daily chunks with `rbindlist()` from `{data.table}` (Dowle & Srinivasan, 2021).

```
# load relevant packages  
library(arrow)  
library(data.table)  
  
# create a vector containing the file paths one week of MiFID data  
# assuming that all seven files are located in the working directory  
mifid_parquet <- dir(pattern = ".parquet")  
  
# import data and assign the result to 'mifid'  
mifid <- rbindlist(lapply(mifid_parquet, read_parquet))
```

Benchmarks

Figure 3 shows the computing time needed to import our baseline MiFID data into Stata and into R.⁹⁾ For both languages we show the timings for the baseline scenario and the optimised scenario. Note that using parquet and R reduces the computing time compared to the optimised Stata approach by 65.1% from 77.6 to 27.1 seconds.

These results carry over to the opposite direction: writing data to disk. Thus, you can save a lot of time by simply storing intermediate datasets as `dta.gz` or `parquet`. The RDSC usually provides data in these two formats.

⁸ For details see <https://arrow.apache.org/>.

⁹ All benchmarks were run on the same Virtual Client (VC). This VC features an Intel Xeon CPU E5-2667 v3 @ 3.20GHz and 192 GB of memory. The data used for the import benchmarks was stored on the same network drive as research projects in order to make the benchmarks representative for visiting researchers at the RDSC.

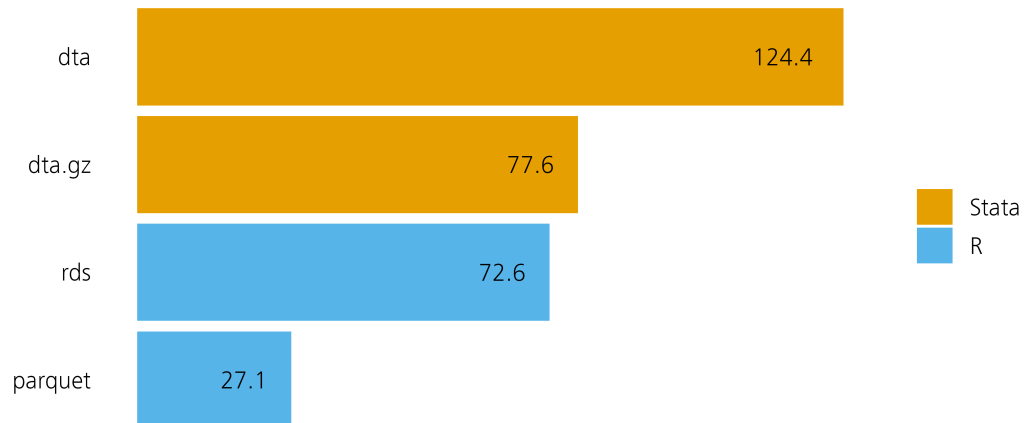


Figure 3: Timings (in seconds) for importing one week of MiFID data. Exact commands below:
 dta: `append using ${mifid_dta}`
 dta.gz: `appendgz using ${mifid_dtagz}`
 rds: `do.call(rbind, lapply(mifid_rds, readRDS))`
 parquet: `rbindlist(lapply(mifid_parquet, read_parquet))`

Just a side note: Using compressed dta files also reduces the file size on disk by approximately 90%.

3.2 Read only relevant variables and time periods into memory

To reduce the size of your dataset you should only use the relevant time period for your analysis when reading the data into memory. Apart from the number of observations, the size of a dataset also depends on the number of columns and their respective formats. One easy and quickly implemented way to reduce the time you need to import a large dataset is by restricting the import to columns of interest. This has the additional benefit of saving memory.

Stata

To read only a subset of columns, you can pass a `varlist` to the `keep()` option from `appendgz`. Using `appendgz` (instead of `append`) we skip the baseline example which we have shown before and read the first 20 columns (`BNGR-KURS`) like this:

```
// create a global macro containing the paths of one week of MiFID data
// assuming that all seven files are located in the working directory
global mifid_dtagz : dir "." files "*.dta.gz", respectcase

// import first 20 columns of data
clear
appendgz using ${mifid_dtagz}, keep(BNGR-KURS)
```

Note: If you import only a single file, `usegz BNGR-KURS using "my_data.dta.gz"` does the

job.

