How to work smarter – efficient completion of international questionnaires using R and Python

Jan Sulavik, Statistics Norway, jan.sulavik@ssb.no

Abstract

An important part of statistics production in the Nordic countries is the delivery of national data through questionnaires to international organisations such as Eurostat. However, these questionnaires are often Excel-based with a strictly defined structure, which often implies time-consuming manual completion. This paper presents a modernised and automated completion of road vehicle kilometres and transport equipment data for the Eurostat/ITF/UNECE Common Questionnaire for Inland Transport Statistics (CQ) recently introduced by Statistics Norway.

Previously, the completion was very time-consuming. Vehicle kilometres data were produced in a complex Excel file, and the completion of transport equipment data was so resource-demanding that it was outsourced to the Norwegian Public Roads Administration. Moreover, the completion lacked transparency and could be prone to errors. Now, Excel file for vehicle kilometres has been replaced with an R program. This simultaneously improves the transparency, reduces the time use, and removes the dependency on proprietary software. R was chosen because it among other things facilitates use of multiple data sources necessary for this delivery.

For transport equipment, Python was chosen, particularly the Polars module, among other things because it efficiently processes large data amounts. Statistics Norway could thus reassume full responsibility for this delivery without requiring more resources, while making the production simultaneously transparent, fast, and maintainable.

R- and Python-based CQ data production was run successfully on both local on-premises data platform and the new cloud-based platform. In consequence, this approach is technology-neutral and can be used across different data platforms. Moreover, both R and Python are open source, which facilitates sharing and maintenance of the production code. Even though only two specific examples are presented here, this approach can be relevant for international questionnaire completion for Nordic statisticians regardless of the data platform used (local or cloud), programming language preferred (R or Python), or statistical domain.

Keywords: open-source code, international questionnaires, transparency, cross-platform, automation

1. Introduction

In an interconnected world of today, statistical data at multiple levels are important to provide a reliable basis for science, decision making and public debate. National statistical institutes (NSIs) collect and analyse data at subnational, e.g., municipal, and national levels, while multiple international organisations such as the European Statistical Office (Eurostat) or the OECD collect and analyse data at higher levels.

Nordic countries are through their EU or EFTA memberships integrated in the European Statistical System, and their NSIs have particular, not the least legal, obligations regarding data delivery to Eurostat and other international organisations [1]. This is thus an important part of the statistics production in their NSIs. Data to international organisations are often delivered through standardised spreadsheet questionnaires, in case of Eurostat uploaded to the EDAMIS system.

Rigid questionnaire structure facilitates standardisation across the different NSIs and automatic validation of uploaded data. However, it also constrains possibilities for automatic completion. In many cases, data are filled in the questionnaires manually or only with minimal automation, which poses three main challenges:

The first is the workload: national statistics production is often demanding, and potentially time-consuming manual questionnaire completion places an additional burden on the statisticians. The second is the transparency and verifiability: with increasing number of manual steps involved, the probability for errors, typos, etc., in the delivered data increases, while the transparency of the completion and verifiability of the data decreases. The third is the flexibility: questionnaires can and do change over time, and retrospective re-submissions of data according to new definitions may prove very difficult if many manual steps were involved in the original delivery.

To avoid manual completion and address these challenges, I developed automated workflows for completion of the Eurostat/ITF/UNECE Common Questionnaire for Inland Transport Statistics (CQ): an R-based workflow for road vehicle kilometres and a Python-based workflow for road transport equipment.

2. Methods

a. Status quo before the new workflows

Before the adoption of the new workflows, road vehicle kilometres' data for CQ were produced in a complex Excel file with main input from internal data from Road traffic volumes statistics [2]. This was already a notable improvement from pure manual completion. However, it was challenging to gain a full overview of a process based on multiple Excel sheets with numerous internal connections and dependencies. It was therefore difficult to maintain and develop the process and to verify the results.

Road transport equipment data for CQ were previously not produced by Statistics Norway, but by Norwegian Public Roads Administration (SVV), owner of the Norwegian vehicle register. I have limited knowledge on methods used by the SVV, but generally, they ran Structured Query Language (SQL)-queries directly from the register. It is unclear why the CQ completion was outsourced to the SVV, i.e., why internal data from Registered vehicles statistics [3] were not used, but presumably due to the associated workload and lack of resources at Statistics Norway.

