Reduktion af RAM-forbrug i det fælles miljø

På serverne i det fælles miljø, også kaldet Forskermaskinen, deles alle brugere om de samme ressourcer, herunder RAM (hukommelse). I Python, R og STATA indlæses data direkte i RAM, så det kan blokere serverne for alle, hvis enkelte brugere indlæser store mængder data.

Derfor lukker DST automatisk processer på Python-, R- og STATA-serverne, når RAM'en er tæt på at blive fyldt. Det kan fx omfatte RStudio-sessioner, Jupyter Notebooks, interaktive Python-sessioner i VS Code, STATA-sessioner, eller jobs, der indlæser store filer.

Det er brugernes eget ansvar at begrænse deres RAM-forbrug, men her giver vi nogle tips til hvordan man kan gøre det, når man arbejder med store datasæt. Vi inkluderer nogle simple kodeeksempler til inspiration i R og Python. *(Vejledningen bliver på sigt udvidet med STATA-eksempler)*

For videre inspiration til kodning kan du læse dokumentationen for relevante pakker/funktioner, fx med afsæt i eksemplerne her. Du kan også bruge AI-programmeringsstøtten på Forskermaskinen (vejledning).

Hvis dine analyser kræver flere systemressourcer end det fælles miljø kan bære, så kan du undersøge muligheden for at bruge en hosted server (vejledning) eller en HPC-analyseplatform (vejledning).

Og husk: Gem din kode ofte, og skriv data til disken undervejs i dine interaktive analyser og automatiske kørsler. Ellers kan du miste du dit arbejde, hvis din proces bliver lukket.

1. Indlæs ikke hele filen, medmindre du har brug for det

Før du indlæser en stor fil, så overvej:

- Hvilke kolonner har jeg brug for?
- Hvilke rækker eller perioder har jeg brug for?
- Kan jeg teste på et lille udsnit først?

Det er for eksempel relevant når du vil udforske data, enten via kode eller visuelt via fx RStudio's Data Viewer eller VS Code's Data Wrangler. Her kan du spare RAM ved at indlæse et mindre udsnit af data.

Eksempel med SAS-fil i R

```
library(haven)

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

Med `haven::read_sas()` kan du vælge kolonner og begrænse rækker via `col_select`, `n_max` og `skip`.

Eksempel med SAS-fil i Python

```
import pandas as pd

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

Med `pandas.read_sas()` kan du indlæse filer i bidder via `iterator=True`.

2. Behandl store filer i bidder

Når du arbejder med store SAS-filer, bør du undgå at indlæse hele filen i RAM, hvis opgaven kan udføres i bidder.

Typiske opgaver der egner sig til behandling i bidder er fx:

- Filtrering af rækker
- Valg af kolonner
- Optællinger eller opsummeringer
- Opdeling af data efter år, gruppe o.l.

Eksempel med SAS-fil i 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()` understøtter indlæsning i bidder via `chunksize`.

3. Brug filtreret indlæsning via Parquet-filer

Du kan spare RAM ved at konvertere store filer til Parquet, og bruge de RAM-besparende indlæsningsmetoder som dette filformat tilbyder.

I R og Python kan du afgrænse Parquet-filer på kolonner og rækker *før* data indlæses i RAM. Udbredte værktøjer til dette er fx `polars` i Python, og kombinationen af `arrow` og `dplyr` i R.

Du kan konvertere til Parquet i programmet StatTransfer, som findes på alle servere i det fælles miljø.

Eksempel i R

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

ds <- open_dataset("BEF202512.parquet")

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

Ved hjælp af `arrow::open_dataset()` kan du bygge en `dplyr`-pipeline, som anvender `filter` og `select` i selve indlæsningen af Parquet-filen fra disken.

Eksempel i 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()
)
```

Med `polars.scan_parquet()` anvendes filtre og kolonnevalg før data indlæses i RAM.

4. Ryd op i dine interaktive sessioner

Interaktiv programmering, fx i RStudio, VS Code og Jupyter Notebook, optager RAM så længe sessionen er åben. Eksempler på god praksis:

- Luk notebooks og RStudio-sessioner, der ikke bruges
- Luk din gamle session og start en ny, når du starter på en ny opgave
- Slet store objekter i RAM, når de ikke længere skal bruges