R

`read_parquet()` has a similar argument: `col_select()`. Here, you have a range of different options to specify the the columns of interest. For details see `?arrow::read_parquet`. Reading only the first 20 columns would be achieved by:

```
# load relevant packages
library(arrow)
library(data.table)

# create vector containing the file paths one week of MiFID data
# assuming that all seven files are located in the working directory
mifid_parquet <- dir(pattern = ".parquet")

# import data and assign the result to 'mifid_subset'
mifid_subset <- rbindlist(
  lapply(mifid_parquet, read_parquet, col_select = 1:20)
)
```

Benchmarks

Figure 4 shows that reading only 20 instead of all 46 columns significantly reduces the time needed to import the data. Using Stata, we save 33.9% of the original time. Using R, the speedup amounts to 39.9%. The optimised R solution is 79% faster than the optimised Stata solution.

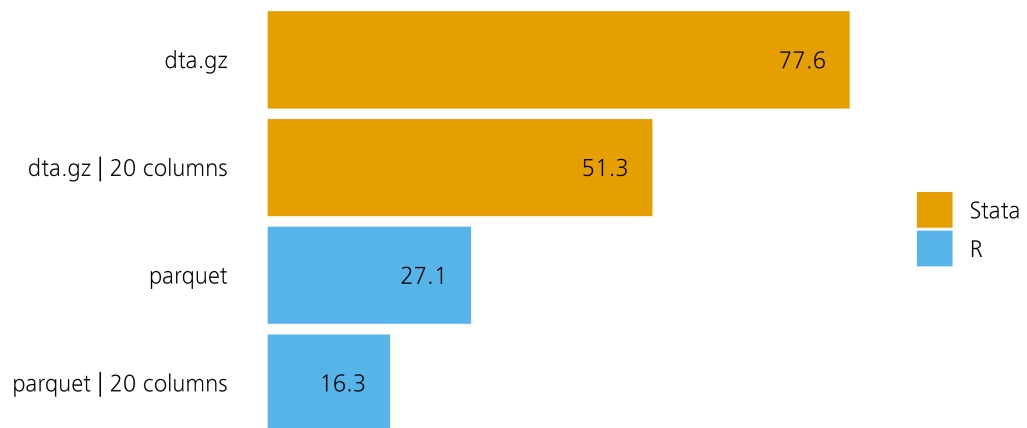


Figure 4: Timings (in seconds) for importing 20 columns from one week of MiFID data. Exact commands below:

```
dta.gz: appendgz using ${mifid_dtagz}
dta.gz | 20 columns: appendgz using ${mifid_dtagz}, keep(BGNR-KURS)
parquet: rbindlist(lapply(mifid_parquet, read_parquet))
parquet | 20 columns: rbindlist(lapply(mifid_parquet, read_parquet,
col_select = 1:20))
```

4 Recommendations on data wrangling

This task is at the heart of each data analysis: Data wrangling. In the context of a research project one could refer to this task as *creating the analysis dataset*.

4.1 Use software packages built for large data

Stata

To increase the data wrangling efficiency we recommend using the {gtools} package (Bravo, 2018), which reimplements a range of Stata commands for data manipulation and statistics in a much faster way. I.e., `gcollapse` replaces `collapse` or `greshape` replaces `reshape`. According to the {gtools} website (<https://gtools.readthedocs.io/en/latest/>), it's commands are often 10 to 20 times faster than their default equivalents. For more information on {gtools} and a list of provided commands see the link above.

Since {gtools} provides a number of commands, we only pick a single one (`gcollapse`) for demonstrational purposes:

```
// see previous Stata code chunks for how the data has been imported

// collapse data
gcollapse (sum) VOLUMEN, by(MP_anon)
```

Here, we collapse the full week of example data and calculate the sum of the transaction volume (`VOLUMEN`) per reporting agent (`MP_anon`). The default Stata command would be identical apart from the `g` at the beginning of the line.

R

For R, we recommend using {data.table}. {data.table} was written to handle “large data (e.g. 100GB in RAM)” and provides all tools needed to mutate, aggregate and join data. You can find an introduction to {data.table} on <https://rdatatable.gitlab.io/data.table/articles/datatable-intro.html> where you also find more advanced topics in the *Vignettes* section.

