Ageing is not just ageing and *Rising Disease Prevalence Signals Epigenetic Degeneration in Humans* — Data Analysis Pipeline

Xavi Marsellach

2025-11-27

Overview

This document reconstructs the master AMAV file and generates all main and supplemental figures used in:

- **Ageing is not just ageing**
- **Rising disease prevalence signals epigenetic degeneration in humans**

The workflow is:

1. Rebuild `output/AMAV_DATA.xlsx` from the published prevalence workbook (`Supplemental_Table_1.xlsx` or `Supplementary_Table_1.xlsx`).
2. Generate **Figure 1** (raw prevalence and loss of studies).
3. Generate **Figures 2–3** (disease-level AMAV and AMAV-derived patterns).
4. Generate **Figure 4** (mental vs physical fold-increase dynamics).
5. Generate **Supplemental Figures**.

All paths are resolved relative to the repository root, so this QMD can be run from within `analysis/` or from the project root.

---

1. Reconstruction of `AMAV_DATA.xlsx` from `Supplemental(ary)_Table_1.xlsx`

Purpose

Prevalence data for **31 diseases** are provided in a single consolidated workbook, `Supplemental(ary)_Table_1.xlsx`.
The Python script in this section reconstructs the **AMAV lines** directly from this workbook and writes the aggregated outputs to `output/AMAV_DATA.xlsx`.

R code to run the Python pipeline

```r
# Works regardless of whether the QMD runs from repo root or inside 'analysis/'.

# --- Find repo root: directory that contains 'scripts/' and a data/Supplemental(ary)_Table_
find_root <- function() {
  cand <- c(".", "..", "../..")
  for (c in cand) {
    script_ok <- file.exists(file.path(c, "scripts", "build_amav_from_supplemental_Table_1.p
    data_sup  <- file.exists(file.path(c, "data", "Supplemental_Table_1.xlsx"))
    data_supp <- file.exists(file.path(c, "data", "Supplementary_Table_1.xlsx"))
    if (script_ok && (data_sup || data_supp)) {
      return(normalizePath(c))
    }
  }
  stop("Can't locate repository root (need 'scripts/' and 'data/Supplemental(ary)_Table_1.xl
}
root <- find_root()

# --- Paths (absolute) ---
script   <- file.path(root, "scripts", "build_amav_from_supplemental_Table_1.py")
inp_sup  <- file.path(root, "data", "Supplemental_Table_1.xlsx")
inp_supp <- file.path(root, "data", "Supplementary_Table_1.xlsx")

if (file.exists(inp_sup)) {
  inp <- inp_sup
} else if (file.exists(inp_supp)) {
  inp <- inp_supp
} else {
  stop("Cannot find Supplemental_Table_1.xlsx or Supplementary_Table_1.xlsx in 'data/'.")
}

outdir <- file.path(root, "output")
out    <- file.path(outdir, "AMAV_DATA.xlsx")
dir.create(outdir, showWarnings = FALSE, recursive = TRUE)

# --- Choose Python interpreter ---
py <- Sys.getenv("PYTHON"); if (!nzchar(py)) py <- Sys.which("python3")
stopifnot(nzchar(py), file.exists(py))

# --- Run and show logs ---
cmd_args <- c(script, inp, out)
cat("$", py, paste(cmd_args, collapse = " "), "\n")
```

```
$ /Users/francesc/miniforge3/envs/ds/bin/python3 /Users/francesc/Library/CloudStorage/Synolo
Docs/1-Feina/3-Home_work/16-Sant_Torne-m-hi_2025/10-Github_repo_ageing_amav/ageing-
```

```
amav/scripts/build_amav_from_supplemental_Table_1.py /Users/francesc/Library/CloudStorage/Sy
Docs/1-Feina/3-Home_work/16-Sant_Torne-m-hi_2025/10-Github_repo_ageing_amav/ageing-
amav/data/Supplemental_Table_1.xlsx /Users/francesc/Library/CloudStorage/SynologyDrive-
Docs/1-Feina/3-Home_work/16-Sant_Torne-m-hi_2025/10-Github_repo_ageing_amav/ageing-
amav/output/AMAV_DATA.xlsx
```

```
res <- system2(py, args = cmd_args, stdout = TRUE, stderr = TRUE)
cat(paste(res, collapse = "\n"), "\n")
```

```
  OK: created /Users/francesc/Library/CloudStorage/SynologyDrive-
Docs/1-Feina/3-Home_work/16-Sant_Torne-m-hi_2025/10-Github_repo_ageing_amav/ageing-
amav/output/AMAV_DATA.xlsx
```

```
# --- Verify output ---
stopifnot(file.exists(out))
cat(sprintf("  OK: created %s\n", out))
```

```
  OK: created /Users/francesc/Library/CloudStorage/SynologyDrive-
Docs/1-Feina/3-Home_work/16-Sant_Torne-m-hi_2025/10-Github_repo_ageing_amav/ageing-
amav/output/AMAV_DATA.xlsx
```

Code for `build_amav_from_supplemental_Table_1.py`

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Rebuild AMAV_DATA.xlsx from a single-table workbook (Supplemental(ary)_Table_1.xlsx).

