EL6: Data formats

data.table, SQL, DuckDB, parquett, SAS, JSON, API

Reading

- [1, ch. 2] on Databases: it is important to understand some basic concepts. You might, however, ignore sections about "(Hyper)graph databases". Try to understand the SQL code even though you might not need to write such code on your own.
- H. Wickham, M. Çetinkaya-Rundel, and G. Grolemund [2] chap 21 describes how to connect to a database from R.
- S. R. Fenk, K. Furu, and I. J. Bakken [3] describe why the current common practice with register data delivery in SAS format should be replaced by parquet files. Until this has happened, we often need to use SAS for data extraction.
- [4] introduces the `data.table` syntax, its general form, how to *subset* rows, *select and compute* on columns, and perform aggregations *by group*.

Recommended references:

- V. Arel-Bundock [5] provides a good comparison between base R, dplyr and data.table.

Optional practice

We won't focus on SQL in this course but it might still be good to know some of the basics. Two useful resources for your own practicing (if you like):

- SQLzoo
- The SQL Tutorial for Data Analysis

Imaginaed scenario

- You have requested register data from Statistics Sweden (SCB) and The National Board of Health and Welfare (NBHW; SoS)
- Aftar a couple of months (it previously took a year), you receive some huge files with file ending `.sas7bdat`
- What do you do now?

.sas7bdat

- Many govermental agencies are using SAS
- Their standard format when delivering big data sets
- SAS is great for handling big data sets (no need to read everything into memory)
- But you don't know how to use SAS 😩

First try

- It is often possible to read medium-sized SAS files to R by `haven::read_sas()`
- Based on the ReadStat C library
- Based on reverse-engineering of binary files!
- You need to put some trust that those wizards where good "hackers" (your analyzis and future patients lifes may depend on it) :-)
- But similar trust always applies when you use open source software, which depend on other peoples software, which depends on ...
- But, yes, it is legal according to EU legislation :-)

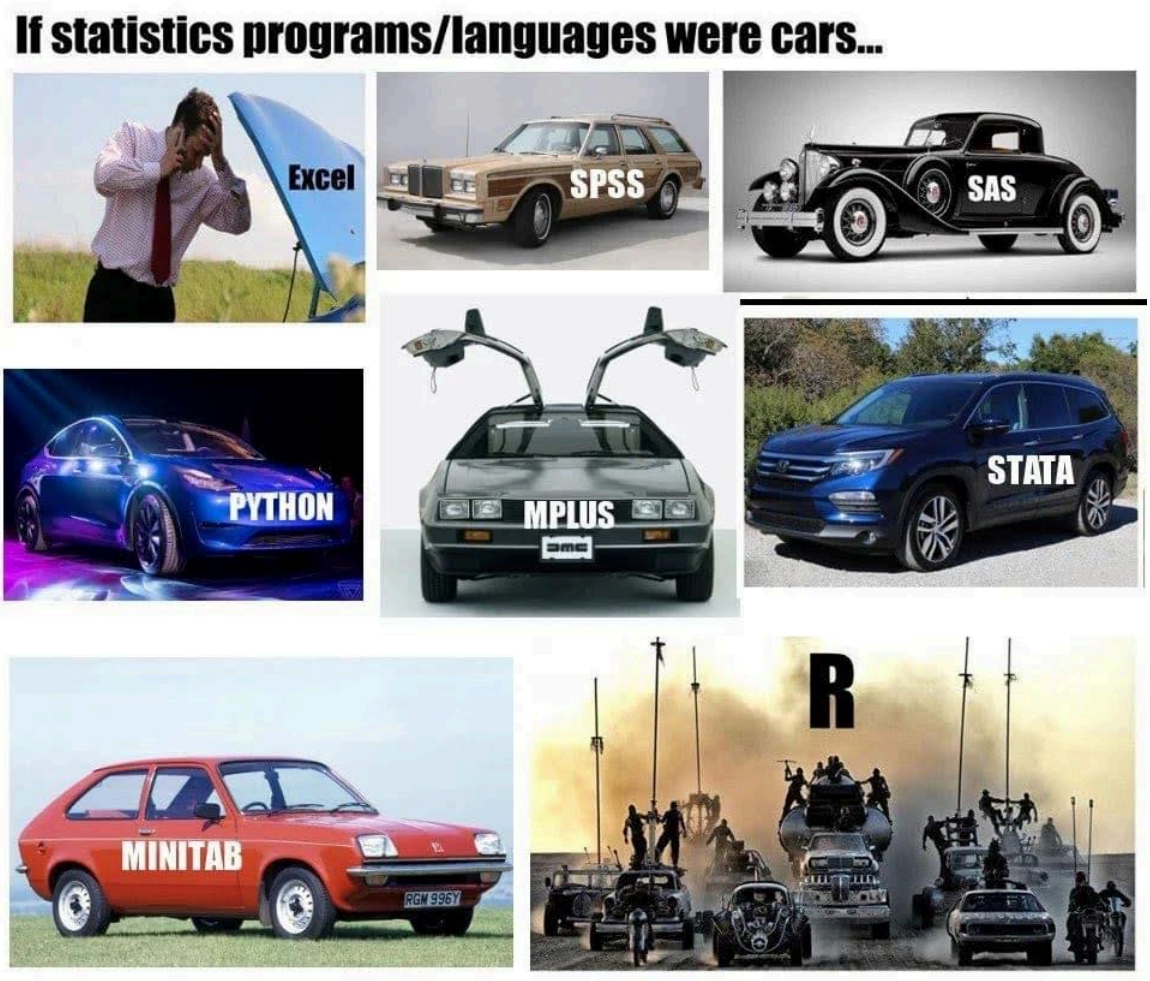
> Directive 2009/24/EC art 6 The authorisation of the rightholder shall not be required where reproduction of the code and translation of its form within the meaning of points (a) and (b) of Article 4(1) are indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs [...]