Here is an example of how to use {data.table} to do the same as `gcollapse` above:

```
# load relevant packages
library(data.table)

# see previous R code chunks for how the data (mifid) has been imported

# collapse 'mifid' data and assign the result to 'mifid_collapsed'
mifid_collapsed <- mifid[, sum(VOLUMEN), by = MP_anon]
```

`mi fid` is the `data.table` created from reading the example week of MiFID data. The base R solution for `data.frames` (hence `df` instead of `mi fid`) is more verbose: `aggregate(df[["VOLUMEN"]], by = list(df[["MP_anon"]]), sum, drop = FALSE)`.

Benchmarks

Since both `{gtools}` and `{data.table}` provide benchmarks on their websites,¹⁰ we refrain from doing many benchmarks on our own. The only benchmark we provide is one on collapsing or aggregating data as described above. The results are depicted in Figure 5. As you can see, `{data.table}` does a terrific job and is 90.1% faster than the optimised Stata solution using `{gtools}` which is already 76.2% faster than the base Stata solution.

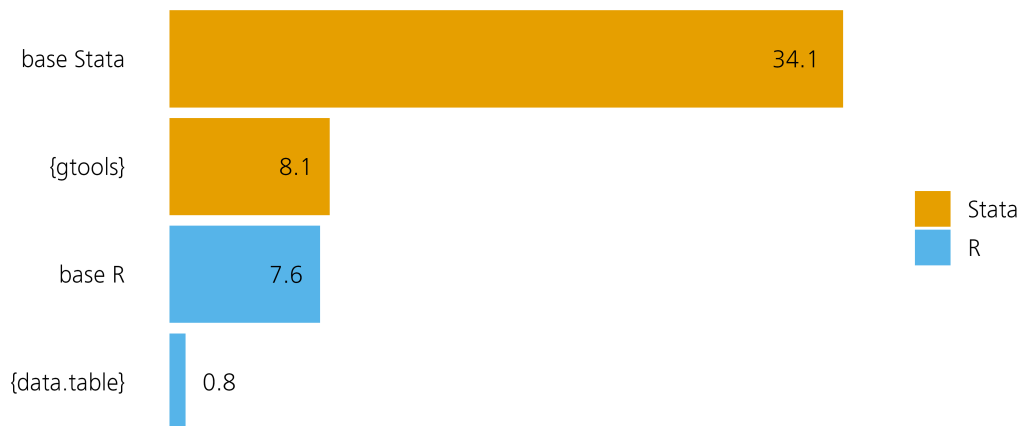


Figure 5: Timings (in seconds) for collapsing data in Stata and R. Exact commands below:
base Stata: `collapse (sum) VOLUMEN, by(MP_anon)`
{gtools}: `gcollapse (sum) VOLUMEN, by(MP_anon)`
base R: `aggregate(df[["VOLUMEN"]], by = list(df[["MP_anon"]]), sum, drop = FALSE)`
{data.table}: `DT[, sum(VOLUMEN), by = MP_anon]`

4.2 Use suitable column types

When working with large data, it is even more important to keep your data as small as possible. This saves memory and disk space and therefore computing time. One possibility to reduce the size of your data is to choose the right type for each column. For example, if a variable `x` contains only whole numbers, store `x` as `integer` instead of `float` (Stata) or `double` (R).

Another aspect of this recommendation is the use of value labels (Stata) and factor variables (R). Especially when your data contains long string variables, these options can drastically reduce its size. But be aware of the fact that this might cause trouble if value labels are assigned inconsistently across several chunks of data.

¹⁰ See <https://gtools.readthedocs.io/en/latest/> and <https://h2oai.github.io/db-benchmark/>.

Stata

Stata comes with the convenient `compress` command. `compress` sets the type of each variable to the minimum required type. Thus, you should `compress` your data at least every time you save it to disk:

```
// here goes your data preparation code

// compress data before saving
compress
savegz "my_data.dta.gz"
```

Note the usage of `{gziptool}'s savegz which saves the data in the compressed dta.gz format.`

See `help table` and `help encode` for documentation on how to convert string variables to numeric without losing information. As said before, this can have an huge impact on the size of your data.