Input expectation (neutral description):
- One Excel workbook containing a single wide table with:
    Phenotype | Citation | 1941 | 1942 | ... | 2023
- Each row is one prevalence trend (study) for a given phenotype/disease.
- A "Citation" column may be present and is ignored for calculations.

What the script computes:
- For each phenotype, piecewise-linear yearly slopes per study between observed points.
- Yearly MAV (mean of study slopes).
- AMAV: cumulative MAV restricted to the last contiguous block of defined MAV.
- AMAV-POS: if AMAV in that block dips below zero, a positive-limb accumulation
  starting after the AMAV valley (MODE='prev'), optionally including negative MAV.
- Aggregated outputs (written to AMAV_DATA.xlsx):
    • AMAV           : one column per phenotype (indexed by Year) using AMAV-POS
                       if it exists, otherwise AMAV.
    • FOLD_YEARLY    : AMAV normalised by the first positive value (by Year).
    • FOLD_RELATIVE  : Same normalisation but re-indexed 0..n per phenotype
                       (relative time).
```

```
    • LOG_used_column: Phenotype name and number of trends parsed.

Default paths (relative to current working directory):
- Input  : data/Supplemental_Table_1.xlsx or data/Supplementary_Table_1.xlsx
           (auto-discovery also checks analysis/data and ./)
- Output : output/AMAV_DATA.xlsx             (directory is created if missing)

CLI:
    python scripts/build_amav_from_supplemental_Table_1.py [input.xlsx] [output.xlsx]

Programmatic (e.g., from R with reticulate):
    from scripts.build_amav_from_supplemental_Table_1 import build_amav
    out_path = build_amav()  # or build_amav("path/to/input.xlsx", "output/AMAV_DATA.xlsx")
"""

from __future__ import annotations

from pathlib import Path
import math
import re
import sys
import warnings
from typing import Dict, List, Tuple

import numpy as np
import pandas as pd

warnings.filterwarnings("ignore")

# ---- Behaviour switches ----
MODE = "prev"        # accumulate MAV from the previous year
INCLUDE_NEG = True   # include negative MAV in AMAV-POS accumulation
EPS = 1e-12

# ---- Default locations (relative to working directory) ----
DEFAULT_INPUT  = Path("data/Supplemental_Table_1.xlsx")
DEFAULT_OUTPUT = Path("output/AMAV_DATA.xlsx")


# ---------------- Utilities ----------------
def to_number(series: pd.Series) -> pd.Series:
    """Robust numeric conversion: handles %, comma decimals, and thousand separators."""
    def _one(x):
        if pd.isna(x):
            return np.nan
```

```
        if isinstance(x, (int, float, np.number)):
            return float(x)
        s = str(x).strip().replace(" ", "")
        is_pct = s.endswith("%")
        s = s.replace("%", "").replace(",", ".")
        # remove thousands separators like 1.234 or 1'234
        s = re.sub(r"(?<=\d)[\.,'’](?=\d{3}\b)", "", s)
        try:
            v = float(s)
        except ValueError:
            return np.nan
        return v / 100.0 if is_pct else v
    return series.apply(_one)


def runs_true(mask: np.ndarray) -> List[Tuple[int, int]]:
    """Return [(start, end)] index intervals where mask is True and contiguous."""
    idx = np.where(mask)[0]
    if idx.size == 0:
        return []
    runs, start, prev = [], idx[0], idx[0]
    for k in idx[1:]:
        if k == prev + 1:
            prev = k
        else:
            runs.append((start, prev))
            start = prev = k
    runs.append((start, prev))
    return runs


# -------- AMAV / AMAV-POS core --------
def build_perstudy_linearslope(raw_df: pd.DataFrame) -> pd.DataFrame:
    """
    raw_df columns: Year | T1 | T2 | ... | Tk
    For each trend, build a piecewise-constant slope between successive observed points.
    """
    df = raw_df.sort_values("Year").reset_index(drop=True)
    years = df["Year"].to_numpy()
    out = pd.DataFrame({"Year": years})

    for col in df.columns:
        if col == "Year":
            continue
        s = pd.to_numeric(df[col], errors="coerce")
```

