Reducing RAM Usage in the Shared Environment

On the servers in the shared environment, also known as the Researcher Machine (Forskermaskinen), all users share the same resources, including RAM (memory). In Python, R, and STATA, data is loaded directly into the RAM, meaning that the servers can become blocked for everyone if individual users load large amounts of data.

Therefore, DST automatically terminates processes on the Python, R, and STATA servers when the RAM is close to being full (p. 1). This can include, for example, RStudio sessions, Jupyter Notebooks, interactive Python sessions in VS Code, STATA-sessions, or jobs loading large files.

It is the users' own responsibility to limit their RAM usage, but here we provide some tips on how to do so when working with large datasets. We include some simple code examples for inspiration in R and Python. (The guide will eventually be expanded with STATA examples).

For further coding inspiration, you can read the documentation for relevant packages/functions, for instance based on the examples here. You can also use the AI programming support on the Researcher Machine (guide).

If your analyses require more system resources than the shared environment can support, you can explore the possibility of using a hosted server (guide) or an HPC analysis platform (guide).

And remember: Save your code often and write data to the disk along the way during your interactive analyses and automated runs. Otherwise, you may lose your work if your process is terminated.

1. Do not load the entire file unless you need to

Before loading a large file, consider:

- Which columns do I need?
- Which rows or periods do I need?
- Can I test on a small sample first?

This is relevant, for example, when you want to explore data, either via code or visually through tools like RStudio's Data Viewer or VS Code's Data Wrangler. Here, you can save RAM by loading a smaller sample of the data.

Example with a SAS file in R

```r
library(haven)

sample <- read_sas(
  "BEF202512.sas7bdat",
  col_select = c(PNR, ALDER, CIVST),
  n_max = 10000
)
```

Using `haven::read_sas()`, you can select columns and limit rows via `col_select`, `n_max`, and `skip`.

Example with a SAS file in Python

```python
import pandas as pd

with pd.read_sas("BEF202512.sas7bdat", iterator=True) as reader:
    sample = reader.read(10_000)
```

Using `pandas.read_sas()`, you can load files in chunks via `iterator=True`.

2. Process large files in chunks

When working with large SAS files, you should avoid loading the entire file into the RAM if the task can be performed in chunks.

Typical tasks suitable for chunk processing include:

- Filtering rows
- Selecting columns
- Counts or summaries
- Splitting data by year, group, etc.

Example with a SAS file in Python

```python
import pandas as pd

filtered_chunks = []

for chunk in pd.read_sas("BEF202512.sas7bdat", chunksize=100_000):
    subset = chunk.loc[chunk["ALDER"] < 18, ["PNR", "KOM"]]
    subset["kbh"] = subset["KOM"] == b"101"
    filtered_chunks.append(subset)

df = pd.concat(filtered_chunks, ignore_index=True)
```

`pandas.read_sas()` supports loading in chunks via `chunksize`.

3. Use filtered loading via Parquet files

You can save RAM by converting large files to Parquet and using the RAM-saving loading methods that this file format offers.

In R and Python, you can restrict Parquet files to specific columns and rows before the data is loaded into the RAM. Widely used tools for this include `polars` in Python and the combination of `arrow` and `dplyr` in R.

You can convert files to Parquet using the program StatTransfer, which is available on all servers in the shared environment.

Example in R

```r
library(arrow)
library(dplyr)

ds <- open_dataset("BEF202512.parquet")


result <- ds |>
  filter(ALDER < 18) |>
  select(PNR, KOM) |>
  mutate(kbh = KOM == 101) |>
  collect()
```

Using `arrow::open_dataset()`, you can build a `dplyr` pipeline that applies `filter` and `select` directly during the loading of the Parquet file from the disk.

Example in Python

```python
import polars as pl

result = (
    pl.scan_parquet("BEF202512.parquet")
      .filter(pl.col("ALDER") < 18)
      .select(["PNR", "KOM"])
      .with_columns(kbh=pl.col("KOM") == "101")
      .collect()
)
```

Using `polars.scan_parquet()`, filters and column selection are applied before the data is loaded into the RAM.

4. Clean up your interactive sessions

Interactive programming, for instance in RStudio, VS Code, and Jupyter Notebook, occupies RAM for as long as the session remains open. Examples of best practices:

- Close notebooks and RStudio sessions that are not in use.
- Close your old session and start a new one when beginning a new task.
- Delete large objects from the RAM when they are no longer needed.