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, Python og STATA.

For videre inspiration 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, VS Code's Data Wrangler eller STATA's Data Editor. Her kan du spare RAM ved at indlæse et mindre udsnit af data.

Eksempel med SAS-fil i R

```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`.

Eksempel med SAS-fil i STATA

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

Ved hjælp af variabellisten og `in` kan du afgrænse kolonnenavne og rækkenumre i `import sas`.

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

Du kan tilføje `if`-betingelser for at afgrænse på værdierne af specifikke kolonner.

2. Behandl store filer i bidder

Når du arbejder med store 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

I R og Python kan du spare RAM ved at konvertere store filer til Parquet, og afgrænse 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. I STATA kan du bruge filtreret indlæsning på SAS- og STATA-filer (se Tip nr. 1.)

OBS: Du kan konvertere til Parquet eller STATA (.dta) i StatTransfer, som findes på alle servere i det fælles miljø. StatTransfer Command Processor kan bruges til at konvertere flere filer ad gangen.

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 `open_dataset()` kan du bygge en `dplyr`-pipeline, som anvender `filter` og `select` i læsningen af Parquet-filen. Brug `%in%` eller `semi_join` til at filtrere ud fra en liste af fx CPR-numre.

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 `scan_parquet()` anvendes filtre og kolonnevalg i læsningen af Parquet-filen. Du kan bruge wildcards i filnavnet, fx `"BEF*.parquet"` for at læse alle BEF-filer i mappen. Brug `is_in` eller `join` til at filtrere ud fra en liste af fx CPR-numre.

Eksempel i STATA

STATA understøtter i sig selv filtreret indlæsning af SAS- og STATA-filer (se Tip nr. 1). Hvis du bruger Parquet-filer i STATA kan du lave filtreret indlæsning ved hjælp af `pq`-pakken.

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

4. Ryd op i dine interaktive sessioner

Interaktiv programmering, fx i RStudio, VS Code, Jupyter Notebook og STATA, optager RAM så længe sessionen er åben. Du kan derfor spare RAM ved at rydde op løbende. Eksempler på god praksis:

- Luk notebooks, RStudio-sessioner og STATA-vinduer, 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

5. Specifikt for STATA: Sæt øvre grænse på RAM-forbrug

I STATA kan du sætte en grænse for hvor meget RAM din session bruger:

```
set max_memory 32g
```

Her sættes fx en grænse på 32 Gb. Hvis du kører en kommando som kræver mere end 32 Gb RAM, så afbrydes kørslen med beskeden `attempt to use too much memory`.