```
        mask = s.notna().to_numpy()
        obs_y = years[mask]
        obs_v = s[mask].to_numpy(float)

        arr = np.full(len(years), np.nan, float)
        for i in range(len(obs_y) - 1):
            y0, y1 = obs_y[i], obs_y[i + 1]
            v0, v1 = obs_v[i],  obs_v[i + 1]
            if y1 > y0:
                slope = (v1 - v0) / (y1 - y0)
                seg = (years > y0) & (years <= y1)
                arr[seg] = slope
        out[col] = arr

    return out


def build_summary_mav_amav(slope_df: pd.DataFrame) -> pd.DataFrame:
    """
    - MAV: mean across study slopes per year.
    - AMAV: cumulative MAV within the last contiguous block where MAV is defined.
    - AMAV-POS: if AMAV dips below zero in that block, accumulate MAV from the year
      after the valley (MODE='prev'), including negative MAV if INCLUDE_NEG=True.
    """
    df = slope_df.sort_values("Year").reset_index(drop=True)
    years = df["Year"].to_numpy()

    study_cols = [c for c in df.columns if c != "Year"]
    M = df[study_cols].to_numpy(float)

    datapoints = np.sum(~np.isnan(M), axis=1)
    with np.errstate(all="ignore"):
        mav = np.nanmean(M, axis=1)

    # AMAV within the last contiguous MAV block
    amav = np.full(len(years), np.nan, float)
    finite_mav = ~np.isnan(mav)
    mav_runs = runs_true(finite_mav)
    last_s = last_e = None
    if mav_runs:
        last_s, last_e = mav_runs[-1]
        acc = 0.0
        # i from last_s+1 .. last_e+1 inclusive (AMAV[i] integrates mav[i-1])
        for i in range(last_s + 1, last_e + 2):
            prev = i - 1
```

```
            if not math.isnan(mav[prev]):
                acc += mav[prev]
            amav[i] = acc

    summary = pd.DataFrame({
        "Year": years,
        "DataPoints": datapoints,
        "MAV": mav,
        "AMAV": amav,
    })

    # AMAV-POS only if AMAV dips below zero in the final block
    if last_s is None:
        return summary

    lo, hi = last_s + 1, last_e + 1  # indices where AMAV is defined
    window_am = amav[lo:hi + 1]
    has_negative = np.isfinite(window_am).any() and (window_am[np.isfinite(window_am)] < -EP
    if not has_negative:
        return summary

    valley_rel = int(np.nanargmin(window_am))
    valley = lo + valley_rel

    amav_pos = np.full(len(amav), np.nan, float)
    run = 0.0

    # Start at valley+1 for MODE='prev'
    start_i = max(valley + 1, lo) if MODE == "prev" else max(valley, lo)
    for i in range(start_i, hi + 1):
        idx_mav = (i - 1) if MODE == "prev" else i
        v = mav[idx_mav]
        if np.isfinite(v):
            contrib = v if INCLUDE_NEG else max(v, 0.0)
            run += contrib
            amav_pos[i] = run

    insert_at = list(summary.columns).index("AMAV") + 1
    summary.insert(insert_at, "AMAV-POS", amav_pos)
    return summary


def first_positive_baseline(s: pd.Series) -> float:
    """First non-zero, non-NA value."""
    for v in s:
```

```
        if pd.notna(v) and v != 0:
            return float(v)
    return np.nan


# -------- Read single table from Supplemental(ary)_Table_1 --------
def find_table_sheet(xlsx: Path) -> str:
    """Find the sheet whose (0,0) cell is 'Phenotype' (fallback: first sheet)."""
    xls = pd.ExcelFile(xlsx)
    for sh in xls.sheet_names:
        test = pd.read_excel(xlsx, sheet_name=sh, nrows=1, header=None)
        if not test.empty and str(test.iat[0, 0]).strip().lower() == "phenotype":
            return sh
    return xls.sheet_names[0]


def read_single_table(input_xlsx: Path) -> tuple[pd.DataFrame, List[int]]:
    """
    Returns:
    - df: columns = Phenotype | <year_int_1> | ... | <year_int_k>
    - year_order: sorted list of integer years

    Drops the 'Citation' column. Removes a header-like duplicate row if needed.
    """
    sheet = find_table_sheet(input_xlsx)
    df = pd.read_excel(input_xlsx, sheet_name=sheet, header=0)
    if df.empty:
        raise SystemExit("Input sheet appears to be empty.")

    if "Phenotype" not in df.columns:
        raise SystemExit("Cannot find 'Phenotype' column in header.")