A prerequisite for the new workflows was an overhaul of Road traffic volumes and Registered vehicles statistics' production consisting of programme translation from SAS to Python and transition from proprietary SAS-data, i.e., sas7bdat, to parquet [4], the new standard data storage format at Statistics Norway. The overhaul and SAS-to-Python translations are beyond the scope of this paper, but at any rate, without the transition to parquet data storage, the new workflows would not have been possible.

b. Common principles of the new workflows

Even though I for specific reasons based the two workflows on two different programming languages, both shared the same common principles:

1) Collect the whole workflow in a single script
2) Use internal data as the main input
3) Use generic and row-specific filters with loops
4) Delay eager query executions as much as possible
5) Format the output exactly as the questionnaire demands

Collecting the workflow in a single script improved the transparency and maintainability. Use of internal data as the main input improved the consistency and verifiability. Use of generic filters facilitated code-reuse, use of row-specific filters increased the flexibility, and delaying of eager query executions reduced the random-access memory (RAM) usage and increased the execution speed. Finally, formatting the output in the exact CQ row-order reduced the overall workload and risk of errors. We will return to these principles when describing the workflows in detail.

c. R-based workflow for road vehicle kilometres

For road vehicle kilometres, I used R. The main reasons were my previous knowledge of this language, and the necessity of including multiple data sources, such as motorcycle and moped data from the report Transport volumes in Norway [5] or bicycle data from the National travel habit survey [6], in addition to the main input, internal data from Road traffic volumes' statistics. The RStudio integrated development environment (IDE) considerably facilitates the use and overview of multiple datasets.

I wrote the whole workflow in a single Quarto Markdown Document (qmd). I chose the qmd-script format over the standard r-script for its markdown support and readability. I used package 'renv' for package management [7]. For treatment of in-memory data, I used 'tidyverse' package collection [8], particularly package 'dplyr'. To delay eager query executions, I used DuckDB R client through 'duckdb' package [9]. With DuckDB, I could create an in-memory database through R database interface from 'DBI' package [10]. Subsequently, I could utilise the properties of parquet to create multiple Views – or virtual datasets – of the main input in the in-memory database.

The main input contained almost 3.8 million rows, corresponding to all vehicles with mandatory roadworthiness tests in Norway, their specifications and yearly driven distances. Eager query execution on the whole dataset would unnecessarily exhaust the available RAM. Instead, I could pre-filter the Views virtually, and only execute the final query step, summation of the yearly driven distances, eagerly. I used generic and row-specific filters, stored as SQL expressions in R lists. Generic filters were reusable, e.g., "vehicle type passenger car", "vehicle age group 1", etc., while row-specific filters

created virtual populations corresponding to specific questionnaire rows, e.g., “row 135” = “passenger car & age group 1”, etc., and comprised about a ⅓ of the script.

I ran a loop of SQL-queries summing the yearly driven distances from the virtual populations, using the row-specific filters in the SQL ‘WHERE’ clause. Only the distance summation was eagerly executed and the sums stored in a data frame, which simultaneously reduced the RAM use and increased the execution speed.

Afterwards, I added distance sums for motorcycles and mopeds since these have no mandatory roadworthiness tests in Norway and lacked in the main input. I thus obtained all data for Table III of the CQ. For Table I, I used data from Table III and additional data on bicycle use. For Table II, I used data from Table I and allocation keys from another Division at Statistics Norway. With all three tables finished, I finalised the output by attaching Eurostat codes to the row numbers. The output was exported as a csv-file with row order matching that of the CQ. Its completion could then be finished by simple pasting of values in Excel. This was one of few manual steps in the completion, together with a few single-value inputs in the script.

d. Python-based workflow for road transport equipment

For road transport equipment, I used Python, particularly module Polars, to get more familiar with this language and utilise Polars’ native handling of parquet data in both eager and non-eager, so-called “lazy”, modes. I could thus use the same syntax across the whole workflow. Moreover, only the main input, internal data from Registered vehicles’ statistics, was involved, hence no need to handle multiple input datasets.

I wrote the whole workflow in a single py-script. JupyterLab IDE can render py-scripts as Notebooks with markdown for better code structuring and readability. I used ‘poetry’ for module and dependency management [11], and module ‘polars’ for most of the workflow [12]. Like DuckDB’s Views, Polars can create virtual datasets, LazyFrames.

With almost 11.9 million rows in the main input, all vehicles in the Norwegian vehicle register, it was even more important to delay eager query executions. I used similar solution as previously, with generic and row-specific filters as Polars expressions in Python dictionaries, which comprised about ⅔ of the script. Adjusted LazyFrame of the main input was sent through three consecutive loops: for number of vehicles, for

number of seats and for total load capacity. I joined output DataFrames from these loops to a single DataFrame and coalesced loop value columns to a single column. Subsequently, I attached Eurostat codes to the row numbers and exported the output as a csv-file. I finished the questionnaire completion by pasting the values to the Excel questionnaire. This was the only manual step involved in the completion. For an overview of the Python-based workflow, see Fig. 1A in the Appendix.