But let's say this did not work for your data! 😩

Large SAS files

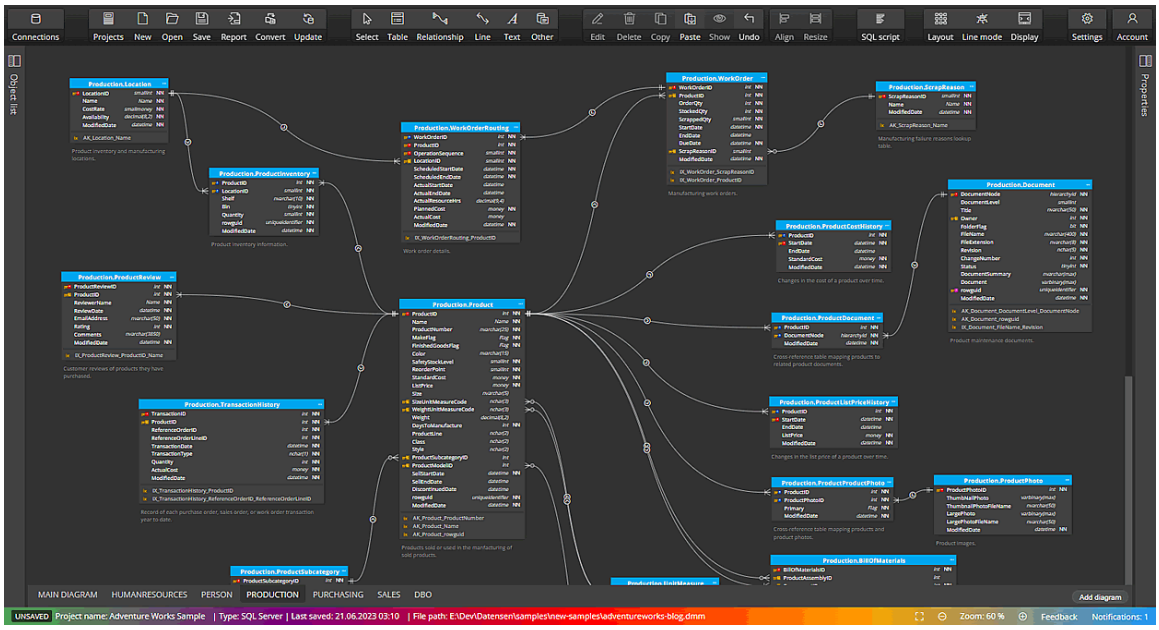
- Best practice for current SAS version (9.4) is to export only nessecary data to csv
  - SAS Viya (another solution not likely replacing SAS 9.4) can export parquet files
- You need a SAS license (expensive) as well as some SAS syntax for this
  - But you might ask GenAI to generate such code for you
- The exported csv file can later be read into R

> **i** SAS in the course?
>
> We are not using SAS in the course (and it is no longer available for students to download ... it's to expensive even for a large university ...). Hence, this is just as a future advice if you will work in an organisation with a SAS license (it is still available for empoyers within GU for a monthly cost).



Databases

- a database is a collection of "tables" (tabular data frames)
- Differences between data frames and database tables:
  - Database tables are stored on disk and can be arbitrarily large.
  - Data frames are stored in memory
  - Database tables almost always have indexes; makes it possible to quickly find rows of interest
    - Data frames and tibbles don't have indexes, but `data.tables` do, which is one of the reasons that they're so fast.
  - No fixed row order in a table



Row vs column oriented

- Most classical databases are optimized for rapidly **collecting** data, **not analyzing existing data**.
- These databases are called row-oriented because the data is stored row-by-row, rather than column-by-column like R.
- More recently, there's been much development of column-oriented databases that make analyzing the existing data much faster.

DBMS

Databases are run by database management systems (DBMS's for short), which come in three basic forms

Client-server

run on a powerful central server, which you connect to from your computer (the client). For example PostgreSQL, MariaDB, SQL Server, and Oracle.

Cloud

like Snowflake, Amazon's RedShift, and Google's BigQuery, are similar to client server DBMS's, but they run in the **cloud**. This means that they can easily handle extremely large datasets and can automatically provide more compute resources as needed.

In-process

like SQLite or duckdb, run entirely on your computer.

Relevance

Client-server

Those are sometimes used in our field but if so, you will hopefully get some initial help from the IT department who will provide acess details etc. Such access is then integrated into R and Positron (as well as RStudio). If thi is relevant, you can most likely use tools such as dbplyr to query the data without to much use of SQL itself.

Cloud

Commercial cloud products are less likely to be used for sensitive health data within our field.

In-process

They're great for working with large datasets locally where you (the statistician) is the primary user. SQLite was a good tool in the past (it might still be but ...) nowadays DuckDB is the most prominent implementation!

SQL

- Different database systems may use different query languages.
- The **Structured Query Language (SQL)** is the most widely used for relational databases.
- Defined by international standards (ANSI/ISO).

What is SQL used for?

SQL is used to:

- Retrieve data (`SELECT`)
- Filter data (`WHERE`)
- Aggregate data (`GROUP BY`)
- Combine tables (`JOIN`)
- Modify data (`INSERT`, `UPDATE`, `DELETE`)

SQL Dialects

- SQL is a standard, but implementations differ slightly.
- These variations are called **dialects**.
  - **T-SQL** (Microsoft SQL Server)
  - **PL/SQL** (Oracle)
  - PostgreSQL SQL dialect
  - MySQL SQL dialect

Most core functionality is similar across systems.