    # Drop Citation column if present
    drop_cit = [c for c in df.columns if str(c).strip().lower() == "citation"]
    if drop_cit:
        df = df.drop(columns=drop_cit)

    # Identify year columns (numeric or string)
    year_cols_orig = []
    for c in df.columns:
        if c == "Phenotype":
            continue
        if isinstance(c, (int, float)) and 1800 <= int(c) <= 2100:
            year_cols_orig.append(c)
        elif isinstance(c, str) and re.fullmatch(r"\s*\d{4}\s*", c or ""):
```

```
            year_cols_orig.append(c)

    if not year_cols_orig:
        raise SystemExit("No year-like columns found in the table.")

    # Normalize year columns to integer names
    year_map = {c: int(str(c).strip()) for c in year_cols_orig}
    df = df.rename(columns=year_map)
    year_cols = sorted(year_map.values())

    # Keep only Phenotype + years
    keep = ["Phenotype"] + year_cols
    df = df[keep].copy()
    df["Phenotype"] = df["Phenotype"].astype(str).str.strip()

    # Remove bogus header-like row
    df = df[df["Phenotype"].str.strip().str.lower() != "phenotype"]

    # Coerce year values to numeric
    for c in year_cols:
        df[c] = to_number(df[c])

    return df, year_cols


def build_disease_series(df: pd.DataFrame, year_order: List[int]) -> Dict[str, pd.Series]:
    """
    For each phenotype, build a per-disease AMAV/AMAV-POS series,
    preferring AMAV-POS when present, else AMAV.
    """
    series_by: Dict[str, pd.Series] = {}
    years = list(year_order)

    for disease, grp in df.groupby("Phenotype", dropna=True):
        raw = pd.DataFrame({"Year": years})
        # Each row in grp is one trend (T1..Tk)
        for i, (_, row) in enumerate(grp.iterrows(), start=1):
            raw[f"T{i}"] = row[years].values

        # Skip if all trends are NA
        if raw.drop(columns=["Year"]).isna().all().all():
            continue

        slope = build_perstudy_linearslope(raw)
        summary = build_summary_mav_amav(slope)
```

```
        chosen_col = "AMAV-POS" if ("AMAV-POS" in summary.columns and summary["AMAV-POS"].no
        ser = summary.set_index("Year")[chosen_col].copy()
        ser.name = disease
        series_by[disease] = ser

    return series_by


# ---------------- I/O helpers ----------------
def resolve_input_output(
    input_xlsx: Path | None,
    output_xlsx: Path | None
) -> tuple[Path, Path]:
    """Resolve paths with sensible defaults and fallbacks."""
    cwd = Path.cwd()

    # Input resolution
    if input_xlsx is None:
        candidates = [
            cwd / "data" / "Supplemental_Table_1.xlsx",
            cwd / "data" / "Supplementary_Table_1.xlsx",
            cwd / "analysis" / "data" / "Supplemental_Table_1.xlsx",
            cwd / "analysis" / "data" / "Supplementary_Table_1.xlsx",
            cwd / "Supplemental_Table_1.xlsx",
            cwd / "Supplementary_Table_1.xlsx",
        ]
        inp = next((p for p in candidates if p.exists()), None)
        if inp is None:
            raise SystemExit(
                "Input file not found. Expected at one of: "
                "data/Supplemental_Table_1.xlsx, data/Supplementary_Table_1.xlsx, "
                "analysis/data/Supplemental_Table_1.xlsx, analysis/data/Supplementary_Table_
                "./Supplemental_Table_1.xlsx, ./Supplementary_Table_1.xlsx"
            )
    else:
        inp = Path(input_xlsx)
        if not inp.exists():
            raise SystemExit(f"Input file not found: {inp}")

    # Output resolution
    out = Path(output_xlsx) if output_xlsx is not None else (cwd / DEFAULT_OUTPUT)
    out.parent.mkdir(parents=True, exist_ok=True)

    return inp, out
```

```python


def write_workbook(amav_df: pd.DataFrame, folds_y: pd.DataFrame,
                   folds_rel: pd.DataFrame, log_df: pd.DataFrame, out_path: Path) -> None:
    """Write the 4 output sheets to an Excel workbook."""
    with pd.ExcelWriter(out_path, engine="xlsxwriter") as xw:
        amav_df.reset_index().rename(columns={"index": "Year"}).to_excel(
            xw, index=False, sheet_name="AMAV"
        )
        folds_y.reset_index().rename(columns={"index": "Year"}).to_excel(
            xw, index=False, sheet_name="FOLD_YEARLY"
        )
        folds_rel.rename_axis("RelYear").reset_index().to_excel(
            xw, index=False, sheet_name="FOLD_RELATIVE"
        )
        log_df.to_excel(xw, index=False, sheet_name="LOG_used_column")