3. Results

I used both workflows successfully on both Statistics Norway's data platforms: local on-premises data platform and new cloud-based data platform. There were no noticeable differences in execution or performance. This data platform neutrality was facilitated by the availability of both IDEs (RStudio and JupyterLab), access to GitHub code storage, and similar data access solutions (Linux storage on-premises and FUSE adapter for Google Cloud Storage [13] on the cloud) on both platforms.

Regarding the workload, the initial writing of the scripts required the largest investment in terms of time. This took several weeks, in part due to my limited knowledge of SQL and Python at that time. However, this time was well invested and provided valuable skills and routines for future work. Subsequent code adjustment could be done relatively quickly, and running the workflows and questionnaire completion itself required only a fraction of the time used previously.

For road vehicle kilometres, running the R-based workflow, completion of the CQ and upload to EDAMIS could be done within a single day, compared to several days or even weeks used previously. I am currently[1] rewriting the R-based workflow to reflect the new streamlined CQ definitions, using package 'duckplyr' [14]. The aim is to further simplify the script, replace SQL with 'rlang' expressions, and make the workflow even more maintainable. For road transport equipment, it is difficult to evaluate the workload reduction as the completion was outsourced before. At any rate, the new Python-based workflow allowed for Statistics Norway to reassume the responsibility for the CQ completion, which also usually requires a single day. However, the most recent

[1] June 2026

completion involving full rewriting of the workflow according to the new streamlined CQ definitions took slightly longer — about a day and a half.

In terms of transparency and verifiability, the new workflows meant a clear improvement. For road vehicle kilometres, I replaced complex multi-sheet Excel file with a single qmd-script with specific and transparent definitions for each CQ row value. Moreover, code version control with git also allows for retrospective re-examinations of previously used scripts and definitions [15]. For road transport equipment, the improvement was even greater as the new Python-based workflow gave Statistics Norway full control over the whole CQ completion process.

The new R- and Python-based workflows considerably increased the flexibility of CQ completion. Use of generic filters in the workflows allows for simple adjustments to any changes in the input, while the row-specific filters provide adjustable definitions for the output. Reduction of manual steps to a necessary minimum combined with rapid execution and code collection in single scripts give full flexibility to re-define and re-run the workflows for a single or multiple statistical periods whenever necessary.

4. Discussion

The delivery of national data to international organisations is an important part of statistics production in the Nordic countries. However, it should not be a burden.

In this paper, I present automated questionnaire completion using R- and Python-based workflows. Both programming languages are open source, which increases the code maintainability. Both are also being adopted by Nordic and other European NSI's, which gives possibilities for future code sharing. In this context, I perceive Eurostat's reliance on proprietary Excel for CQ as disputable. A switch to generic formats, e.g., csv or dat, used in Statistics Norway's Statbank [16], could simplify CQ data deliveries.

Even though only two specific examples are presented here, this approach to questionnaire completion can be relevant for Nordic statisticians regardless of the data platform used (local or cloud), programming language preferred, or statistical domain. In my experience, the initial workload costs of script writing are paid off manifold in future workload reduction and clear increase in transparency, verifiability and flexibility. The result is simply a more efficient and smarter way to work.

5. Appendix: List of references

[1] “The Statistics Act,” *SSB*, Aug. 12, 2021. https://www.ssb.no/en/omssb/ssbs-virksomhet/styringsdokumenter/statistikkloven (accessed May 13, 2026).

[2] “Road traffic volumes,” *SSB*. https://www.ssb.no/en/transport-og-reiseliv/landtransport/statistikk/kjorelengder

[3] “Registered vehicles,” *SSB*. https://www.ssb.no/en/transport-og-reiseliv/landtransport/statistikk/bilparken

[4] “Apache Parquet,” *Apache.org*, 2018. https://parquet.apache.org/

[5] B. L. Flotve, “Transport volumes in Norway 1946-2024 - Transportøkonomisk institutt,” *Transportøkonomisk institutt*, May 13, 2025. https://www.toi.no/publications/transport-volumes-in-norway-1946-2024-article39213-29.html (accessed May 13, 2026).

[6] “Den nasjonale reisevaneundersøkelsen,” *Statens vegvesen*, 2016. https://www.vegvesen.no/vegprosjekter/nasjonal-transportplan/den-nasjonale-reisevaneundersokelsen/om-den-nasjonale-reisevaneundersokelsen/ (accessed May 13, 2026).