A Simple Example

```sql
SELECT name, age
FROM students
WHERE age >= 18
ORDER BY age;
```

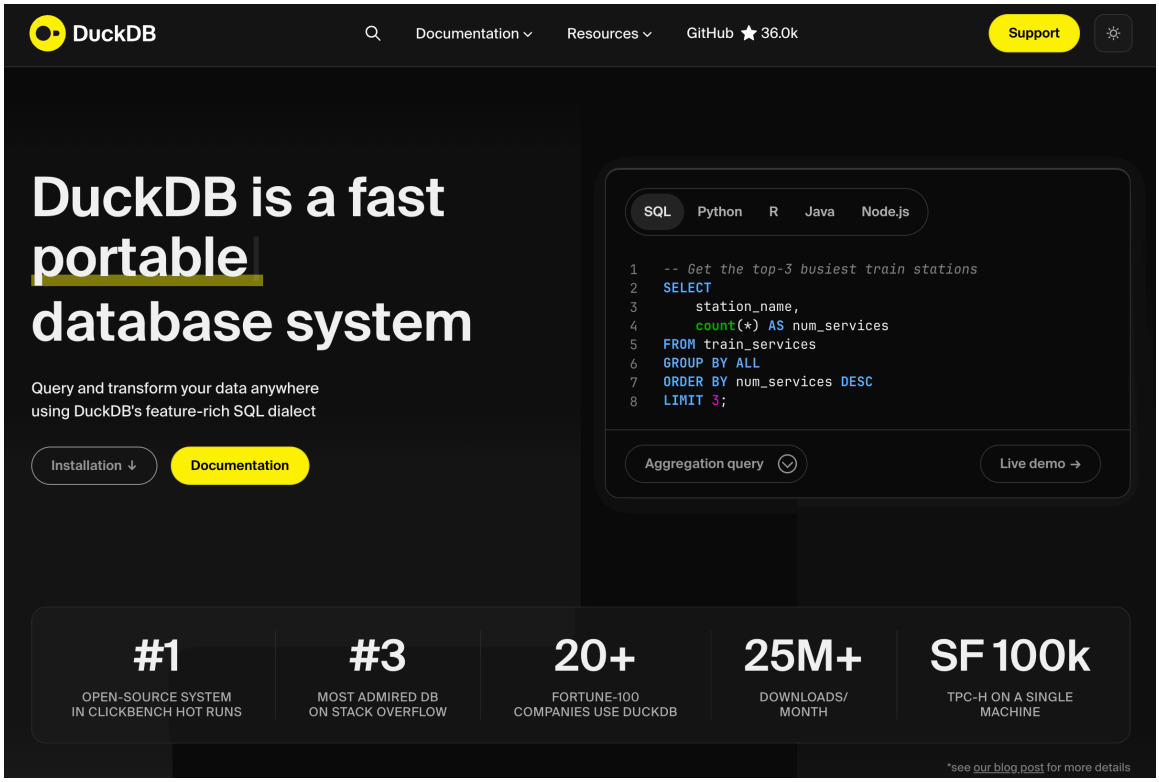
This query:

- Selects two columns

- Filters rows
- Sorts the result

Relevance?

- I don't think SQL is a necessary skill to master for most statisticians
- You should nevertheless be aware of its existence and understand simple code examples as above
- If an employer requires SQL skills, you can probably learn enough if/when needed
- Nevertheless, the more languages/tools etc you know, the more competitive you become (but don't sacrifice statistical knowledge for technical.



DuckDB

- Free and open source (backed by foundation)
- Column based
- Very easy to set up
- Very efficient!
- SQL is an option as API language but not a requirement
- Integration by DBMS and "hidden" SQL under the hood: dbplyr
- A more "native" implementation by duckplyr
  - Better (but perhaps not as widely used yet and therefore a bit less mature/stable)
- Can be used both with disk data and data in RAM

Example

```
# pak::pak("tidyverse/duckplyr")
library(duckplyr)
```

```
Loading required package: dplyr
```

```

Attaching package: 'dplyr'
```

```
The following objects are masked from 'package:stats':

    filter, lag
```

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
The duckplyr package is configured to fall back to dplyr when it encounters an
incompatibility. Fallback events can be collected and uploaded for analysis to
guide future development. By default, data will be collected but no data will
be uploaded.
ℹ Automatic fallback uploading is not controlled and therefore disabled, see
  `?duckplyr::fallback()`.
✔ Number of reports ready for upload: 99.
→ Review with `duckplyr::fallback_review()`, upload with
  `duckplyr::fallback_upload()`.
ℹ Configure automatic uploading with `duckplyr::fallback_config()`.
```

```
✔ Overwriting dplyr methods with duckplyr methods.
ℹ Turn off with `duckplyr::methods_restore()`.
```

```
conflicted::conflicts_prefer(dplyr::filter)
```

```
[conflicted] Will prefer dplyr::filter over any other package.
```

```
# Extra tools to connect to internet data
db_exec("INSTALL httpfs")
db_exec("LOAD httpfs")