# ---------------- Orchestrator ----------------
def build_amav(input_xlsx: str | Path | None = None,
               output_xlsx: str | Path | None = None) -> Path:
    """
    High-level entry point. Returns the Path of the created Excel workbook.
    """
    inp, out = resolve_input_output(
        Path(input_xlsx) if input_xlsx is not None else None,
        Path(output_xlsx) if output_xlsx is not None else None,
    )

    df, year_order = read_single_table(inp)
    series_by = build_disease_series(df, year_order)
    if not series_by:
        raise SystemExit("No diseases could be parsed into AMAV/AMAV-POS series.")

    # AMAV: union of years across all diseases
    all_years = sorted(set().union(*[set(s.index) for s in series_by.values()]))
    amav_df = pd.DataFrame(index=all_years)
    for disease, s in series_by.items():
        amav_df[disease] = s.reindex(all_years)

    # FOLD_YEARLY: normalised by first positive baseline, indexed by Year
    folds_y = pd.DataFrame(index=amav_df.index)
    for col in amav_df.columns:
        v = amav_df[col]
        b = first_positive_baseline(v.dropna())
```

```
        folds_y[col] = v / b if pd.notna(b) else np.nan

    # FOLD_RELATIVE: same but re-indexed 0..n per disease
    rel_dict: Dict[str, pd.Series] = {}
    max_len = 0
    for col in amav_df.columns:
        v = amav_df[col].dropna()
        b = first_positive_baseline(v)
        rel = (v / b) if pd.notna(b) else pd.Series(dtype=float)
        rel.index = range(len(rel))
        rel_dict[col] = rel
        max_len = max(max_len, len(rel))

    folds_rel = pd.DataFrame(index=range(max_len))
    for col, s in rel_dict.items():
        folds_rel[col] = s.reindex(folds_rel.index)

    # LOG sheet: phenotype + number of trends parsed
    log_df = (
        df.groupby("Phenotype", dropna=True)["Phenotype"]
        .count()
        .rename("Trends")
        .reset_index()
        .rename(columns={"Phenotype": "Disease"})
        .sort_values("Disease")
    )

    write_workbook(amav_df, folds_y, folds_rel, log_df, out)
    print(f"  OK: created {out}")
    return out


# ---------------- CLI ----------------
def main() -> None:
    # CLI overrides: [input.xlsx] [output.xlsx]
    argv = sys.argv
    in_arg = Path(argv[1]) if len(argv) >= 2 else None
    out_arg = Path(argv[2]) if len(argv) >= 3 else None
    build_amav(in_arg, out_arg)