[7] “Project Environments [R package renv version 1.1.4],” *R-project.org*, Mar. 2025, doi: https://cran.r-project.org/package=renv.

[8] “Tidyverse,” *Tidyverse.org*, 2025. https://tidyverse.org/

[9] “DBI Package for the DuckDB Database Management System [R package duckdb version 1.5.2],” *R-project.org*, Apr. 2026, doi: https://cran.r-project.org/package=duckdb.

[10] “R Database Interface [R package DBI version 1.2.3],” *R-project.org*, Jun. 2024, doi: https://cran.r-project.org/package=DBI.

[11] “Poetry - Python dependency management and packaging made easy,” *python-poetry.org*. https://python-poetry.org/

[12] “polars,” *PyPI*, Apr. 11, 2025. https://pypi.org/project/polars/

[13] “Cloud Storage FUSE,” *Google Cloud Documentation*, 2025. https://docs.cloud.google.com/storage/docs/cloud-storage-fuse/overview

[14] “A DuckDB-Backed Version of dplyr,” *Tidyverse.org*, 2022. https://duckplyr.tidyverse.org/index.html (accessed Jun. 15, 2026).

[15] Git, “Git,” *Git-scm.com*, 2024. https://git-scm.com/

[16] “Statbank Norway,” *SSB*, Jun. 21, 2021. https://www.ssb.no/en/statbank/

6. Appendix: Code examples and figures

a. *R-based workflow: examples of generic filters in SQL*

```
g_f_tab3 <- list(
        kjtg_pbil = "biltype == '1'", # passenger cars
        aldr_gr1 = "alderbil < 2", # age group 1, vehicle age under 2 years
            ) # generic filters as SQL expressions in R list
```

b. *R-based workflow: an example of a row-specific filter in SQL*

```
s_f_tab3 <- list (
        filt_L135 = str_glue("{g_f_tab3$kjtg_pbil} AND {g_f_tab3$aldr_gr1}"), # passenger cars & age group 1, corresponds to CQ ROADVKM row 135: "Passenger cars/Vehicle-km (Millions)/By age of vehicle/Under 2 years"
                ) # row-specific filters as composite SQL expressions in R list
```

c. *R-based workflow: an example of looped summation of yearly driven distances*

```
output_tab3 <- vector("list", length(s_f_tab3)) # empty output storage list
for (i in 1:length(s_f_tab3)) {
        lengde_naa <- lengde_naa + 1 # simple tracking of loop progress
        sql_i <- str_glue("SELECT SUM (kmar) # sum of 'kmar', yearly driven distances in km
                        FROM prodfil_view # DuckDB View of internal data
                        WHERE {s_f_tab3[[i]]} AND aargang=={stataar}" # virtual filtering by the row-specific filter and statistical year
                        ) # full row-specific SQL-query
        output_i <- dbGetQuery(con, sql_i) # eager execution of the full row-specific SQL-query
        output_i <- as.numeric(round(output_i/1000000, 0)) # CQ values are in millions km
        output_tab3[[i]] <- output_i # storage of row-specific output value in the output storage list
                        } # R for-loop traversing the list with row-specific filters
```

d. *Python-based workflow: examples of generic filters as Polars Expressions*

```
gen_filt = {
        "pbil_tekn" : pl.col("TEKN_TKNAVN").is_in(["M1","M1G"]), # passenger cars
        "aldr_gr1"  : pl.col("alder").le(2), # age group 1, vehicle age up to 2 years
          } # generic filters as Polars expressions in Python dictionary
```

e. *Python-based workflow: an example of a row-specific filter as a Polars Expression*

```
spes_filt = {
        "filt_L27" : gen_filt["pbil_tekn"] & gen_filt["aldr_gr1"], # passenger cars & age group 1,
        corresponds to CQ ROAD row 27: "Passenger cars/Number at 31.12 (Unit)/By age/<= 2
        years"
        } # row-specific filters as composite Polars expressions in Python dictionary
```

f. *Python-based workflow: an example of looped summation of number of vehicles*

```
dict_output = [] # empty output storage list
for i in range(18,168):
        key = f"filt_L{i}"
        filter_expr = spes_filt[key]
        output_i = (lf_kjreg_A    # LazyFrame of internal data, registered vehicles
                    .filter(filter_expr) # virtual filtering by the row-specific filter
                    .select(pl.len()) # select LazyFrame height (i.e., number of vehicles)
                    .collect() # eager execution of the select-query
                    .item() # row-specific output value as a scalar

                    )
        dict_output.append({"Statistikkår": stataar, "Linje": f"L{i}", "Verdi_loop1": output_i})
        # appends the row-specific value in the output storage list

        # Python for-loop; range of CQ row numbers for registered vehicles + 1
```