# Use some online data which is to big to keep in memory
```

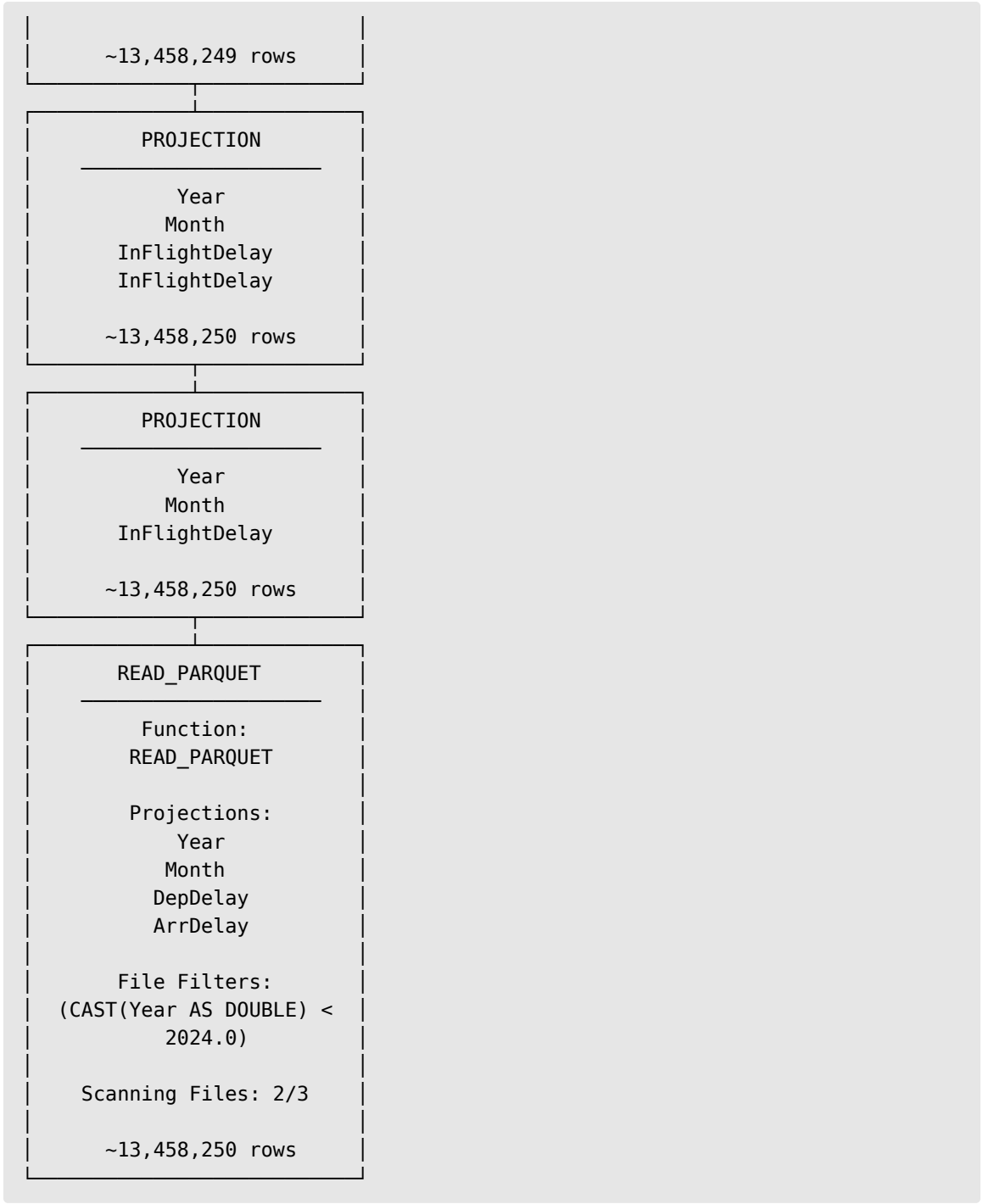
```r
year <- 2022:2024 # therefore, select just 3 years
base_url <- "https://blobs.duckdb.org/flight-data-partitioned/"
files <- paste0("Year=", year, "/data_0.parquet")
urls <- paste0(base_url, files)


# Connect to the data (without downloading)
flights <- read_parquet_duckdb(urls)
# nrow(flights) # It's not in memory!
count(flights, Year) # processing outside memory
```

```
# A duckplyr data frame: 2 variables
   Year       n
  <dbl>   <int>
1  2022 6729125
2  2023 6847899
3  2024 3461319
```

Complex queries can be executed on the remote data. Note how only the relevant columns are fetched and the 2024 data isn't even touched, as it's not needed for the result.

```r
out <-
  flights |>
  mutate(InFlightDelay = ArrDelay - DepDelay) |>
  summarize(
    .by = c(Year, Month),
    MeanInFlightDelay = mean(InFlightDelay, na.rm = TRUE),
    MedianInFlightDelay = median(InFlightDelay, na.rm = TRUE),
  ) |>
  filter(Year < 2024)