R

In R, we have `type.convert()` from the `{utils}` package (R Core Team, 2020). Compared to Stata's `compress`, it is less rigorous so you might need to revert a more manual approach in order to set optimal column types.

```
# here goes your data preparation code

# compress data before saving
df <- type.convert(df)
write_parquet(df, "my_data.parquet")
```

If you use `data.table`'s (instead of `data.frames`) in your project, you can efficiently convert string variables to factors like this:

```
# take variable 'var' from data.table 'DT' and convert it from
# character to factor
DT[, .(var := factor(var, levels = 1:10, labels = letters[1:10]))]
```

Here, `DT` is a `data.table` and the variable `var` is converted to factor. To make sure that factor levels and labels are consistent across data chunks, we recommend to set these explicitly as in the (arbitrary) example above.

5 General recommendations

5.1 Start with a small subset of your data

Despite all optimisation efforts, your code might still need quite some time to compute all results on a large dataset. Here, it can be particularly annoying if it suddenly breaks after a long runtime due to a programming error and you have to run it all over again and potentially book another visit at the RDSC's premises.¹¹⁾ In order to avoid this and to reduce the amount of time you spend during your code's programming or testing phase we highly recommend starting with a small sample. Hence, you should first test your code on a subset, e.g. on a few days, to ensure that it will not produce any errors due to programming mistakes. If everything went smooth you can extend your analysis to the full dataset.

5.2 Use Virtual Clients

Regardless of the programming language, the RDSC recommends using Virtual Clients (VCs) for resource-intensive tasks. The RDSC's visiting researcher workstations have 4 CPU's, 8 threads and 16 GB RAM. In addition, visiting researchers can apply to access one of ten VC that have 8 CPUs with 16 threads and up to 128 GB of RAM (regular 32 GB) per VC. Especially for resource-intensive tasks it might be beneficial to use the VCs. Additionally, VC's have a much faster connection to the network drive where the data is stored, which speeds up reading and writing tasks.

¹¹ Advanced R users should take a look at {targets} (Landau, 2021) to mitigate this challenge.

6 Conclusion

In this technical report, we provided a number of recommendations for working with large datasets in the RDSC's environment. Due to the availability of the RDSC's software packages for SDC, we limited our analysis to Stata and R. We showed that common tasks have a lot of potential for optimisation.

For example, reconsider the benchmarks on importing data into memory: Figure 6 shows that (using Stata) you can reduce the time to import one week of MiFID data by 73 seconds. Projecting this improvement to the complete MiFID data, you would save roughly 10.6 hours of computing time by changing only one line of code. Note that just importing the data would still take 7.4 hours.

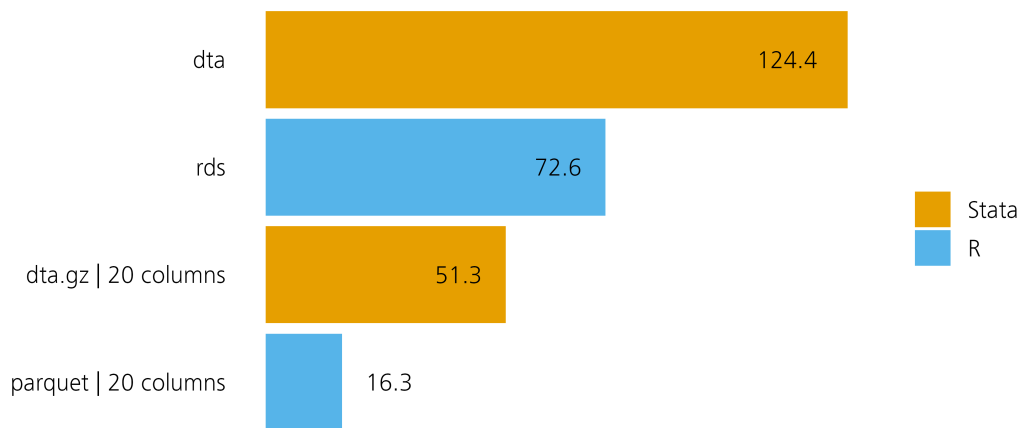


Figure 6: Timings (in seconds) for importing one week of MiFID data without optimisation versus full optimisation. Exact commands below:

```
dta: append using ${mifid_dta}
rds: do.call(rbind, lapply(mifid_rds, readRDS))
dta.gz | 20 columns: appendgz using ${mifid_dtagz}, keep(BGNR-KURS)
parquet | 20 columns: rbindlist(lapply(mifid_parquet, read_parquet,
col_select = 1:20))
```