g. *Figure 1A: An overview of the Python-based workflow*

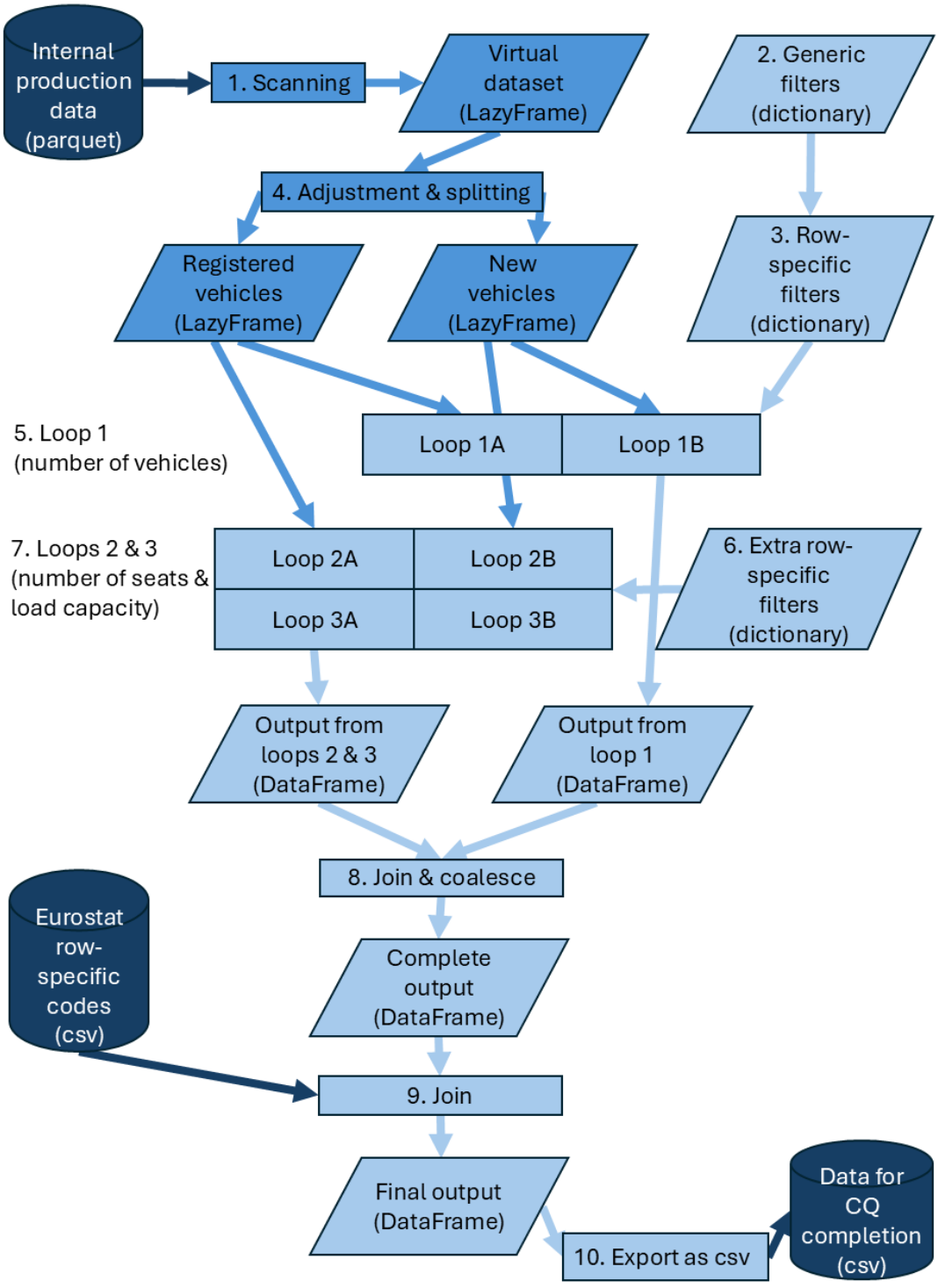

Figure 1A. An overview of the Python-based workflow. Cylinders are physically stored data (datatype in parentheses), lozenges are in-process data (datatype in parentheses), rectangles and arrows are processes. Darkest blue denotes physically stored data and associated processes, dark blue virtual in-process data and light blue in-memory in-process data.