if __name__ == "__main__":
    main()
```

2. Main Figures — Ageing is not just ageing

All subsequent chunks assume that:

- `root` points to the repository root (defined above); and
- `output/AMAV_DATA.xlsx` has been created by the previous step.

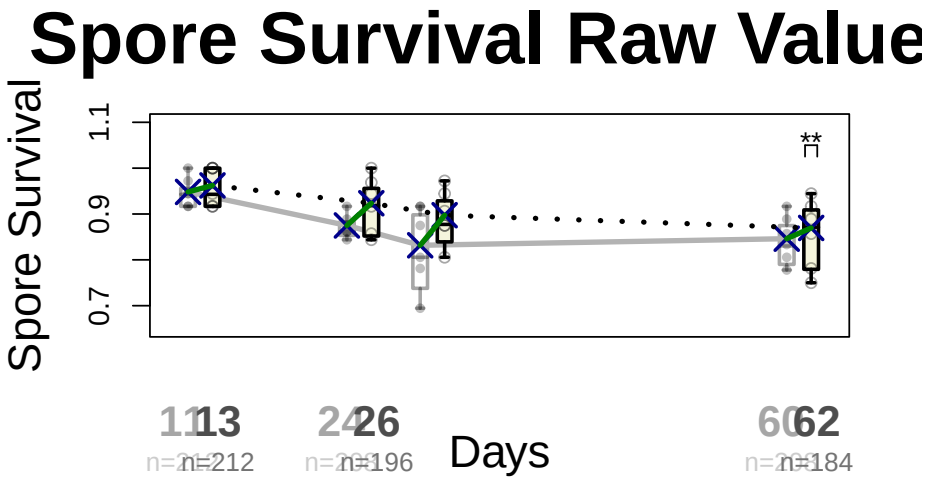
We call the existing R scripts in `scripts/`, which read from:

- `data/*.csv` for **Figure 1**
- `output/AMAV_DATA.xlsx` (sheets `AMAV`, `FOLD_YEARLY`, `FOLD_RELATIVE`) for **Figures 2–4**.

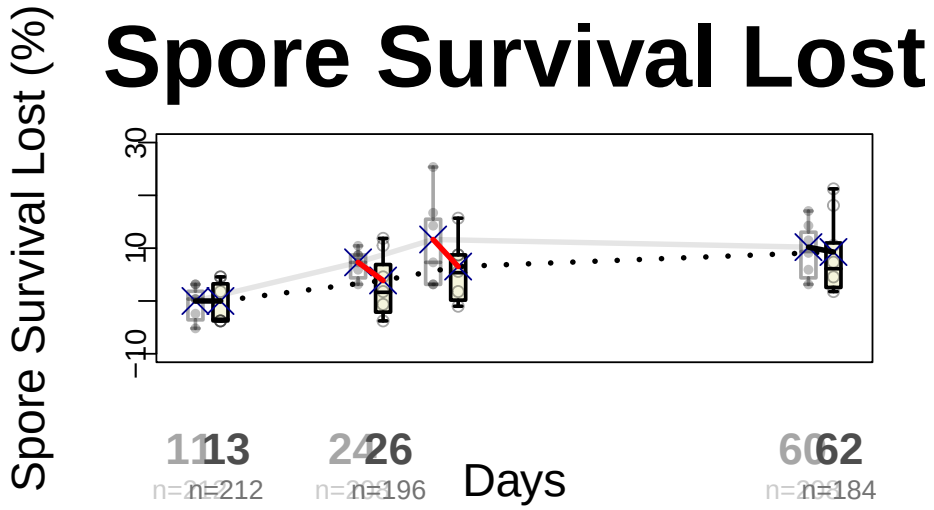
The scripts write figure files into `output/` (or subfolders).

2.1 Figure 1 — Raw prevalence and loss of studies

```
# Temporarily switch working directory to repo root
old_wd <- getwd()
setwd(root)
on.exit(setwd(old_wd), add = TRUE)

source(file.path("scripts", "Figure_1_RAW.R"))
```




```
source(file.path("scripts", "Figure_1_LOSS.R"))
```



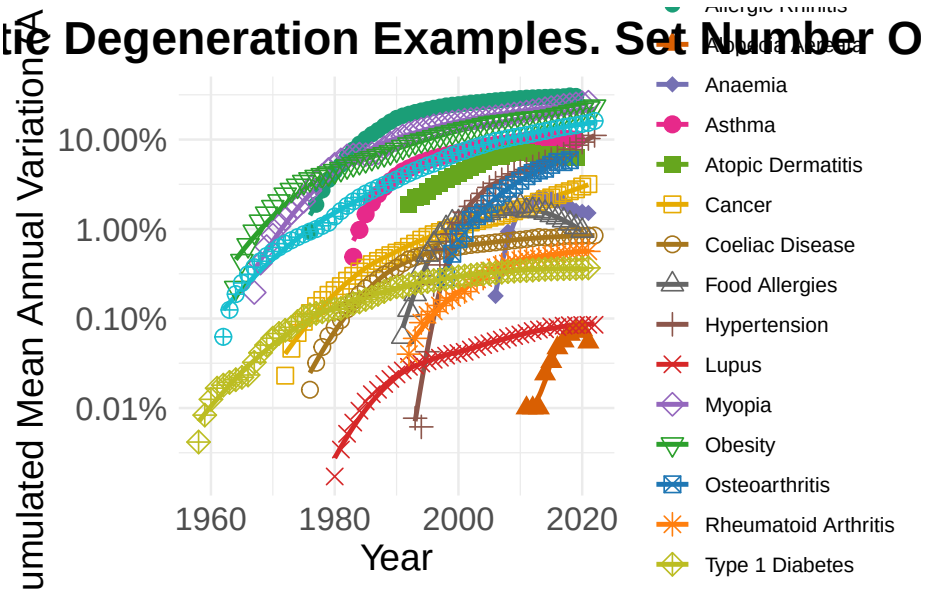