out |>
  explain()
```

```
┌─────────────────────────────┐
│       HASH_GROUP_BY         │
│    ─────────────────────    │
│          Groups:            │
│             #0              │
│             #1              │
│                             │
│        Aggregates:          │
│          mean(#2)           │
│         median(#3)          │
```



DuckDB and Data Files

DuckDB can query files directly:

- Parquet
- CSV

- JSON

No data import (to your computers Random Access Memory [RAM]) is required!

Example

- Let's say you used SAS to export your data to a csv file
- DuckDB can query the relevant data from that file directly and collect only the data you actually need!
- (If you knew you only needed i smaller data set you might have performed similar work already in SAS but you often query the data multiple times for different purposes so it still good to have a readable version of everything)

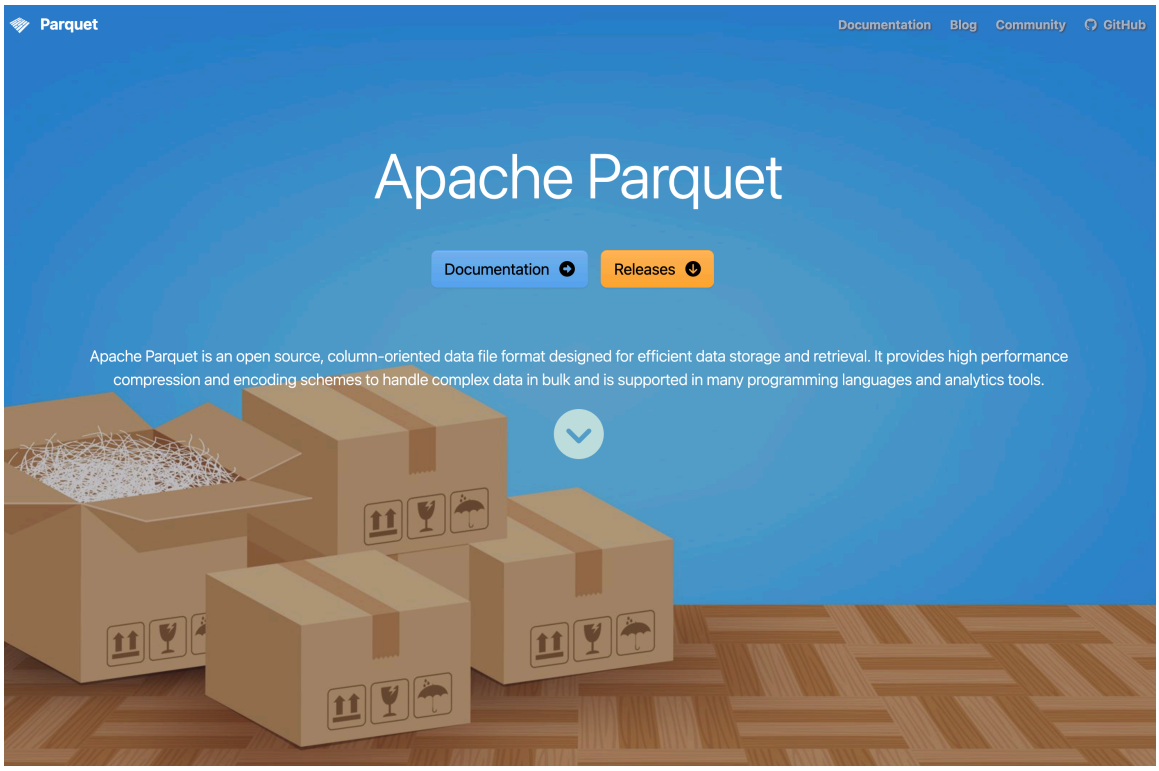
```
# This is not writing from SAS but to illustrate the point :-)
csv_file <- tempfile(fileext = ".csv")
write.csv(NHANES::NHANES, csv_file)

read_csv_duckdb(csv_file) |>
  filter(Age >= 18) |>
  summarise(BMI_mean = mean(as.numeric(BMI), na.rm = TRUE), .by = Gender)
```

```
Warning: There were 2 warnings in `summarise()`.
The first warning was:
ℹ In argument: `BMI_mean = mean(as.numeric(BMI), na.rm = TRUE)`.
ℹ In group 1: `Gender = "male"`.
Caused by warning in `mean()`:
! NAs introduced by coercion
ℹ Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
# A duckplyr data frame: 2 variables
  Gender BMI_mean
  <chr>     <dbl>
1 male       28.7
2 female     28.7
```

Parquet

- From Apache Arrow
- Open format
- flat files with possible hierarchical structure by transparent structure of folders and filenames
- very fast to read and write



Example

- So we might work with CSV + DuckDB but it is even more efficient if we could convert the CSV to a parquet file
- The solution below reads the CSV data and then exports it to a new parquet file
- This means that all data must still fit into memory
  - If it doesn't, do it chunk-wise (piece by piece, 1 M rows at the time or similar)

```
# install.packages("arrow")
library(arrow)
parquet_file <- tempfile(fileext = ".parquet")
read_csv_arrow(csv_file) |>
  write_parquet(parquet_file)

# Now
read_parquet_duckdb(parquet_file)
```

```
# A duckplyr data frame: 77 variables
      C0    ID SurveyYr Gender   Age AgeDecade AgeMonths Race1 Race3
Education
   <int> <int> <chr>    <chr>  <int> <chr>         <int> <chr> <chr>
<chr>
 1     1 51624 2009_10  male      34 " 30-39"        409 White <NA>  High
```