Analogously, R users could reduce the time needed to import the data from 10.5 to 2.4 hours. Please note that these savings are somewhat theoretical, because the RDSC provides large data only as dta.gz and parquet. Nonetheless, the idea carries over to intermediate datasets stored in your project directory and highlights the fact that small changes in your code can result in large performance gains.

This also shows that common tasks can usually be accomplished much faster using R than using Stata. However, we also pointed out that you should not only focus on computing time alone as shorter computing times might be offset by longer programming time.

Therefore, you need to decide for yourself which tool makes most sense for your research project dealing with large data. If you are a proficient R user, possibly familiar with `{data.table}`, you should definitely prefer R to Stata. Otherwise, there is no simple answer. You should carefully weigh the benefits and the learning curve of programming in a new language.

References

- Bravo, M. C. (2018). *GTOOLS: Stata module to provide a fast implementation of common group commands*. Statistical Software Components, Boston College Department of Economics. Retrieved from <https://ideas.repec.org/c/boc/bocode/s458514.html>
- Cagala, T., Gomolka, M., Krüger, M., & Sachs, K. (2021). *Markets in Financial Instruments Directive (MiFID), Data Report 2021-13 – Metadata ID 1.0*. Deutsche Bundesbank, Research Data; Service Centre. Retrieved from <https://www.bundesbank.de/resource/blob/634886/3414886ff73afe8ce02a009084b812e5/mL/2021-13-mifid-data.pdf>
- Dowle, M., & Srinivasan, A. (2021). *data.table: Extension of 'data.frame'*. Retrieved from <https://CRAN.R-project.org/package=data.table>
- Gomolka, M. (2018). *GZIPUTIL: Stata module to provide access to gzipped files*. Statistical Software Components, Boston College Department of Economics.
- Gomolka, M., & Becker, T. (2021). *sdLog: Tools for Statistical Disclosure Control in Research Data Centers*. Retrieved from <https://CRAN.R-project.org/package=sdLog>
- Gomolka, M., Schäfer, M., & Stahl, H. (2021). *Monthly Balance Sheet Statistics (BISTA), Data Report 2021-10 – Metadata Version BISTA-Doc-v3-0*. Deutsche Bundesbank, Research Data; Service Centre. Retrieved from <https://www.bundesbank.de/resource/blob/747114/29fb691901403642127353c7935116ef/mL/2021-10-bista-data.pdf>
- Landau, W. M. (2021). The targets R package: a dynamic Make-like function-oriented pipeline toolkit for reproducibility and high-performance computing. *Journal of Open Source Software*, 6(57), 2959. Retrieved from <https://doi.org/10.21105/joss.02959>
- R Core Team. (2020). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from <https://www.R-project.org/>
- Research Data and Service Centre. (2021). *Rules for visiting researchers at the RDSC, Technical Report 2021-02 - Version 1-0*. Deutsche Bundesbank, Research Data; Service Centre. Retrieved from <https://www.bundesbank.de/resource/blob/826176/ffc6337a19ea27359b06f2a8abe0ca7d/mL/2021-02-gastforschung-data.pdf>
- Richardson, N., Cook, I., Keane, J., François, R., Ooms, J., & Apache Arrow. (2021). *arrow: Integration to 'Apache' 'Arrow'*. Retrieved from <https://CRAN.R-project.org/package=arrow>
- SAS Institute Inc. (2016). *SAS software 9.4*. Cary, North Carolina.
- StataCorp. (2019). *Stata statistical software: Release 16*. College Station, TX: StataCorp LLC.
- The Mathworks, Inc. (2018). *MATLAB R2018b*. Natick, Massachusetts.
- Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, California: CreateSpace.
- Wickham, H., & Grolemund, G. (2016). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Inc.