```
cat("  Figure 1 scripts executed (RAW + LOSS).\n")
```

```
  Figure 1 scripts executed (RAW + LOSS).
```

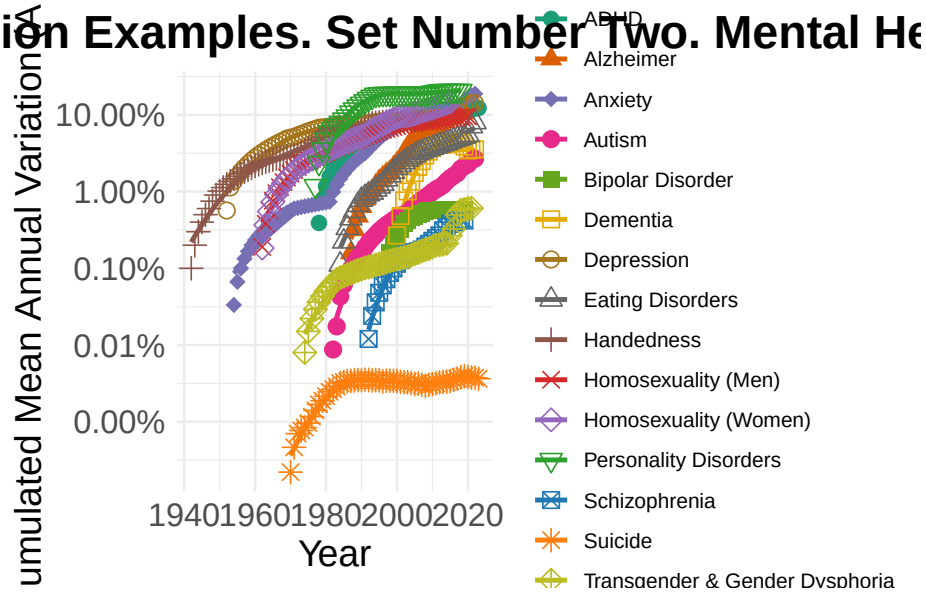
2.2 Figures 2 and 3 — AMAV-derived disease trajectories

```
old_wd <- getwd()
setwd(root)
on.exit(setwd(old_wd), add = TRUE)

source(file.path("scripts", "Figure_2_3_from_AMAV_DATA.R"))
```

```
`geom_smooth()` using formula = 'y ~ x'
```




```
`geom_smooth()` using formula = 'y ~ x'
```



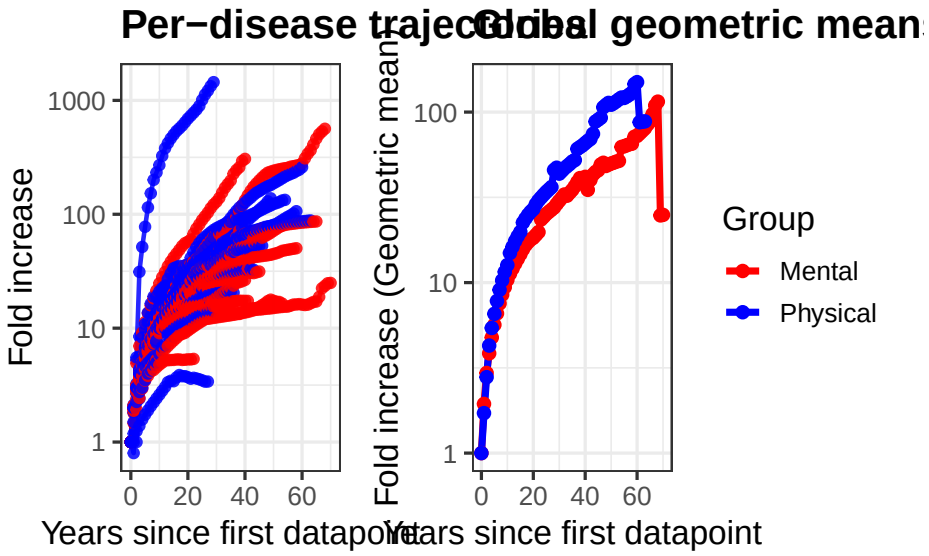

```
cat("  Figures 2 and 3 generated from AMAV_DATA.xlsx.\n")
```

```
  Figures 2 and 3 generated from AMAV_DATA.xlsx.
```

2.3 Figure 4 — Mental vs physical fold-increase dynamics