```
School
 2     2 51624 2009_10  male      34 " 30-39"        409 White <NA>  High
School
 3     3 51624 2009_10  male      34 " 30-39"        409 White <NA>  High
School
 4     4 51625 2009_10  male       4 " 0-9"           49 Other <NA>
<NA>
 5     5 51630 2009_10  female    49 " 40-49"        596 White <NA>  Some
Colle…
 6     6 51638 2009_10  male       9 " 0-9"          115 White <NA>
<NA>
 7     7 51646 2009_10  male       8 " 0-9"          101 White <NA>
<NA>
 8     8 51647 2009_10  female    45 " 40-49"        541 White <NA>  College
Gr…
 9     9 51647 2009_10  female    45 " 40-49"        541 White <NA>  College
Gr…
10    10 51647 2009_10  female    45 " 40-49"        541 White <NA>  College
Gr…
# ℹ more rows
# ℹ 67 more variables: MaritalStatus <chr>, HHIncome <chr>, HHIncomeMid <int>,
#   Poverty <dbl>, HomeRooms <int>, HomeOwn <chr>, Work <chr>, Weight <dbl>,
#   Length <dbl>, HeadCirc <dbl>, Height <dbl>, BMI <dbl>,
#   BMICatUnder20yrs <chr>, BMI_WHO <chr>, Pulse <int>, BPSysAve <int>,
#   BPDiaAve <int>, BPSys1 <int>, BPDia1 <int>, BPSys2 <int>, BPDia2 <int>,
#   BPSys3 <int>, BPDia3 <int>, Testosterone <dbl>, DirectChol <dbl>, …
```

tidy data

- `duckplyr` let's you use the familiar dplyr/tidyverse syntax
- If certain steps in your pipeline are not compatible with DuckDB, data might be collected into memory and the pipeline contonous nevertheless
- After filteringen/aggregation etc you can always choose to `collect()` the data and use your ordinary workflow from there

Data files

`.qs2`

A fast, compressed R-specific binary format used for caching by the `{targets}` package. Data must be fully read from disk into memory (RAM), typically via `targets::tar_read()`.

`.sas7bdat`

A proprietary binary format used by SAS and commonly encountered in register data deliveries.

It can be sometimes be imported into R (`haven::read_sas()`, which also allow selection of specific columns at import).

The dataset is read into memory.

`.csv` (Comma-Separated Values)

A plain-text format following a loosely defined standard.
It is human-readable but inefficient for large datasets.
Tools such as DuckDB (via `{duckplyr}`) can read only the required columns and rows and perform filtering and aggregation during import.

`.parquet` (Apache Parquet)

A columnar, compressed, and standardized binary format designed for efficient analytics.
It supports selective column access and integrates well with modern analytical engines such as DuckDB and Arrow.

Archiving

- Swedish law say that we must archive the data (within the public sector).
- General practice is to store every "official document" forever ... which is a long time :-)
- Usualy conceptualized as 500 years.
- Retention (dokumenthanteringsplan/gallringsplan) usually limit the time needed to store source material for statistcal reporting/research etc
  - Varies between organisations but usually 10 years (25 for some research, which is also regulated by the EU)
- Can you read a 25 year old data file?
  - Yes, if storeed as an inefficient CSV-file ...
  - Maybe not if using another less human-readable format ...
- The work of a professional archivist might contain responsibilities for data conversion from time to time (CD-ROMS or floppy disks etc might otherwise not be accasible)
- It seems, however, less common that organizational standardd etc imply that statisticians are limited to certain formats (it is more often up to you and your collegues)

Faster in memory?

- DuckDB and `{duckplyr}` makes it possible to work with data which does not fit into memory
- But what if data would fit (which is more common)?
  - You can use tibbles and dplyr etc as usual ...
    - ... but it might be slooooooooooooow ... for big enough data
    - (Although dplyr also gets some speed improvements from time to time)
- `{data.table}` is much more efficient!

data.table

- An R package (not a data storage format or a database system)
- Comparable to working with `data.frame` in base R or `tibble` in the tidyverse
- Sometimes described as a Domain Specific Language (DSL) within R
- Introduced in 2006 (mature, stable, and widely used — older than the tidyverse)
- Provides faster alternatives to several base R functions: `fifelse()`, `fcase()`, `forder()`, `fread()`, `fcoalesce()`,
  `as.IDate()`, `%chin%`
- Core components implemented in C for performance

- Uses concepts similar to databases:
  - Integer-based keys and indices
  - Efficient joins and grouping
- Reference semantics
- Very flexible and compact syntax
- Strongly appreciated by many statisticians —

  unknown to some — and slightly intimidating to others 🙂

Copy-on-modify

- Assume `df` is a large `data.frame`/`tibble` with many rows and columns, one of them called `old`. Let `f()` be some function.
- What happens when you add a new column using
  `df$new <- f(df$old)` or `df <- mutate(df, new = f(old))`?
- Even though you reuse the name `df`, R typically creates a modified copy of the object.
- In R, objects use *copy-on-modify* semantics:
  - When an object is changed, a new copy is created (if needed).
  - The name `df` is just a reference to an object in memory.
  - After reassignment, `df` refers to the new object.
- The previous object may remain in memory temporarily until garbage collection runs.
- For very large objects, this can result in substantial temporary memory use

Reference semantics

- In `data.table`, reference semantics are used instead.
- You can modify the existing object directly: `df[, new := f(old)]`
- Or delete columns in place: `df[, old := NULL]`
- No full copy is made
- There is still only one object in memory (and it is still called `df`).

Example

```
library(data.table)
dt <- NHANES::NHANES
setDT(dt)
setkey(dt, ID) # Use the ID column as index
# How many adults by gender do we have
dt[Age >= 18, .N, Gender]
```

```
   Gender     N
   <fctr> <int>
1:   male  3686
2: female  3795
```

```r
# Mean BMI by gender
dt[Age >= 18, mean(BMI, na.rm = TRUE), Gender]
```

```
   Gender       V1
   <fctr>    <num>
1:   male 28.67528
2: female 28.66978
```

```r
# Convert all factors to characters
dt[, names(.SD) := lapply(.SD, as.character), .SDcols = is.factor]
```

joins

