Reducing RAM Usage in the Shared Environment

On the servers in the shared environment, also called Forskermaskinen, all users share the same resources, including RAM (memory). In Python, R and STATA, data are loaded directly into RAM, so it can block the servers for everyone if individual users load large amounts of data.

Therefore, DST automatically closes processes on the Python, R and STATA servers when RAM is close to being full. This can include RStudio sessions, Jupyter Notebooks, interactive Python sessions in VS Code, STATA sessions, or jobs that load large files.

Users are responsible for limiting their own RAM usage, but here we provide some tips for how to do this when working with large datasets. We include some simple code examples for inspiration in R, Python and STATA.

For further inspiration, you can read the documentation for relevant packages/functions, e.g. using our examples as a starting point. You can also use the AI coding support on Forskermaskinen (guide).

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

You can check your current RAM usage by opening the Taskmanager shortcut on the server desktop. Select the Users tab, find your project ID, and view the RAM usage under Memory.

Remember: Save your code often, and write data to disk during your interactive analyses and automated runs. Otherwise, you may lose your work if your process is closed.

1. Do not load the whole file unless you need it

Before loading a large file, consider: Which columns do I need? Which rows or periods do I need? Can I test on a small subset first?

This is relevant, for example, when you want to explore data, either via code or visually via, for example, RStudio's Data Viewer, VS Code's Data Wrangler or STATA's Data Editor. Here you can save RAM by loading a smaller subset of 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
)
```

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

Example with a SAS file in Python

```
import pandas as pd

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

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

Example with a SAS file in STATA

```
import sas PNR ALDER CIVST in 1/1000 using BEF202512.sas7bdat
```

Using the variable list and `in`, you can limit column names and row numbers in `import sas`.

```
import sas PNR KOM if ALDER < 18 using BEF202512.sas7bdat
```

You can add `if` conditions to limit based on the values of specific columns.

2. Process large files in chunks

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

Typical tasks suitable for chunked processing include:

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

Example with a SAS file in 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

In R and Python, you can save RAM by converting large files to Parquet and limiting columns and rows before data are loaded into RAM. Common tools for this include `polars` in Python and the combination of `arrow` and `dplyr` in R. In STATA, you can use filtered loading for SAS and STATA files (see Tip no. 1).

Note: You can convert to Parquet or STATA (.dta) in StatTransfer, which is available on all servers in the shared environment. Use StatTransfer Command Processor to convert multiple files at once.

Example in R

```
library(arrow)
library(dplyr)

ds <- open_dataset("BEF202512.parquet")

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

Using `open_dataset()`, you can build a `dplyr` pipeline that applies `filter` and `select` when reading the Parquet file. Use `%in%` or `semi_join` to filter based on a list of, for example, CPR numbers.

Example in 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()
)
```

With `scan_parquet()`, filters and column selection are applied when reading the Parquet file, before the data are loaded into RAM. You can use wildcards in the file name, e.g. `"BEF*.parquet"`, to read all BEF files in the folder. Use `is_in` or `join` to filter based on a list of, for example, CPR numbers.

Example in STATA

STATA itself supports filtered loading of SAS and STATA files (see Tip no. 1). If you use Parquet files in STATA, you can perform filtered loading using the `pq` package.

```
pq use PNR KOM using BEF202512.parquet, if (ALDER < 18)
```

4. Clean up your interactive sessions

Interactive programming, e.g. in RStudio, VS Code, Jupyter Notebook and STATA, uses RAM as long as the session is open. You can therefore save RAM by cleaning up regularly. Examples of good practice:

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

5. Specific to STATA: Set an upper limit on RAM usage

In STATA, you can set a limit for how much RAM your session uses:

```
set max_memory 32g
```

This sets a limit of 32 GB, for example. If you run a command that requires more than 32 GB of RAM, the run is interrupted with the message `attempt to use too much memory`.