```
old_wd <- getwd()
setwd(root)
on.exit(setwd(old_wd), add = TRUE)

if (file.exists(file.path("scripts", "remove_monotonic_prefix_AMAV_FR.R"))) {
  source(file.path("scripts", "remove_monotonic_prefix_AMAV_FR.R"))
}

source(file.path("scripts", "FIGURE_4_PANELS_A_and_B.R"))
```

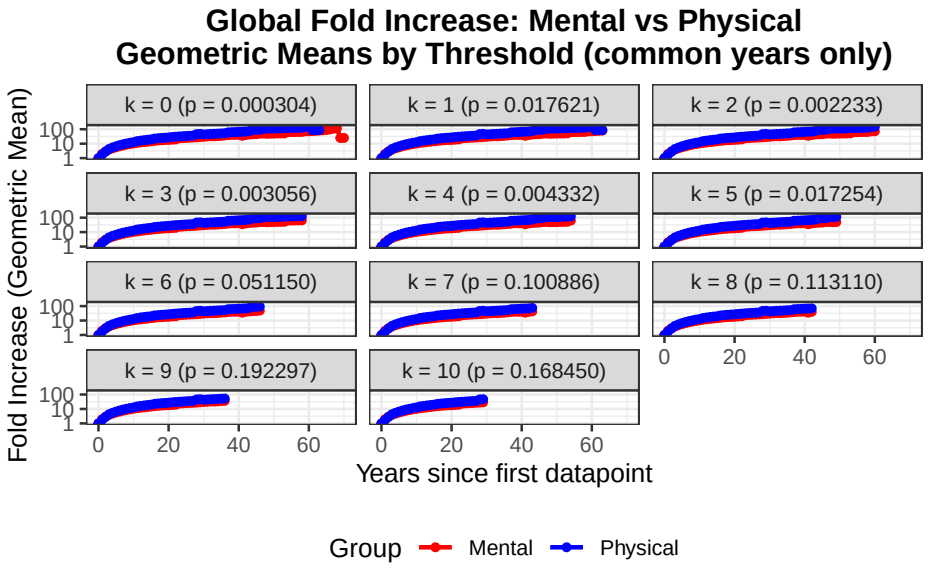
```
Attaching package: 'dplyr'

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

    filter, lag

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

    intersect, setdiff, setequal, union


Attaching package: 'tidyr'

The following object is masked from 'package:reshape2':

    smiths

Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```




```
source(file.path("scripts", "FIGURE_4_PANEL_C.R"))
```

```
Warning: Removed 7 rows containing missing values or values outside the scale range
(`geom_point()`).
```

```
Warning: Removed 7 rows containing missing values or values outside the scale range
(`geom_point()`).
```




```
cat("  Figure 4 panels generated.\n")
```

```
  Figure 4 panels generated.
```

3. Supplemental Figures

Supplemental figures are generated from the same AMAV_DATA.xlsx file plus additional scripts for specific panels.

3.1 Supplemental Figure S1B (example)

```
old_wd <- getwd()
setwd(root)
on.exit(setwd(old_wd), add = TRUE)

if (file.exists(file.path("scripts", "FIGURE_S1B.R"))) {
  source(file.path("scripts", "FIGURE_S1B.R"))
  cat("  Supplemental Figure S1B generated.\n")
} else {
  cat("  FIGURE_S1B.R not found; skipping.\n")
}
```

```
Error: `path` does not exist: 'output/AMAV_DATA-no-monotonic_AUTO.xlsx'
```

3.2 Additional supplemental figures via Python script

```
old_wd <- getwd()
setwd(root)
on.exit(setwd(old_wd), add = TRUE)

sup_script <- file.path("scripts", "make_supplemental_figres.py")
if (file.exists(sup_script)) {
  py2 <- Sys.getenv("PYTHON"); if (!nzchar(py2)) py2 <- Sys.which("python3")
  stopifnot(nzchar(py2), file.exists(py2))

  cmd_args2 <- c(sup_script)
  cat("$", py2, paste(cmd_args2, collapse = " "), "\n")
  res2 <- system2(py2, args = cmd_args2, stdout = TRUE, stderr = TRUE)
  cat(paste(res2, collapse = "\n"), "\n")
  cat("  Additional supplemental figures generated by Python.\n")
} else {
  cat("  make_supplemental_figres.py not found; skipping Python supplemental figures.\n")
}
```

```
  make_supplemental_figres.py not found; skipping Python supplemental figures.
```

4. Summary

This Quarto document provides an end-to-end, executable record of:

1. Reconstruction of `AMAV_DATA.xlsx` from the published prevalence workbook.

2. Generation of all **main figures** (1–4).

3. Generation of the **supplemental figures**.

It is intended as an internal, fully reproducible logbook of how the figures for both manuscripts are produced from the underlying data.