```
x[i, on, nomatch]
| |  |   |
| |  |   \__ If NULL only returns rows linked in x and i tables
| |  \____ a character vector or list defining match logic
| \_____ primary data.table, list or data.frame
\____ secondary data.table
```

Join type	data.table	SQL	dplyr
Right join	`DT1[DT2]`	RIGHT JOIN	`right_join(DT2, DT1)`
Left join	`DT2[DT1]`	LEFT JOIN	`left_join(DT1, DT2)`
Inner join	`DT1[DT2, nomatch = 0]`	INNER JOIN	`inner_join(DT1, DT2)`
Full join	`merge(DT1, DT2, all = TRUE)`	FULL OUTER JOIN	`full_join(DT1, DT2)`
Anti join	`DT1[!DT2]`	NOT EXISTS	`anti_join(DT1, DT2)`
Semi join	`DT1[DT2, nomatch = 0][, unique(.SD)]`	EXISTS	`semi_join(DT1, DT2)`
Rolling join	`DT1[DT2, on = "date", roll = TRUE]`	—	`join_by()` + `rolling`
Non-equi join	`DT1[DT2, on = .(x >= lower, x <= upper)]`	—	`join_by(x >= lower, x <= upper)`
Update join	`DT1[DT2, value := i.value]`	UPDATE … JOIN	`DT1 %>% left_join(DT2) %>% mutate(...)`

External API

- We previously assumed you get some data delivery from a register holder

- Or mayby you work within the organization where data is collected, and therefore have access to the original data base.
- Some data can also be openly accessed by an Programming Application Interface (API)
- The PX-WEB API is used by a large number of statistical authorities (and others) world-widea to provide access to aggregated data
- The `{pxweb}` R package simplifies the process

Available resources

name	value
api.scb.se	Statistics Sweden
statfin.stat.fi	Statistics Finland
pxwebapi2.stat.fi	Statistics Finland (old version)
statistik.sjv.se	The Swedish Agricultural Agency
fohm-app.folkhalsomyndigheten.se	The Public Health Agency of Sweden
statistik.konj.se	The Swedish National Institute of Economic Research
prognos.konj.se	The Swedish national institute of economic research, forecast database
statdb.luke.fi	LUKE Natural Resources Institute Finland
vero2.stat.fi	Verohallinto - Finnish Tax Administration
px.hagstofa.is	Statistics Iceland
statistik.linkoping.se	Linköping municipality in Sweden
pxwebb2017.vgregion.se	Vastra Gotaland Region in Sweden
bank.stat.gl	Statbank Greenland
px.rsv.is	Icelandic Centre for Retail Studies
statbank.hagstova.fo	Statistics Faroe Islands
data.ssb.no	Statistics Norway
pxweb.asub.ax	Statistics Aland
makstat.stat.gov.mk	State Statistical Office of the Republic of Macedonia
data.stat.gov.lv	Latvia - official statistics
statbank.statistica.md	Statistics Moldova
www.pxweb.bfs.admin.ch	Statistics Switzerland
askdata.rks-gov.net	Statistics Kosovo
trafi2.stat.fi	Finnish Transport Safety Agency
w3.unece.org	United Nations Economic Commission for Europe

name	value
www.grande-region.lu	Portail statistique de la Grande Région
visitfinland.stat.fi	Visit Finland (Rudolf service)
stat.hel.fi	Helsingin seudun aluesarjat -tilastotietokanta
andmed.stat.ee	Estonia - official statistics
pxweb.nordicstatistics.org	Nordic Statistics Database
pxweb.stat.si	SiStat Database

Example

- The first time you need the data, use `pxweb_interactive()` from your console.
- It will guide you through all the necessary steps.
- It will then provide you the necessary R code to replicate the query

---

```
library(pxweb)
```

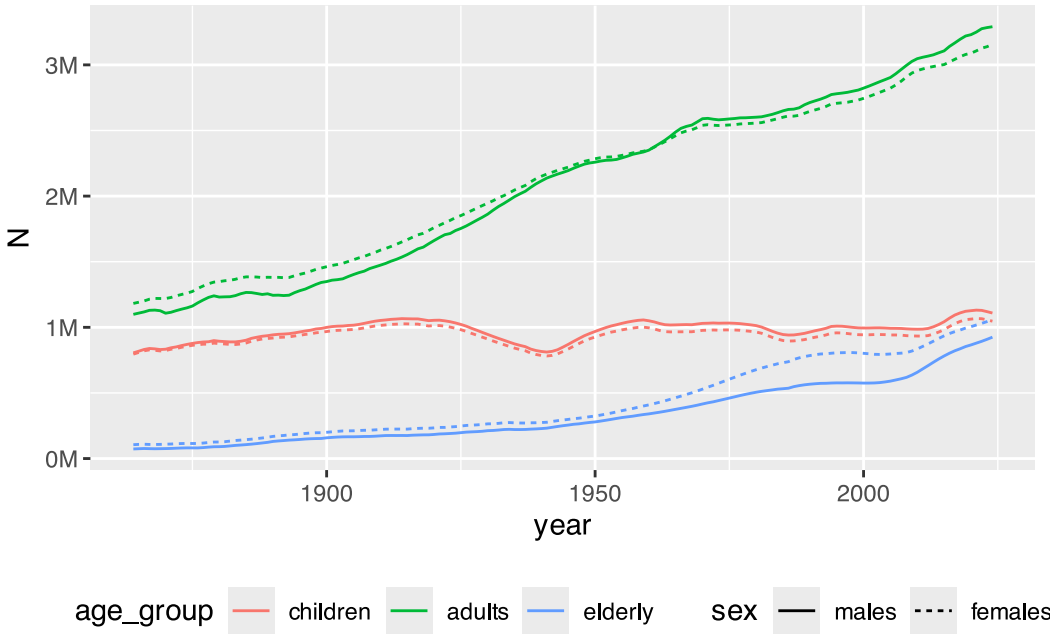
```
pxweb 0.17.1: R Interface to PXWEB APIs.
https://github.com/ropengov/pxweb
```

```
library(data.table)
library(ggplot2)

url <- "https://api.scb.se/OV0104/v1/doris/sv/ssd/BE/BE0101/BE0101A/
BefolkningR1860N"

# PXWEB query
pxweb_query_list <-
  list(
    "Alder" = "*",
    "Kon" = c("1", "2"),
    "ContentsCode" = c("0000053A"),
    "Tid" = as.character(1864:2024)
  )

px_data <-
  pxweb_get(
    url = url,
    query = pxweb_query_list
  )

# A data.table with population numbers
bef <-
```

```
  px_data |>
  as.data.frame() |>
  setDT()

# Some data cleaning
setnames(
  bef,
  c("ålder", "kön", "år", "Antal"),
  c("age", "sex", "year", "N")
)
bef[, `:=`(
  age = as.numeric(gsub(" år", "", age)),
  sex = factor(sex, c("män", "kvinnor"), c("males", "females")),
  year = as.integer(year)
)]
```

```
Warning in eval(jsub, SDenv, parent.frame()): NAs introduced by coercion
```

```
bef <- bef[!is.na(age)] # Remove totals

# Aggregate for age groups
bef[,
  age_group := cut(
    age,
    c(-Inf, 17, 66, Inf),
    c("children", "adults", "elderly")
  )
]
bef_ag <- bef[, .(N = sum(N)), .(age_group, sex, year)]

# Visualize
gg <- bef_ag |>
  ggplot(aes(year, N, color = age_group, linetype = sex)) +
  geom_line() +
  theme(legend.position = "bottom") +
  scale_y_continuous(
    labels = scales::label_number(scale = 1e-6, suffix = "M")
  )
```



Hierachical data

- We like tabular data!
- If we get wide data we can transform it to long data (and vice versa)
- But we might sometimes encounter more hierarchical data structures
- JSON (JavaScript Object Notation) most popular
- XML is another format (used for example by the IRS/Skatteverket sometimes)

```
{
  "system": "ICD-10",
  "chapter": {
    "code": "Chapter I",
    "title": "Certain infectious and parasitic diseases",
    "range": "A00–B99",
    "blocks": [
      {
        "code": "A00–A09",
        "title": "Intestinal infectious diseases",
        "categories": [
          {
            "code": "A00",
            "title": "Cholera",
            "includes": ["Vibrio cholerae infection"],
            "excludes": ["Carrier state"],
            "subcategories": [
              {
                "code": "A00.0",
```

```
            "title": "Cholera due to Vibrio cholerae 01, biovar cholerae"
          },
          {
            "code": "A00.1",
            "title": "Cholera due to Vibrio cholerae 01, biovar eltor"
          }
        ]
      }
    ]
  }
]
}
```


Example

```
# db_exec("INSTALL json")
db_exec("LOAD json")
url <- "https://raw.githubusercontent.com/LuChang-CS/icd_hierarchical_
structure/main/ICD-10-CM/diagnosis_codes.json"
(icd <- duckplyr::read_json_duckdb(url))
```

```
# A duckplyr data frame: 3 variables
   code    desc
children
   <chr>   <chr>
<list>
 1 A00-B99 Certain infectious and parasitic diseases
<df>
 2 C00-D49 Neoplasms
<df>
 3 D50-D89 Diseases of the blood and blood-forming organs and certain …
<df>
 4 E00-E89 Endocrine, nutritional and metabolic diseases
<df>
 5 F01-F99 Mental, Behavioral and Neurodevelopmental disorders
<df>
 6 G00-G99 Diseases of the nervous system
<df>
 7 H00-H59 Diseases of the eye and adnexa
<df>
 8 H60-H95 Diseases of the ear and mastoid process
<df>
 9 I00-I99 Diseases of the circulatory system
<df>
10 J00-J99 Diseases of the respiratory system
```

```
<df>
# ℹ more rows
```

Flattening

So you now have a duckplyr data frame. One of its columns is nested and, with additional data frames as children) If we unnest the children for chapter 1, we see that those children also have children … and so it continous. To get a simple translation between individual codes and their description, you might need to define some recursive function to unnest until you have found the last children in line.

```
(chap1 <- icd |> slice(1) |> collect())
```

```
# A tibble: 1 × 3
  code    desc                                      children
  <chr>   <chr>                                     <list>
1 A00-B99 Certain infectious and parasitic diseases <df [22 × 3]>
```

```
chap1 |>
  select(children) |>
  tidyr::unnest(children)
```

```
# A tibble: 22 × 3
   code    desc
children
   <chr>   <chr>
<list>
 1 A00-A09 Intestinal infectious diseases
<df>
 2 A15-A19 Tuberculosis
<df>
 3 A20-A28 Certain zoonotic bacterial diseases
<df>
 4 A30-A49 Other bacterial diseases
<df>
 5 A50-A64 Infections with a predominantly sexual mode of transmission
<df>
 6 A65-A69 Other spirochetal diseases
<df>
 7 A70-A74 Other diseases caused by chlamydiae
<df>
 8 A75-A79 Rickettsioses
<df>
 9 A80-A89 Viral and prion infections of the central nervous system
```

```
<df>
10 A90-A99 Arthropod-borne viral fevers and viral hemorrhagic fevers
<df>
# ℹ 12 more rows
```

Bibliography

[1] A. Nguyen, *Hands-on healthcare data: taming the complexity of real-world data*, First edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2022.

[2] H. Wickham, M. Çetinkaya-Rundel, and G. Grolemund, "R for Data Science (2e)." [Online]. Available: https://r4ds.hadley.nz/

[3] S. R. Fenk, K. Furu, and I. J. Bakken, "Improve data management in register-based research: Transition from CSV to Parquet," doi: 10.1101/2025.10.15.25337992.

[4] *Data analysis using data.table*. 2026. [Online]. Available: https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html

[5] V. Arel-Bundock, "data.table vs. base vs. dplyr – Vincent Arel-Bundock." [Online]. Available: https://arelbundock.com/posts/dt_tb_df/index.html